Comparison of three back-propagation training algorithms for two case studies

Özgür Kişio̧ & Erdal Uncuoğlu

Hydraulics Division, Civil Engineering Department, Engineering Faculty, Erciyes University, Kayseri, Turkey
Geotechnology Division, Civil Engineering Department, Engineering Faculty, Erciyes University, Turkey

Received 23 June 2004; accepted 26 May 2005

This paper investigates the use of three back-propagation training algorithms, Levenberg-Marquardt, conjugate gradient and resilient back-propagation, for the two case studies, stream-flow forecasting and determination of lateral stress in cohesionless soils. Several neural network (NN) algorithms have been reported in the literature. They include various representations and architectures and therefore are suitable for different applications. In the present study, three NN algorithms are compared according to their convergence velocities in training and performances in testing. Based on the study and test results, although the Levenberg-Marquardt algorithm has been found being faster and having better performance than the other algorithms in training, the resilient back-propagation algorithm has the best accuracy in testing period.

IPC Code: G06N3/02

Many of the activities associated with the planning and operation of the components of a water resource system require forecasts of future events. For the hydrologic component, there is a need for both short-term and long-term forecasts of stream-flow events in order to optimize the system or to plan for future expansion or reduction.

Artificial neural networks (ANNs) have proven to be an efficient alternative to traditional methods for modeling qualitative and quantitative water resource variables1-9. Recently, numerous ANN-based rainfall-runoff models have been proposed to forecast stream-flow10-18. ANN concepts and applications in hydrology have been discussed recently by the ASCE Task Committee on Application of ANN in Hydrology19, which conclude that ANNs may be perceived as alternative modeling tools worthy of further exploration. It is important to note that on average 90% of the application cases in hydrology have used the conventional feed forward neural network, namely the standard multi-layer perceptron (MLP) trained with the back-propagation algorithm20.

Recently, Maier and Dandy21 dealt with the use of the ANN model for the prediction and forecasting of water resources variables. It is evident from the literature that no study has been carried out to utilize the input-output mapping capability of Levenberg-Marquardt (LM), conjugate gradient (CG) and resilient back-propagation (RB) neural network training algorithms in the forecasting of stream-flow. Therefore, the first case in the present study was devoted to the utilization of the input-output mapping capabilities of these three neural network algorithms in daily stream-flow forecasting.

Prediction of the in-situ state of stress in soils is of major importance with regard to a variety of geotechnical problems. Numerous investigators22 have addressed this problem and have achieved varying degrees of success. Although a substantial data base has been developed, it is still not possible to predict exactly the in-situ state of stress in most natural soil deposits, because they have undergone a complex stress history of loading and unloading which is difficult to reconstruct precisely. The geostatic vertical stress can be estimated from a profile of effective overburden stress with depth. The in-situ horizontal stress, however, is highly dependent on the geological history of the soil.

A review of the literature reveals that ANNs have been used successfully in pile capacity prediction, modeling soil behaviour, site characterization, earth retaining structures, settlement of structures, slope stability design of tunnels and underground openings, liquefaction, soil permeability and hydraulic conductivity, soil compaction, soil swelling and classification of soils23. In the second case study, the comparison of the NN algorithms in lateral stress estimation was employed.
Neural Network Training Algorithms

Three different artificial neural network (ANN) training algorithms, Levenberg-Marquardt, conjugate gradient and resilient back-propagation, are used in the present study. This was done with a view to see which algorithm produces better results and has faster training for the application under consideration.

The objective of training is to reduce the global error $E$ defined as

$$E = \frac{1}{P} \sum_{p=1}^{P} E_p$$  \hspace{1cm} ... (1)

where $P$ is the total number of training patterns; and $E_p$ is the error for training pattern $p$. $E_p$ is calculated by the following formula:

$$E_p = \frac{1}{2} \sum_{i=1}^{N} (o_i - t_i)^2$$  \hspace{1cm} ... (2)

where $N$ is the total number of output nodes, $o_i$ is the network output at the $i^{th}$ output node, and $t_i$ is the target output at the $i^{th}$ output node. In every training algorithm, an attempt is made to reduce this global error by adjusting the weights and biases.

