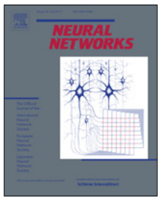
Neural Networks 37 (2013) 182–188

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



The *No-Prop* algorithm: A new learning algorithm for multilayer neural networks

Bernard Widrow[*], Aaron Greenblatt, Youngsik Kim, Dookun Park

ISL, Department of Electrical Engineering, Stanford University, CA, United States

ARTICLE INFO

Keywords:
Neural networks
Training algorithm
Backpropagation

ABSTRACT

A new learning algorithm for multilayer neural networks that we have named *No-Propagation (No-Prop)* is hereby introduced. With this algorithm, the weights of the hidden-layer neurons are set and fixed with random values. Only the weights of the output-layer neurons are trained, using steepest descent to minimize mean square error, with the *LMS* algorithm of Widrow and Hoff. The purpose of introducing nonlinearity with the hidden layers is examined from the point of view of Least Mean Square Error Capacity (LMS Capacity), which is defined as the maximum number of distinct patterns that can be trained into the network with zero error. This is shown to be equal to the number of weights of each of the output-layer neurons. The *No-Prop* algorithm and the *Back-Prop* algorithm are compared. Our experience with *No-Prop* is limited, but from the several examples presented here, it seems that the performance regarding training and generalization of both algorithms is essentially the same when the number of training patterns is less than or equal to LMS Capacity. When the number of training patterns exceeds Capacity, *Back-Prop* is generally the better performer. But equivalent performance can be obtained with *No-Prop* by increasing the network Capacity by increasing the number of neurons in the hidden layer that drives the output layer. The *No-Prop* algorithm is much simpler and easier to implement than *Back-Prop*. Also, it converges much faster. It is too early to definitively say where to use one or the other of these algorithms. This is still a work in progress.

1. Introduction

The most widely used method for training multi-layer perceptron networks is the *Back-Propagation* algorithm (*Back-Prop*), invented by Werbos (1974). This has proven to be a very robust and solidly working algorithm which has led to many successful applications for neural networks. This algorithm is based on the method of steepest descent, adjusting the weights of the network to minimize mean square error. When presented with an input pattern, the difference between the output response of the network and the desired response (which is presented during training) is the error which is to be minimized.

The mean square error is a function of the weight settings. With the presentation of each input vector during training, an instantaneous gradient is obtained of the mean square error with respect to each of the weights (the synaptic weights). Iteratively changing the weights in the direction of the negative gradient leads to an approximate minimum mean square error solution. The training patterns are presented repeatedly to bring the mean square error, averaged over the set of training patterns, down to minimum. The *Back-Propagation* algorithm is known to find optimal solutions to the weight settings that are in fact local optima.

A new algorithm is presented that seems to perform in many cases equivalently to *Back-Prop* but is much simpler and converges much faster. We call this algorithm *No-Propagation*, or *No-Prop*. Furthermore, we can now define a Least Mean Square Error Capacity (LMS Capacity) of a multi-layered neural network. *The LMS Capacity is defined as the number of patterns that can be trained into a neural network with zero mean square error.*

2. The *Back-Propagation* algorithm

The *Back-Propagation* algorithm as applied to multi-layer neural networks back propagates the errors of the output layer throughout the network to derive errors for all of the neurons in the hidden layers. During training, instantaneous gradients are obtained with these errors. The *Back-Prop* training algorithm is very well known, so it will not be described or derived here. For those who would like more details, please refer to Werbos's work (Werbos, 1974), Rumelhart and McClellan's books (Rumelhart & McClellan, 1986), and Simon Haykin's book (Haykin, 1999). A very simple derivation is given in the paper by Widrow and Lehr (1992).

Fig. 1 shows a fully-connected three layer feed-forward neural network. The network inputs are pattern vectors. If the inputs were derived from a black and white visual image, each individual input would have a value equal to the gray-scale contents of each pixel.

[*] Corresponding author.
E-mail addresses: widrow@stanford.edu (B. Widrow), aarong@stanford.edu (A. Greenblatt), youngsik@stanford.edu (Y. Kim), dkpark@stanford.edu (D. Park).

