Neural Network Ensembles

LARS KAI HANSEN AND PETER SALAMON

Abstract—**We propose several means for improving the performance and training of neural networks for classification. We use crossvalidation as a tool for optimizing network parameters and architecture. We show further that the remaining residual "generalization" error can be reduced by invoking ensembles of similar networks.**

Index Terms—**Crossvalidation, fault tolerant computing, neural networks, *N*-version programming.**

I. Introduction

RECENT schemes for training neural networks involving hidden neurons have caused a resurgence of interest in nonalgorithmic supervised learning. A supervised learning scheme is implemented using a *database* which consists of a set of input patterns (a sample from the set of possible inputs) together with the corresponding targets (classifications). The objective of the training is to let the trainee extract relevant information from the database in order to classify future input patterns: in other words to *generalize* [1].

The present paper advocates a number of procedures for analyzing and improving classification by networks. The procedures that we introduce require very little specifics of the neural network involved; we may think of the network merely as a pattern recognition device with tunable parameters which has been trained under supervision, i.e., which has been shown a given set of input–output pairs. For most architectures, these tunable parameters take the form of the weights w. This includes feed-forward nets [2], Boltzmann machines [3], Madalines [4], recurrent nets [5], etc.

As our principal device for analyzing performance, we apply the concept of *crossvalidation* to neural networks. The basic idea is to use only a portion of the database in training the network and to use the rest of the database in assessing the capacity of the network to generalize. Such a procedure is in fact well-known within statistical pattern recognition; we argue below that it is also an important procedure for neural networks. Once we can assess the performance of the network, we can optimize such performance by varying network characteristics and architecture.

A residual error will typically remain even after optimizing all available network characteristics [6]. To further reduce this error we propose to use a device from fault tolerant computing [7]. We run not a single network but an *ensemble* of networks, each of which have been trained on the same database. The basic idea is to classify a given input pattern by obtaining a classification from each copy of the network and then using a consensus scheme to decide the collective classification by vote.

II. Crossvalidation for Network Optimization

For supervised learning we employ a database as described above including a representative sample of the set of possible input patterns i^β and the corresponding output patterns o^β. In general the network is supposed to deduce a "noisy rule" from the database. This rule is stored in the particular matrix of weights w which is selected so as to minimize the number of errors in the assignments $i^\beta \rightarrow \hat{o}(i^\beta)$ made by the network on a subset of the inputs. Hence the input–output relation can be encoded by a well trained w, i.e., by a significantly lower number of bits than were contained in the database used in its training. The extra bits in the database represent redundancy and noise which we try to avoid modeling. From this perspective, the choice of a network architecture amounts to a choice of parametrization for this input–output relation. Crossvalidation is the standard tool in deciding between alternative choices of parametrization for a data set by statistical methods [8].

By contrast, the standard procedure for training a neural network involves training on the complete database by minimizing the accumulated misclassification of inputs in the dataset [2]. Since the overall goal is not to minimize errors on the dataset but rather to minimize misclassification on a much larger set of conceivable inputs, crossvalidation gives a much better measure of expected ability to generalize.

As an example, consider choosing the number of hidden neurons in the network. Statistically this can be interpreted as the size of the parameter set used to model the data. As measured by the ability to generalize there is a limit to the desired size of any network architecture. This follows from the basic observation that it might not be optimal to train a neural network to perfection on a given finite database because of noise contamination. But then exactly how much should it be trained in order to

Manuscript received March 20, 1989; revised March 14, 1990. Recommended for acceptance by R. De Mori. The work of L. K. Hansen was supported by the Danish Teknologiradet. The work of P. Salamon was supported by the Naval Ocean Systems Center under Contract N66001-87-D0136.

L. K. Hansen was with the Department of Mathematical Sciences, San Diego State University, San Diego, CA 92182. He is now with ANDREX Radiation Products A/S, Halfdansgade 8, DK-2300 Copenhagen S, Denmark.

P. Salamon is with the Department of Mathematical Sciences, San Diego State University, San Diego, CA 92182.

IEEE Log Number 9036965.

produce the best generalizations? An unbiased estimate can be obtained by bipartitioning the database into specific training and test sets. Overtraining will then show up as poorer performance on the test set, i.e., when crossvalidating.

The method is not limited to choosing the number of neurons. Crossvalidation also allows comparing different architectures, i.e., which neurons are hooked to which others. For example, one can use it to compare the performance of one and two layers of hidden neurons in a feed forward network. One can also use it to select the number of copies of the network used in parallel. This is the topic of the next section.

III. Ensembles of Neural Networks

Standard practice [2], [6], [9], [10], [11] dictates that we perform some trial tunings to find an acceptable architecture and tuning for the network and then trust all future classifications to the best network we find. It turns out however that it is preferable to keep the complete set of networks (or at least a *screened* subset) and run them all with an appropriate collective decision strategy. In analogy with physical theory, we will refer to the set of neural networks used as an *ensemble* [1].

Ensembles are desirable due to the basic fact that selection of the weights w is an optimization problem with many local minima. All global optimization methods in the face of many local minima yield "optimal" parameters (w) which differ greatly from one run of the algorithm to the next, i.e., which show a great deal of randomness stemming from different initial points (w^0) and sequencing of the training examples. This randomness tends to differentiate the errors of the networks, so that the networks will be making errors on different subsets of the input space.

The networks will differ in the values of the weights w for several reasons discussed in the following section. These different weights correspond to different ways of forming generalizations about the patterns inherent in the training set. As each network makes generalization errors on different subsets of the input space, we shall argue that the collective decision produced by the ensemble is less likely to be in error than the decision made by any of the individual networks. Our argument is further supported by our experiments. The conclusion is that the ensemble can be far less fallible than any one network.

