LARRIE V. HUTTON

USING STATISTICS TO ASSESS THE PERFORMANCE OF NEURAL NETWORK CLASSIFIERS

Neural network (NN) approaches to pattern classification problems both complement and compete with statistical approaches. Each approach has unique strengths that can be exploited in the design and evaluation of classifier systems. In the spirit of emphasizing this complementary nature, four points are made in this article. First, classical (statistical) techniques can be used to evaluate the performance of NN classifiers. Second, NN classifiers often outperform classical techniques. Third, NN classifiers may have advantages even when their ultimate performance on a training set can be shown to be no better than the performance of a classical classifier. Finally, it is suggested that methods that are routinely used in statistics, but not in NN approaches, should be adopted for the latter as well.

INTRODUCTION

Neural network (NN) technologies are being investigated by several groups at the Applied Physics Laboratory. This article focuses on the use of NN's as pattern classifiers, particularly multilayer perceptrons (MLP's) using the backpropagation algorithm. For example, Eberhart et al.[1] and Wilson et al.[2] classified electroencephalographic (EEG) waveforms with MLP's. Similarly, Bankman et al.[3] detected EEG K-complex waves with NN classifiers, and Eberhart and Dobbins[4] diagnosed appendicitis by using both MLP's and adaptive resonance theory (ART) networks. Olsen et al.[5] expanded upon Eberhart's work by developing a system for automatically detecting seizures in epileptics by using MLP's (and classical statistical methods).

Outside of biomedical research, Lee[6] used MLP's to recognize ship images. In addition, Sigillito et al.[7] classified radar returns from the ionosphere with MLP's, and in a related experiment involving the aurora, Newell et al.[8,9] identified source regions of ionic precipitation with MLP's.

Neural networks have also been employed in automatic target recognition (ATR) tasks at APL. For example, Roth[10] detected targets in high clutter environments with both feedforward and Hopfield networks, and Boone et al.[11] fused sensors with NN's in the development of a ship recognition classifier. Finally, Sigillito et al.[12] used MLP's in the development of a system to detect defects shown in gray-scale images of fuzes.

Because classification is often associated with classical statistics, it is significant that each of the studies just cited used NN technologies either instead of, or along with, classical approaches to develop classifier systems. It is also significant that an article on NN classifiers was sought for inclusion among the articles on statistics in this issue of the *Technical Digest*. Although NN and classical approaches can be used in the development of classifiers, both approaches are valid in the appropriate context and can coexist with each other. Indeed, NN's can better be said to *do* statistics than to *replace* statistics. But just as some statistical approaches are better than others, some NN approaches may occasionally be superior to a more classical approach to pattern recognition and classification.

To make the ensuing discussion more concrete, assume a classification system with n_i (binary or real-valued) input elements and n_o output elements. (See the boxed insert for terminology.) Each of the n_i input elements might represent sensor information from the environment, the gray-level value of a pixel, or other pertinent information. Each of the n_o output elements represents the presence or absence of a category, for example, a particular type of tank or plane. When multiple categories exist, the output element with the highest value defines the active category. The combination of input and output vectors at any one time defines a pattern vector, and the classification system produces an output vector that correctly classifies the corresponding input vector. A special example is a classification system with a single element in the output vector; if the value of that single element is above some threshold value, the presence of the object or category is assumed; otherwise, it is not.

ASSESSING THE PERFORMANCE OF CLASSIFIERS

Two major issues will be addressed in this section: (1) categorical measures of the performance of a classification system, and (2) squared-error measures of performance. Assessment of performance during learning is typically of greater concern for users of NN tools than it is for users of statistics, because NN users are more likely to have to contend with decisions of when to stop training. The ultimate performance, however, is equally important to users of either approach.

EXPLANATION OF NEURAL NETWORK TERMINOLOGY

Many of the terms used in this article are based on the following definitions:

$$E_T = \frac{1}{2} \sum_{p=1}^{n_p} \sum_{k=1}^{n_o} (t_{kp} - o_{kp})^2 ,$$

$$o_{kp} = f\left(\sum_{j=1}^{n_k} w_{kj} o_{jp} - \beta_k \right) ,$$

where E_T is the total error across all n_o output nodes for all n_p patterns in the training set, t_{kp} is the target (desired) value for the kth output node and pth pattern (alternatively, the kth category or the kth element in the output vector), w_{kj} is the weight (conductance) from the jth node to the kth node in the layer above, and β_k is the current bias term for the kth node. The squashing function $f(x)$ is typically the logistic function, $1/(1 + e^{-x})$. To simplify notation, the subscript k indicates the kth output node, j the jth hidden node, and i the ith input node. A fully connected feedforward architecture is assumed to be present unless indicated otherwise. In other words, each input node o_i has a weight to each hidden node o_j, and each hidden node has a weight to each output node o_k. Thus, w_{kj} is a hidden-to-output weight, and w_{ji} is an input-to-hidden weight. The details of the backpropagation algorithm that is assumed here can be found in Rumelhart et al.[13]

The learning rule for the hidden-to-output weights is

$$\Delta w_{kj}(t) = \eta \delta_k o_j + \alpha \Delta w_{kj}(t - 1) ,$$

differing only in the subscripts for the input-to-hidden weights. The learning rate typically lies between 0.01 and 0.7; $\delta_k o_j$ is an estimate of $\partial E / \partial w_{kj}$, and α is a so-called momentum term (typically in the range from 0 to 0.7), which minimizes the effect of high-frequency fluctuations in the error surface. A multilayer perceptron is assumed to have at least one hidden layer of neurons, whereas a logistic perceptron, which also incorporates a squashing function, has connections only between input and output elements.

