The Back-Prop and No-Prop Training Algorithms

Bernard Widrow, Youngsik Kim, Yizheng Liao, Dookun Park, and Aaron Greenblatt

***Abstract*—Back-Prop and No-Prop, two training algorithms for multi-layer neural networks, are compared in design and performance. With Back-Prop, all layers of the network receive least squares training. With No-Prop, only the output layer receives least squares training, whereas the hidden layer weights are chosen randomly and then fixed. No-Prop is much simpler than Back-Prop. No-Prop can deliver equal performance to Back-Prop when the number of training patterns is less than or equal to the number of neurons in the final hidden layer When the number of training patterns is increased beyond this, the performance of Back-Prop can be slightly better than that of No-Prop. However, the performance of No-Prop can be made equal to or better than the performance of Back-Prop by increasing the number of neurons in the final hidden layer. These algorithms are compared with respect to training time, minimum mean square error for the training patterns, and classification accuracy for the testing patterns. These algorithms are applied to pattern classification and nonlinear adaptive filtering.**

***Index Terms*—Neural Networks, Back-Propagation, No-Prop, Least Squares Training, Capacity, Pattern Classification, Nonlinear Adaptive Filtering.**

I. Introduction

THE Back-Prop (Backpropagation) algorithm of Paul Werbos is the most widely used method for training multiple-layered neural networks. It is based on the method of steepest descent. An instantaneous gradient of the mean square error with respect to each of the weights (synapses) of the network is computed, and the weights are changed by small amounts in the direction of the negative gradient. The process is repeated over again and again as the mean square error becomes smaller. The objective is to minimize mean square error. Details of this well-known algorithm are available in the literature [1][2].

A new training algorithm for multi-layered neural networks has been introduced by Widrow et. al.[3]. We call this algorithm No-Prop. Closely related prior work by Huang et. al. describing an algorithm that they call the Extreme Learning Machine (ELM) has been proposed [4][5][6][7]. Under many circumstances, the ELM and No-Prop algorithms can be substitutes for Back-Prop, offering faster and more predictable convergence, and greater simplicity. Back-Prop is a remarkable algorithm. No-Prop on the other hand is a very simple option. The purpose of this paper is to compare Back-Prop and No-Prop and to further explore the properties of No-Prop.

With the Back-Prop algorithm, all of the weights of the multi-layered network are changed during each training cycle. Errors at the output layer are back propagated throughout the network in order to compute the components of the gradient. With the No-Prop algorithm, only the weights of the output layer are changed during each training cycle. The weights of the hidden layers are not trained and the output errors are not

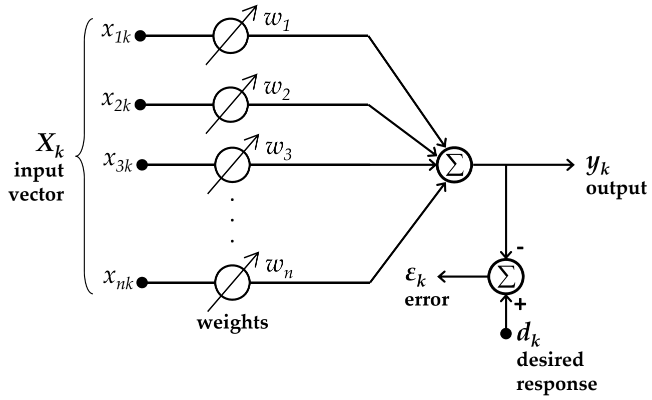

Fig. 1: An adaptive linear combiner

back propagated. Hence the name No-Prop. The weights of the hidden layers are initially set randomly and independently, and they remain fixed. The ELM algorithm also fixes the hidden-layer weights and trains the output layer.

Back-Prop and No-Prop are supervised training algorithms. No-Prop is easy to build in hardware and easy to implement in software. No-Prop is less flexible than Back-Prop, but the lack of adaptivity in the hidden layers has in many cases no effect or only a small effect on performance. The performance of No-Prop can always be enhanced by increasing the number of weights of the output layer neurons, thus allowing No-Prop to be equivalent, for many applications, to Back-Prop.

It should noted that the earliest neural network configured with a random hidden layer and an adaptive output layer was Rosenblatt's perceptron [8] [9] [10]. The perceptrion adaptation rule, however, was not based on least squares.

II. The linear combiner and its LMS capacity

In order to compare the Back-Prop and No-Prop algorithms, the LMS capacity of neural networks will be defined. Capacity has to do with the number of patterns that can be perfectly trained in, with zero mean square error [3]. The simplest way to study this would be to begin with a single neuron and its weights, the adaptive linear combiner of Figure 1. This is a trainable element that works as follows. A set of input training patterns, input vectors, are presented together with associated desired responses. The k th input pattern vector is X_k, and its desired response, a scalar, is d_k. Least squares methods can be used to find a set of weights $w_1, w_2, \dots, w_n$ that yield outputs y_k that match the desired responses d_k as well as possible for the set of training patterns. The objective is to set the weights to values that minimize the mean square error, averaged over the set of training patterns. For each input pattern, the error

ϵ_k is the difference between the output response y_k and the desired response d_k. The weights are the components of the weight vector W_k. Let the number of input training patterns be N. Referring to Figure 1,

$$\begin{aligned} y_k &= {X_k}^T W_k \\ \epsilon_k &= d_k - y_k = d_k - {X_k}^T W_k \end{aligned} \tag{1}$$

The equations (1) can be written as simultaneous equations:

$$\begin{cases} x_{11}w_1 + x_{12}w_2 + \cdots + x_{1n}w_n = y_1 = d_1 - \epsilon_1 \\ x_{21}w_1 + x_{22}w_2 + \cdots + x_{2n}w_n = y_2 = d_2 - \epsilon_2 \\ \quad\vdots \qquad\qquad \vdots \qquad \vdots \\ x_{N1}w_1 + x_{N2}w_2 + \cdots + x_{Nn}w_n = y_N = d_N - \epsilon_N \end{cases} \tag{2}$$

It is clear that for a given set of input pattern vectors $X_1, X_2, \ldots, X_N$ with their respective desired responses $d_1, d_2, \ldots, d_N$, a set of weights $w_1, w_2, \ldots, w_n$ will exist such that all the errors $\epsilon_1, \epsilon_2, \ldots, \epsilon_N$ will be zero as long as $N \leq n$, and the training vectors $X_1, X_2, \ldots, X_N$ are linearly independent. If the training vectors are linearly independent, the maximum number that can be perfectly trained in with zero error is n, equal to the number of weights. If the number of training patterns is greater than the number of weights, i.e. $N > n$, the training patterns cannot all be linearly independent. Weight values can be found that minimize the mean square error (MSE), but the minimum MSE will generally not be zero.

