ORNL/CP-102700

Single-Iteration Learning Algorithm for Feed-Forward Neural Networks*

[†]V. Protopopescu, [†]Jacob Barhen, and [‡]R. Cogswell

[†]Center for Engineering Science Advanced Research
Computer Science and Mathematics Division
Oak Ridge National Laboratory
P. O. Box 2008-6355
Oak Ridge, TN 37831

[‡]Department of Mathematics and Computer Science
Monmouth College
Monmouth, IL 61462

Submitted to: Proceedings of the Fifth Conference of Information Systems Analysis and Synthesis (ISAS'99), and the Third Conference of Systemics, Cybernetics, and Informatics, (SCI'99), Orlando, FL, July 31 – August 4, 1999.

* This research was funded in part by the *DeepLook* Consortium under Agreement Number ERD-97-1506 with Lockheed Martin Energy Research Corporation. Additional funding was provided by the Engineering Research Program of the Office of Basic Energy Sciences under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation. R. Cogswell's participation was supported by the Great Lakes Colleges Association/Associated Colleges of the Midwest Oak Ridge Science Semester, sponsored by the Office of University Science Education, Oak Ridge National Laboratory.

DISCLAIMER

Single-Iteration Learning Algorithm for Feed-forward Neural Networks

V. Protopopescu, J. Barhen, and R. Cogswell*
Center for Engineering Science Advanced Research,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6364, USA

Abstract. A new methodology for neural learning is presented, whereby only a single iteration is required to train a feed-forward network with near-optimal results. To this aim, a virtual input layer is added to the multi-layer architecture. The virtual input layer is connected to the nominal input layer by a special nonlinear transfer function, and to the first hidden layer by regular (linear) synapses. A sequence of alternating direction singular value decompositions is then used to determine precisely the inter-layer synaptic weights. This algorithm exploits the known separability of the linear (inter-layer propagation) and nonlinear (neuron activation) aspects of information transfer within a neural network.

1. Introduction. Artificial neural networks are adaptive systems that process information by nonlinear response to discrete or continuous input [1]. Neural networks can provide practical solutions to a variety of artificial intelligence problems, including pattern recognition [2], autonomous knowledge acquisition from observations of correlated activities [3], real-time control of complex systems [4], and fast adaptive optimization [5]. At the heart of such advances lies the development of efficient computational methodologies for "learning" [6]. The development of neural learning algorithms has generally been based upon the minimization of an energy-like error function [7]. Gradient-based techniques have typically provided the main computational mechanism for carrying out the minimization process, often resulting in excessive training times for the large-scale networks needed to address real-life applications. Consequently, to date, considerable efforts have been devoted to: (1) speeding up the rate of convergence [7,8] and (2) designing more efficient methodologies for deriving the gradients of these functions or functionals with respect to the parameters of the network [9,10].

In a major departure from the above paradigms, Biegler-König and Bärman proposed a learning approach solely based on linear algebraic methods [11]. In their seminal paper, they observed that it is possible to separate the linear (inter-layer propagation) and nonlinear (individual neuron activation function) aspects of information processing within a neural network. Using linear least squares, they obtained the synaptic weights for each layer, and invoked the inverse of the activation function to propagate any remaining error back into the preceding layer of the network. The essence of their approach was to minimize the learning error on each layer separately, rather than globally for the entire network. An accelerated training algorithm based on a similar scheme was more recently proposed by Tam and Chow [12].

The methodology presented in this letter builds upon these seminal ideas, but introduces significant innovations. For instance, the algorithms in [11,12] solve each of a sequence of least square problems by using Householder transformations followed by a QR decomposition [13]. Since for most practical applications the number of training examples exceeds the dimensionality of the training patterns, approximate results are typically obtained. Our proposed algorithm, on the other hand, implements a sequence of alternating direction singular value decompositions

* On leave of absence from Monmouth College, Department of Mathematics and Computer Science, Monmouth, IL 61462, USA.