In a typical example, a sample is drawn from some population of n_p patterns, in which each pattern consists of $n_i + n_o$ elements; a pattern vector, therefore, consists of both an input and a target vector. The sample is usually split into a training set on which the neural network or classical classifier is trained, and a test set on which the performance of the classifier is assessed. The performance on the test set is taken as a measure of the performance on the population from which the sample that included both training and test sets was drawn.

Categorical Measures of Performance

Although "percent correct" is probably the most commonly used performance measure for a classification system, it is not the only one. A number of other useful measures are shown in Table 1.

One particularly useful measure of overall performance, which combines sensitivity and specificity measurements, is the receiver operating characteristic (ROC) curve. An ROC curve (see Fig. 1) is generated by plotting sensitivity as a function of false alarm rate. The false alarm and sensitivity values for each point on the curve are generated by varying the detection threshold. A completely random system, which would be just as likely to predict a signal as a false alarm, would produce a straight line from lower left to upper right on the ROC graph. A perfect discriminator would produce two straight lines, one going from lower left to upper left, and then from upper left to upper right. A curve bowed in the opposite direction from lower left to lower right to upper right is possible but not likely, since such performance would be worse than chance. It is easy to construct an ROC curve for an MLP, since one needs only the real-valued output (and associated target value) for each pattern in the database of interest. The trade-off between sensitivity and specificity (which is 1, the false alarm rate) can be read directly from the graph. A commonly used statistic, d', is the distance between the upper left corner and the ROC curve as measured along the minor diagonal.

In addition to using the ROC curve to examine the trade-off between false alarms and sensitivity, the area under the curve can be used to derive an overall index of goodness that is independent of the prior probabilities.[14,15] An advantage to using this area, which should range between 0.5 and 1.0 for all practical purposes, as opposed to using a single point on the curve, is that the area summarizes the results over all threshold values, and thus provides a convenient method of comparing two diagnostic systems over a broad range of conditions. A significance measure for the difference between areas is provided by Meistrell.[16]

The information required to generate each point on an ROC curve is obtained by enumerating, for a given threshold, the number of true positives, true negatives, false positives, and false negatives for one pass through the pattern database. The other points are generated by changing the threshold (i.e., the output value required to classify an input pattern as an instance of the category).

Although ROC curves are said to be independent of the prior probabilities, one should not ignore them. A simple exercise in Bayesian statistics makes this point clear. Assume that you have a population in which 1% are at risk for AIDS. You have an instrument that correctly identifies persons without AIDS 95% of the time, and persons with AIDS 95% of the time. A person, randomly sampled from the population, tests positive. Does he/she have AIDS? Probably not. In fact, the person is 5 times more likely not to have AIDS than to have it. This exercise illustrates the importance of prior probabilities—and incorporating them into training sets—and the dangers of overreliance on any single statistic.

The 2×2 table (true positive, false positive, false negative, true negative) used to generate the ROC curves and associated statistics is actually a special type of $n_o \times n_o$ table where n_o is the number of decision classes. In such a table, the number of correct classifications can

Table 1. Commonly used measures of performance in diagnostic systems.

Measure	Description	Computation (× 100 for percent)
Hit	True positive	TP
Correct rejection	True negative	TN
False alarm	False positive	FP
Miss	False negative	FN
Percent correct, accuracy	Number right/number of cases	(TP + TN) / (TP + TN + FP + FN)
Sensitivity, true positive ratio	Percent of positive instances detected	TP / (TP + FN)
Specificity, true negative ratio	Percent of negative instances detected	TN / (TN + FP)
False alarm rate, false positive ratio	Percent of negative instances incorrectly diagnosed	FP / (TN + FP)
Positive predictive value, selectivity	Percent of time that a positive signal indicated a positive instance	TP / (TP + FP)

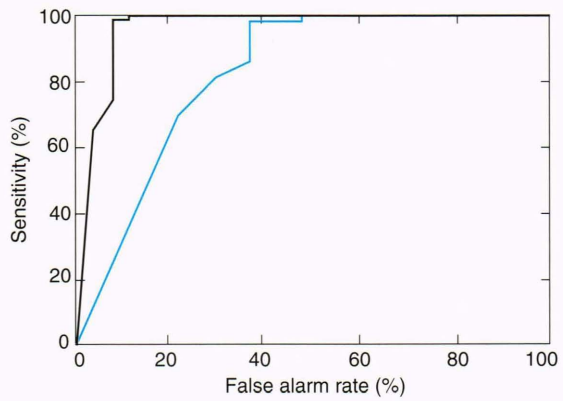


Figure 1. Two examples of receiver operating characteristic curves. The black example is a much worse discriminator than the blue example. A random discriminator would produce a straight line along the main diagonal. (Reprinted from Ref. 7, p. 265.)

be obtained by summing the entries along the diagonal, and misclassifications can be obtained by summing the off-diagonal elements. A very general metric for assessing the performance is to compute a utility function,

$$U = k \sum_{j=1}^{n_o} \sum_{i=1}^{n_o} n_{ij} f(U_{ij}), \qquad (1)$$

where k is a scaling (or normalizing) constant, n_{ij} is the frequency with which the ith class is placed in the jth category, and $f(U_{ij})$ is the cost (or utility) of each entry in cell ij of the $n_o \times n_o$ matrix. For example, if $k = 1/n_p$, $f(U_{ij}) = 1$ for all $i = j$, and $f(U_{ij}) = 0$ for all $i \neq j$, then U simply reduces to the proportion correctly classified. For $n_o > 2$, the construction of ROC curves requires appropriate definitions for sensitivity and false alarm rate. If FAR is defined as the probability that a nonoccurrence of some event is misclassified as an instance of that event, then n_o ROC curves must be generated to provide the coverage that a single ROC curve would provide in the 2×2 example.