0893-6080/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2012.09.020

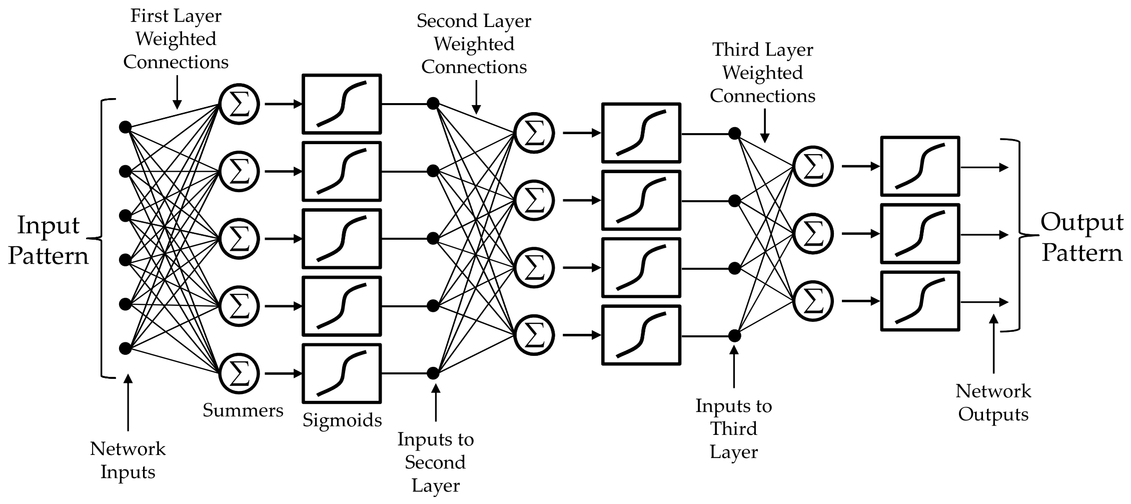


Fig. 1. A three-layer neural network.

Each individual input is a component of the input vector. The input vector is applied to the first layer neurons. The sigmoidal outputs of the first layer comprise an input vector for the second layer, and so forth.

To train the network of Fig. 1, it is common to establish random initial values for all the weights. Let the network be trained by *Back-Prop* with a given set of training patterns. For each input training pattern, there is a given desired response pattern. The desired response pattern is compared with the actual response pattern for the given input pattern, and the difference, the error pattern or error vector, is back-propagated throughout to compute the instantaneous gradient of the magnitude square of the error vector with respect to the weights. The weights of the network are then changed in proportion to the negative of the instantaneous gradient. The weight-change cycle is repeated with the presentation of the next input training pattern, and so forth.

Suppose that the network has already been trained with a given set of training patterns and the error vectors are now small. The weights are stabilized and are essentially unchanging. New input patterns may now be applied to the network. Referring to Fig. 1, the inputs to the third layer neurons are obtained from the new input patterns as they are mapped nonlinearly through the first two layers.

The components of the input pattern vectors are assumed to be gray scale, and therefore will have analog values. As such, an infinite number of distinct input patterns could exist and could be inputted to the neural network. Each distinct input pattern, by nonlinear mapping, would cause a distinct input pattern at the third layer, the output layer. This happens because the nonlinear elements of the network, the sigmoids, are monotonic and memoryless. (Pathological cases can occur where this will not be true, but they are rare and extremely unlikely.)

If the training patterns are distinct, the corresponding input patterns to the third layer will also be distinct. If the number of training patterns is less than or equal to the number of weights of each of the individual output layer neurons, then the input vectors to the third layer will not only be distinct patterns but they will also be linearly independent, even if the training patterns may or may not be linearly independent. With this linear independence, the output layer neurons will be capable of delivering the desired gray-scale response patterns perfectly and the network can thus be perfectly trained with zero error. Linear independence created by propagation through a nonlinear layer or layers is assumed and the rationale for this will be discussed in more detail below.

Linear independence with the input patterns to the output layer, the third layer for the example of Fig. 1, will result when the number of distinct training patterns is less than or equal to the number of weights of the output neurons. Each of the output neurons is assumed to have the same number of weights as all the others. The LMS Capacity of the network will be equal to the number of weights of each of the individual output neurons. If the number of training patterns is under Capacity or at Capacity, weight values for the output neurons will exist that will allow the output neurons to deliver the desired output patterns perfectly, without error, i.e. with zero mean square error. If the number of distinct training patterns is greater than Capacity, then the input patterns to the output layer cannot be linearly independent and it will not be possible in general to train the network without error. The network will be able to be trained, but there will be some residual mean square error in this case. When there is residual error, "underfitting" takes place. When there is no residual error, "overfitting" takes place (except when training exactly at Capacity).