Levenberg-Marquardt algorithm

Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical in training feed forward networks), then the Hessian matrix can be approximated as

$$H = J^T J$$  \hspace{1cm} ... (3)

and the gradient can be computed as

$$g = J^T e$$  \hspace{1cm} ... (4)

where $J$ is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights and biases, and $e$ is a vector of network errors. The Jacobian matrix can be computed through a standard back-propagation technique that is much less complex than computing the Hessian matrix.

The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e$$  \hspace{1cm} ... (5)

When the scalar $\mu$ is zero, this is just Newton’s method, using the approximate Hessian matrix. When $\mu$ is large, this becomes gradient descent with a small step-size. Newton’s method is faster and more accurate near a minimum error, so the aim is to shift towards Newton’s method as quickly as possible. Thus, $\mu$ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm. The Levenberg-Marquardt optimization technique is more powerful than the conventional gradient descent techniques. The application of Levenberg-Marquardt to neural network training is described elsewhere.

Conjugate gradient algorithm

The basic back-propagation algorithm adjusts the weights in the steepest descent direction (the most negative of the gradients). This is the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions.

In most of the conjugate gradient algorithms the step-size is adjusted at each iteration. A search is made along the conjugate gradient direction to determine the step-size, which will minimize the performance function along that line.

All of the conjugate gradient algorithms start out by searching in the steepest descent direction on the first iteration.

$$p_0 = -g_0$$  \hspace{1cm} ... (6)

A line search is then performed to determine the optimal distance to move along the current search direction:

$$x_{k+1} = x_k + \alpha_k g_k$$  \hspace{1cm} ... (7)
Then the next search direction is determined so that it is conjugate to previous search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction:

$$p_k = -g_k + \beta_k p_{k-1}$$  \hspace{1cm} \ldots (8)$$

The various versions of conjugate gradient algorithms are distinguished by the manner in which the constant $\beta_k$ is computed. For the Fletcher-Reeves update the procedure is

$$\beta_k = \frac{g^T_k g_k}{g^T_{k-1} g_{k-1}}$$  \hspace{1cm} \ldots (9)$$

This is the ratio of the norm squared of the current gradient to the norm squared of the previous gradient. This update procedure was used in the study and denoted as CGF (conjugate gradient with Fletcher-Reeves).

Resilient back-propagation algorithm

Multi-layer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called “squashing” functions, since they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slope must approach zero as the input gets large. This causes a problem when using steepest descent to train a multi-layer network with sigmoid functions, since the gradient can have a very small magnitude; and therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

The purpose of the resilient back-propagation (RB) training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by a factor whenever the derivative changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating the weight change will be reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change will be increased. A complete description of one such type of algorithms known as the RPROP algorithm is given elsewhere.

Case Study I: Stream-flow Forecasting

Description of data

The daily stream-flow data belongs to Derecikviran Station (Station No: 1335) on Filyos Stream in Turkey were used in the applications. The data of October 01, 1999 to September 30, 2000 were chosen for calibration, and data for October 01, 2000 to September 30, 2001 were used for validation, arbitrarily. The daily statistical parameters of stream-flow data are given in Table 1. In this table, $x_{\text{mean}}$, $s_x$, $c_v$, $c_{sx}$, $x_{\text{min}}$, $x_{\text{max}}$ denote the mean, standard deviation, variation coefficient, skewness, minimum and maximum of the data, respectively. The all data sets have high skewness (2.65-4.06) especially the test data. $x_{\text{min}}$ and $x_{\text{max}}$ values fall in the range 21.6-1181 m$^3$/s. The validation data set extremes ($x_{\text{min}} = 9.8$ m$^3$/s, $x_{\text{max}} = 495$ m$^3$/s). $x_{\text{min}}$ for calibration data is higher than the corresponding validation set value. This may cause some extrapolation difficulties in forecasting low flow values. The auto-correlation functions (ACF) of the stream-flow data are shown in Fig. 1. The ACF of the training and entire data is close and parallel to each other. However, the test data’s ACF is decreasing and