Continuous Measures of Performance

The preceding section stressed categorical measures of performance. In this section, measures that treat real values of the output nodes are discussed. The concern is thus not just with final classifications, but how close each output is to the target classification of 0 or 1.

A common and useful measure of an NN classifier is the rms error,

$$E_{rms} = \left[\frac{\sum_p \sum_k (t_{kp} - o_{kp})^2}{n_p n_o} \right]^{1/2}, \qquad (2)$$

which is merely the square root of the average squared error. The notation used here and elsewhere is t_{kp} (value of the kth target on the pth pattern), o_{kp} (value of the kth output node on the pth pattern), n_p (number of patterns), and n_o (number of output categories). Sometimes the k or p subscripts will be dropped if they are not required for clarity.

Since both target and system output range between 0 and 1, E_{rms} ranges between 0 and 1 as well. It can measure the progress (or ultimate performance) of a single system or can be used to compare several different systems. One advantage of E_{rms}, as opposed to a simple sum of squared error measure (which is usually the objective function in an MLP), is that it is normalized by the number of output variables and the number of patterns. Thus, it can be used to compare the performance of systems that differ in these two regards.

A related measure is average absolute error, defined as

$$E_{\text{abs}} = \frac{\sum_p \sum_k \left| t_{kp} - o_{kp} \right|}{n_{\text{p}} n_{\text{o}}} , \tag{3}$$

where $|t_{kp} - o_{kp}|$ is the absolute value of the error on the pth pattern for the kth output node. An E_{abs} value that is substantially smaller than E_{rms} suggests that some patterns are generating higher-than-average errors (large absolute values for $t_{kp} - o_{kp}$ affect E_{rms} more than E_{abs}), and that there is something unusual about those patterns. For example, such patterns may have been misclassified when category assignments were made when the training or test set was created, or they may be associated with input features that were not incorporated in the pattern database.

Another useful measure is the proportion of variance (*POV*) accounted for. The *POV* is simply the square of the Pearson product–moment correlation, r, and is defined for the kth output class as

$$POV_k = r_k^2 = \frac{\left(s_{t_k o_k}^2 \right)^2}{s_{t_k}^2 s_{o_k}^2} , \tag{4}$$

where $s_{t_k o_k}^2$ is the covariance of the kth target and kth output vectors for all n_{p} patterns, and $s_{t_k}^2$ and $s_{o_k}^2$ are the target and output variances, respectively. The *POV* is equivalent to the squared cosine of the angle between output and target vectors. It is useful because it is normalized not just by the number of pattern presentations, as are E_{rms} and E_{abs}, but by the variances of the target and output measures as well. It can be interpreted as the proportion of variance in the kth target class that is accounted for by the NN or statistical model.

Pineda[17] has suggested a related approach. Let s_t^2 be the target variance for the kth class for all n_{p} patterns. The average squared error between target and output for the kth category is

$$E_{\text{m}} = \frac{\sum_p (t_p - o_p)^2}{n_p} , \tag{5}$$

for any output class k. A performance measure that, like *POV*, approaches 1 as the error approaches 0 is

$$P = 1 - \frac{E_{\text{m}}}{s_t^2} . \tag{6}$$

Note that P can be negative if $E_{\text{m}} > s_t^2$ (which is typically true at the beginning of training), is equal to 0 when $E_{\text{m}} = s_t^2$, and approaches 1 as E_{m} approaches 0.

Measures of Training Progress

The behavior of P and E_{m} can be used as a measure of training progress. Once training begins, E_{m} quickly approaches s_t^2. It is instructive to see why this happens, because it has implications for initializing a network. Initially, the input-to-hidden and hidden-to-output weights are small in magnitude and centered around zero. Hence, changes in the input vector will tend to have a small effect on changing the value of any hidden node. In particular, each hidden node's output (after applying a logistic squashing function) should be around 0.5 regardless of the input vector. But the outputs of the hidden nodes are attenuated by the near-zero hidden-to-output weights, so the average value of an output node should be around 0.5 as well. Consequently, the input vectors initially have little effect on the value of an output node, and the system can most quickly minimize error by adjusting o_p through the bias term. The bias for the output nodes is relatively labile, compared with the more slowly changing hidden-to-output weights, because it is affected directly by the target values for each pattern rather than being shielded by the hidden nodes. Recall that o_p is initially affected inconsequentially by the input vector, so the bias term disproportionately affects o_p. Since o_p will initially be constant through all pattern presentations, and because the fixed value of o_p that minimizes E_{m} is the mean of t_p, the system learns to set o_p to the mean of t_p. The system thus effectively learns the prior probability of target present without using information from the input vectors. Others[18–21] have shown that output values reflect the Bayesian posterior probabilities after training. When initializing the weights, it seems likely that input-to-hidden weights that are set to zero should facilitate the learning process described previously because that would minimize the interference from the input patterns as the system learns the prior probabilities. Lee[6] reported that faster training does occur when the input-to-hidden weights are set to 0.

Other measures of network progress have been used by Sigillito and Eberhart,[22] also at APL. In one method, inspired by Pineda,[17] the angle between a vector whose elements are the values of all the weights in a network, and a vector with all elements equal, is found. Although the angle between the vectors does not itself give much information (it is possible that the final weight vector could be either farther from or nearer to the reference vector than it was at the beginning of the training), the rate at which the angle is changing indicates the speed at which the weight vector is approaching its final destination in weight space.