The maximum number of patterns that can be perfectly trained in (zero MSE), equal to the number of weights n, is defined to be the least mean square capacity, the LMS capacity of the linear combiner, the capacity of the single neuron and its weights (synapses). Henceforth, the LMS capacity will be simply called the capacity.

The linear combiner of Figure 1 can be trained with any least squares method. It is quite common to do this with the Widrow-Hoff LMS algorithm [11] [12] [13], as follows:

$$\begin{aligned} W_{k+1} &= W_k + 2\mu\epsilon_k X_k \\ \epsilon_k &= d_k - {X_k}^T W_k \end{aligned} \tag{3}$$

Each input pattern X_k and its associated desired response d_k are presented in sequence, and the present weight vector W_k is changed a small amount to obtain the next weight vector W_{k+1}, reducing the error ϵ_k with each iteration. The parameter μ controls the speed of convergence and stability. This training algorithm will find a set of weights to bring the MSE to zero if the training patterns are linearly independent, but if not, a set of weights will be found that minimizes MSE. This algorithm is based on the method of steepest descent, using an instantaneous gradient with each iteration. The mean square error can be shown to be a quadratic function of the weights [11] [12] [13]. When optimizing by following the gradient, there will be no local optima. The LMS algorithm was invented in 1959. Today, it is the world's most widely used learning algorithm. It is used in adaptive filters which are used in telecom systems for echo cancelling and channel equalization, and for many other applications.

The capacity of the linear combiner, as stated above, is equal to the number of weights. If the number of training patterns is less than capacity, the weights will not be totally constrained by least squares training and this is called "overfitting." If the number of training patterns is equal to or greater than capacity and the weights are totally constrained by training, this is called "underfitting." The ideas of capacity, overfitting, and underfitting will be extended to apply to layered neural networks. For this purpose, some ideas about nonlinear mapping will be needed.

Knowledge of capacity of layered neural networks will be needed when comparing the Back-Prop and No-Prop algorithms.

III. Nonlinear mapping and linear independence

A three-layer neural network is diagrammed in Figure 2. The first two layers, the "hidden layers", have neurons with sigmoidal activation functions. The neurons of the third layer are linear combiners without sigmoids, like the neuron of Figure 1. The two hidden layers provide a nonlinear mapping of the network inputs into a set of inputs for the third layer, the output layer.

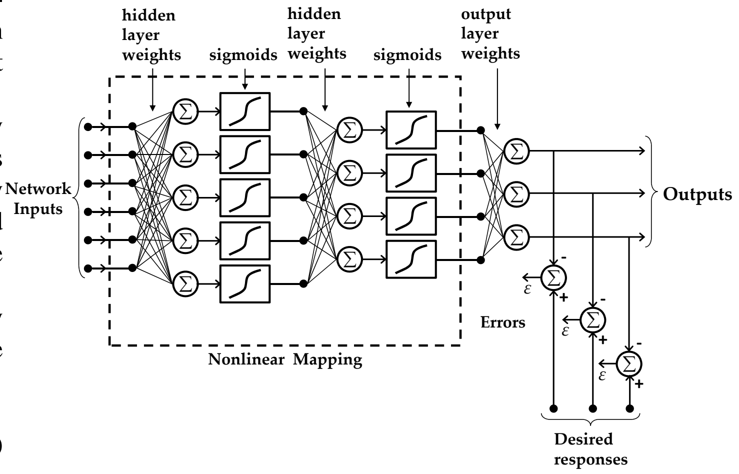

Fig. 2: A Three-Layer Neural Network

When training the neural network of Figure 2, input patterns and their desired response patterns are presented. The weights are changed to minimize the mean of the sum of the squares of the errors, averaged over all the training patterns. The errors are the difference between the outputs and the respective desired responses. Two methods will be used for training networks like this one, Back-Prop and No-Prop.

When the weights of the first two layers are fixed, whether fixed after training with Back-Prop or initially fixed with No-Prop, the nonlinear mapping shown in Figure 2 will be memoryless. For every distinct pattern presented at the network inputs, a distinct pattern will be presented at the inputs to the output layer neurons. The mapping will be one-to-one from the network input to the input of the output layer.

An interesting effect of the nonlinear mapping shown in Figure 2 is the following: If the number of network input patterns is less than or equal to the number of weights of

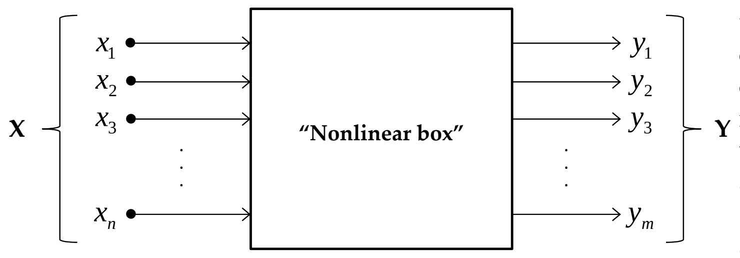

Fig. 3: Memoryless nonlinear mapping

each of the output layer neurons and all of these patterns are distinct, the input patterns to the output layer will be linearly independent vectors, whether or not the network input patterns are linearly independent. This effect holds for 1 hidden layer, 2 hidden layers, or any number of hidden layers. At present, we have no rigorous proof of linear independence, but we have a strong plausibility argument and experimental confirmation [3]. Huang et. al. have shown linear independence for a single hidden layer by means of a statistical argument [5].

Linear independence is the key to the idea of capacity of a layered neural network. An argument for linear independence follows. In Figure 3, a "nonlinear box" is shown doing nonlinear mapping. One example of a nonlinear box is the nonlinear mapping seen in Figure 2.

In Figure 3, a set of X vectors are inputs to the nonlinear box. A corresponding set of Y vectors are outputs of the nonlinear box. The set of X vectors are distinct and may or may not be linearly independent. The input X vectors have n components, and the output Y vectors have m components. The box is memoryless, and distinct input vectors will yield distinct output vectors. The mapping is instantaneous and one-to-one.