This is dramatically illustrated in Fig. 1 which shows three networks trained to identify points in the first and third quadrants. While none of the three individual networks have correctly learned the rule, their majority vote does remarkably well. The experiment is discussed further in the experimental section below.

The performance in Fig. 1 is unusual. As shown by Eckhardt and Lee [7], using collective performance for fault-tolerant applications by either hard or software does not necessarily improve the reliability of the output obtained. Central to this issue is the extent to which errors tend to coincide. In fact, for independently trained neural

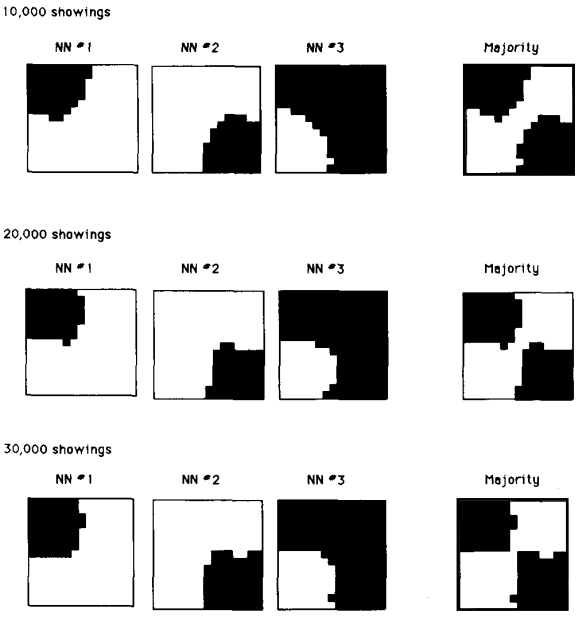


Fig. 1. Three networks trained to identify inputs (x, y) as belonging to the first and third quadrants (black) or the second and fourth quadrants (white). The figure shows the classification by each of the networks and their majority decisions.

networks, the extent of coincidence is likely to be low. We begin our analysis of ensemble performance with a discussion of why this is the case.

IV. The Problem of Many Local Minima

A. How Local Minima Differ

The multiple minima structure of the objective function used in tuning the weights w follows from the symmetries of the networks. While all symmetry-related nets produce identical output, the system is likely to end up in "mixed" configurations that resemble different symmetry related solutions in different regions. It is well known from the analysis of disordered materials [12] that multiple discrete symmetries lead to frustration and a combinatorial explosion of local minima.

To illustrate this fact, we specify a particular feed forward network of the type for which Rummelhart *et al.* devised the Back propagation method. We consider a three layer network combined from an input slab, a layer of hidden neurons and an output slap as depicted in Fig. 2. The computing neurons perform calculations of the form

$$s_p = g\left(\sum_q w_{p,q} s'_q\right) \tag{1}$$

where s'_q is a neuron in the layer preceding s_p and $g(\cdot)$ is the sigmoid function as exemplified in Fig. 2. We have represented the tunable *thresholds* by additional fixed-state neurons in the preceding slabs.

With any form of the sigmoid function there is a discrete symmetry (i.e., a transformation leaving the output invariant) related to a simple permutation of indexes

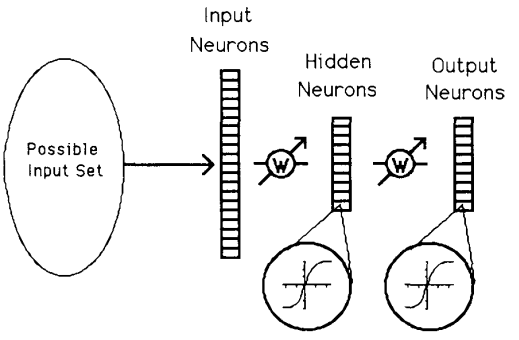


Fig. 2. A Rumelhardt architecture is shown using 20 input neurons, a single layer of 10 hidden neurons, and an output layer of 10 neurons. This architecture is employed for the experiments on classification of random patterns described below. The inset shows the sigmoid function used as the nonlinearity in the neurons.

among the hidden neurons. As the typical sigmoid has the two properties of antisymmetry and saturation for large arguments, additional symmetries follow. From the former property arrives a discrete symmetry by inverting the state of a given hidden neuron and changing signs on the corresponding w's. The latter leads to an approximate continuous symmetry. Due to the saturation we can multiply the weights approaching a given neuron by an arbitrary scale factor without significantly changing the output. This approximate continuous symmetry indicates that the solution space has the form of a family of "cones."

B. Different Search Methods and Objectives

Despite the fact that our ultimate objective function is the ability to generalize, the objective for the tuning phase of the project is rather the sum of squared deviations between the outputs obtained for the training set and the outputs produced by the network. Either objective suffers from the problem of many local minima.

Tuning algorithms will end up at different w's following independent runs for many reasons which may include different starting weights, different sequences of the training samples, different partitionings of the database used during training, or different random numbers generated during a stochastic search algorithm. Finding good algorithms for solving global optimization problems in the face of many local minima is a highly active field without firm recipes. We note that the conventional implementation of "gradient descent" for neural nets [11] contains a significant component of "self-annealing." By this we mean that it allows occasional "uphill" moves characteristic of methods like simulated annealing [13] rather than greedy algorithms like steepest descent. It is customary to update the weights using only an estimate of the local gradient of the error function obtained from one or at most a few input samples of the database. The noise-level of this estimate can be suppressed in part by "momentum" smoothing parameters [6]. The approach resembles a dynamical system interacting with an external stochastic source (a heatbath) simulated with *molecular dynamics* [14].