<table>
<thead>
<tr>
<th>Data</th>
<th>$x_{\text{mean}}$ (m$^3$/s)</th>
<th>$s_x$ (m$^3$/s)</th>
<th>$c_v$</th>
<th>$c_{sx}$ (m$^3$/s)</th>
<th>$x_{\text{min}}$ (m$^3$/s)</th>
<th>$x_{\text{max}}$ (m$^3$/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>132</td>
<td>159</td>
<td>1.20</td>
<td>2.65</td>
<td>21.6</td>
<td>1181</td>
</tr>
<tr>
<td>Test</td>
<td>47</td>
<td>55</td>
<td>1.17</td>
<td>4.06</td>
<td>9.8</td>
<td>495</td>
</tr>
<tr>
<td>Entire</td>
<td>89</td>
<td>125</td>
<td>1.40</td>
<td>3.53</td>
<td>9.8</td>
<td>1181</td>
</tr>
</tbody>
</table>

Fig. 1—Autocorrelation functions of stream-flow data.
being far from the ACF of the entire data after six days of lag.

**Application and results**

In general, the architecture of multi-layer ANN can have many layers where a layer represents a set of parallel processing units (or nodes). The three-layer ANN used in this study contains only one intermediate (hidden) layer. Multi-layer ANN can have more than one hidden layer, however theoretical works have shown that a single hidden layer is sufficient for ANNs to approximate any complex non-linear function\(^{30,31}\). Indeed many experimental results seem to confirm that one hidden layer may be enough for most forecasting problems\(^{20,30}\). Therefore, in this study, one hidden layered ANN was used.

A difficult task with ANNs involves choosing parameters such as the number of hidden nodes, the learning rate, and the initial weights. Determining an appropriate architecture of a neural network for a particular problem is an important issue, since the network topology directly affects its computational complexity and its generalization capability. Here, the hidden layer node numbers of each model were determined after trying various network structures since there is no theory yet to tell how many hidden units are needed to approximate any given function. In the training stage, the initial weights that gave the minimum error were found for each ANN networks after being tried fifty different initial weights. The tangent sigmoid and pure linear functions are found appropriate for the hidden and output node activation functions, respectively.

Three MATLAB language codes were written for the ANN algorithms. Before applying the ANN to the data, the training input and output values were normalized using the equation

\[
a \frac{x_i - x_{\text{min}}}{x_{\text{max}} - x_{\text{mean}}} + b \quad \ldots \quad (10)
\]

where \(x_{\text{min}}\) and \(x_{\text{max}}\) denote the minimum and maximum of all training and testing data. Different values can be assigned for the scaling factors \(a\) and \(b\). There are no fixed rules as to which standardization approach should be used in particular circumstances\(^{33}\). In this study, the \(a\) and \(b\) values were taken as 0.6 and 0.2, respectively. The learning and momentum rates were taken as 0.01 and 0.9, respectively. It is seen that choosing high values like 0.5 for the learning rate, as done by Raman and Sunilkumar\(^{34}\), throws the network into oscillations or saturates the neurons\(^7\).

Let \(Q_t\) represent the stream-flow at time \(t\). The input combinations of flow data evaluated in the present study are: (i) \(Q_{t-1}\); (ii) \(Q_{t-1}\) and \(Q_{t-2}\); (iii) \(Q_{t-1}, Q_{t-2}\,\text{and}\,Q_{t-3}\); (iv) \(Q_{t-1}, Q_{t-2}, \ldots, Q_{t-4}\); (v) \(Q_{t-1}, Q_{t-2}, \ldots, Q_{t-5}\); (vi) \(Q_{t-1}, Q_{t-2}, \ldots, Q_{t-6}\). The output layer had one neuron for the current flow \(Q_t\).