Sigillito and Eberhart[22] suggested that the length of the weight vector indicates network progress, since weights start out as small random weights and increase in average absolute magnitude as output nodes are driven to their limits with training. Finding the variance of the normalized weight vector gives an index of dispersion that indicates movement away from the relatively equal weights that were set during initialization. Hanson and Burr[23] also reported an increase in average absolute weight after training, and that the weight variance was a function of the average absolute weight, at least for very large networks. Although this result is consistent with Weber's Law, a well-known psychophysical law, it is not clear whether Hanson and Burr's observations apply generally.

RELATIVE PERFORMANCE OF NEURAL NETWORK AND STATISTICAL CLASSIFIERS

An NN classifier is asymptotically equivalent to a statistical procedure when, given enough time, the performance of the NN converges to the performance of the statistical procedure. Although asymptotically equivalent networks are not uncommon (NN's do statistics), only the most common will be mentioned here.

Perhaps the best known equivalence of NN and classical methods is provided by Stone,[24] who showed that the delta rule used in linear perceptrons implements a multivariate multiple linear regression. Thus, the linear perceptron will, in principle, develop weights that are equivalent to the coefficients in multiple linear regression, where the linear perceptron's bias is equal to the intercept of the regression analysis. If zero-mean inputs are used, the intercept should be zero. The *POV* will be identical as well. Since a Fisher linear discriminant analysis has been shown to be a special example of multiple regression analysis,[25] a linear perceptron is also asymptotically equivalent to a Fisher linear discriminant analysis.

Another example of a close correspondence, if not asymptotic equivalence, between a statistical tool and an NN classifier is found in the logistic perceptron, which approximates the performance of a logistic regression analysis.[18] The MLP, which is the most widely used NN classifier, can be used to perform a nonlinear regression analysis—again, a common statistical method. Summarizing, linear perceptrons, which use the delta rule, are equivalent to multivariate multiple linear regression approaches and to linear discriminant analysis. Logistic perceptrons, which use the logistic function but do not have hidden layers, are equivalent to logistic regression, and MLP's, which use the generalized delta rule, can be used to perform nonlinear regression analyses.

The original perceptron learning rule has not been discussed in this article. Although the classical perceptron can be shown to minimize classification error for any problem that is linearly separable, it uses a nondifferentiable threshold function. Thus, it does not minimize a squared-error objective function and so is not discussed further.

No universal agreement has been reached as to whether NN or classical classifiers are better, and the question as to which should be preferred is not easy to answer. Reports that classical classifiers can outperform NN classifiers are not common, and the one unequivocal report reviewed here used questionable methodology.[26] On the other hand, several studies[27-29] have shown that MLP's or their variants can approximate any continuous function with arbitrary precision. In fact, any arbitrary function can be approximated with arbitrary precision.[30,31] It is hard to imagine how a classical classifier could do better than that. Furthermore, simulations have confirmed that MLP's generally do at least as well as, if not better than, quite sophisticated classical classifiers.[27,28]

The battle for best thus rages on with no end in sight, and readers who expected this issue to be resolved here may be disappointed. As a working hypothesis, however, we will assume that the two approaches are about equal in terms of ultimate performance on the training set.

WHY USE NEURAL NETWORK CLASSIFIERS WHEN THEY ARE ONLY ASYMPTOTICALLY EQUIVALENT?

The claim that NN classifiers will generally outperform classical classifiers hinges on the assumption that NN and statistical classifiers may exhibit similar, if not equivalent, performance if trained to convergence on the training set, but that only the NN classifier can validate its performance on the test set while it is learning on the training set.[28] The fact that training and test sets are likely to be different suggests the following thought experiment.

Consider that training and test sets are ideally both perfectly representative of the population from which they are presumably drawn, but typically at least one set is not perfectly representative of the population. When both training and test sets are equally representative of the population, an asymptotically equivalent NN classifier offers no advantage. The reason is simply that asymptotic equivalence on the training set translates into equivalence by definition on the test set. Thus, the training set gives a perfect indication of performance on the test set.

If the two sets are not equally representative, and especially if irregular decision boundaries are involved, a statistical classifier that does not train iteratively will almost always overtrain. The performance of the statistical classifier cannot be affected by its performance on the test set because the classifier is designed to optimize performance on the training set. On the other hand, it is possible to assess performance over time with an NN classifier by examining its performance on the test set,[32,33] since overlearning is manifested by deteriorating performance (after an initial rise to maximum) on the test set, and the NN weights can be frozen at that point. Thus, NN classifiers can be made to maximize performance on the test set, whereas classifiers that do not permit iterative changes of the coefficients tend to overtrain on the training set. This is an argument against optimization techniques, which use efficient methods to find a local minimum of the error surface on the training set. The logic of this argument has been discussed by Hecht-Nielsen,[32] although he does not suggest that weight updates be discontinued when performance on the test set is optimum. Other researchers,[34,35] however, have used performance on the test set as the criterion by which training is stopped. This procedure seems somewhat "unfair" in that it measures the classifier where its performance is optimum. Nevertheless, it is arguable that performance on a randomly sampled test set is a better indicator of performance on the population from which both training and test sets are drawn than any single stopping criterion based solely on performance on the training set. Whether or not this assumption is reasonable for realistic data sets remains an open theoretical and empirical question. The deterioration of performance on the validation set that occurs because of outliers in the training set data is widely recognized, however, in the field of machine learning, where methods exist to prune branches of decision trees that have been induced from the training set.[36] The pruned branches are those that are likely to lead to poor generalization of the population data.