V. Models of Collective Performance

An ensemble of independently trained networks can make a collective classification several ways. The most powerful voting rule appears to be *plurality* in which the collective decision is the classification reached by more networks than any other. Simpler to analyze is the *majority* voting rule which chooses the classification made by more than half the networks. When there is no agreement among more than half the networks, the result is considered an error. Note that a correct decision by majority is perforce a correct decision by plurality.

A. Independent Errors

The simplest model of network performance is to assume that all neural networks arrive at the correct classification with a certain likelihood $1 - p$ and that they make independent errors. The chances of seeing exactly k errors among N copies of the network is then

$$\binom{N}{k} p^k (1 - p)^{N-k} \tag{2}$$

which gives the following likelihood of the majority rule being in error

$$\sum_{k>N/2}^{N} \binom{N}{k} p^k (1 - p)^{N-k}. \tag{3}$$

It can be shown by induction for odd N (or separately for even N) that provided $p < 1/2$, (3) is monotonically decreasing in N. In other words, if each network can get the right answer more than half the time, and if network responses are independent, then the more networks used, the less the likelihood of an error by a majority decision rule. In the limit of infinite N, the ensemble error rate goes to zero. The conclusion is of course reversed for $p > 1/2$.

To find the probability that a plurality decision makes an error is considerably more complicated. In particular, this needs some assumptions about the type of errors made, i.e., when two networks both make an error in classification, how likely are they to choose the same incorrect classification. This is difficult to predict and is likely to be rather problem dependent. As an illustration consider the possibility that classifications are difficult because pairs of inputs bear a strong resemblance to their partners. In this case there are effectively only two choices at each input and plurality becomes the same as majority. We proceed by adopting what we will refer to as the assumption of "random errors." Specifically, we assume that when an error in classification occurs, the incorrectly identified output is chosen with equal likelihood from among the $M - 1$ remaining possible classifications. The exact value of M appropriate to a certain analysis then becomes a parameter which we call the effective "degree of confusion." It can reveal important information about the structure of a problem. Below we estimate this parameter $M_{\text{effective}}$.

Using the random errors assumption, we calculate the probability of making an error with plurality voting by N networks choosing between M outputs. Let n_i be the number of networks identifying a given input with output pattern i, $i = 1, \cdots M$. Without loss of generality we assume that the correct pattern for the identification is $i = 1$. For a given value of n_1, an error occurs if and only if at least one of the n_i, $i > 1$, is at least as large as n_1. The probability of an error with plurality consensus then becomes

$$\sum_{n_1=0}^{N} \text{Prob}\,[n_1]\left(1 - \text{Prob}\left[\left(\max_{i>1} n_i\right) < n_1 \middle| n_1\right]\right). \quad (4)$$

Letting p again denote the probability that any one of the N networks makes an error we find

$$\text{Prob}\,[n_1] = \binom{N}{n_1} p^{n_1}(1-p)^{N-n_1}. \quad (5)$$

The other probability in (4) is harder to compute. Given the value of n_1, we are faced with making $N - n_1$ choices from among $M - 1$ equally likely alternatives. The required probability asks for the likelihood that the maximum of the frequency of the most commonly chosen pattern is less than n_1. This probability can be obtained using the closed form summability of certain generating functions [15]. To use this summability one maps the problem into the equivalent one of counting the number of functions from a set with $M - 1$ elements to the set $\{1, \cdots n_1 - 1\}$ such that the sum of the values taken on by this function is exactly $N - n_1$. Omitting the details, the calculation gives

$$\text{Prob}\left[\left(\max_{i>1} n_i\right) < n_1 \middle| n_1\right] = \frac{\sum_{k=0}^{M-1} (-1)^k \binom{M-1}{k} \binom{N - n_1(1+k) + M - 2}{N - n_1(1+k)}}{\binom{N - n_1 + M - 2}{M-2}} \quad (6)$$

for $n_1 \geq 1 + (N-1)/M$. For smaller n_1 we find

$$\text{Prob}\left[\left(\max_{i>1} n_i\right) < n_1 \middle| n_1\right] = 0. \quad (7)$$

Substitution of (5), (6), and (7) into (4) yields the probability $P_{\text{plurality}}$ of failure by an N network ensemble with random errors using plurality voting among M patterns

$$P_{\text{plurality}}(p) = \sum_{n_1=0}^{\lfloor 1+(N-1)/M \rfloor} \binom{N}{n_1} p^{n_1}(1-p)^{N-n_1} + \sum_{n_1=\lceil 1+(N-1)/M \rceil}^{N} \binom{N}{n_1} p^{n_1}(1-p)^{N-n_1} \left[1 - \frac{\sum_{k=0}^{M-1} (-1)^k \binom{M-1}{k} \binom{N - n_1(1+k) + M - 2}{N - n_1(1+k)}}{\binom{N - n_1 + M - 2}{M-2}}\right]. \quad (8)$$

Either of the above models predicts performance far superior to the performance we see experimentally. This is due to the fact that the behavior of the networks is in fact *not* independent.

