Feed-Forward Network Training Using Optimal Input Gains

Sanjeev S. Malalur, Student Member, IEEE, Michael Manry, Sr. Member, IEEE

Abstract—In this paper, an effective batch training algorithm is developed for feed-forward networks such as the multilayer perceptron. First, the effects of input transforms are reviewed and explained, using the concept of equivalent networks. Next, a non-singular diagonal transform matrix for the inputs is proposed. Use of this transform is equivalent to altering the input gains in the network. Newton’s method is used to solve for the input gains and an optimal learning factor. In several examples, it is shown that the final algorithm is a reasonable compromise between first order training methods and Levenburg-Marquardt.

I. INTRODUCTION

The multilayer perceptron (MLP) has been shown to have several properties that make it of interest to investigators. First, it can be trained using gradient approaches such as backpropagation (BP) [1] and Levenburg-Marquardt (LM) [2]. It has the universal approximation property [3]. With proper training, the MLP approximates the Bayes classifier [4] or the minimum mean-square error (mmse) estimator [5]. The MLP has found use in many applications including character recognition [6] [7], power load forecasting [8], prognostics [9], well logging [10], and data mining [11].

Unfortunately, MLP training is sensitive to many parameters of the network and its training data, including the input means and the initial network weights. In addition, MLP training is sensitive to the collinearity of its inputs [12].

In this paper, a fast, convergent training algorithm is developed which attempts to compensate for input collinearity. In section II, some MLP matrix-vector notation is introduced for BP in the network’s input weights.

Non-singular input transforms are reviewed in section III. It is shown that non-singular orthogonal transforms of inputs are useless. In section IV, the emphasis is therefore on non-orthogonal transforms. The simplest such transform, consisting of a diagonal matrix, is introduced. It is shown to be equivalent to modifying the gains on the network inputs. A combination of the optimal input gains (OIGs) and optimal learning factor (OLF) are then found, using Newton’s method. In section V, the computational burdens of the OIG and other training algorithms are presented for comparison. Numerical results and conclusions are presented in sections VI and VII.
the $k^{th}$ hidden unit’s activation output is denoted as $O_p(k)$, where $O_p(k) = f(n_p(k))$. $f(\cdot)$ denotes the hidden layer activation, which is usually nonlinear, the most popular one being the sigmoid [1].

For the $p^{th}$ pattern, the $i^{th}$ element $y_p(i)$ of the M-dimensional output vector $y_p$ is

$$y_p(i) = \sum_{n=1}^{N_v+1} w_{oi}(i, n) \cdot x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i, k) \cdot O_p(k) \quad (3)$$

which can be summarized as

$$y_p = W_{oi} \cdot x_p + W_{oh} \cdot O_p \quad (4)$$

where $O_p$ is the $N_p$-dimensional hidden unit activation vector. The last rows of $W$ and $W_{oi}$ respectively store the hidden unit and output unit threshold values.

B. Training using Backpropagation

A typical error function used in training the MLP is the mean-squared error (MSE) described as

$$E = \frac{1}{N_v} \sum_{i=1}^{N_v} [t_p(i) - y_p(i)]^2 \quad (5)$$

The $N_h$ by $(N+1)$ negative input weight Jacobian matrix for the $p^{th}$ pattern’s input weights is

$$G = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p \cdot x_p^T \quad (6)$$

where $\delta_p = [\delta_p(1), \delta_p(2), \ldots, \delta_p(N_h)]^T$ is the $N_h$ by 1 column vector of hidden unit delta functions [1]. If steepest descent is used to modify the input weight matrix $W$, then $W$ is updated in a given iteration as

$$W \leftarrow W + z \cdot G \quad (7)$$

so

$$\Delta W = z \cdot G \quad (8)$$

where $z$ is the learning factor. In output weight optimization-backpropagation (OWO-BP), we alternately use BP to modify input weights, as in (7), and solve linear equations for output weights. In standard BP, all weights are changed as in (7). Both algorithms converge if $z$ denotes the OLF for that algorithm.

III. LINEAR TRANSFORMATION OF INPUTS

In this section we analyze the effects of transforming the augmented input vectors $x_p$.