Let there be three distinct input vectors that cause three distinct output vectors. In other words, $X_1 \rightarrow Y_1$, $X_2 \rightarrow Y_2$, and $X_3 \rightarrow Y_3$. These vectors are

$$X_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} \quad X_2 = \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} \quad X_3 = \begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} \tag{4}$$

$$Y_1 = F\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} \quad Y_2 = F\begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} \quad Y_3 = F\begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} \tag{5}$$

where F is a memoryless nonlinear mapping function. The question is, does $Y_3 = \alpha Y_1 + \beta Y_2$? In other words, does

$$F\begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} = \alpha F\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} + \beta F\begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} ? \tag{6}$$

Would a set of constants α and β exist for the nonlinear equation (6) to be correct? The answer is generally no, and as such, Y_1, Y_2, and Y_3 would be linearly independent. It would be very unlikely but not impossible for the above equation to hold. A pathological case could occur with a carefully chosen set of X input vectors and a carefully chosen nonlinear mapping function. If equation (6) holds, then the three Y vectors would be linearly dependent. Otherwise, they are linearly independent.

Given the randomly chosen nonlinear mapping function as illustrated in Figure 2, where the hidden layer weights are chosen randomly and fixed. It is highly unlikely that equation (6) would hold, and linear independence would be almost perfectly assured. It is impossible to prove linear independence deterministically, since linear dependence could actually happen, but this would be extremely rare. A set of simulation experiments have been performed to demonstrate this rarity.

IV. Linear Independence Experiments

By computer simulation, an extensive set of experiments was performed in order to confirm or contradict the above argument about linear independence, as well as this could be resolved by experimentation. In each case, the number of input patterns was chosen to be equal to the number of outputs m of the nonlinear box of Figure 3. The corresponding output patterns or output vectors were tested for linear independence by means of the MATLAB function called "**rank**". If the rank of the set of output vectors equalled the number of output vectors, the output patterns were linearly independent. Otherwise, the output patterns were linearly dependent. For these experiments, the nonlinear box of Figure 3 was actually a neural network with randomly chosen fixed weights. The size of the network and the number of layers was varied in the course of the experiments.

For all of the experiments, the input patterns were random vectors whose component values were gray scale from -1 to +1. Thirty percent of the patterns were independently generated. Seventy percent were linear combinations of the first thirty percent. These input patterns were all distinct, but clearly not linearly independent. In spite of this, the question is, would the output patterns from the nonlinear box be linearly independent?

The first experiments were done with the hidden layers of the neural network of Figure 2, except that only one hidden layer was used. The weights of this layer were randomly independently chosen and fixed. The number of neurons in this layer was 400, giving 400 hidden layer outputs. Four hundred input patterns were generated and the 400 hidden layer output patterns were tested and found to be linearly independent. This experiment was repeated 1000 times with different random weights. In every single case, the output patterns were linearly independent.

A second set of experiments was done, again with only one hidden layer. The number of neurons in that layer being alternatively 100 and 200, tested with 100 and 200 input patterns respectively. These experiments were repeated 1000 times and in every single case, the output patterns were linearly independent.

A third set of experiments was performed with networks having two fixed hidden layers. The first layer had alternatively 100, 200, and 400 neurons connected in turn with all combinations of 100, 200, and 400 neurons in the second fixed layer. The input pattern vectors had 200 components generated as above, and the number of input patterns chosen in each case was equal to the number of outputs of the second layer. The second layer output vectors were linearly independent. The experiments were repeated 1000 times, and in every case the output vectors were linearly independent.

A fourth set of experiments was performed with networks having three fixed hidden layers, 200 neurons in the first fixed layer, 100 neurons in the second fixed layer, connected with 100, 200, or 400 neurons in the third fixed layer. The number of input patterns chosen in each case was equal to the number of neurons in the third hidden layer. These experiments were repeated 1000 times and in every case, the third-layer output vectors were linearly independent.

The total number of experiments was 15000. In every case, the output vectors created by nonlinear mapping were linearly independent. Not a single case of linear dependence was observed. Linear dependence is highly unlikely and we can now accept this with a great deal of confidence.

V. Capacity of layered neural networks

The capacity (the LMS capacity) of the single neuron of Figure 1, defined as the number of linearly independent patterns that can be perfectly trained in with zero MSE, is equal to the number of weights. This idea can be generalized for layered neural networks. Consider the 3-layer network of Figure 2. Let the weights of the two hidden layers be randomly chosen and fixed. Let the weights of the third layer, the output layer, be trained with the LMS algorithm. The question is, how many patterns can be perfectly trained in, with zero mean square error? The capacity of the network is defined to be the answer to this question.

Perfect training of the output layer requires that the error for each output-layer neuron be zero for each of the network input training patterns. This will be possible if the input patterns to the output layer are linearly independent. Note that all the output-layer neurons have the same input patterns. Their input patterns will be linearly independent as long as the network input patterns are distinct and their number is less than or equal to the number of weights of each output-layer neuron.

The number of network input patterns that can be perfectly trained in, the network capacity, is therefore equal to the number of weights of each of the neurons of the output layer. The capacity does not depend on the number of hidden layers. The capacity can be stated as being equal to the number of neurons in the final hidden layer, just before the output layer. The number of neurons in the other hidden layers would have no effect on capacity. The capacity is not affected by the network input patterns being linearly dependent or linearly independent, as long as they are distinct. The capacity of this network does not depend on the random choice of the fixed weights of the hidden layers, as long as they are chosen with finite non-zero values, negative or positive. The random weights are usually chosen to have zero mean.

The capacity of a network is an important property, as will be seen with the application examples to be presented below. It is an important property when comparing the Back-Prop and No-Prop algorithms.

VI. Nonlinear mapping; 1-to-1 mapping

The Back-Prop and No-Prop algorithms will be compared as training algorithms for multi-layer neural networks used for pattern classification. One might wonder about the effects of nonlinear mapping, that this might destroy whatever structure the input patterns have and make the classification task more difficult. This raises the immediate question, what is the purpose of the hidden layer or layers for pattern classification?

Without the hidden layers, only linear classification would be possible. The input patterns would need to be linearly separable [14][15]. With one or more hidden layers, this limitation is overcome and more general nonlinear separation becomes possible.

Memoryless nonlinear mappings with layered neural networks having sigmoidal activation functions map small neighborhoods into small neighborhoods. They map small clusters of points into small clusters of points, even when going through hidden layers with random fixed weights, as with No-Prop. The reason is that Jacobian matrices for mapping from input space to output space exist throughout the input space.

Consider the nonlinear box of Figure 3. Let an input vector be X_A. A matrix of partial derivatives, the Jacobian matrix, can be written as follows:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}_A \tag{7}$$

Now let another input vector be X_B. The Jacobian can be written as

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}_B \tag{8}$$

For layered neural networks with sigmoidal activation functions, these derivatives will exist, and will vary continuously if the input vector were varied gradually, as if for example one wished to morph vector X_A into vector X_B. Small changes in the input vector will cause small changes in the output vector. Neighborhoods will map into neighborhoods.