3. The *No-Prop* algorithm

Referring again to the three layer network of Fig. 1, let the number of distinct training patterns be less than or equal to Capacity. The weights of the first two layers could be randomized and fixed, not set by *Back-Prop* training. The training patterns will undergo a fixed random mapping, and the inputs to the third layer neurons will be distinct and linearly independent. By training only the weights of the third layer neurons, all of the corresponding desired response patterns will be able to be perfectly realized at the network output. The weights being trained are analogous to the "unknowns" of linear simultaneous equations. The number of equations is equal to the number of training patterns. Training at Capacity is analogous to the number of equations being equal to the number of unknowns.

So, without back-propagating the output errors throughout the network and by adapting only the output layer after randomizing and fixing the weights of the first two layers, the "hidden layers", we have the "*No-Prop*" algorithm. It is a very much simpler algorithm than *Back-Prop* and does not require the training of all the weights of the network, only the output layer weights. This algorithm will provide network performance under many conditions equivalent to that of *Back-Prop*.

4. Linear independence

It was related above that pattern vectors at the input to the weights of the output layer neurons will be linearly independent as long as the number of input patterns is less than or equal to the

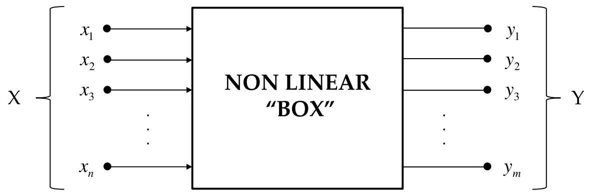


Fig. 2. A nonlinear transformation.

number of weights of each of the output neurons, and as long as the network input patterns propagate through at least one or more nonlinear layers before reaching the weights of the output layer. Linear independence is key to the idea of Capacity.

An explanation for linear independence follows. In Fig. 2, a nonlinear transformation or nonlinear box is shown. A set of X vectors are its inputs, and a corresponding set of Y vectors are its outputs. 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.

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

$$X_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} \qquad X_2 = \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} \qquad X_3 = \begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} \tag{1}$$

$$Y_1 = F\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} \qquad Y_2 = F\begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} \qquad Y_3 = F\begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} \tag{2}$$

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

$$F\begin{bmatrix} x_{13} \\ x_{23} \\ \vdots \\ x_{n3} \end{bmatrix} = \alpha F\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} + \beta F\begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} ? \tag{3}$$

Would a set of constants α and β exist for the above equation to be correct? The answer is generally no, and then 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 with a given set of X input vectors. If Eq. (3) holds, the three Y vectors would be linearly dependent. This is highly unlikely, and linear independence is almost perfectly assured.

It is impossible to prove linear independence, since linear dependence could actually happen, but this would be extremely rare. A set of simulation experiments have been performed to demonstrate this rarity.

5. Linear independence experiments

By computer simulation, an extensive set of experiments was performed to verify or deny the above argument about linear independence, this could also 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 Fig. 2. The output patterns were tested for linear independence by means of the MATLAB function called "*rank*". If the rank of the set of output patterns equalled the number of output patterns, the output patterns were linearly independent. Otherwise, the output patterns were linearly dependent. The nonlinear box of Fig. 2 was a neural network with randomly chosen fixed weights. For these experiments, the size of the network and the number of layers was varied.

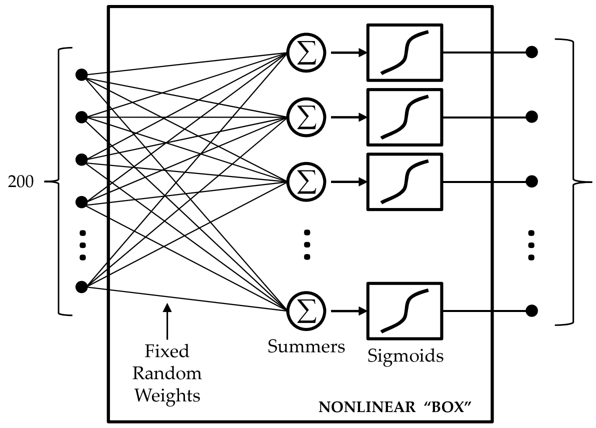


Fig. 3. Network with a single fixed layer used for some of the linear independence experiments. Bias weights, not shown, were used.

For all of the experiments, input patterns were 200 component 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, could the output patterns from the nonlinear box be linearly independent?