A. Derivation

Let MLP-1, be trained using the input vectors $x_p$ as described in section II-B. In a second network called MLP-2, the input vectors $x_p'$ are defined as $x_p' = A \cdot x_p$ where $A$ is an $N'$ by $(N+1)$ rectangular transformation matrix, for some $N' \geq (N+1)$. MLP-2 has parameters $w'(k, n)$, $w_{oh}(i, k)$, and $w_{oi}(i, n)$. Therefore the net function vector and output vector for MLP-2 are respectively

$$n_p' = W' \cdot x_p' \quad (9)$$

and

$$y_p' = W_{oi}' \cdot x_p' + W_{oh}' \cdot O_p' \quad (10)$$

Let MLP-2 be equivalent to MLP-1, which means that it has the same output vectors, so $y_p' = y_p$. One way that this can occur is if (i) $O_p' = O_p$, which leads to $W_{oh}' = W_{oh}$, and (ii) the transformation from $x_p$ to $x_p'$ is invertible. Next, $n_p' = n_p$ so (9) leads to

$$n_p = W \cdot x_p = W' \cdot x_p' = W'A \cdot x_p \quad (11)$$

and

$$W' \cdot A = W \quad (12)$$

Finally,

$$W_{oi}' \cdot A = W_{oi} \quad (14)$$

The negative Jacobian matrix for training input weights in MLP-1 is given in (6). The negative Jacobian for training input weights in MLP-2 is then

$$G' = \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p \cdot (x_p')^T$$

$$= \frac{1}{N_v} \sum_{p=1}^{N_v} \delta_p \cdot x_p^T A^T \quad (15)$$

$$= G \cdot A^T$$

Suppose that the negative Jacobian $G'$ for MLP-2 is mapped back to modify input weights in MLP-1, using (14). This mapped negative Jacobian for MLP-1 is then

$$G'' = G \cdot A^T A \quad (16)$$

Lemma 1: If we are at a local minimum in the weight space of the original network, we are also at a local minimum in the weight space of the transformed network. This is clear for the input weights from (16).

Lemma 2: If BP is used to train the transformed network’s input weight matrix $W'$, this is not equivalent to applying BP to the original network’s weight matrix $W$ unless $A$ is an orthogonal matrix. Again, this follows from (16). For any orthogonal matrix $A$, (16) becomes

$$G'' = G \quad (17)$$

When optimal learning factors are used with BP to train the input weights, training with the original data is equivalent to training with the transformed data. Orthogonal transform matrices are therefore useless [13].
B. Effect of Input Collinearity

In the previous subsection, if the matrix $A$ is rectangular, with $N' > (N + 1)$, so that $A$ has more rows than columns, then the elements of $x'_p$ are linearly dependent or collinear. From Lemma 2, training changes when collinear inputs are used. $BP$ in the collinear or dependent data network is not equivalent to steepest descent in the non-collinear network.

To show that training is different for a collinear network and a non-collinear network, we carry out a simple simulation. Given a training data set, we generate additional dependent inputs and initialize the collinear network. Specifically, $A$ consists of an $(N + 1)$ by $(N + 1)$ identity matrix, followed by $K$ rows of random numbers. $K$ is then the number of dependent inputs. Using the concept of equivalent networks, the non-collinear network is initialized as in (12) and (14). The two networks are then trained separately.

![Fig. 2. Effect of collinearity on OWO-BP training](image)

Fig. 2 compares the training on dependent data and independent data, using OWO-BP. The two networks are equivalent at the start of training. Clearly, the training is different for the two networks as stated in Lemma 2.

Considering Lemma 2, why have some investigators proposed the use of orthogonal transform matrices $A$, as a means of improving training? Perhaps because this approach removes input collinearity. However, there is no guarantee that training for a collinear input network is always worse.

C. A Useful Non-Orthogonal Transform

Suppose that the input vectors $x_p$ are biased such that $E[x_p] = m$. A zero-mean version of $x_p$ is $x'_p$, which satisfies

$$x_p = x'_p + m$$

(18)

It is well-known that networks train more effectively with unbiased inputs [14]. Now, $x'_p$ can be expressed as $A \cdot x_p$, where

$$A = \begin{bmatrix} 1 & 0 & \ldots & 0 & -m_1 \\ 0 & 1 & \ldots & 0 & -m_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & -m_N \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}$$

(19)

From Fig. 2, we see that some non-orthogonal transform matrices make training worse. However, the non-orthogonal transform matrix above improves training, since it makes the inputs zero-mean.