It is common for an input pattern, a vector with n components, to be represented as a point in an n-dimensional space. A pure noise free input pattern is thereby a point in the space. A collection of noisy versions of the pure pattern is a cluster of points in the space. Let all the points of the cluster be contained within a manifold. The manifold itself is an infinite set of points in the space. The input manifold and all the points therein will map into an output manifold with internal points corresponding to the internal points of the input manifold. This is a consequence of a neighborhood mapping into a neighborhood. The output manifold will usually be a distorted form of the input manifold.

Now suppose that there is a second pure noise free input pattern, and a second collection of noisy versions of it. Let there be a second manifold that encloses the second cluster in the input vector space. Assume that the two manifolds do not overlap and have no points in common. The question is, could the two output manifolds overlap?

This is a difficult question to answer in general. If the output manifolds do not overlap, this would be desirable. Separability of the input manifolds would lead to separability of the output manifolds. In that case, nonlinear mapping would "do no harm" from the point of view of pattern classification. If the output manifolds were to overlap, then the nonlinear mapping would instill confusion and perfect pattern separation would not be possible with the outputs of the nonlinear map. Harm will have been done.

Nonlinear mapping is useful, providing linear independence and allowing nonlinear separation, but it sometimes could cause overlap of clusters that were originally separated. The clusters and manifolds sketched in Figure 4 illustrate the idea.

Referring to Figure 4, points in the input space map into

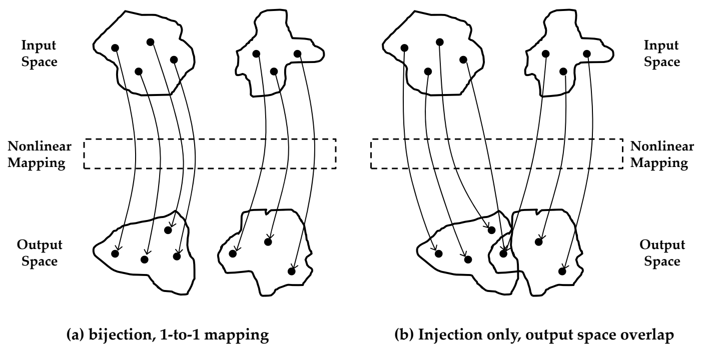

Fig. 4: Nonlinear Mapping: (a) bijection, one-to-one mapping, (b) injection, but not bijection.

points in the output space. This is injection. In the reverse or inverse direction, points in the output space correspond to unique points in the input space, one-to-one correspondence, in Figure 4(a). This is bijection. In Figure 4(b), not all points in the output space correspond to unique points in the input space. Where the manifolds overlap, points in the overlap region correspond to points in both input manifolds and are not uniquely identifiable.

Figure 5 shows simple one-layer neural networks that perform nonlinear mapping. The vector X is the input. The vector Y' is the output. The weights are chosen randomly, and are represented by the matrix W whose rows correspond to the weight vectors of the individual neurons. The linear sums are represented by the vector Y. The nonlinearities are sigmoids. The network of Figure 5(a) has 3 inputs and 3 outputs. The network of Figure 5(b) has 5 input and 3 outputs.

For every X vector, there will be a unique Y vector and a

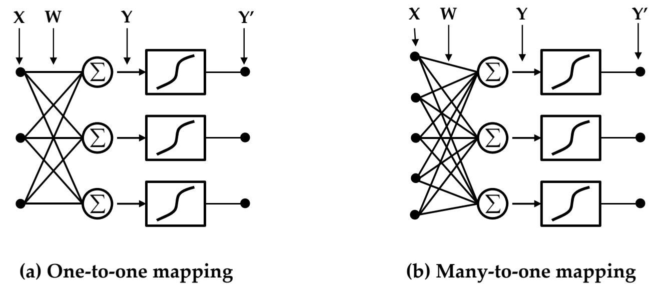

Fig. 5: Neural Networks for Nonlinear Mapping

unique Y' vector. The question is, is there a unique X vector that corresponds to every Y' vector? The answer is, possibly yes for the network of Figure 5(a), and no for the network of Figure 5(b).

Because of the monotonic nature of the sigmoids, for every Y' there will be a unique Y, so the question can be changed to, is there a unique X vector that corresponds to every Y vector? Referring to Figure 5(a), the vector Y can be written as

$$Y = WX \tag{9}$$

If W^{-1} exists, then X can be written as

$$X = W^{-1}Y \tag{10}$$

and there will be a unique X for every Y. Therefore, there will be a unique X for every Y'. This will not be the case for the neural network of Figure 5(b). The W matrix here is not square and does not have an inverse.

Why would the W-matrix of equation (9) have an inverse? It turns out that since the elements of W are randomly chosen, there is a high probability that W^{-1} exists. The probability gets closer and closer to 1 as the number of neurons is increased, as the dimensions of the W-matrix increases. Mathematical justification for this statement is provided by reference [16].

With 10 or more neurons, it is almost certain that W^{-1} exists. We have tested the rank of 50000 10×10 random matrices, and not a single case of singularity has been observed. Figure 4(a) represents the mapping behavior of the neural network of Figure 5(a), and Figure 4(b) represents a possible mapping behavior of the neural network of Figure 5(b).

By induction we conclude that a layered neural network having equal numbers of neurons in each hidden layer, with the number of neurons in each layer made equal to the number of components of the input vector, with the weights of the hidden layers chosen randomly, the hidden layers will provide 1-to-1 bijection mapping and will not cause cluster overlap when the input clusters do not overlap. This is not affected by the number of neurons in the output layer. Other network configurations might create overlap, which would make pattern classification more difficult or impossible. It should be noted that although overlap would be possible with certain network configurations, overlap will not automatically occur with all sets of input patterns.

VII. Training classifiers with Back-Prop and No-Prop

Classification experiments were performed to compare the behavior of the Back-Prop and No-Prop algorithms. Experiments were done with numbers of training patterns less than or equal to the network capacity, and with numbers of training patterns greater than capacity. The training and testing patterns were obtained from a set of 20,000 Chinese characters. Each character had 20×20 pixels. The content of each pixel was either black or white, represented numerically as +0.5 or -0.5 respectively. Each character was able to be translated up or down by one pixel, and left or right by one pixel. For these experiments, nine distinct patterns were constructed by translation from each Chinese character. An example is shown in Figure 6.