The ANN algorithms were compared according to the mean absolute relative errors (MARE) and the mean square errors (MSE) criteria. These criteria are defined as:

\[
MARE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{Y_i^{\text{observed}} - Y_i^{\text{predicted}}}{Y_i^{\text{observed}}} \right| \times 100 \quad \ldots \quad (11)
\]

\[
MSE = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{Y_i^{\text{observed}} - Y_i^{\text{predicted}}}{Y_i^{\text{observed}}} \right)^2 \quad \ldots \quad (12)
\]

where \(N\) and \(Y\) denote the number of data sets and stream-flow data, respectively.

For each input combination, ANN was trained using three different training algorithms, that is, LM, CGF and RB. After training was over, the weights were saved and used to test (validate) the each network performance on test data. The ANN results were transformed back to the original domain, and the MARE and MSE were computed. The iterations were stopped when the error difference between two epochs was too small. Table 2 shows comparisons of the results of network training done by using error LM, CGF and RB schemes after the 50, 100 and 150 iterations, respectively. It is obviously seen from the table that the LM has smaller training time than the other algorithms for each input combination. The RB seems faster than CGF in training. The LM has also better training (learning) performances (MAREs and MSEs) than the RB and CGF. It can be said that the LM is better than the other two algorithms in function approximation.

The testing results of the networks in which their training results are given in Table 2, are represented in Table 3. A number of nodes in the hidden layer that gave the lowest MARE was determined for each network. The node number in the hidden layer was found to vary between 2 and 9. Out of the three ANN algorithms, the RB algorithm has the lowest MARE (9.13 m\(^3\)/s) for the fourth input combination. This
means that the LM algorithm could not learn the phenomenon exactly in the training period (Table 2).

The algorithms that have the lowest MARE in testing period (Table 3) are compared in Fig. 2 in the form of hydrograph and scatter plot. As can be seen from the hydrographs all the ANN algorithms’ forecasts are close to observed values. The RB algorithm gave a $R^2$ coefficient of 0.899, which was higher than the values of 0.897 and 0.890 obtained using the LM and CGF algorithms. The RB has 269 forecasts lower than the 10% relative error in testing period while the LM and CGF have the 261 and 226 forecasts lower than the 10% error, respectively. Furthermore, the LM and CGF have the 174 and 112 forecasts lower than the 5% relative error, respectively, while the RB has 210 forecasts lower than the 5% error.

The results are also tested by using analysis of variance (ANOVA) and $t$-test for verifying the robustness of the models. Both tests are set at a 95% significant level. The statistics of the tests are given in Table 4. The RB-based models yield small testing values (-0.033 and 0.001) with a high significance level (0.487 and 0.974) in both the $t$-test and ANOVA. The values of the LM- and CGF-based models are also statistically acceptable. According to the test results, the RB seems to be more robust in stream-flow forecasting than the LM and CGF.

Table 2—Training results of each ANN training algorithm

<table>
<thead>
<tr>
<th>Model input</th>
<th>CPU time (second)</th>
<th>MARE (m³/s)</th>
<th>MSE (m⁶/s²)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LM</td>
<td>CGF</td>
<td>RB</td>
</tr>
<tr>
<td>$Q_t$</td>
<td>1.87</td>
<td>5.00</td>
<td>2.8</td>
</tr>
<tr>
<td>$Q_{t-1}$</td>
<td>1.49</td>
<td>4.34</td>
<td>3.07</td>
</tr>
<tr>
<td>$Q_{t-2}$</td>
<td>1.76</td>
<td>5.93</td>
<td>3.19</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$</td>
<td>2.10</td>
<td>2.58</td>
<td>3.13</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$, $Q_{t-4}$</td>
<td>2.47</td>
<td>5.71</td>
<td>4.67</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$, $Q_{t-4}$, and $Q_{t-5}$</td>
<td>2.47</td>
<td>5.22</td>
<td>3.84</td>
</tr>
</tbody>
</table>