IV. Non-Singular, Non-Orthogonal Transformation of Inputs

So far, it is clear that $A$ should not be orthogonal, and should not have more rows than columns. In this section, we discuss the utility of non-singular, non-orthogonal $A$ matrices, with $N' = (N + 1)$.

A. A Diagonal Transform Matrix

The simplest non-orthogonal, nonsingular transform matrix $A$ is diagonal. For this case, let $a(k)$ initially denote the $k^{th}$ diagonal element of $A^T A$. Also, the elements of $x'_p$ are simply scaled versions of $x_p$.

Following (16) we get

$$G'' = G \cdot \begin{bmatrix} a(1) & 0 & \ldots & 0 & 0 \\ 0 & a(1) & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & a(N) & 0 \\ 0 & 0 & \ldots & 0 & a(N + 1) \end{bmatrix}$$

(20)

Instead of using the negative Jacobian elements $g(k, n)$ in training the network, we use $g(k, n) \cdot a(n)$. Note also that the optimal learning factor (OLF) $z$ can be absorbed into the gains $a(n)$.

Assume that the MLP is being trained using OWO-BP. Given the negative Jacobian $G$ of dimension $N_h$ by $(N + 1)$, the error function being minimized with respect to the $a(n)$’s is given in (5), where

$$y_p(i) = \sum_{n=1}^{N+1} w_{ai}(i, n)x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i, k) \cdot f \left( \sum_{n=1}^{N+1} (w(k, n) + a(n) \cdot g(k, n))x_p(n) \right)$$

(21)

The first partial of $E$ with respect to $a(m)$ is

$$g(m) = \frac{\partial E}{\partial a(m)} = -\frac{2}{N_v} \sum_{p=1}^{N_v} x_p(m) \sum_{i=1}^{M} \left[ t_p(i) - \sum_{k=1}^{N_h} w_{oh}(i, k)f(n_p(k)) \right] \cdot \sum_{k=1}^{N_h} w_{oh}(i, k)f'(n_p(k))g(k, m)$$

(22)
Here, \( g(k, m) \) is an element of the negative Jacobian matrix \( G \) in equation (6), and \( f'(n_p(k)) \) denotes the derivative of \( f(n_p(k)) \) with respect to its net function. Then,

\[
t'_p(i) = t_p(i) - \sum_{n=1}^{N+1} w_{ai}(i, n)x_p(n)
\]

\( n_p(m) = \sum_{n=1}^{N+1} (w(m, n) + a(n) \cdot g(m, n))x_p(n) \) (23)

\( v(i, m) = \sum_{k=1}^{N_h} w_{oh}(i, k)f'(n_p(k))g(k, m) \)

The second partial derivatives are given by

\[
g_2(m, m) = \frac{\partial^2 E}{\partial a(m)^2} = \frac{2}{N_v} \sum_{p=1}^{N_v} x_p^2(m) \sum_{i=1}^{M} v^2(i, m) \quad (24)
\]

\[
g_2(m, u) = \frac{\partial^2 E}{\partial a(m)\partial a(u)} = \frac{2}{N_v} \sum_{p=1}^{N_v} x_p(m)x_p(u) \sum_{i=1}^{M} v(i, m)v(i, u) \quad (25)
\]

B. Optimal Input Gain Algorithm

Given \( g \) and the Hessian \( G^2 \), we minimize \( E \) with respect to the vector \( a \) using Newton’s method. Now we have a choice. In each iteration, we can (i) use \( A^T A \) to transform the gradient matrix as in (16), or we can (ii) decompose \( A^T A \) to find \( A \) using an SVD approach. We can then transform the input data and use OWO-BP with the optimal learning factor. However, this latter approach is too inefficient to consider, even when \( A \) is diagonal. We use the first approach.

In each iteration of the training algorithm, the steps are as follows:

(i) Calculate the input weight Jacobian \( G \) using BP.
(ii) Calculate the OLF-input gain products \( a(n) \)
(iii) Update the input weights as
\[
w(k, n) \leftarrow w(k, n) + a(n)g(k, n)
\]
(iv) Solve linear equations for all output weights

Here, the optimal input gain (OIG) procedure has been inserted into the OWO-BP algorithm. It can be inserted into other algorithms as well, including standard BP.