(SVD) [13], on a network architecture having a monotonically decreasing number of nodes in successive layers. We also modify the traditional neural network architecture by introducing a *virtual input* layer connected to the input nodes through a nonlinear transfer function. The virtual layer acts as a preprocessor of the input vectors, and replaces a highly overdetermined linear system with a solvable one. Our algorithm has several new, important features, namely: (i) the network is trained in a single iteration; (ii) the method does not rely on calculating gradients, therefore it is not affected by local minima; (iii) the algorithm achieves near-optimal training results, and (iv) it generalizes well to vectors outside the training set.

2. Architecture. We consider a generalized multi-layer feed-forward neural network. In general, the nodes (or neurons) are organized in layers, namely: (i) *input*, (ii) (one or several) *hidden*, and (iii) *output*. In addition to these traditional layers, we introduce a *virtual input* layer between the input layer and the (first) hidden layer. The virtual input layer allows for a specific type of preprocessing of the input vectors, and the associated computations between the actual and virtual input layers differ from the usual inter-layer neural network computations.

The feed-forward neural network has (L+1) layers, numbered from zero (input layer) to L (output layer). An input vector is propagated forward through the neural network as a sequence of intermediate vectors at each successive layer. For the implementation of the proposed method, the number of nodes in successive layers must satisfy the inequalities $N_\ell \geq N_{\ell+1}$, $\ell = 1, \ldots, L$. The N_ℓ nodes at the layer ℓ receive (local) input values and produce N_ℓ output values (*i.e.* a vector of dimension N_ℓ) which are propagated to the next layer of nodes, or (for the output layer) represent the calculated output vector corresponding to the initial inputs. For the hidden and output layers, the processing can be described as follows. Assume that an initial input vector is propagated through the network. Let $\mathbf{s}$ denote the row vector of outputs of the nodes at the layer ℓ, and let $\mathbf{t}$ denote the row vector of inputs to the nodes at the layer (ℓ+1). When bias weights are used, the vector $\mathbf{t}$ is calculated as

$$\mathbf{t} = \mathbf{s}\,\mathbf{W} + b\,\mathbf{v}, \tag{1}$$

where $\mathbf{W}$ is the matrix of interconnection weights, b is a constant between 0 and 1 (*e.g.* b=0.5), and $\mathbf{v}$ is a row vector of bias weights. Eq. 1 can be represented in an equivalent way by appending b to $\mathbf{s}$ to form $\hat{\mathbf{s}}$, adding the row vector $\mathbf{v}$ to $\mathbf{W}$ as the last row of the augmented matrix $\hat{\mathbf{W}}$, and calculating $\mathbf{t} = \hat{\mathbf{s}}\hat{\mathbf{W}}$. For the training problem, we consider the K training input vectors as rows of a matrix $\mathbf{S}_{KN_0}$, where N_0 is the number of (scalar) input values. At the layer ℓ, the output vectors comprise the rows of a matrix $\mathbf{S}_{KN_\ell}$, where the column dimension N_ℓ is the number of nodes of layer ℓ, and the ith row is the output at layer ℓ corresponding to the ith training input vector. Similarly, the input vectors for layer (ℓ+1) form rows of a matrix $\mathbf{T}_{KN_{\ell+1}}$, where $N_{\ell+1}$ denotes the number of nodes in layer (ℓ+1). The propagation of information between layers is given by the linear matrix equation

$$\mathbf{T}_{KN_{\ell+1}} = \mathbf{S}_{KN_\ell}\,\mathbf{W}_{N_\ell N_{\ell+1}} + \mathbf{b}_K\;\mathbf{v}_{N_{\ell+1}} \tag{2}$$

where $\mathbf{W}_{N_\ell N_{\ell+1}}$ is the matrix of interlayer connection weights from layer ℓ to layer (ℓ+1), $\mathbf{b}_K$ is a column vector of constants b_i, and $\mathbf{v}_{N_{\ell+1}}$ is a row vector of $N_{\ell+1}$ bias weights. If we form $\hat{\mathbf{S}}_{KN_\ell}$ from $\mathbf{S}_{KN_\ell}$ by appending $\mathbf{b}_K$, and append $\mathbf{v}_{N_{\ell+1}}$ to $\mathbf{W}_{N_\ell N_{\ell+1}}$ to form $\hat{\mathbf{W}}_{N_\ell N_{\ell+1}}$, we can rewrite (2) as

$$\mathbf{T}_{KN_{\ell+1}} = \hat{\mathbf{S}}_{KN_\ell}\, \hat{\mathbf{W}}_{N_\ell N_{\ell+1}}. \tag{3}$$