The first experiments were done with the nonlinear box being the network shown in Fig. 3. The input patterns had 200 components. The first layer of 400 neurons provided 400 outputs. The weights of the neurons in this layer were randomly chosen and fixed. 400 input patterns were generated as described above. The corresponding output patterns were tested for linear independence. There were 400 output pattern vectors, each having 400 components. The rank of the set of output patterns of the fixed layer was determined to be equal to 400, indicating that the 400 output vectors were linearly independent. This experiment was repeated 1000 times, each time with a different set of input patterns generated as above. In all thousand cases, the output pattern vectors were linearly independent.

A second set of experiments was performed with networks having input pattern vectors with 200 components, and having one fixed layer like in Fig. 3, but alternatively having 100 outputs, and 200 outputs. With 100 outputs, 100 input patterns were applied to the network. With 200 outputs, 200 input patterns were applied to the networks. The input patterns were generated as above. The output patterns of the fixed layer were found to be linearly independent. These experiments were repeated 1000 times. In every single case, the output pattern vectors were linearly independent.

A third set of experiments was performed with networks having two fixed 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 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

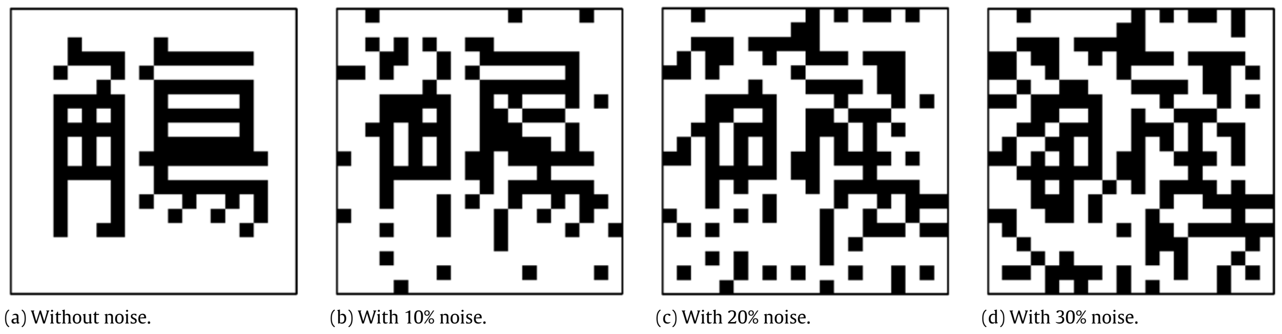


Fig. 4. An example of a Chinese character with its noisy versions.

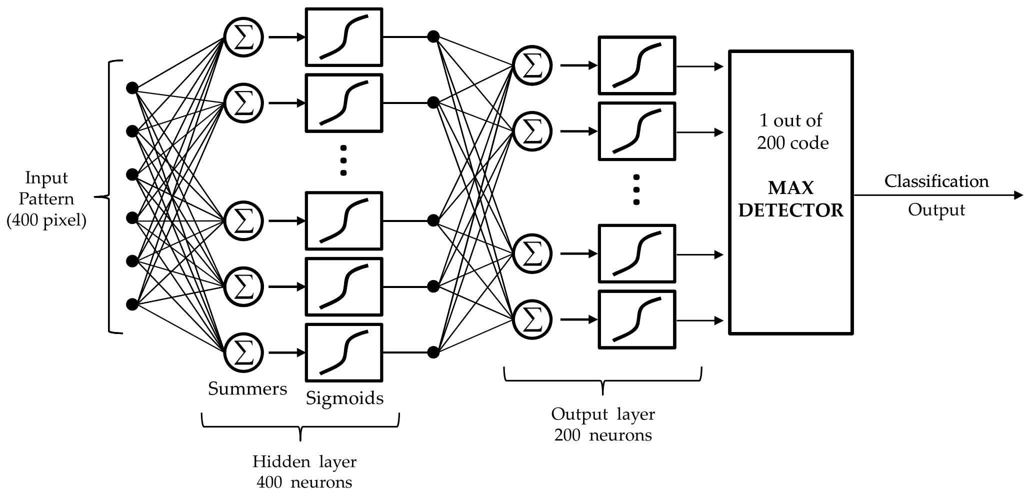


Fig. 5. A trainable neural-net classifier.

chosen in each case was equal to the number of neurons in the third 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 15 000. In every case, the output vectors were linearly independent. Not a single case of linear dependence was observed. Linear dependence seems to be highly unlikely and we are no longer concerned with this issue. This provides critical support to the idea that *the Capacity of a layered neural network depends only on the number of weights of each of the neurons of the output layer, regardless of network configuration and regardless of the nature of the training algorithm.*

6. Training classifiers with *Back-Prop* and *No-Prop*

Both the *Back-Prop* and *No-Prop* algorithms are very useful for training neural networks. One might expect equivalent performance for these two algorithms when the number of training patterns is less than or equal to Capacity. When training with a number of patterns greater than Capacity, it would seem that *Back-Prop* would deliver superior performance. In any event, neither of these algorithms can train over Capacity with zero mean square error. *Back-Prop* might provide a lower mean square error however.