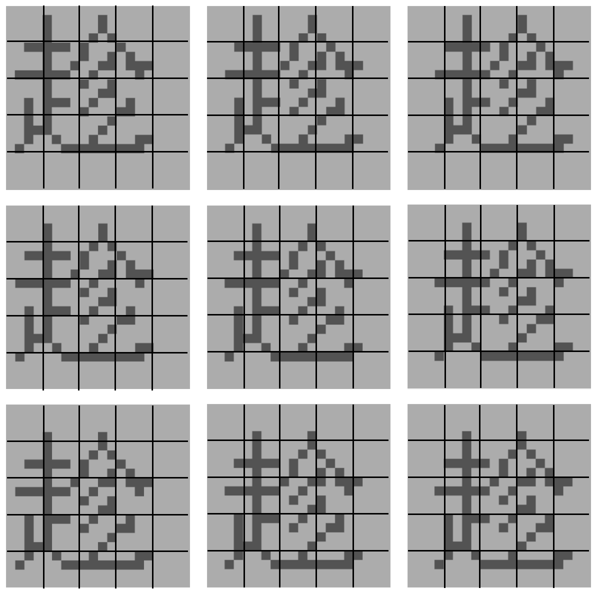

Fig. 6: Nine translations, left-right, up-down, of a Chinese character. Grids are shown to help illustrate the translations. Each of these patterns had 20×20 pixels, black or white.

Noisy patterns were constructed by adding various amounts of independent random noise to the pixels of the pure binary patterns. Figure 7 shows probability density distributions for different amounts of noise, on a percent basis. The percent noise refers to the ratio of noise variance to the variance of the pure pixels. Noise applied to a +0.5 pixel had the probability densities on the right in Figure 7. For a -0.5 pixel, the additive noise had the probability densities on the left. Figure 8 shows

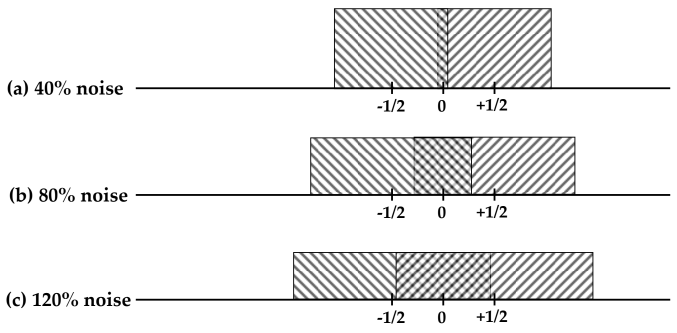

Fig. 7: Probability density distributions for noise additive to binary Chinese characters.

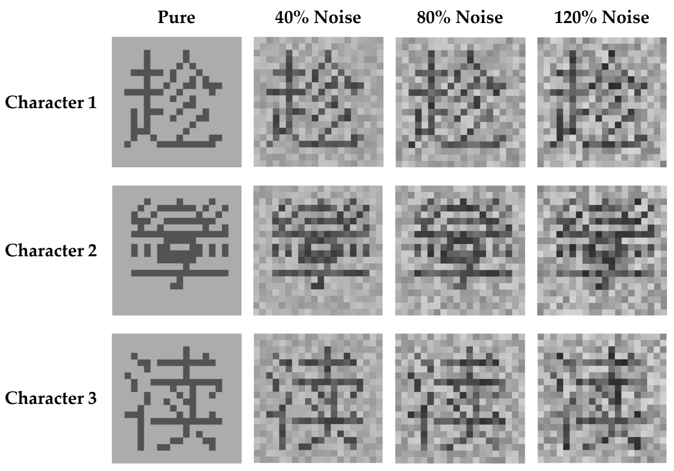

Fig. 8: Examples of Chinese characters with various amounts of noise.

examples of Chinese characters with various amounts of noise.

The neural network that was used for comparing Back-Prop and No-Prop in a classifier application is shown in Figure 9. It is a three-layer network having two hidden layers and one output layer. With Back-Prop, all three layers are trained. With No-Prop, the hidden layers have their weights set randomly and fixed, while the third layer, the output layer, is trained. The input patterns were 20×20, 400 pixels, so the inputs to the neural network were pattern vectors having 400 components. The number of output layer neurons corresponded to the number of pattern classes. For these experiments, there were 50 classes, each class corresponding to a single character. The character is of the same class in all nine positions.

Each output neuron was assigned to a pattern class and was trained to produce a +1 output for that class while all other neurons were trained to -1 outputs for that class. After training, a test pattern would be classified by identification with the output layer neuron with the most positive output. This is a one-out-of-many code.

The network of Figure 9 had 50 output neurons. The two hidden layers had various numbers of neurons but always

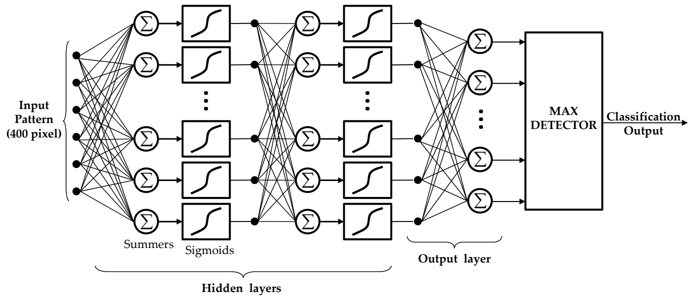

Fig. 9: A trainable neural-net classifier

the same number of neurons in each layer. For different experiments, the hidden layers had 400 neurons, 800 neurons, or 1200 neurons. The network capacities were 400, 800, and 1200, respectively. When trained with Back-Prop with a capacity of 400, the algorithm was designated as BP 400. When trained with No-Prop with a capacity of 400, the algorithm was NP 400. With a capacity of 800, the algorithm was NP 800, and with a capacity of 1200, the algorithm was NP 1200. For all of the experiments, the network was trained with 50 Chinese characters in all 9 positions, with a small amount of independent noise added to the training patterns, 10%. Deliberately adding a small amount of noise to the original pure characters turned out to be beneficial when the trained network was tested with noisy versions of the 50 Chinese characters in any of their nine positions.

The mean square error was minimized by the training process. Errors were observed before training, and the mean of the squares of the errors was the initial MSE. The percent MSE was the ratio of MSE to the mean square of the desired output values, multiplied by 100. Learning curves that show percent MSE as a function of the number of adapt cycles or iterations are shown in Figure 10 for Back-Prop and No-Prop learning.