As a single initial input vector is propagated forward, each node in the (hidden or output) layer ℓ uses a transfer function φ that maps the presynaptic vector $\mathbf{t}$ at layer ℓ to a new postsynaptic output vector $\mathbf{s}$ (also at layer ℓ) one component at a time. We assume φ is the sigmoid function with gain γ so that the ith component s_i of $\mathbf{s}$ is calculated using

$$s_i = \varphi(t_i) = \frac{1}{1+\exp(-\gamma t_i)} \tag{4}$$

For the training problem, the output matrix $\mathbf{S}_{KN_{\ell+1}}$ for the layer (ℓ+1) is

$$\mathbf{S}_{KN_{\ell+1}} = \varphi(\mathbf{T}_{KN_{\ell+1}}), \tag{5}$$

where the transfer function φ is applied to each of the $K \times N_{\ell+1}$ matrix entries individually. The final output patterns, which are vectors of dimension N_{L+1}, form the rows of the output matrix $\mathbf{S}_{KN_{L+1}}$.

The processing between the input and virtual layers is realized in a different way. For a given set of training vectors, we assume there exists a particular nonlinear transfer function, ψ, that maps input vectors to virtual input vectors by directly applying the function ψ rather than the separate calculation of the vector $\mathbf{t}$ and the (component-wise) nonlinear transfer function at each node:

$$\mathbf{s}_{virtual} = \psi(\mathbf{s}_{input}). \tag{6}$$

In the training algorithm, the function ψ (which is not altered in the learning algorithm) maps the input pattern matrix $\mathbf{S}_{KN_0}$ (treated as the postsynaptic output of the input-layer nodes) to the postsynaptic output matrix $\mathbf{S}_{KN_1}$ of the virtual layer:

$$\mathbf{S}_{KN_1} = \psi(\mathbf{S}_{KN_0}). \tag{7}$$

3. Single-iteration neural learning algorithm. The goal in the learning process is to determine the weight parameters $\{\mathbf{W}_{N_\ell N_{\ell+1}}\}$ that minimize (some norm of) the difference between the calculated output values and the target outputs. Our new approach is to decouple the nonlinearities of the transfer functions at each layer from the linear interlayer pattern propagation. To simplify the description, and without loss of generality, we discuss a network

architecture having $N_0 = I$ input nodes, $N_1 = V$ virtual input nodes, one hidden layer with $N_2 = H$ nodes, and $N_3 = O$ output nodes. We assume that the interconnection weight matrices $\mathbf{W}_{VH}$ and $\mathbf{W}_{HO}$ and bias weight vector $\mathbf{v}_O$ are randomly initialized. The learning algorithm begins with the output layer. A full iteration consists of a backward pattern propagation to the virtual layer, followed by a forward sweep to the output layer of the neural network. (In the following, a bar indicates a calculated quantity, as opposed to a "target" or initialized quantity, and a superscript f denotes a forward calculation.) We note that since $\varphi:\Re\rightarrow(0,1)$ is bijective, the inverse φ^{-1} is well-defined. The following two equations relate the presynaptic inputs at the output layer:

$$\mathbf{T}_{KO} = \varphi^{-1}(\mathbf{S}_{KO}); \tag{8}$$

$$\overline{\mathbf{T}}_{KO} = \overline{\mathbf{S}}_{KH}\,\mathbf{W}_{HO} + \mathbf{b}_K\ \mathbf{v}_O. \tag{9}$$

The first phase of the learning algorithm is implemented to minimize $\left\|\mathbf{T}_{KO} - \overline{\mathbf{T}}_{KO}\right\|$, by solving the system

$$(\mathbf{T}_{KO} - \mathbf{b}_K\ \mathbf{v}_O) = \overline{\mathbf{S}}_{KH}\,\mathbf{W}_{HO} \tag{10}$$

for $\overline{\mathbf{S}}_{KH}$. Since $\mathbf{T}_{KO}$ is known and $\mathbf{W}_{HO}$ and $\mathbf{v}_O$ were initialized, we compute $\overline{\mathbf{S}}_{KH}$ using the SVD of $\mathbf{W}_{HO}$ from the right. When $\mathbf{W}_{HO}$ is a nonsingular square matrix, this step reduces to inverting $\mathbf{W}_{HO}$; however even in that case the SVD may be less expensive, especially for large matrices.