Classification experiments were performed to compare the behavior of the two algorithms. A set of 20,000 Chinese characters were available and many of these were used as training and testing patterns. Each character had 20×20 pixels. Each pixel was either black or white, represented by $+1$ or -1 respectively. Random noise was introduced to these patterns by changing black to white or white to black, randomly over the pixels. The noise was introduced first to 5% of the pixels. The random changes were then maintained, and additional random changes were introduced to the remaining pixels to comprise 10% noise. All these changes were maintained, and further changes were introduced for a total of 15% noise, and so forth. In this way, sets of Chinese characters with 5%, 10%, 15%, 20%, 25%, and 30% noise were generated. An example of one of the Chinese characters, pure and without noise, can be seen in Fig. 4(a). This character with 10% noise is shown in Fig. 4(b), with 20% noise in Fig. 4(c), and with 30% noise in Fig. 4(d). It is difficult to determine what the character is from inspection of any of the noisy versions, as verified by Chinese readers. Yet the neural network was able to classify them, as will be noted below.

The neural network that was used for comparing *Back-Prop* and *No-Prop* in a classifier application is shown in Fig. 5. It is a two-layer network having one hidden layer and one output layer. With *Back-Prop*, both layers were trained. With *No-Prop*, the fixed layer, the hidden layer, had its weights set randomly and fixed, while the second layer, the output layer, was trained. The input patterns were 20×20, 400 pixels, so the inputs to the neural network were pattern vectors having 400 components. The hidden layer had 400 neurons, each with 400 weights, and the output layer had 200 neurons, each with 400 weights. The Capacity of the network was 400 and the number of outputs was 200.

Two hundred Chinese characters were chosen. Each of the output neurons was trained to produce a $+1$ output when a correspondingly selected Chinese character free of noise was presented at the network input, and a -1 output for all other inputted characters. Training was stopped after 10,000 iterations. After training, noisy versions of the 200 characters never seen before were presented, and these inputs were identified with the output layer neuron which produced the most positive analog output. The network output is connected to a maximum detector for a 1 out of 200 code. In most cases, the number of classification errors was small.

Table 1
Training with 200 noise-free characters; Testing with 100 noisy versions of each character.

% noise in testing patterns		5%	10%	15%	20%	25%	30%
Classification errors	*Back-Prop* trained	1	8	24	114	464	2225
	No-Prop trained	0	28	110	440	2219	7012

Table 2
Training with 10% noise, 50 noisy versions each of the 200 noise-free characters; Testing with 100 noisy versions of the 200 noise-free characters.

% noise in testing patterns		5%	10%	15%	20%	25%	30%
Classification errors	*Back-Prop* trained	0	19	63	166	438	1728
	No-Prop trained	2	26	92	213	697	2738

Table 3
Training with 20% noise, 50 noisy versions each of the 200 noise-free characters; Testing with 100 noisy versions of the 200 noise-free characters.

% noise in testing patterns		5%	10%	15%	20%	25%	30%
Classification errors	*Back-Prop* trained	6	34	107	274	550	1555
	No-Prop trained	6	36	126	274	584	1885

For each experiment, the total number of training patterns was 10,000, consisting of 50 noisy versions of the 200 characters. For the first experiment, the network was trained only with the 200 noise-free characters. For the second experiment, the network was trained with 10,000 noisy characters, each with 5% noise. The third experiment with 10% noise, the fourth experiment with 15% noise, the fifth experiment with 20% noise, the sixth experiment with 25% noise, and the seventh experiment with 30% noise. Each of these experiments was done alternatively with *Back-Prop* and with *No-Prop*. Results of these experiments are the following.

With the first experiment, when the network was trained with the 200 noise-free characters, the errors at the outputs of all 200 output layer neurons were essentially zero for all 200 training patterns. After training, testing the network with the 200 noise-free characters resulted in perfect classification. There were no discrimination errors. These results were the same for *No-Prop* and *Back-Prop*. This was no surprise since the number of training patterns was less than Capacity. *No-Prop* required considerably less computation.