Generally speaking, performance on the training set is relevant, but it is not the most important measure of performance. Performance on the test set is more important since such performance is typically the defined measure of population performance. It would be remarkable if the two sets were uncorrelated because then the performance on the training set would tell us nothing useful. Likewise, a perfect correlation would imply that the two sets are substitutable. Real world data sets are neither uncorrelated nor perfectly correlated, however. Therefore, we want our classification system to learn about the training set only insofar as it reflects pertinent features of the testing set. Statistical approaches (or NN approaches) that blindly minimize the error on the training set will not reliably reflect those pertinent features.

I have assumed that NN classifiers, and not classical classifiers, will be designed to optimize performance on some presumed population data. In practice, this assumption may not hold. The methods described here are valid for either classical or NN approaches as long as the classifier does not produce its solution in a single pass, but instead produces intermediate solutions that can be used to test against the validation data. Neural network applications that do not avail themselves of the techniques described here suffer from the same problems that befall classical classifiers if the NN classifier either cannot produce intermediate solutions (i.e., it is not iterative) or does not consider test set performance when setting the coefficients for the field data. It is true, however, that NN approaches that use backpropagation are capable, in principle, of exploiting the information obtained from intermediate solutions; many classical approaches are not.

A few additional points should be mentioned: First, if performance on the statistical classifier can be assessed at regular intervals, and the statistical classifier performs as well as the NN classifier, then it is more difficult to choose between the two methods. Considerations other than ultimate performance, such as speed, computational complexity, storage requirements, human comprehensibility, and general acceptance of the method chosen (e.g., political considerations), would then necessarily dictate the choice of methods.[37]

Second, MLP's (and NN methods in general) have an advantage over any classical method that makes assumptions about population parameters. Although one of the virtues of NN techniques is that they do not require an underlying model (i.e., they are universal approximators), it must be remembered that nonparametric methods also exist in statistics. In fairness, such desirable attributes are not exclusively the domain of NN models. Of course, if the underlying distribution can be fully described in terms of its parameters, one can do no better than to use those parameters, but usually that is not the kind of problem that motivates one to use neural networks in the first place.

Finally, the dangers of overtraining with small data sets cannot be overemphasized. Assume that you are locked in a room with some data. Your job is to produce a classifier that will do the best possible job on data to which you do not currently have access. (If you have access to more data, you should train using those data as well. If you have access to all possible data, there is no point in training with a subset at all, because you really want a lookup table.) Theoretical statements about how well the system could perform in principle are not useful; you need to produce a set of weights that will do the best possible job on data that are not available for training. The rule is "quit when you're ahead on the test set." (You have, of course, taken the data in your room and split it into a training and a test set.) To see why this rule is important, consider the following thought experiment.

Assume that you have some data that have been dutifully split into a training and a test set; unbeknownst to you, all the data were generated with a random-number generator. What is the best way to minimize the squared error of such a set? The answer is "guess the mean." This method effectively uses the prior probabilities of the targets, which is something the classifiers will learn to do quickly, as we have seen. Error on the training set, and somewhat less reliably on the test set, will decrease until it approaches the normalized variance, where it should stop. If the training and test sets are different—and they will be unless the samples are very large and equally representative—then continuing to train on the training set to some predetermined level of error will force the classifier to exploit differences that do not really exist (recall, the data were random).

Improved performance on the training set now comes at the expense of performance on the test set and, more to the point, poorer performance on the population that the test set represents. This problem can be avoided if performance on the test set is monitored and that information is used when producing the effective weights. I have conducted simulations with small data sets, which maximize the possibility that training sets are unrepresentative, and confirmed the overtraining effect. In fact, weight matrices were consistently produced that resulted in performances on the test set that were worse than guessing the mean! This result is to be expected under the conditions described. No one wants a classifier that performs at that level in the field.

The importance of considering test set performance can be seen in Figure 2, which shows the error as a function of the number of passes through the training set for the training set itself (5 pattern vectors drawn randomly from a population of 100 pattern vectors), a test set (also 5 random pattern vectors), and the entire population of 100 pattern vectors from which both stratified random samples were drawn. It is apparent in this admittedly contrived example that, almost unbelievably, one would be better off if the statistics on the training set were ignored. The test set data clearly track the population data better than the training set data.

Of course, no one uses NN classifiers to classify random data, and if no divergence between the training and test sets occurs, this admonition does not apply. With small data sets or unlucky splits, however, both of which can occur with real data, training to criterion on the training set will almost surely result in less-than-optimal performance on the population data. And it is the population data where performance really counts. The differences may not be as dramatic as in the experiment described here, but they do exist, and they will be magnified if the MLP has a large number of hidden nodes.

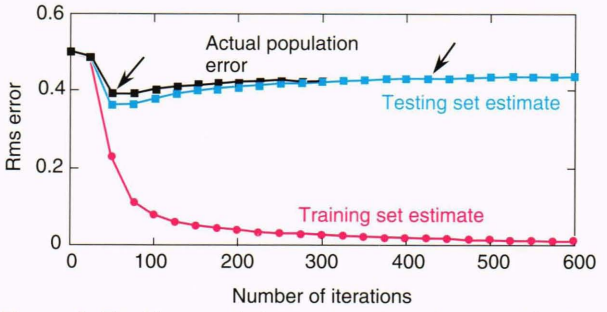


Figure 2. Tracking population error: effect of overtraining in a very noisy environment. The left arrow shows performance in the field (population data) if the weight matrix is saved when test set error is at a minimum. The right arrow shows field performance if training is stopped at an arbitrary rms error of 0.025. Overlearning on the training set exaggerates the importance of the test set performance.