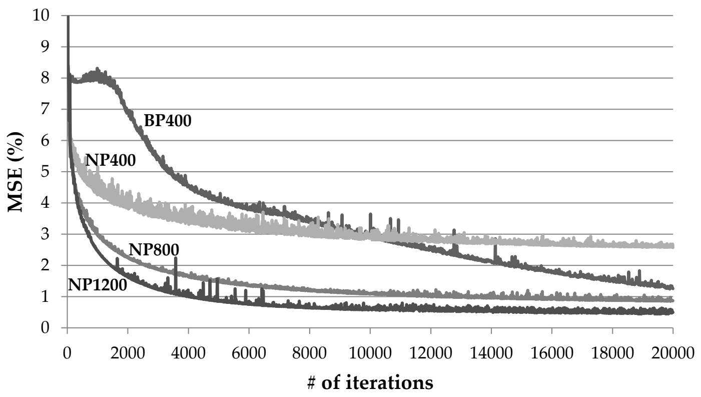

Fig. 10: Back-Prop and No-Prop learning curves. Three-layer neural network, training with noisy patterns with small noise (10%), 50 classes.

The training patterns were 50 Chinese characters. For each character in each translated position, a cluster of 10 noisy versions were constructed with 10% noise making the total

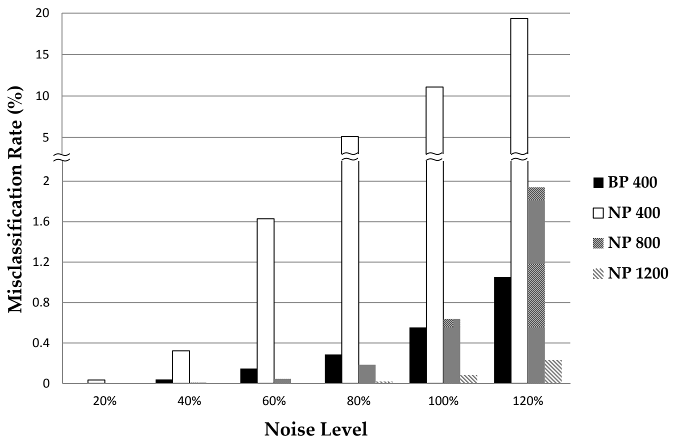

Fig. 11: Classification errors when testing with noisy Chinese characters.

number of training patterns 4500. In all cases, training was underfitted, with the number of training patterns greater than capacity. Back-Prop was performed with a capacity of 400. No-Prop was performed with capacities of 400, 800, and 1200.

The learning curves of Figure 10 show that all the No-Prop algorithms converge with less adaptation cycles than Back-Prop 400. It should be noted that each cycle of Back-Prop 400 with the 3-layer network described above required 1.7M floating operations per training pattern. Increasing the capacity increases the computational cost. With Back-Prop, computation increases with the square of capacity. With No-Prop, computation increases linearly with capacity. For the above network, No-Prop 400 required 0.1M floating operations per cycle, No-Prop 800 required 0.2M floating operations per cycle, and No-Prop 1200 required 0.3M floating operations per cycle. Comparing BP 400 with NP 1200, Back-Prop required almost 5 times as much computation per cycle, and No-Prop converges with many times less cycles than Back-Prop. Both algorithms yielded very low levels of mean square error, less than 0.5% with 4500 training patterns.

Once the network was trained, the question arose about the effectiveness of the network as a classifier of the Chinese characters with additional noise. The issue is generalization with noisy test patterns. Figure 11 presents results of classification experiments with additive noise ranging from 20% to 120%. The misclassification rate, the percent of classification errors over hundreds of noisy versions of each of the 50 characters in all 9 positions, is shown in the figure. The error rates are surprisingly small even with noise at the 120% level. The rate of misclassification was 1% for the BP 400 network, 2% for the NP 800 network, 0.2% for the NP 1200 network.

For these experiments, the performance of No-Prop was made equal to or better than that of Back-Prop by increasing the network capacity by a factor of 3, comparing BP 400 with NP 1200. There was no need to train the hidden layers, and training the output layer with the LMS algorithm of Widrow

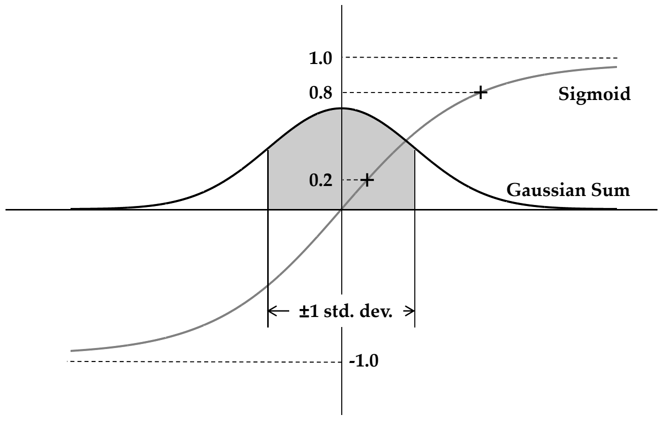

Fig. 12: A Gaussian sum superposed on a sigmoid characteristic.

and Hoff allowed one to precisely predict learning rate and misadjustment [12][13]. With No-Prop, convergence was at a global optimum. With Back-Prop, the rate of convergence was hard to predict and convergence to a global optimum could not be assured.

VIII. Choice of Random Weight Distributions for the Hidden Layers When Using No-Prop

For the experiments with No-Prop described above, the weights of the hidden layers were randomly selected from uniformly distributed probability densities. The choices of the variances were not critical, but some thought was warranted to insure that the sigmoids were not always saturated and alternatively not always operated at low levels making them essentially linear devices.

Consider the example of the Chinese characters. They are 20×20, providing 400 inputs to the first hidden layer neurons. Each pixel has a value of +0.5 or -0.5, plus noise, having a variance of the order of 0.25, plus that of possible additive noise. In the first hidden layer neurons, Gaussian sums (Central limit theorem) are generated whose variances are the variances of the input image pixels multiplied by the variance of the independently randomly chosen weights, multiplied by 400. If the weights are chosen from a uniform density, the variance of the weights will be $\frac{1}{12}$ the square of the width of the uniform density.

In each of the neurons, the sums are applied to sigmoids. The probability density of a Gaussian sum is sketched in Figure 12, superposed on the sigmoid characteristic. Intuitively, the variance of the first layer random weights can be chosen so that abscissas at ± 1 standard deviation of a typical Gaussian sum would intersect its sigmoid at a range of its output values of ± 0.2 to ± 0.8 and this would be fine. This is not critical.