After training the network with the 200 noise free characters by *Back-Prop* and then repeating this by *No-Prop*, the *Back-Prop* and *No-Prop* trained networks were tested with 100 noisy versions of each of the 200 characters. There were 100 × 200 or 20,000 noisy test patterns for each level of noise. Noise levels of the test patterns were 5%, 10%, 15%, 20%, 25%, 30%. The results of this experiment are given in Table 1. Both methods yielded quite low numbers of classification errors with test patterns having noise up to 20%. With 20% test pattern noise, the *Back-Prop* trained network had 114 classification errors with 20,000 test patterns of 0.6% error. The *No-Prop* trained network had 440 errors or 2.2% error. Of course, the rate of classification errors increased with the increase in test pattern noise beyond 20%, and the *Back-Prop* trained network performed better than the *No-Prop* trained network.

Table 2 shows results with another experiment, only here the network was trained with noisy characters having 10% noise. The sets of training and testing patterns were independently generated. The training patterns were 50 noisy versions of each of the 200 noise-free characters, 50 × 200 or 10,000 training patterns. Now the number of training patterns is far greater than Capacity, but with only 10% noise, they were close enough to the original 200 patterns which, if trained on, would be of a number less than Capacity. The *Back-Prop* trained network performed better than *No-Prop* trained network, but the noise in the training patterns was quite beneficial for the *No-Prop* network. In fact, the performance of both networks had become similar.

Table 3 shows results with another repeat experiment. Here the network was trained with noisy characters having 20% noise. The training and testing patterns were independently generated. There were 10,000 training patterns, far greater than Capacity, but still close enough to the original 200 patterns to cause the training to behave somewhat like training under Capacity. The additional noise in the training patterns further improved the performance of the *No-Prop* trained network which is now very close to that of the *Back-Prop* trained network. Higher noise in the training patterns causes lower error rate for the noisier test patterns, with noises in the 25% and 30% range, especially for the *No-Prop* trained network. With 25% noise in the test patterns, the *No-Prop* network made 2.92% output error. The *Back-Prop* network made 2.75% output error. This is good performance by both algorithms, and there is not much difference between them. Training with noisy patterns helps with subsequent classification of noisy patterns.

7. Training autoassociative networks with *Back-Prop* and *No-Prop*

There are many applications for autoassociative neural networks. An application of great interest to the authors is to data retrieval in "cognitive memory". The subject of cognitive memory involves the study of human memory and how it works, with the goal of designing a human-like memory for computers. Applications for such memory are to the field of pattern recognition, learning control systems, and to other problems in artificial intelligence. Cognitive memory will not be described here. Papers reporting early work on the subject are reproduced on the website, http://www-isl.stanford.edu/widrow/publications.html, which can also be reached by googling 'Bernard Widrow's Stanford University home page'.

Autoassociative networks are multilayered neural networks like the one in Fig. 1. They can be trained either with *Back-Prop* or *No-Prop*. (If used for principal component analysis, all layers must be trained with *Back-Prop*.) What makes them autoassociative is how they are trained. During training, patterns presented to the network are both input patterns and at the same time are the desired response patterns. Thus, autoassociative networks are trained to produce output patterns that match the input training patterns. When inputting patterns that were not part of the training set, the outputs will not match the inputs. Thus, an autoassociative network will register a "hit" (very low difference between an input pattern and its output pattern) or "no hit" (big difference between an input pattern and its output pattern) depending on the input pattern, if it was in the training set or not. This type of network plays a major role in data retrieval in human-like cognitive memory systems according to the above Widrow et al. references.

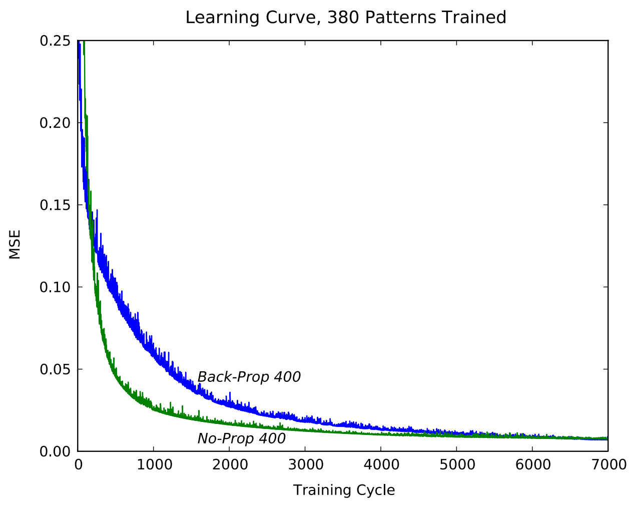


Fig. 6. Training a network under its capacity.