B. A Model Incorporating Input Difficulty

The dominant cause for dependence among errors in classification is the fact that a specific input has an associated difficulty independent of which network is doing the classification. For the example in Fig. 1, points near the origin are harder to classify correctly than points near the corners of the square. On inputs which are difficult to classify, all of the neural networks have a lower probability of coming up with the right output. Eckhardt and Lee [7] have proposed a model incorporating such input specific difficulty within a study of N-version programming. Their model as applied to neural nets is briefly as follows. For each possible input α, we define $\theta(\alpha)$ to be the fraction of (an infinite) ensemble of neural networks trained according to a fixed algorithm (or distribution of algorithms) which fails to correctly classify input α. Then the performance of an ensemble of N networks on this input under majority voting becomes

$$\sum_{k>N/2}^{N} \binom{N}{k} \theta^k (1-\theta)^{N-k}. \quad (9)$$

We now define the classification difficulty distribution μ. By μ we shall mean a measure on the set of θ values so that $\mu(\theta)\,d\theta$ is the fraction of inputs α on which a fraction between θ and $\theta + d\theta$ of the networks make an erroneous classification. Then the predicted performance by majority rule among N networks is

$$\int \sum_{k>N/2}^{N} \binom{N}{k} \theta^k (1-\theta)^{N-k} \mu(\theta)\,d\theta. \quad (10)$$

At this point in their development of the theory, Eckhardt and Lee state that this majority rule need not perform bet-

ter than an individual network. Since the integrand is smaller for N networks than for 1 provided $\theta < 1/2$, this can only happen when the majority of the inputs have high θ values, i.e., when μ has significant mass in the interval $[1/2, 1]$. For neural networks, this will happen only if the input distribution during operation of the neural network is very different from the input distribution during training or if the networks are very poorly trained, i.e., if their performance on the training set identifies the correct outputs less than $1/2$ the time. For $N \to \infty$, the error rate tends to $\int_{0.5}^{1} \mu(\theta)\, d\theta$.

The performance of an ensemble using plurality with an effective degree of confusion M and an input difficulty distribution μ is computed analogously to (10).

$$\int P_{\text{plurality}}(\theta)\mu(\theta)\, d\theta. \tag{11}$$

These models are not useful without some knowledge of the distribution μ of classification difficulties. This quantity is observable as a discrete distribution over the θ values $0, 1/K, 2/K, \cdots (K-1)/K, 1$, where K is the number of networks in the ensemble used to observe μ. For each input α, the estimate of $\theta(\alpha)$ is the fraction of the K networks which classify α incorrectly. The estimate of $\mu(i/K) \equiv \mu_i$ is then the fraction of questions in a large sample which result in i of the K networks making erroneous classifications.

One can make a useful guess of the distribution μ using maximum entropy techniques. We note that for an infinite ensemble, the average error rate $\bar{p}$ for all networks trained according to the given regimen must be the mean $\int \theta\mu(\theta)\, d\theta$. Since this average error rate is easily (and standardly) measured for each network in the ensemble, we can choose μ to be the distribution μ^* which is the most random distribution on $[0, 1]$ with mean $\bar{p}$ in the sense that it maximizes the integral $\int \mu(\theta) \ln \mu(\theta)\, d\theta$.

To understand the meaning of most random in the present context, consider again the discrete case where θ can take on only the values i/K. Consider the performance of K copies of the system on L classifications. We have got to distribute a certain total number $T = \bar{p}KL$ of errors among the networks. Each such distribution of errors will result in a corresponding empirical μ_i which is the number of problems on which i of the K networks erred. The number of ways we can achieve a certain set of values for the μ_i's is given by

$$\frac{L!}{\Pi(\mu_i L)!} \tag{12}$$

with

$$\sum_{i=1}^{K} \mu_i = 1 \quad \text{and} \quad \sum_{i=1}^{K} i\mu_i = \bar{p}K. \tag{13}$$

Maximizing the entropy then corresponds to maximizing this number of ways of realizing μ with a given mean error rate $\bar{p}$. It is usually justified by assuming that all μ's consistent with the mean are equally likely and then involving the sharply peaked nature of the multinomial distribution. This gives μ^* which is of the Boltzmann form

$$\mu^*(\theta_i) = \frac{e^{-\lambda\theta_i}}{\sum_j e^{-\lambda\theta_j}}. \tag{14}$$

This estimate is useful when experimental μ's are unavailable. A value of λ can be selected so (13) holds and then formulas (10) and (11) above can be used to predict ensemble performance.

Note that even these models do not fully treat the dependence between neural networks. While this model gives a good qualitative fit to observed data, the real measure of ensemble performance must come from crossvalidation.

C. A Model Incorporating Network Proficiency

We now model a different source of correlation in the performance of the networks: their individual proficiency. An accurate model of performance with individual proficiencies can provide a criterion for screening the networks for membership in the ensemble. As an illustration, consider development of a neural network application which resulted in one copy which achieves an error rate of 0.1 and two other copies that each have an error rate of 0.2. Will the ensemble of three networks make fewer errors than the best network alone?

The predictive power of the present models, can only be relied on for qualitative information. The answer, however, comes out affirmative even when the other two networks are much less well trained. In the present paper we examine only the simplest case: independent performance using three networks. Qualitatively this case already shows the important features. Using only three networks avoids the complication of an unknown M since the three network consensus makes an error if and only if at least two of the participating networks make an error regardless of whether their erroneous identifications coincide.