V. COMPUTATIONAL BURDEN

In this section, we describe the computational burden for using the training algorithms compared in this paper. Let \( N_u = N + N_h + 1 \) denote the number of weights connected to each output. The total number of weights in the network is denoted as \( N_w = M(N + N_h + 1) + N_h(N + 1) \). The number of multiplies required to solve for output weights using the Orthogonal Least Squares [22] is \( M_{ols} \), which is described by

\[
M_{ols} = N_u(N_u + 1) \left[ M + \frac{1}{6} N_u(2N_u + 1) + \frac{3}{2} \right] \quad (26)
\]

The numbers of multiplies required for training using BP, OWO-BP, OIG and LM are respectively given by

\[
M_{bp} = N_{it}\{N_u[MN_u + 2Nh(N + 1) + M(N + 6Nh + 4)] + N_w\} \quad (27)
\]

\[
M_{owo–bp} = N_{it}\{N_u[2Nh(N + 2) + M(N_u + 1) + \frac{N_u(N_u + 1)}{2} + M(N + 6Nh + 4)] + M_{ols} + Nh(N + 1)\} \quad (28)
\]

\[
M_{oig} = M_{owo–bp} + N_{it}\{N_v[(N + 1)(3MN_h + MN + 2(M + N) + 3) - M(N + 6Nh + 4) - Nh(N + 1)] + (N + 1)^3\} \quad (29)
\]

\[
M_{lm} = M_{bp} + N_{it}\{N_v[MN_u(N_u + 3Nh(N + 1)) + 4N_h^2(N + 1)^2] + N_w^2 + N_h^2\} \quad (30)
\]

where \( N_{it} \) is the number of training iterations.

Note that \( M_{oig} \) consists of \( M_{owo–bp} \) plus the required multiplies for calculating optimal input gains. Similarly, \( M_{lm} \) consists of \( M_{bp} \) plus the required multiplies for calculating and inverting the Hessian matrix.

VI. RESULTS

Here we present the results for the OWO-BP algorithm, modified using the optimal input gain method. We compare its performance with BP, OWO-BP and LM, where optimal learning factors (OLFs) were used in the latter three algorithms. In BP and LM, all weights are varied in each iteration. In OWO-BP, we alternately use BP for the input weights (with the OLF) and solve linear equations for the output weights.

For a given network, we obtain the training error and the number of multiplies required for each training iteration. We also obtain the validation error for a fully trained network. This information is used to subsequently generate the plots and compare performances.

We use the \( k \)-fold cross-validation procedure to obtain the training and validation errors. Given a data set, we split the set into \( k \) non-overlapping parts of equal size, and use \((k - 1)\) parts for training and the remaining one part for validation. The procedure is repeated till we have exhausted all \( k \) combinations (\( k = 10 \) for our simulations).

All the data sets used for simulation are publicly available. In all data sets, the inputs have been normalized to be zero-mean and unit variance.
A. Prognostics Data Set

This data file is available on the Image Processing and Neural Networks Lab repository [19]. It consists of parameters that are available in the Bell Helicopter health usage monitoring system (HUMS), which performs flight load synthesis, which is a form of prognostics [9]. The data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995. The seventeen input features include cockpit available signals including CG F/A load factor, pitch attitude, roll attitude, yaw rate, and several others. The nine desired outputs are loads on various mechanical components.

For this data file, which is called F17, we trained an MLP having 15 hidden units. In Fig. 3, the average mean square error (MSE) for training from 10-fold validation is plotted versus the number of iterations for each algorithm. In Fig. 4, the average training MSE from 10-fold validation is plotted versus the required number of multiplies (shown on a $\log_{10}$ scale).

From Fig. 3 and Fig. 4, the proposed optimal input gain algorithm converges faster than BP or OWO-BP, and it is much faster than LM.

B. Remote Sensing Data Set

This data file is available on the Image Processing and Neural Networks Lab repository [19]. It consists of 16 inputs and 3 outputs and represents the training set for inversion of surface permittivity, the normalized surface rms roughness, and the surface correlation length found in back scattering models from randomly rough dielectric surfaces [15].