Typically we need to modify the calculated solution $\overline{\mathbf{S}}_{KH}$, since in the forward calculations the corresponding entries of $\mathbf{S}_{KH}$ are output values of the sigmoid function φ, with range (0,1). Some of the calculated values of $\overline{\mathbf{S}}_{KH}$ may be outside the range of φ. As described below, we shall find an invertible renormalization matrix Γ with the property that

$$[\tilde{\mathbf{S}}_{KH} \mid \mathbf{b}_K] = [\overline{\mathbf{S}}_{KH} \mid \mathbf{b}_K]\ \Gamma. \tag{11}$$

Then, $\tilde{\mathbf{S}}_{KH}$ has values in the range of φ, while preserving the final column $\mathbf{b}_K$. Here, $[\tilde{\mathbf{S}}_{KH} \mid \mathbf{b}_K]$ denotes a matrix with K rows and $H + 1$ columns. We use $\tilde{\mathbf{S}}_{KH}$ to determine $\tilde{\mathbf{T}}_{KH} = \varphi^{-1}(\tilde{\mathbf{S}}_{KH})$. If more than one hidden layer is used, one minimizes $\left\|\tilde{\mathbf{T}}_{KN_\ell} - \overline{\mathbf{T}}_{KN_\ell}\right\|$ as above for each preceding hidden layer ℓ, until reaching the first hidden layer (counting from the input end).

When the presynaptic input matrix $\tilde{\mathbf{T}}_{KH}$ for the (first) hidden layer has been calculated, we alter the process and begin the forward sweep, in which the weights are updated. We observe that $\tilde{\mathbf{T}}_{KH}$ and the postsynaptic output $\overline{\mathbf{S}}_{KV}$ of the virtual input layer are related by

$$\tilde{\mathbf{T}}_{KH} = \overline{\mathbf{S}}_{KV}\,\mathbf{W}_{VH}, \tag{12}$$

where $\tilde{\mathbf{T}}_{KH}$ is known from the backward sweep and $\overline{\mathbf{S}}_{KV}$ is calculated from $\mathbf{S}_{KI}$. Thus we do not need to compute $\tilde{\mathbf{S}}_{KV}$. Rather, we carry out an SVD of $\overline{\mathbf{S}}_{KV}$ from the left to determine the interconnection weight matrix $\overline{\mathbf{W}}^f_{VH}$. Using Eq. (12), we derive a forward estimate of $\mathbf{T}_{KH}$:

$$\overline{\mathbf{T}}^{f}_{KH} = \overline{\mathbf{S}}_{KV}\, \overline{\mathbf{W}}^{f}_{VH}\,. \tag{13}$$

The forward estimate of $\mathbf{S}_{KH}$ is

$$\overline{\mathbf{S}}^{f}_{KH} = \varphi(\overline{\mathbf{T}}^{f}_{KH}\ ). \tag{14}$$

The augmented weight matrix between the hidden layer and the output layer is updated using the inverse of the renormalization matrix:

$$\hat{\mathbf{W}}^{f}_{HO} = \Gamma^{-1}\ \ \hat{\mathbf{W}}_{HO}. \tag{15}$$

Finally, a forward estimate of $\mathbf{T}_{KO}$ is obtained:

$$\overline{\mathbf{T}}^{f}_{KO} = \hat{\mathbf{S}}^{f}_{KH}\ \ \hat{\mathbf{W}}^{f}_{HO}, \tag{16}$$

which yields

$$\overline{\mathbf{S}}^{f}_{KO} = \varphi(\overline{\mathbf{T}}^{f}_{KO}\ ). \tag{17}$$

The error is measured as some norm (*e.g.* L_2) of the difference between the provided target output pattern matrix and the corresponding forward estimate:

$$E = \left\| \mathbf{S}_{KO} - \overline{\mathbf{S}}^{f}_{KO} \right\| . \tag{18}$$

If additional hidden layers are used, each hidden layer matrix $\overline{\mathbf{S}}_{KN_\ell}$ is renormalized by post-multiplying by an appropriate matrix Γ_ℓ as in Eq. (11) during the backward sweep, and the augmented weight matrix $\hat{\mathbf{W}}_{N_\ell N_{\ell+1}}$ is updated by pre-multiplying by Γ_ℓ^{-1} , as in Eq. (15).