Consider three networks with proficiencies p_1, p_2, and p_3. Without loss of generality we take the minimum error to be p_1. The condition that the consensus error rate be less than the best individual error rate is

$$p_1 > p_1p_2p_3 + p_1p_2(1 - p_3) + p_1p_3(1 - p_2) + p_2p_3(1 - p_1). \tag{15}$$

For given p_1, this represents the region in the first quadrant of the p_2, p_3 plane below a rectangular hyperbola which crosses the axes at $(0, 1)$ and $(1, 0)$, respectively. Thus one of the networks can be quite poor provided the other is sufficiently good. To get an idea how good they have to be, consider the case $p_2 = p_3 \equiv p$. Expanding the criterion to second order in p_1 this becomes

$$p < \sqrt{p_1} - p_1 + \frac{3}{2}p_1^{3/2} - 2p_1^2 + o(p_1^2). \tag{16}$$

For the illustration above with $p_1 = 0.1$ and $p = 0.2$ we see that it does pay to use consensus classification.

VI. Experiments

Below we illustrate the above ideas on two examples. Both examples involve a known rule and hence constitute toy problems. This made it possible to mimic the cross-validation by generating any desired number of test patterns cheaply. This made good statistics possible for our experiments. For real examples this function can be performed, albeit to lesser accuracy, using crossvalidation.

Both of our experiments employ a simple feed forward architecture which we tuned by backpropagation [2]. Validation schemes were used to measure the effects of ensemble size and of network architecture.

Backpropagation being a standard technique, we present only the details needed to make our experiments reproducible. We have found it useful to allow for some flexibility in the choice of the gradient descent parameter η. For the first five showings of the training set, we used $\bar{\eta} = \eta/100$ to absorb the strong gradients in the transient phase. In addition we allow for different η's for the different components of the w-gradient; we have used η's specific to weights and thresholds as described below.

The iteration scheme indicated above has been initialized with random w's as well as random "velocities":

$$w_{k,1}^0 = 2.0\eta r,$$
$$\delta w_{k,1}^0 = 2.0\eta r', \qquad (17)$$

with r and r' uniform in $[-1, 1]$.

We have enforced a vote from each network by letting the output neuron with the maximum output define the class.

A. Example 1: The Generalized XOR

One of the classical problems which cannot be solved without "hidden" neurons is the XOR (or Parity) problem. In its standard form it calls for reproducing the truth-table of the logical function XOR. With two binary input neurons representing the operands and one output neuron representing the value, it can be shown that at least two hidden neurons are needed within the Feed-forward architecture of Fig. 2 [2]. We have generalized the problem by accepting continuously valued input and letting the classification be: $o(x, y) = \text{sign}\,(x * y)$. The example is included for its visual impact. Fig. 1 shows the performance of three independently trained Feed-forward networks. We have included six hidden neurons in each network. Each square represents a scan of the classifications within the domain, $[-0.5, 0.5] \times [-0.5, 0.5]$, after presentation of 10 000 uniformly chosen random points. For the *test set*, the unit square has been discretized in a 15 by 15 lattice and the classification of each cell center has been calculated using the individual nets as well as the majority rule.

The parameter values used for training were: η (weights) = 0.05, η (thresholds) = 0.01, and momentum parameter = 0.99.

B. Example 2: A Model Problem

As our second and more realistic example we have chosen the classification of a number of regions in the 20-dimensional hypercube. The regions are defined by 10 "pure" patterns chosen at random. Each pure pattern (corner of the hypercube) defines a class which is designated 1–10. Samples from each class are generated by perturbing the corresponding pure pattern by bit-inversion with specified probability p. The class-coding utilizes the standard form [6] having ten output neurons and letting output-pattern j be defined by neuron j being "on" $(+1)$ and the rest "off" (-1). The training set was created with $p = 0.1$ while we use test sets with p: 0.0, 0.05, 0.1, 0.15 and 0.2. This models extrapolation as well as interpolation. We note that regions used in test sets with $p = 0.20$, are expected to have a significant overlap as two random pure patterns differ by ~ 10 bits while there is about 20% probability that 6 or more bits are changed.

We have performed training sessions in two modes, in the first mode we trained each network on the same training set of 100 examples (10 from each region), while in the second mode we used independently generated sets of 100 examples.

The parameter values used for training were: η (weights) = 0.05, η (hidden thresholds) = 0.01, η (output thresholds) = 0.00001, and momentum parameter = 0.8.

VII. Results

In this section we present some numerical evidence for our claims as well as an application of our analytical tools.

A. The Generalized XOR

The XOR problem for real-valued input is harder than the corresponding discrete problem. This does not imply that the problem cannot be solved by a single hidden layer; we have explicitly constructed a solution involving six hidden neurons.

The nets of Fig. 1 each have six hidden neurons but none of them has found a satisfactory solution. Each has errors in the range 25–30%. We found that only about half of the single networks could come up with the correct picture, i.e., errors in the range 0–5%. The other half retained error rates of 25–30% despite extended training. The figure is meant to underline the dramatically improved performance obtained by using a consensus decision rule even with an ensemble of only three networks.

B. Classification of Random Patterns

This model problem is our main source of experimental back-up. The problem is approximately linearly separable, as the region of each pure pattern can be "cut out" of the hypercube by a hyperplane. Disregarding the chance that two regions may overlap, the problem is solvable by a simple perceptron [16].

With fewer than ten hidden neurons, the network has to provide some data reduction. We expect a lower bound of around ten neurons needed in the hidden layer for good

performance without much training [3]. If we include the possibility that best performance involves encoding, the lower bound goes down to 4 hidden neurons ($10 < 2^4$).

The first set of experiments use a shared training set (mode one). To find the optimal number of hidden neurons, we perform a crossvalidation experiment as presented in Fig. 3. Each net has been trained by 5000 pattern presentations (showings) using patterns randomly chosen from the 100-member training set. Each point represents the average performance of three networks with different initial w's. As expected, we see that the nets with fewer than ten hidden neurons perform significantly worse than the limiting performance seen for nets with more than ten. Combined with the observation that the training time grows linearly with the number of hidden neurons, this shows that the optimal configuration has ~ 13 neurons.