USING STATISTICAL METHODS TO ENHANCE THE PERFORMANCE OF NEURAL NETWORK CLASSIFIERS

The first section of this article emphasized that statistical methods must be used to evaluate the performance of both NN and statistical classifiers. The next two sections suggested that even when neural network and statistical classifiers are comparable in terms of their ultimate performance, NN approaches may have other advantages to support their use. This section offers suggestions for evaluating the performance of NN classifiers and for increasing their efficiency. The suggestions will be familiar to any statistician.

How Should Input Data Be Preprocessed for a Neural Network?

One frequently used transformation in statistics is to convert each element in the input vector to its corresponding z-score[35,38] for the corresponding data column. A common variant is merely to convert each data column to a zero-mean vector,[39] which minimizes interference between input elements.[40] The z-score is obtained by subtracting the mean of the data column in question from each element, and then dividing the resultant value by the standard deviation of that column; the data column is thus linearly transformed to a zero-mean vector with unit variance. This simple method has several advantages: (1) It seems to speed up the learning process; (2) weights become interpretable as measures of the importance of deviations from the mean value, as opposed to deviations from zero; (3) initializing the input-to-hidden weights to zero has the effect of setting all weights immediately to the mean value for that element; (4) the average squared score becomes equal to one; and (5) every score in any column becomes equivalent to a corresponding score in any other column in terms of percentile ranks (assuming the original scores were normally distributed). These advantages theoretically should not be reflected in ultimate performance, but they do seem to speed up learning, and they make the input-to-hidden weight matrix easier to interpret.

How Can the Number of Input Nodes Be Minimized?

A more general solution is likely to be generated with an input vector that is stripped of elements that do not contribute to improving the performance. One approach is to use a strategy adopted in stepwise regression, where one input variable at a time is added to or subtracted from the input vector and a regression analysis is performed. The variable that maximizes the change in *POV* is kept (or deleted). With NN classifiers (or any nonlinear classifier), throwing variables out makes more sense than adding variables. For example, suppose you have a database with four input variables that together predict the effectiveness of a drug. Suppose further that one variable (e.g., eye color) was weakly related to the dependent variable, and the other three variables (e.g., dosage, age, and sex) were unrelated to each other and to the dependent variable. A three-way interaction occurs, however, such that moderate doses of the drug are quite effective for males in their mid-thirties, but it has no effect otherwise. By adding variables one at a time, only eye color would make it into the model, because none of the other three variables alone would ever increase the *POV* accounted for. On the other hand, starting with all the variables would result in a catastrophic drop in *POV* accounted for as soon as any one of the three variables involved in the three-way interaction was removed. Removing eye color, however, would not affect the *POV* accounted for (even though it is weakly correlated with the dependent variables), since the interaction term (in this example) can fully account for the results.

Another, less computationally intensive, approach would be to replace each input vector with the vector of principal component scores for which significant eigenvalues exist. Although this approach is likely to reduce the input dimensionality, it may not be helpful if nonlinear relationships dominate the input/output mapping.[41] Large eigenvalues provide information about the most important eigenvectors if the problem is a linear one. It is quite easy, however, to construct problems in which no variance occurs in the eigenvalues; this would happen, for example, if the output were strictly determined by interactions between the independent variables.

How Should the Training and Test Sets Be Split To Provide the Best Estimate of Error on the Test Set?

The obvious answer is to use large random samples of the population for both the training and test sets; alas, realistic samples are frequently so small that assuming good random samples is likely to be gratuitous. A general strategy for using small samples is to use a stratified random sample, or to resample many times on the pattern vectors that are available. Resampling techniques include the jackknife method[42,43] and other variations that include the well-known leaving-one-out method, k-fold sampling methods, and bootstrap methods.[26,42,43] In bootstrap methods, for example, the training and test sets are originally pooled into one large sample. The training set is generated by sampling the pooled set with replacement (i.e., each sample that is taken is replaced before another sample is taken) until a training set is developed that is

equal in size to the original pooled sample. In one variant of the bootstrap method, the e0 bootstrap, those pattern vectors not sampled constitute the test set, and the error rate on that set is the estimate of the true error rate for the sample. Typically, many of these partitions are generated, and the average of the estimated sample error rates defines the estimated population error rate.

Although such resampling methods are computationally very expensive, they are no longer infeasible. These powerful methods can and should be adopted by the developers of NN classifiers. The computational burden is even greater for NN sampling methods, because the effects of learning rate and momentum terms, as well as the effects of initial random weights, for example, must also be taken into account.

Although such methods permit accurate estimates of the true error rate, they do not give us the weight matrix that results in that error rate; obviously, the weight vectors cannot be merely averaged over all samples. Thus, the suggestion that weight training should stop once the error rate on the test set becomes flat or begins to increase is still valid. In other words, the weight-stopping strategy is relevant once one needs a set of weights to solve the problem that the classifier was designed to solve. The resampling methods give good estimates of the true error rate, but they cannot provide an actual set of weights.

CONCLUSIONS

Developers of applications that employ NN components have much to gain by incorporating the tools that are routinely used by statisticians. Neural networks are powerful tools, but using them well requires foresight and careful interpretation. Best results require a consideration of the NN architecture, appropriate transformations of the input and output vectors, sampling methods, strategies for selecting the final weight matrix that avoid the overlearning effect, and the most appropriate performance measures. The approaches suggested here can serve well but, like any set of rules, slavish devotion is to be avoided if the particulars of a situation dictate principled changes to them.

Neural network classifiers should not be looked at as competitors to classical approaches, but rather as extensions of them. As NN classifiers prove their worth, it is hoped that statisticians will embrace them as one more set of useful statistical tools.

REFERENCES

[1] Eberhart, R. C., Dobbins, R. W., and Webber, W. R. S., "Case Net: A Neural Network Tool for EEG Waveform Classification," in *Proc. IEEE Symp. on Computer Based Medical Systems*, IEEE Computer Society Press, Minneapolis, Minn., pp. 60–68 (1989).