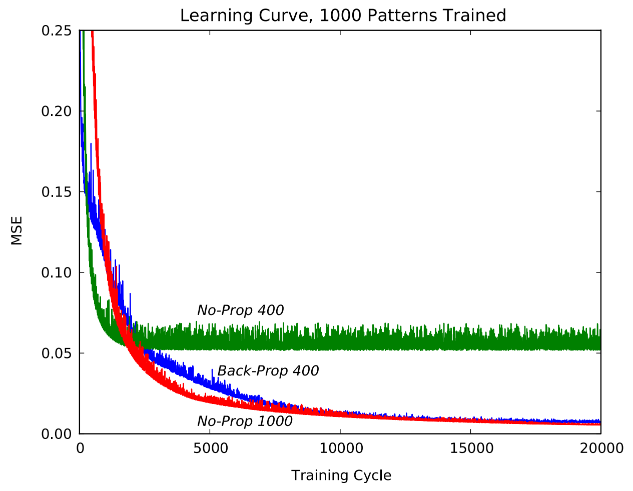


Fig. 7. Training a network over its Capacity ('*Back-Prop* 400' and '*No-Prop* 400'), and training the network under its Capacity ('*No-Prop* 1000').

Experiments have been performed with autoassociative networks, trained with *Back-Prop* and with *No-Prop*. The networks were trained for minimization of mean square error. The difference between the input pattern vector and the output pattern vector is the error vector. The sum of the squares of the error vector components is computed for each training pattern and the networks were trained to minimize the average of this quantity over the set of training patterns. This average is the MSE.

The same set of 20,000 Chinese characters was used for the autoassociative experiments, for training and testing. Fig. 1 shows the network configuration that was used for these experiments. There were 400 inputs in correspondence with the 400 pixels of the characters. There were 400 outputs, corresponding to the 400 inputs. The first layer, a hidden layer, had 400 neurons. The second layer, another hidden layer, had 400 neurons. The output layer had 400 neurons. The Capacity of the network was 400. The weights of the hidden layers were trained with *Back-Prop*, and the weights of the hidden layers were randomly fixed when using *No-Prop*.

With 380 Chinese characters chosen for training from the available set of 20,000, learning curves were obtained for *Back-Prop* and *No-Prop*, and they are shown in Fig. 6. Additional experiments were done with 1000 Chinese characters chosen for training from the set of 20,000, and leaning curves were obtained for *Back-Prop* and *No-Prop* and they are shown in Fig. 7.

When training with 380 patterns, the number of training patterns was less than the network Capacity of 400. These patterns could be trained perfectly, with the mean square error going to zero. The asymptotic MSE level for the curves of Fig. 6 is theoretically zero. One curve is labeled '*Back-Prop* 400'. This represents learning with *Back-Prop* with a network Capacity of 400. The second curve is labeled '*No-Prop* 400'. This represents learning with *No-Prop* with a network having a Capacity of 400. Both curves are very similar, with similar learning rates and similar asymptotic levels.

The *Back-Prop* learning process was terminated after 7000 training cycles, and the network was tested with the 20,000 Chinese characters. All 380 training patterns made hits, with low errors. The remainder of the 20,000 patterns did not make hits, having high errors. The *Back-Prop* network performed perfectly.

The *No-Prop* learning process was also terminated after 7000 training cycles, and the network was tested with the 20,000 Chinese characters. All 380 training patterns made hits, with low errors. The remainder of the 20,000 patterns did not make hits, having high errors. The *No-Prop* network performed perfectly.

When training the network under Capacity, with a number of patterns less than Capacity, both *Back-Prop* and *No-Prop* performed equivalently, regarding training and generalization which, in this case, was making hits with the training patterns and not making hits with patterns that were not trained in.

Fig. 7 shows learning curves for the network trained with 1000 patterns. Curves are shown for '*Back-Prop* 400', training the network with a Capacity of 400, and '*No-Prop* 400', training the network with a Capacity of 400. In both cases, the network is trained over its Capacity. The performance of *Back-Prop* is clearly superior to that of *No-Prop*. The asymptotic MSE level of *Back-Prop* is much lower that that of *No-Prop*. Both learning processes were terminated at 20,000 cycles. The two trained networks were tested with all 20,000 Chinese characters. Both networks made hits with the 1000 training patterns. The *Back-Prop* trained network perfectly rejected the rest of the 20,000 characters, but the *No-Prop* trained network yielded 434 hits with the rest of the 20,000 characters, with a false hit rate of 2.2%. For training and generalization, *Back-Prop* is the superior algorithm in this over Capacity case.