To further bring down the error rate, we examine two options: using an ensemble of copies of our network with one hidden layer as well as the more conventional tactic of adding a second hidden layer.

The experiment of Fig. 4 presents the performance of an ensemble of seven nets using a single hidden layer. We have given the average performance, the performance of the best net in the ensemble and the results of collective inference by majority and plurality. In order to facilitate counting, we have enforced a valid vote from each network by letting the output neuron with the maximum output define the class, i.e., we set that neuron to 1 and the rest to -1. To assess the importance of ensemble size we have performed the experiment presented in Fig. 5 using ensembles with 3, 5, 11, and 15 members. While there is a substantial gain on going from three to five nets, we note that the gain is leveling off for the larger ensembles.

We compare the performance of an ensemble of one hidden layer nets with the average performance of two layer nets in Fig. 6. The figure shows the performance of ensembles of three and five nets, each with 10 hidden neurons, compared to two-layer nets having, respectively, 15 and 25 neurons in each layer. The constructions have pairwise the same number of hidden neurons but the performance using ensembles is better than the performance of the more complex network. We note, however, that there is an improvement in individual performance on going to two-layer nets. Further improvement yet can be obtained by going to ensembles of two layer nets. This is illustrated in Fig. 7 which shows the collective performance of an 11 copy ensembles of 2 hidden layer nets.

While individual performance remained unaffected, the use of independent training sets (mode two) gave markedly better results than using the same training set for all copies (mode one). This is shown in Fig. 8 for seven networks using one hidden layer.

Fig. 9 compares the experimental problem difficulty distributions μ with shared and independent training sets as well as with our maximum entropy prediction for seven networks tested on inputs with a 20% chance of bit inversion. Note that the predicted μ lies much closer to the data for the networks with independent training. This is

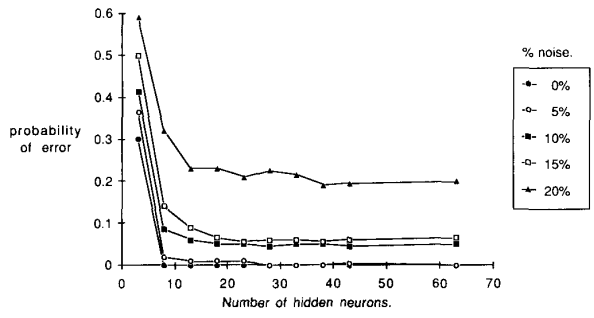


Fig. 3. Network architecture as assessed by crossvalidation for classification of random patterns. The figure shows performance versus number of hidden neurons using test sets with 0, 5, 10, 15, and 20% probability of bit inversion in the input.

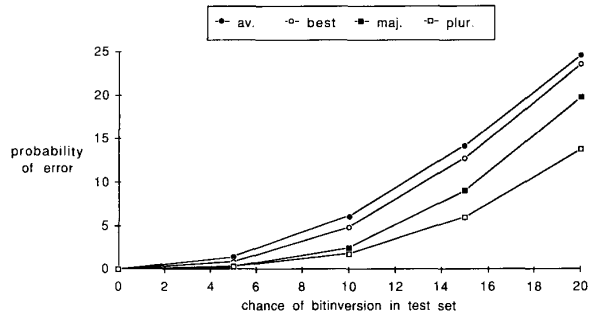


Fig. 4. Performance versus noise level in the test set is shown for individual and for consensus decisions. Data displayed shows the average and the best network, as well as collective decisions using majority and plurality for seven networks trained on individual training sets.

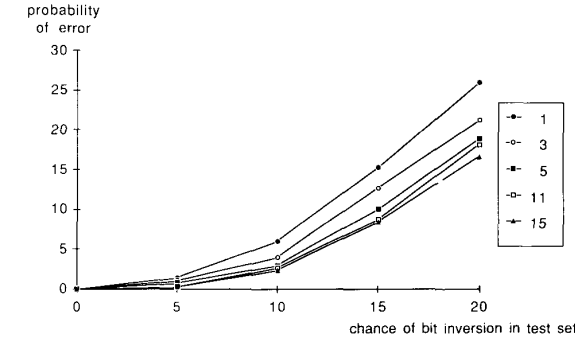


Fig. 5. The figure examines performance versus noise level in the test set. Plurality performance is shown for ensembles of 1, 3, 5, 11, and 15 networks trained on shared training sets.

the general trend for all our predictions. Note that even at this high noise level where the generated patterns overlap, the probability $\mu(\theta > 0.5)$ is only around 0.2. This is the limiting error rate for an infinite ensemble using majority; plurality can only do better. For error rates of 15%, 10%, 5%, and 0% this limiting probability is 0.09, 0.03, 0.004, and 0, respectively.

Using the maximum entropy model for the problem difficulty distribution μ, gives excellent predictions of performance for the independently trained networks. This is

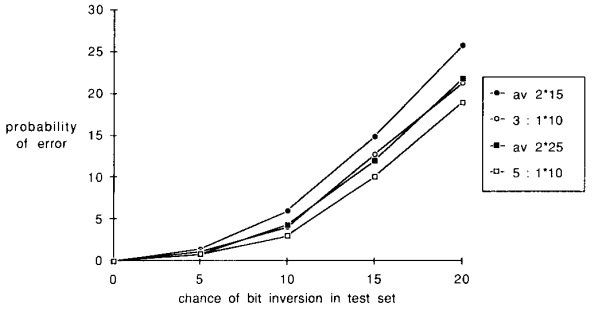


Fig. 6. The figure compares hidden neuron effectiveness as part of a second layer or as part of another network used in the ensemble. Performance curves are shown for ensembles of 3 and 5 single layer nets with 10 hidden neurons each which are to be compared with the average performance of nets with 15 and 25 neurons in each of two hidden layers. Note that the ensembles outperform the corresponding two layer nets.