[2] Wilson, K., Webber, W. R. S., Lesser, R. P., Fisher, R. S., Eberhart, R. C., and Dobbins, R. W., "Detection of Epileptiform Spikes in the EEG Using a Pattern-Independent Neural Network," in *Proc. IEEE Symp. on Computer Based Medical Systems*, IEEE Computer Society Press, Baltimore, Md., pp. 264–271 (1991).

[3] Bankman, I. N., Sigillito, V. G., Wise, R. A., and Smith, P. L., "Detection of the EEG K-Complex Wave with Neural Networks," in *Proc. IEEE Symp. on Computer Based Medical Systems*, IEEE Computer Society Press, Baltimore, Md., pp. 280–287 (1991).

[4] Eberhart, R. C., and Dobbins, R. W., "Neural Networks versus Bayesian Diagnosis of Appendicitis," in *Proc. 12th Annual International Conf. of the IEEE Engineering in Medicine and Biology Society*, Vol. 12, pp. 1447–1448 (1990).

[5] Olsen, D. E., Cristion, J. A., and Spaur, C. W., "Automatic Detection of Epileptic Seizures Using Electroencephalographic Signals," *Johns Hopkins APL Tech. Dig.* **12**, 182–191 (1991).

[6] Lee, D. G., "Preliminary Results of Applying Neural Networks to Ship Image Recognition," in *Proc. IEEE and INNS International Joint Conf. on Neural Networks*, Erlbaum, Hillsdale, N.J., Vol. II, p. 576 (1989).

[7] Sigillito, V. G., Wing, S. P., Hutton, L. V., and Baker, K. B., "Classification of Radar Returns from the Ionosphere Using Neural Networks," *Johns Hopkins APL Tech. Dig.* **10**, 262–266 (1989).

[8] Newell, P. T., Wing, S., Meng, C.–I., and Sigillito, V. G., "A Neural-Network-Based System for Monitoring the Aurora," *Johns Hopkins APL Tech. Dig.* **11**, 291–299 (1990).

[9] Newell, P. T., Wing, S., Meng, C., and Sigillito, V., "The Auroral Oval Position, Structure, and Intensity of Precipitation from 1984 Onward: An Automated On-Line Data Base," *J. Geophys. Res.* **96**, 5877–5882 (1991).

[10] Roth, M. W., "Neural Networks for Extraction of Weak Targets in High Clutter Environments," *IEEE Trans. on Systems, Man, and Cybernetics* **19**, 1210–1217 (1989).

[11] Boone, B. G., Constantikes, K. T., Fry, R. L., Gilbert, A. S., and Kulp, R. L., "New Directions in Missile Guidance: Signal Processing Based on Neural Networks and Fractal Modeling," *Johns Hopkins APL Tech. Dig.* **11**, 28–36 (1990).

[12] Sigillito, V. G., Sadowsky, J., Bankman, I. N., and Willson, P., "Neural Network Analysis of Images for Non-Destructive Evaluation," in *Proc. IEEE and INNS International Joint Conf. on Neural Networks*, Erlbaum, Hillsdale, N.J., Vol. II-A, p. 933 (1991).

[13] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Rumelhart, D. E., and McClelland, J. L. (eds.), MIT Press, Cambridge, Vol. 1, pp. 318–362 (1986).

[14] Swets, J. A., "Measuring the Accuracy of Diagnostic Systems," *Science* **240**, 1285–1293 (1988).

[15] Adlassing, K.–P., and Scheithauer, W., "Performance Evaluation of Medical Expert Systems Using ROC Curves," *Comput. Biomed. Res.* **22**, 297–313 (1989).

[16] Meistrell, M. L., "Evaluation of Neural Network Performance by Receiver Operating Characteristics Analysis: Examples from the Biotechnology Domain," in *Proc. 13th Annual Symp. on Computer Applications in Medical Care*, IEEE, Washington, D.C., pp. 295–301 (1989).

[17] Pineda, F. J. "Dynamics and Architecture for Neural Computation," *J. Complexity* **4**, 216–245 (1988).

[18] Hripcsak, G., "The Estimation of Posterior Probabilities Using Neural Networks," in *Proc. 12th Annual Symp. on Computer Applications in Medical Care*, IEEE, Washington, D.C., pp. 240–244 (1988).

[19] Horne, B., and Hush, D., "On the Optimality of the Sigmoid Perceptron," in *Proc. IEEE and INNS International Joint Conf. on Neural Networks*, Erlbaum, Hillsdale, N.J., Vol. I, pp. 269–272 (1990).

[20] Lippman, R. P., "A Critical Overview of Neural Network Pattern Classifiers," in *Proc. 1991 IEEE Workshop: Neural Networks for Signal Processing*, IEEE Signal Processing Society and the IEEE Neural Networks Council, Piscataway, N.J., pp. 266–275 (1991).

[21] Makhoul, J., "Pattern Recognition Properties of Neural Networks," in *Proc. 1991 IEEE Workshop: Neural Networks for Signal Processing*, IEEE Signal Processing Society and the IEEE Neural Networks Council, Piscataway, N.J., pp. 173–187 (1991).

[22] Sigillito, V. G., and Eberhart, R. C., "Network Analysis," in *Neural Network PC Tools: A Practical Guide*, Eberhart, R. C., and Dobbins, R. W. (eds.), Academic Press, San Diego, pp. 177–188 (1990).

[23] Hanson, S. J., and Burr, D. G., "What Connectionist Models Learn: Learning and Representation," *Behav. Brain Sci.* **13**, 471–488 (1990).