To choose the variance of the random weights of the second hidden layer, the same method can be used. The inputs to the second hidden layer are outputs of the sigmoids of the first hidden layer, and these outputs will have a variance of the order of 0.25.

It should be noted that Back-Prop does not need this choice since it adapts the hidden layer weights automatically, although making this choice when using Back-Prop would insure good initial conditions and would aid the convergence process.

IX. Training Nonlinear Adaptive filters With Back-Prop and No-Prop

Adaptive filters and their applications are described in reference [12] and in many other books and papers. They are an important part of the repertory of digital signal processing. Linear adaptive filters become linear filters after learning, when their weights converge and stablize. Nonlinear adaptive filters become nonlinear filters when their weights converge and stablize. In this section, an adaptive filter that is a combination of linear and nonlinear adaptive filters is described. The nonlinear filter components contained neural networks that were trained alternatively with both Back-Prop and No-Prop. The linear component was trained with LMS. Both components were trained with the same error signal, and both contributed to minimization of the MSE.

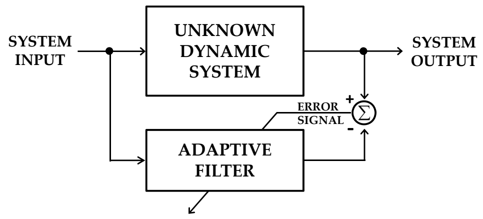

Fig. 13: Using an adaptive filer for modeling an unknown dynamic system.

Figure 13 illustrates an application for adaptive filtering, that of modeling an unknown dynamic system. In the terminology of the control system field, this is called "plant identification." An input signal is applied to both the unknown dynamic system to be modeled and to the adaptive modeling filter. The output of the unknown system becomes the desired response for the adaptive system. The difference between the output of the unknown system and the adaptive filter is the error signal that is used by the adaptive algorithm for training the adaptive filter. Low error indicates a good match. Note that a nonlinear adaptive filter is required for accurate modeling of a nonlinear unknown dynamic system.

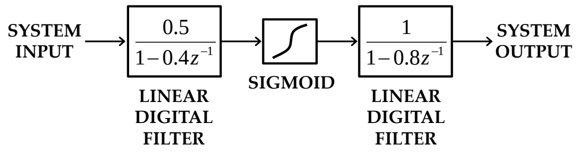

Fig. 14: The unknown dynamic system of this study.

For this study, a nonlinear dynamic system was computer simulated as follows. A sigmoid was flanked by linear digital filters, as is shown in Figure 14. The nonlinearity was created by the sigmoid, and the dynamic aspects were created by the two digital filters. The problem was to model this specific nonlinear dynamic system using the standardized general purpose nonlinear adaptive filter shown in Figures 15 and 16.

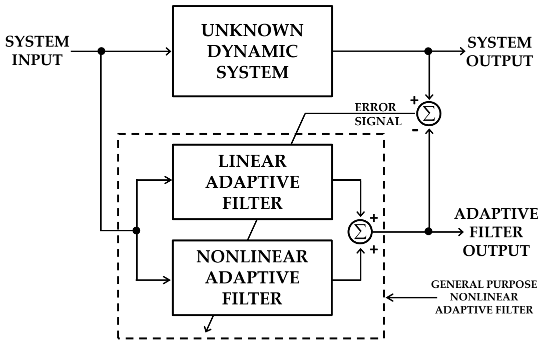

Fig. 15: A general purpose nonlinear adaptive filter, having linear and nonlinear components, modeling an unknown dynamic system. The same error signal is used for training the weights of both adaptive filters.

The linear adaptive filter component is FIR (finite impulse response), a tapped delay line trained with LMS. The nonlinear adaptive filter component is a tapped delay line whose taps provide input signals to a 3-layer neural network trained with Back-Prop or No-Prop. The same error signal is used for both components with the mutual purpose of minimizing mean square error. The experiments of this study compared No-Prop with Back-Prop training of the nonlinear component.

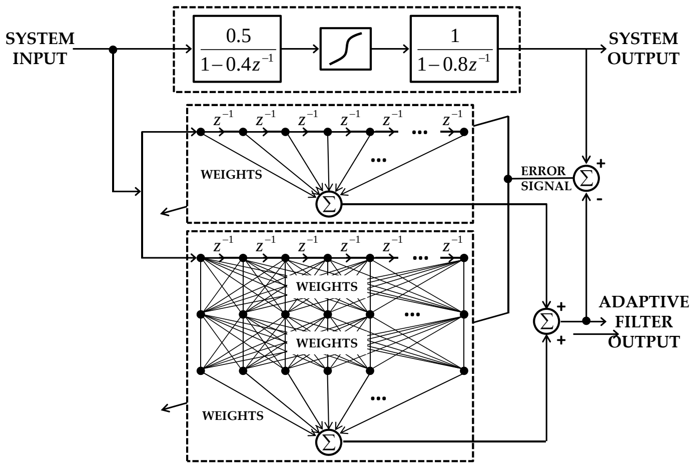

Fig. 16: Details of Figure 15.

For computer simulation, the sampling rate was set at 16 kHz for the unknown dynamic system and for the linear and nonlinear adaptive filters. The tapped delay lines for these filters each had 100 taps. For the nonlinear adaptive filter, both hidden layers had 400 neurons for Back-Prop and 1200 neurons for No-Prop. The random fixed weight values for No-Prop were chosen by observing the criteria of Section VIII above. The system input signal during training was a Gaussian first-order Markov process.

Learning curves for the general purpose nonlinear adaptive system contrasting training with BP400 and training with NP1200 is shown in Figure 17. The two curves are almost identical. These curves give percent MSE versus the number of training cycles. The percent MSE was calculated as the ratio of the mean square of the error signal to mean square of the unknown system output signal, multiplied by 100.

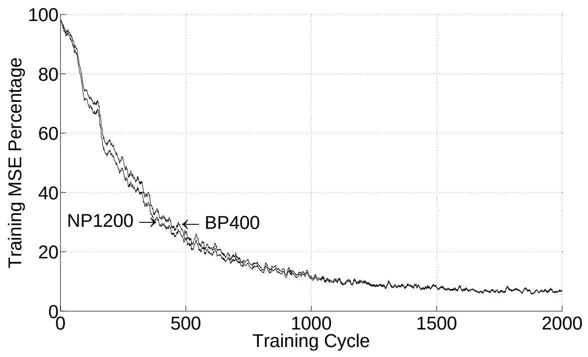

Fig. 17: Learning curves for Training with BP400 and NP1200.