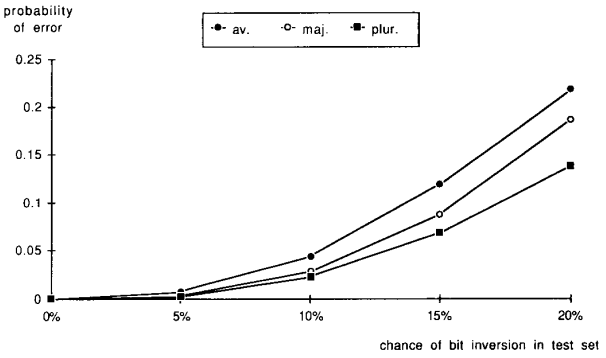


Fig. 7. The figure shows the improvement obtained by using ensembles of networks with two hidden layers. The average performance of a single network is compared to the majority and plurality performance of 11 copies of a two hidden layer network having 25 neurons in each hidden layer. Error rates were measured on test sets with 10 000 inputs.

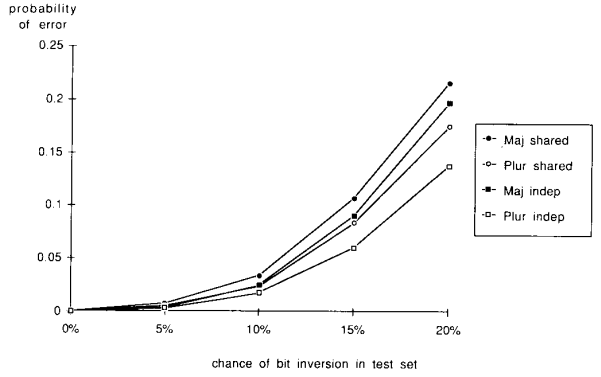


Fig. 8. The effect of using independent versus shared training sets is shown for majority and plurality consensus among seven networks.

illustrated in Fig. 10 for performance using majority consensus among seven networks. Using our difficulty distribution μ, and the plurality performance $_{\mathrm{P}}$ measured at one ensemble size, we can select an effective degree of confusion parameter M_{eff} so the observed performance comes as close as possible to the performance predicted by (8).

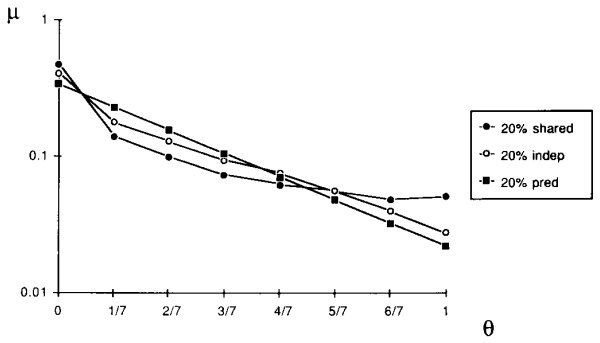


Fig. 9. The problem difficulty distribution predicted by maximum entropy for a test set with 20% noise is compared to experimental distributions for seven networks trained independently and for seven networks that shared a training set.

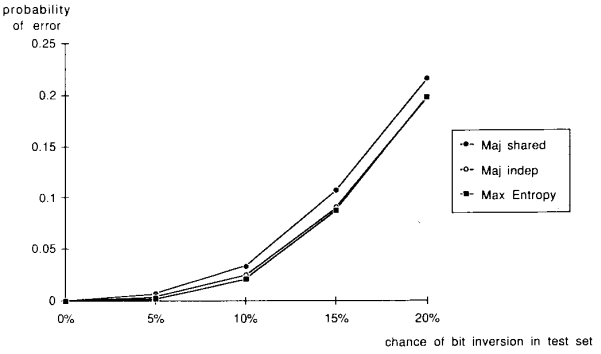


Fig. 10. Majority performance predicted by maximum entropy is compared to experimental performance for seven networks trained independently and for seven networks that shared a training set.

M_{eff} can then be used to predict the performance for other ensemble sizes. If we also have a measured value of the majority performance, a slightly less noisy value of M_{eff} can be estimated by requiring that the improvement of plurality performance over majority performance match the predicted improvement. Specifically, we choose M_{eff} so the ratio of (8) to (5) match the observed ratio as closely as possible. By either of these methods we find M_{eff} values of approximately 4, 5, 7, and 9 for noise levels of 5%, 10%, 15%, and 20%, respectively. The performance for 15% chance of bit inversion using $M_{\text{eff}} = 7$ is shown in Fig. 11 next to the experimental data for independently trained networks.