For this data file, which is called Single2, we trained an MLP having 15 hidden units. In Fig. 5, the average training MSE from 10-fold validation is plotted versus the number of iterations for each algorithm. In Fig. 6, the average training MSE from 10-fold validation is plotted versus the required number of multiplies.

From Fig. 5 and Fig. 6, the optimal input gain algorithm again converges faster than BP and OWO-BP, and it has smaller training error. In this example, it trains as well as LM, almost two orders of magnitude fewer multiplies.

C. Federal Reserve Economic Data Set

This file contains some economic data for the USA from 01/04/1980 to 02/04/2000 on a weekly basis. From the given features, the goal is to predict the 1-Month CD Rate [17]. It has 15 inputs and one output per pattern, with a total of 1049 patterns. For this data file, which is called TR, we trained an MLP having 15 hidden units. In Fig. 7, the average training MSE from 10-fold validation is plotted versus the number of iterations for each algorithm. In Fig. 8, the average training MSE from 10-fold validation is plotted versus the required number of multiplies.

From Fig. 7 and Fig. 8, the optimal input gain algorithm has a training error close to that of LM, with far fewer
D. Housing Data Set

This data file is available on the DELVE dataset repository [20]. It was designed on the basis of data provided by the US Census Bureau (under Lookup Access: Summary Tape File 1). The data were collected as part of the 1990 US census. These are mostly counts cumulated at different survey levels. For the purpose of this data set a level State-Place was used. Data from all states was obtained. Most of the counts were changed into appropriate proportions [18].

These are all concerned with predicting the median price of houses in a region based on demographic composition and the state of the housing market in the region. For Low task difficulty, more correlated attributes were chosen as signified by univariate smooth fit of that input on the target. Tasks with high difficulty have had their attributes chosen to make the modeling more difficult due to higher variance or lower correlation of the inputs to the target.

The training data consists of 16 inputs and 1 output per pattern, with a total of 22,784 patterns. For this data file, we trained an MLP having 15 hidden units. In Fig. 9, the average training MSE from 10-fold validation is plotted versus the number of iterations for each algorithm. In Fig. 10, the MSE from 10-fold validation is plotted versus the required number of multiplies.

From Fig. 9 and Fig. 10, the optimal input gain algorithm has a training error close to that of LM, with far fewer multiplies per iteration.

E. Concrete Compressive Strength Data Set

This data file is available on the UCI Machine Learning Repository [21]. It contains the actual concrete compressive strength (MPa) for a given mixture under a specific age (days) determined from laboratory. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash,
water, super plasticizer, coarse aggregate, and fine aggregate. The data set consists of 8 inputs and one output per pattern, with a total of 1030 patterns. For this data file, we trained an MLP having 15 hidden units. In Fig. 11, the average training MSE from 10-fold validation is plotted versus the number of iterations for each algorithm. In Fig. 12, the average training MSE from 10-fold validation is plotted versus the required number of multiplies.

From Fig. 11 and Fig. 12, the optimal input gain algorithm has a training error close to that of LM, with far fewer multiplies per iteration.

Table I compares the average training and validation errors of the proposed OIG algorithm with BP, OWO-BP and LM on different data sets. For each data set, the training and validation errors again come from 10-fold cross validation.

We can see that the proposed OIG algorithm sometimes has a performance comparable to or better than the popular LM algorithm.

**VII. CONCLUSIONS**

We have derived a second order method for simultaneously optimizing input gains and the OLF. The method has been successfully demonstrated on five datasets. Results show that this approach performs much better than two common first order algorithms with comparable complexity, namely BP and OWO-BP. It comes close to LM in terms of the training error, but with orders of magnitude less computation. This is evident in all of the plots of training error versus the required number of multiplies and also from the expressions for the numbers of multiplies.

Although LM works very well in practice it has a high computational burden and is sub-optimal in the way it handles the 'scaling' factor, $\lambda$. OIG on the other hand uses a Newton type update and combines the optimal learning factor, leaving little room for heuristics.

Much work remains to be done. We hope to extend our
approach to additional network parameters, yielding a fast second order method that more often rivals the performance of LM, but with greatly reduced complexity.

REFERENCES