4. Construction of the renormalization matrix. The matrix Γ rescales the backward computation of $\overline{\mathbf{S}}_{KH}$ so that its elements are in the range of φ (where again we assume only one hidden layer). A simple method for rescaling is the transformation $s_{ij} \rightarrow \alpha \cdot s_{ij} + \beta$ for appropriate constants α, β. The augmented matrix $\hat{\mathbf{S}}_{KH} = [\overline{\mathbf{S}}_{KH} \mid \mathrm{b_K}]$ includes a final column $\mathrm{b_K}$. The rescaled matrix $[\tilde{\mathbf{S}}_{KH} \mid \mathrm{b_K}] = [\overline{\mathbf{S}}_{KH} \mid \mathrm{b_K}]\ \Gamma$ should preserve the K values of the vector $\mathrm{b_K}$ in the last column. The transformations

$$(s_{ij} \rightarrow \alpha \cdot s_{ij} + \beta\ , 1 \le j \le H\ ;\ b_i \rightarrow b_i\ ),\ 1 \le i \le K \tag{19}$$

can be implemented by the matrix

$$\Gamma = \begin{bmatrix} \alpha & 0 & 0 & \cdots & 0 & 0 \\ 0 & \alpha & 0 & \cdots & 0 & 0 \\ 0 & 0 & \alpha & \cdots & & 0 \\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & \alpha & 0 \\ \beta & \beta & \beta & \cdots & \beta & 1 \end{bmatrix}.$$

For positive values of α, the matrix Γ is nonsingular and easily invertible.

Notes:

1. The range (0,1) of φ corresponds to domain values $(-\infty, \infty)$. We chose to restrict values to the subrange [0.1, 0.9], which corresponds to the finite subdomain [-ln(9), ln(9)].
2. We use $\alpha \leq 1$ to contract the values $\{ s_{ij} \}$ to fit in an interval of length ≤ 0.8. If the range of $\{ s_{ij} \}$ is less than 0.8, it is not necessary to expand the range, and we let $\alpha = 1$.
3. The value β represents a shift of the values to fit within [0.1, 0.9]. We have modified our choices of α and β so that the average value μ for the set $\{ s_{ij} \}$ is mapped to 0.5, the center of the interval [0.1, 0.9], and the value s_{ij} farthest from μ is mapped to either 0.1 or 0.9.

5. Predicted error of learning results. Due to the SVD computations, the learning algorithm introduces "minimum norm" type approximations for the solutions of each of the linear systems. Due to the nonlinear transformations effected by each layer, even if each layer were solved optimally, this still would not guarantee global optimal results. However, we can predict near-optimal results for the learning algorithm if the following three conditions are met:

1. The training set does not contain duplicate input vectors.
2. The number H of hidden-layer nodes is greater than the number O of output nodes. (If more than one hidden layer is used, the number of hidden-layer nodes decreases in successive hidden layers.)
3. The mapping from the input matrix $\mathbf{S}_{KI}$ to the virtual-layer matrix $\overline{\mathbf{S}}_{KV}$ produces a nonsingular square matrix (with $V = K$).

If condition (1) is not met, then the training set contains either duplicate copies of the same (input,output) pair, or contradictory patterns in which the same input is matched with more than one output pattern. Preprocessing of the training patterns can be introduced to handle such situations.

Condition (2) implies that the linear system $(\mathbf{T}_{KO} - \mathbf{b}_K \mathbf{v}_O) = \overline{\mathbf{S}}_{KH} \mathbf{W}_{HO}$ is underdetermined (provided that the rows of $\mathbf{W}_{HO}$ are linearly independent, as is usually the case) and hence the "minimum norm" solution for $\overline{\mathbf{S}}_{KH}$ should be exact (one of infinitely many).