When modeling an unknown nonlinear dynamic system, the model, being generic, would invariably have a different architecture from that of the system being modeled. This is certainly true of the case being studied and illustrated in Figure 16. The model may fit for the training data, and Figure 17 shows low error after training, but would the model fit for input data that it was not specifically trained on? Statistical generalization requires low error on new input data having the same statistical properties as the training data. This did indeed work for the system being tested. Table I shows percent MSE after different number of training cycles, stopping training for testing, then starting training again. BP400 had slightly less error than NP1200.

Training Cycle	MSE Percentage	
	BP400	NP1200
2000	5.48 %	6.63 %
5000	4.37 %	5.44 %
10000	4.12 %	5.10 %
15000	4.00 %	4.96 %

TABLE I: Statistical generalization. Performance results with random input signals not used for training.

The statistical generalization results were quite good for both training algorithms. Further testing for generalization was done with sinusoidal inputs that were of course quite different from the first-order Markov training signals. The results were not perfect, but surprisingly not bad. They are shown in Figure 18 for a 100 Hz input and in Figure 19 for a 3 kHz input.

This study of nonlinear adaptive filtering is a work in progress. This is a very important subject in the field of adaptive signal processing, and will be a subject for future

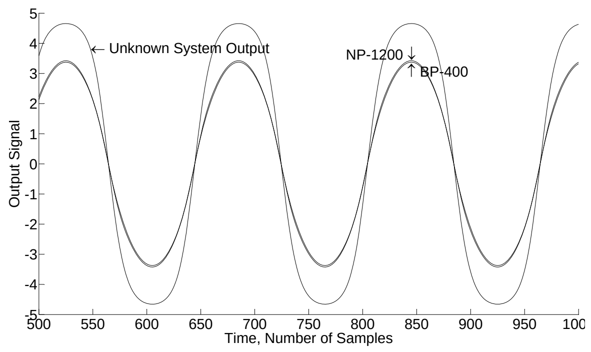

Fig. 18: Generalization with a 100 Hz sinusoidal input signal. Comparison of unknown system output with outputs of adaptive models trained with BP400 having 10.42% MSE and NP1200 having 9.61% MSE. The difference in performance of the two adaptive models is indistinguishable.

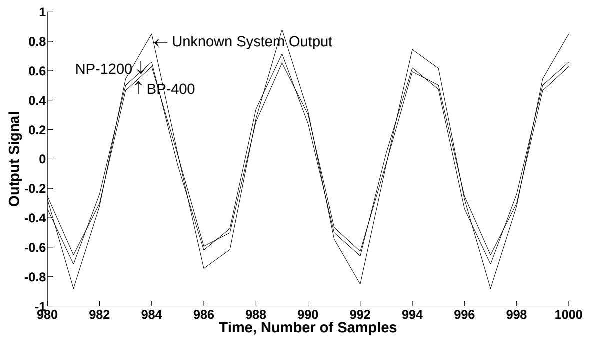

Fig. 19: Generalization with a 3 kHz sinusoidal input signal. Comparison of unknown system output with outputs of adaptive models trained with BP400 having 4.45% and NP1200 having 3.65% MSE. The difference in performance of the two adaptive models is very slight.

research. Both Back-Prop and No-Prop have worked well.

X. Conclusion

Back-Prop and No-Prop are least squares training algorithms for layered neural networks. The purpose of this paper is to compare them regarding structure and performance. Applications to adaptive pattern classification and nonlinear adaptive filtering are described and used in the comparison.

In making the comparison, a useful idea is that of LMS capacity or simply the capacity. The capacity of a layered neural network is equal to the number of neurons in the final hidden layer which is equal to the number of weights in the output layer neurons. Equivalent performance for the two algorithms has been achieved by increasing the capacity of the network when using No-Prop by an approximate factor of 3. With No-Prop, the training algorithm is simpler and has no relative optima, but one may need to use more neurons in the final hidden layer.

The hidden layers of a neural network play an important role. They allow nonlinear separation for adaptive pattern classification, and they provide the required nonlinearity for nonlinear adaptive filtering. When the weights of the hidden layers are randomized and fixed, a concern might be that the input patterns will be scrambled and information might be lost. But the random hidden layers can be designed so that input patterns in neighborhoods that do not overlap can produce output patterns in neighborhoods that also do not overlap. As such, no harm is done by the hidden layer or layers.

The Back-Prop algorithm is very useful and very popular. The No-Prop algorithm could also be very useful and very popular. No-Prop is easy to implement and well worth trying.

References

[1] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” *Harvard University*, 1974.

[2] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Fondations*. MIT press, 1987.

[3] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, “The no-prop algorithm: A new learning algorithm for multilayer neural networks,” *Neural Networks*, vol. 37, pp. 182–188, Jan. 2013.

[4] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” in *IEEE International Joint Conference on Neural Networks*, 2004, pp. 985–990.

[5] ——, “Extreme learning machine: theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[6] ——, “Extreme learning machine for regresion and multiclass classfication,” *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 863–878, 2006.

[7] G. B. Huang, H. Zhu, X. Ding, and R. Zhang, “Extreme learning machine for regresion and multiclass classfication,” *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, vol. 42, no. 2, pp. 513–529, 2012.

[8] F. Rosenblatt, *Principles of neurodynamics: perceptrions and the theory of brain mechanism*. Spartan Books, 1962.

[9] H. D. Block, “The perceptron: A model for brain functioning. i,” *Reviews of Modern Physics*, vol. 43, no. 1, pp. 123–135, 1962.

[10] H. D. Block, B. W. J. Knight, and F. Rosenblatt, “Analysis of a four-layer series-coupled perceptron. ii,” *Reviews of Modern Physics*, vol. 43, no. 1, pp. 135–142, 1962.

[11] B. Widrow and M. E. Hoff Jr., “Adaptive switching circuits,” in *IRE WESCON Convention Record*, 1960, pp. 96–104.

[12] B. Widrow and S. D. Stearns, *Adaptive signal processing*. Prentice-Hall, 1985.

[13] B. Widrow and E. Walach, *Adaptive Inverse Control, Reissue Edition: A Signal Processing Approach*. Wiley-IEEE Press, 2008.

[14] T. M. Cover, “Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition,” *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 3, pp. 326–334, 1965.

[15] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience, 2000.

[16] M. Rudelson, “Norm of the inverse of a random matrix,” in *Foundations of Computer Science, 2006. FOCS ’06. 47th Annual IEEE Symposium on*, 2006, pp. 487–496.