VIII. Conclusions

The basic conclusion of the present paper is that using an ensemble of neutral networks with a plurality consensus scheme can be expected to perform far better than using a single copy. Slightly inferiorly trained networks are a free by-product of most tuning algorithms; it is desirable to use such extra copies even when their performance is significantly worse than the best performance found. Better performance yet can be achieved through careful planning for an ensemble classification by using

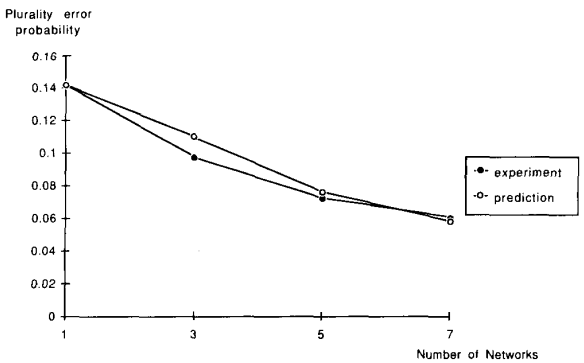


Fig. 11. Plurality performance for independently trained neural networks is compared to predicted performance on data with 15% chance of bit inversion. The predictions use an estimated effective degree of confusion M and the difficulty distribution obtained by maximum entropy.

the best available parameters and training different copies on different subsets of the available database.

Crossvalidation is the ultimate test of performance. This measure of performance is applied after the training set is learned well and measures the true objective function in the training of neutral networks: the number of errors which will be made during their execution phase of operation, i.e., the capacity to generalize. We have shown how cross validation may be used to optimize network architecture; in particular, we have analyzed the effect of varying the number of hidden neurons and layers in a Feed-forward network.

The models were tested using experiments on a simple problem which modeled the essential features of a linearly separable problem with a noisy rule. Our models of ensemble performance gave good quantitative predictions for identically trained networks on different training sets chosen from the database. The prediction of performance by a majority consensus is based only on the mean error rate of one network. The predictions for plurality consensus also need an estimated effective "degree of confusion" parameter.

We presented arguments to explain the excellent performance of consensus schemes among neutral networks. An intuitive picture of these arguments is: the search for good weights takes place in a space with many "traps" which correspond to different ways of "generalizing" the rule hidden in the training set.

Acknowledgment

We would like to thank the participants of the neural network seminar at S.D.S.U. for helpful comments and discussions. In addition we would like to thank S. Brunak for helpful criticisms, W. Root for some combinatorial assistance, and L. Liao for pointing out the relation between our problem and fault tolerant computing.

References

[1] E. Levin, N. Tishby, and S. Solla, "A statistical approach to learning and generalization in layered neural networks," *Proc. IEEE (Special Issue on Neural Networks)*, C. Lau, Guest Ed., 1990, to be published.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[3] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive Sci.*, vol. 9, pp. 147–169, 1985.

[4] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer*, vol. 25, Mar. 1988.

[5] F. J. Pineda, "Generalization of backpropagation to recurrent neural networks," *Phys. Rev. Lett.*, vol. 59, pp. 2229–2232, 1987.

[6] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.*, vol. 1, pp. 145–168, 1987.

[7] D. E. Eckhardt, Jr., and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1511–1517, 1985.

[8] G. T. Toussaint, "Bibliography on estimation of misclassification," *IEEE Trans. Inform. Theory.*, vol. IT-20, pp. 472–479, 1974.

[9] N. Qian and T. J. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models," *J. Molecular Biol.*, vol. 202, pp. 865–884, 1989.

[10] H. Bohr, J. Bohr, S. Brunak, R. M. J. Cotterill, B. Lautrup, L. Nørskov, O. H. Olsen, and S. B. Petersen, "Protein secondary structure and homology by neural networks," *Fed. European Biochem. Soc. Lett.*, vol. 241, pp. 223–228, 1988.

[11] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.

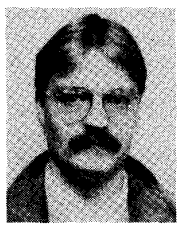
[12] R. G. Palmer, "Broken ergodicity," *Advances Phys.*, vol. 31, pp. 669–735, 1982.

[13] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[14] T. Schneider and E. Stoll, "Molecular-dynamics study of a three dimensional one-component model for distortive phase transitions," *Phys. Rev. B*, vol. 17, p. 1302, 1978.

[15] J. Riordan, *An Introduction to Combinatorial Analysis*. Princeton, NJ: Princeton University Press, 1980.

[16] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan, 1959.



Lars Kai Hansen received the M.Sc. and Ph.D. degrees in physics from the University of Copenhagen, Denmark, in 1984 and 1986, respectively.

Since 1987 he has been with Andrex Radiation Products A/S of Copenhagen, doing research in computer vision for automation of nondestructive testing systems. From August 1988 to March 1989 he was Adjunct Professor at the Department of Mathematical Sciences of San Diego State University, San Diego, CA, working with applications of neural networks. His research interests are in the areas of applied statistical mechanics, machine vision, and parallel computer architectures.

Dr. Hansen was awarded the Gold Medal of the University of Copenhagen in 1986.

Peter Salamon received the B.A. degree in mathematics from Lindenwood College, St. Charles, MO, the M.S. degree in applied mathematics from Drexel University, Philadelphia, PA, and the Ph.D. degree in chemical physics from the University of Chicago, Chicago, IL.

He served as a Postdoctoral Fellow in the Chemistry Institute at Tel Aviv University during 1978–1979 and as a Visiting Professor in Mathematics at Arizona State University during 1979–1980. Since 1980, he has been with the Department of Mathematical Sciences at San Diego State University, San Diego, CA. In 1986–1987 he was a Guest Professor in the Physics Laboratory of the Oersted Institute in Copenhagen and he was an Alexander von Humboldt stipendiate in Theoretical Physics at the University of Heildeberg during the Summer and Fall of 1987. His research interests are in the areas of stochastic algorithms and in-principle limits for physical processes.