Condition (3) implies that $\overline{\mathbf{S}}_{KV}$ is invertible, and the singular value decomposition gives the inverse of $\overline{\mathbf{S}}_{KV}$. Thus the "minimum norm" solution for $\mathbf{W}_{VH}$ should also be exact.

Since the inverse of the transfer function φ is known explicitly, the preceding analysis predicts that, in theory, the learning algorithm should result in error $E = 0$ (or as close to zero as can be expected from repeated finite-precision calculations with matrices).

6. Conversion from input layer to virtual-input layer. Condition (2) above illustrates the critical role of the virtual-input layer. It is common for the number K of training examples to exceed the number of input nodes I. If we attempt to use the training process without the virtual layer, the linear system $T_{KH} = S_{KI}\, W_{IH}$ is overdetermined for W_{IH}, and the resulting "minimum norm" solution may be a poor approximation. (For example, for the standard XOR function with $I = 2$ binary inputs and $K = 4$ training examples, the training algorithm applied without the virtual input layer gives $\overline{S}^{f}_{KO} = [\ 0.5\ \ 0.5\ \ 0.5\ \ 0.5\ ]^T$ as an approximation of the target output vector $S_{KO} = [\ 0\ \ 1\ \ 1\ \ 0\ ]^T$.)

In the following, we present a mapping ψ which always produces a nonsingular matrix $\overline{S}_{KV}$. Suppose the input vectors have dimension $I > 1$. For each component $m = 1, 2, \ldots, I$, construct a $K \times K$ matrix $\overline{S}^{(m)}_{KV}$ in which

$$\overline{S}^{(m)}_{KV}(i,j) = 1 - \frac{|s(i)_m - s(j)_m|}{D_m}\ ,\ i,j = 1, \ldots, K,\ m = 1, \ldots, I, \tag{21}$$

where $s(i)_m$ denotes the mth component of the ith training vector $s(i)$ and D_m is the maximum of $|\, s(i)_m - s(j)_m \,|$ over all $i, j = 1, \ldots, K$. Let $\overline{S}_{KV}$ be the block diagonal matrix whose mth block is $\overline{S}^{(m)}_{KV}$. The determinant of the full matrix is [14]:

$$\prod_{m=1}^{I}\left(\frac{1}{2}\prod_{j=1}^{K-1}\frac{2d_m(j)}{D_m}\right) \tag{22}$$

This guarantees that the matrix is nonsingular, but – for large I - severely increases the size of the matrix. As an alternative, we define the mapping ψ by taking the (i,j)th component of $\overline{S}_{KV}$ to be

$$1 - \frac{\|s(i) - s(j)\|}{D},\ i,j = 1, \ldots K, \tag{23}$$

where D is the maximum distance between input vectors in the training set. In general, for some training sets, the matrix $\overline{S}_{KV}$ obtained by using the transformation (23) may be singular. However, the SVD method does not require an invertible matrix. If the linear system is overdetermined, the SVD will generate a "least squares" best fit approximation.

We have studied several virtual-layer transformations to be used within this general approach to neural network training. Different virtual layer transformations may provide better representation of the data, yielding better learning results and/or better ability of the trained neural network to generalize. The choice of an *optimal* mapping ψ from S_{KI} to $\overline{S}_{KV}$ is still under investigation.

7. Implementation. We implemented this algorithm on several examples with excellent results [14]. Here we present just one example, for illustration purposes. The function f is given by:

$$f(x_1, x_2, x_3, x_4, x_5) = |x_1 - 0.25| + \sin(6x_2) - x_3 e^{0.5-x_3} + 2x_4 + x_5(0.5 - x_5),\ 0 \le x_i \le 1.$$

A neural network was constructed with five input nodes, 1000 virtual-layer nodes, a single hidden layer with 20 nodes, and one output node, using the component-wise virtual-layer transformation described by Eq. (21). The training set comprised 200 randomly generated input vectors (x_1, x_2, x_3, x_4, x_5) in $[0,1]^5$, with corresponding output values of f. The network was trained, and the absolute error was computed for each training input vector. The mean error was 0.000006, with a maximum error of 0.000021. We tested the trained network's ability to generalize using 1600 randomly generated vectors in $[0,1]^5$. The function values are between –1.75 and 3.4. The L_2 norm for the error was 0.0268, with a maximum absolute error of 0.1348.