By increasing the Capacity of the network, it became possible for a *No-Prop*-trained network to perform as well as the *Back-Prop* trained network. The third curve labeled '*No-Prop* 1000' in Fig. 7 is a learning curve for the network, now with a Capacity of 1000, trained with the *No-Prop* algorithm. It performed perfectly with the same set of input patterns. The increased Capacity was obtained by raising the number of neurons in the second hidden layer from 400 to 1000. The number of neurons in the first and third layers remained the same. The *No-Prop*-trained network made hits with the 1000 training patterns and rejected the rest of the 20,000.

From these experiments with autoassociative training, it appears that when *Back-Prop* training yields performance superior to *No-Prop* training, *No-Prop* can be made to catch up with *Back-Prop* by increasing the network Capacity. So there is a trade-off here. One could use a more complicated training algorithm, *Back-Prop*, or one could use a simpler training algorithm, *No-Prop*, with more neurons in the network.

Back-Prop is a fine algorithm, although it may converge on a local optimum. The *No-Prop* algorithm always converges on a global optimum. With *No-Prop*, the mean square error is a quadratic function of all the weights of the output-layer neurons. This type of function has a unique optimum. The reader is referred to the Widrow–Stearns book (Widrow & Stearns, 1985), the Widrow–Walach book (Widrow & Walach, 1995), and the Haykin book (Haykin, 1999).

When using a trained autoassociative network, a hit or no hit is determined by the magnitude of the error, the difference between the input patterns vector and the output pattern vector. There is a hit when the magnitude of the error vector is below a threshold. The question is, how is the threshold determined? For this work, the threshold was set at the end of the training process to be the highest error magnitude among all of the training patterns.

For all the networks described in this paper that were trained with *No-Prop*, the output-layer neurons were designed to produce linear outputs, with the sigmoids omitted. This simplified the network configuration and allowed the output neurons to be trained with the simplest of all training algorithms, the *LMS* algorithm of Widrow and Hoff (1960). Omitting the sigmoids in the output layer did not significantly affect performance, but caused the MSE to be a quadratic function of the weights.

8. Conclusions

The *No-Prop* algorithm trains multilayer neural networks by training the output layer only. This can be done using the *LMS* algorithm of Widrow and Hoff. *No-Prop* has the advantage of not requiring error backpropagation throughout the network, making it simpler to build in hardware or to code in software.

When the number of training patterns in less than the network Capacity (equal to the number of weights of each of the output layer neurons), the *No-Prop* network and the *Back-Prop* network perform equivalently. When training over Capacity, *Back-Prop* performs generally better than *No-Prop*. But by increasing the number of neurons in the layer before the output layer, i.e. by increasing the network Capacity, the performance of *No-Prop* can be brought up to that of the *Back-Prop* algorithm.

Perhaps this work will lend some insight into the training of neural networks in animal brains. It may not be necessary to train all layers, only the output layer. That would make mother nature's job much simpler.

Acknowledgment

This research was partially supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program.

References

Haykin, S. (1999). *Neural networks: a comprehensive foundation*. New Jersey: Prentice Hall.
Rumelhart, D. E., & McClellan, J. L. (Eds.) (1986). *Parallel distributed processing*. Cambridge, MA: MIT press.
Werbos, P. J. (1974). Beyond regression: new tools for prediction and analysis in the behavioral sciences. *Ph.D. Thesis* Harvard University, Cambridge, MA.
Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record*.
Widrow, B., & Lehr, M. A. (1992). Backpropagation and its applications. In *Proceedings of the INNS summer workshop on neural network computing for the electric power industry*. Stanford, 21–29, August.
Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing*. New Jersey: Prentice Hall.
Widrow, B., & Walach, E. (1995). *Adaptive inverse control*. New Jersey: Prentice Hall.

Further reading

Rosenblatt, F. (1958). The perceptron: a probablistic model for information storage and organization in the brain. *Psychological Review*, *65*(6).
Widrow, B., & Lehr, M. A. (1990). 30 Years of adaptive neural networks: perceptron, Madaline, and backpropagation. *Proceedings of the IEEE*, *78*(9), 1415–1442.