[24] Stone, G. O., "An Analysis of the Delta Rule and the Learning of Statistical Associations," in *Parallel Distributed Processing*, Rumelhart, D., and McClelland, J. (eds.), MIT Press, Cambridge, Vol. 1, pp. 444–459 (1986).

[25] Ferguson, G. A., *Statistical Analysis in Psychology and Education*, McGraw-Hill, New York (1981).

[26] Weiss, S., and Kulikowski, C. A., *Computer Systems That Learn*, Morgan Kaufman Publishers, San Mateo, Calif. (1991).

[27] Ng, K., and Lippman, R. P., "A Comparative Study of the Practical Characteristics of Neural Network and Conventional Pattern Classifiers," in *Advances in Neural Information Processing Systems*, Morgan Kaufman, San Mateo, Calif., pp. 970–976 (1991).

[28] Tsoi, A. C., and Pearson, R. A., "Comparison of Three Classification Techniques, CART, C4.5 and Multilayer Perceptions," in *Advances in Neural Information Processing Systems*, Morgan Kaufman, San Mateo, Calif., pp. 963–969 (1991).

[29] White, H., "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," *Neural Networks* **3**, 535–550 (1990).

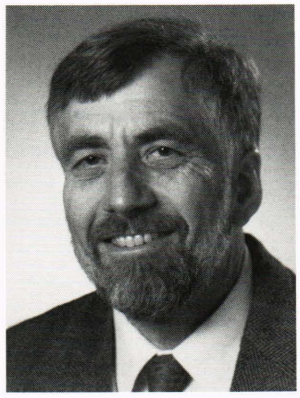
[30] Hornik, K., Stinchonke, M., and White, M., "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks* **3**, 551–560 (1990).

[31] Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks* **2**, 192–198 (1989).

[32] Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, Reading, Mass. (1990).
[33] Hutton, L. V., Sigillito, V. G., and Johannes, R. S., "An Interaction Between Auxiliary Knowledge and Hidden Nodes on Time to Convergence," in *Proc. 13th Annual Symp. on Computer Applications in Medical Care*, IEEE Computer Society Press, Washington, D.C., pp. 302–306 (1989).
[34] Morgan, N., Wooters, C., and Hermansky, H., "Experiments with Temporal Resolution for Continuous Speech Recognition with Multi-Layer Perceptrons," in *Proc. 1991 IEEE Workshop: Neural Networks for Signal Processing*, IEEE Signal Processing Society and the IEEE Neural Networks Council, Piscataway, N.J., pp. 405–410 (1991).
[35] Intrator, N., and Tajchman, G., "Supervised and Unsupervised Feature Extraction from a Cochlear Model for Speech Recognition," in *Proc. 1991 IEEE Workshop: Neural Networks for Signal Processing*, IEEE Signal Processing Society and the IEEE Neural Networks Council, Piscataway, N.J., pp. 461–469 (1991).
[36] Mingers, J., "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Machine Learning* **4**, 227–243 (1989).
[37] White, H., "Learning in Artificial Networks: A Statistical Perspective," *Neural Computation* **1**, 425–464 (1989).
[38] Wu, L., and Fallside, F., "Vector Quantization with a Codebook-Excited Neural Network," in *Proc. 1991 IEEE Workshop: Neural Networks for Signal Processing*, IEEE Signal Processing Society and the IEEE Neural Networks Council, Piscataway, N.J., pp. 432–441 (1991).
[39] Lang, K. J., Waibel, A. H., and Hinton, G., "A Time-Delay Neural Network Architecture for Isolated Word Recognition," *Neural Networks* **3**, 23–43 (1990).
[40] Hinton, G. E., and Plaut, D. C., "Using Fast Weights to Deblur Old Memories," in *Proc. 9th Annual Conf. of Cognitive Science Society*, Erlbaum, Hillsdale, N.J., pp. 177–186 (1987).
[41] Intrator, N., "Feature Extraction Using an Unsupervised Neural Network," in *Proc. 1990 Connectionist Models Summer School*, Touretzky, D. S, Elmer, S. C., Sejnowski, T. J., and Hinton, G. E. (eds.), Morgan Kaufman, San Mateo, Calif. (1990).
[42] Efron, B., *The Jackknife, the Bootstrap, and Other Resampling Plans*, Society for Industrial and Applied Mathematics, Philadelphia (1982).
[43] Efron, B., and Tibshirani, R., "Statistical Data Analysis in the Computer Age," *Science* **253**, 390–395 (1991).

ACKNOWLEDGMENTS: The author wishes to thank Jim Spall for his patience and critical comments, Bob Fry for pointing out the importance of Bayesian interpretations, Bob Jenkins for his recognition that initial conditions had interesting effects upon network performance and deserved to be investigated, Dave Aha and John Sommerer for their suggestions concerning representation of the results, and Vince Sigillito, who first convinced me that test set performance was indeed an appropriate measure of a network's ability to generalize.

THE AUTHOR



LARRIE V. HUTTON is a member of APL's Senior Professional Staff and has worked in the Mathematics and Information Science Group of the Milton S. Eisenhower Research Center since 1988. He attended Michigan State University and received his Ph.D. in experimental psychology from Ohio University in 1969. He has taught courses in psychology, computer science, and artificial neural networks at the Homewood campus of The Johns Hopkins University and in the Continuing Professional Program at APL. Current interests include using artificial neural networks as models of behavioral and cognitive systems, the development of a connectionist system to track multiple targets, and the evaluation of diagnostic systems. Dr. Hutton is a member of the IEEE, the International Neural Network Society, and Sigma Xi.