We also trained and tested a neural network using the alternative virtual-layer transformation of Eq. (23), using 5 input nodes, 200 virtual-layer nodes, and 20 hidden-layer nodes. For the training vectors, the mean error was 0.000189, with a maximum error of 0.000637. Over the test set, the L_2 norm error was 0.2812, with a maximum absolute error of 1.014. The component-wise transformation (21) produced more accurate results for this function. The training set comprised 200 randomly generated input vectors (x_1, x_2, x_3, x_4, x_5) in $[0,1]^5$, with corresponding output values of f. The network was trained, and the absolute error was computed for each training input vector. The mean error was 0.000087, with a maximum error of 0.000295. We tested the trained network's ability to generalize using 1600 randomly generated vectors in $[0,1]^5$. The function values are between 0 and 3.5. The L_2 norm for the error was 0.3968, with a maximum absolute error of 1.34.

8. Conclusions. We presented a new training algorithm for multi-layer feed-forward neural networks. The new algorithm has several important features: (i) the ability to train in a single iteration; (ii) avoidance of local minima of the error function; (iii) near-optimal training results; and (iv) the ability to generalize well.

These features are based on a new architecture that includes a virtual input layer and on a new algorithmic idea. The algorithm calculates backward from the output layer, alternating between inverting nonlinear transfer functions within layers and using SVD to solve linear systems between layers, and then moving forward through the neural network, updating interconnection weights, to complete the training process in a single iteration. The use of a virtual input layer reduces a highly overdetermined linear system to a solvable one. We showed that in theory (with infinite-precision calculations), the method trains the neural network with error $E = 0$.

The implementation showed that even with finite-precision calculations with large matrices, the observed training error is near zero. In addition, the examples demonstrated that the trained neural networks can generalize well for input vectors which are not in the training set.

Acknowledgments

This research was funded in part by the *DeepLook* Consortium under Agreement Number ERD-97-1506 with Lockheed Martin Energy Research Corporation. Additional funding was provided by the Engineering Research Program of the Office of Basic Energy Sciences under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation. R. Cogswell's participation was supported by the Great Lakes Colleges Association/Associated Colleges of the Midwest Oak Ridge Science Semester, sponsored by the Office of University Science Education, Oak Ridge National Laboratory.

References

1. M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press (1995).

2. C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press (1997).

3. M. Beckerman, *Adaptive Cooperative Systems*, Wiley Interscience (1997).

4. D. A. White and D. A. Sofge, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold (1992).

5. A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, Wiley Interscience (1993).

6. P. Mars, J. R. Chen, and R. Nambiar, *Learning Algorithms*, CRC Press (1996).

7. Y. Chauvin and D. E. Rumelhart, *Backpropagation: Theory, Architectures, and Applications*, Lawrence Erlbaum (1995).

8. J. Barhen, S. Gulati, and M. Zak, "Neural Learning of Constrained Nonlinear Transformations", *IEEE Computer*, **22**(6), 67-76 (1989).

9. J. Barhen, N. Toomarian, and S. Gulati, "Applications of Adjoint Operators to Neural Networks", *Appl. Math. Lett.*, **3**(3), 13-18 (1990).

10. N. Toomarian and J. Barhen, "Learning a Trajectory using Adjoint Functions and Teacher Forcing", *Neural Networks*, **5**, 473-484 (1992); *ibid.*, *U.S. Patent No.* 5,428,710 (March 28, 1995).

11. F. Biegler-König and F. Bärman, "A Learning Algorithm for Multilayered Neural Networks based on Linear Least Squares Problems", *Neural Networks*, **6**, 127-131 (1993).

12. Y. F. Tam and T. W. S. Chow, "Accelerated Training Algorithm for Feedforward Neural Networks Based on Least Squares Method", *Neural Processing Letters*, **2**(4), 20-25 (1995).

13. G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press (1983).

14. J. Barhen, R. Cogswell, and V. Protopopescu, "Single-Iteration Training Algorithm for Multi-layer Feed-forward Neural Networks", *Neural Processing Letters,* submitted.