Table 3—The MARE and the MSE for different ANN applications in testing period

<table>
<thead>
<tr>
<th>Model input</th>
<th>Nodes in hidden layer</th>
<th>MARE (m³/s)</th>
<th>MSE (m⁶/s²)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LM</td>
<td>CGF</td>
<td>RB</td>
</tr>
<tr>
<td>$Q_t$</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$Q_{t-1}$</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$, $Q_{t-4}$</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>$Q_{t-1}$, $Q_{t-2}$, $Q_{t-3}$, $Q_{t-4}$, and $Q_{t-5}$</td>
<td>6</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4—Analysis of variance and $t$-test for stream-flow forecasting

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ANOVA</th>
<th>t-test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F-Statistic</td>
<td>Resultant Significance Level</td>
</tr>
<tr>
<td>LM</td>
<td>0.090</td>
<td>0.764</td>
</tr>
<tr>
<td>CGF</td>
<td>0.140</td>
<td>0.709</td>
</tr>
<tr>
<td>RB</td>
<td>0.001</td>
<td>0.974</td>
</tr>
</tbody>
</table>

Case Study II: Determination of Lateral Stress in Cohesionless Soils

Description of data

In these applications, the data obtained from the experiments done by Saglamet and Ozer. In these experimental studies, they performed odometer tests to obtain the lateral stress occurred in the sand samples against to vertical stress. The physical characteristics of the data are given in Table 5. In this table, $D_60$, $D_10$, $\gamma$, $\sigma_v$ and $\sigma_h$ denote the relative density, unit weight, the diameter of $D_60$ and $D_10$, vertical stress and lateral stress, respectively. The statistical properties of the data are represented in Table 6. All the data have low skewness except the $\gamma$. The 176 data set were chosen for calibration of the networks and the remaining 88 data were used for validation, arbitrarily.

Application and results

In these applications also the three-layered ANN was used as in the first case study. The best initial
Fig. 2—Observed and forecasted daily stream-flows in testing period.

<table>
<thead>
<tr>
<th>Place where samples are obtained</th>
<th>Sand Type</th>
<th>$D_R$ (mm)</th>
<th>$\gamma_s$ (g/cm$^3$)</th>
<th>$D_{60}$ (mm)</th>
<th>$D_{10}$ (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sile</td>
<td>Medium</td>
<td>0.33, 0.61, 0.89</td>
<td>2.63</td>
<td>0.389</td>
<td>0.348</td>
</tr>
<tr>
<td>Ayvalık</td>
<td>Medium</td>
<td>0.33, 0.64, 0.86</td>
<td>2.64</td>
<td>0.630</td>
<td>0.470</td>
</tr>
<tr>
<td>Sile</td>
<td>Coarse</td>
<td>0.61</td>
<td>2.65</td>
<td>0.940</td>
<td>0.820</td>
</tr>
<tr>
<td>Yaliköy</td>
<td>Coarse</td>
<td>0.33, 0.83</td>
<td>2.66</td>
<td>0.870</td>
<td>0.860</td>
</tr>
<tr>
<td>Kilyos</td>
<td>Fine</td>
<td>0.47, 0.89</td>
<td>2.72</td>
<td>0.160</td>
<td>0.120</td>
</tr>
</tbody>
</table>

Table 6—Statistical properties of the soil data

<table>
<thead>
<tr>
<th>Data</th>
<th>$x_{\text{mean}}$</th>
<th>$S_x$</th>
<th>$c_v$</th>
<th>$c_{sk}$</th>
<th>$x_{\text{min}}$</th>
<th>$x_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_R$</td>
<td>0.6400</td>
<td>0.2300</td>
<td>0.3590</td>
<td>-0.2800</td>
<td>0.3300</td>
<td>0.8900</td>
</tr>
<tr>
<td>$\sigma_s$</td>
<td>4.9012</td>
<td>2.7087</td>
<td>0.5526</td>
<td>0.1082</td>
<td>0.3096</td>
<td>9.6850</td>
</tr>
<tr>
<td>$\gamma_s$</td>
<td>2.6673</td>
<td>0.0340</td>
<td>0.0127</td>
<td>0.7295</td>
<td>2.6300</td>
<td>2.7200</td>
</tr>
<tr>
<td>$D_{60}$</td>
<td>0.5573</td>
<td>0.2897</td>
<td>0.5190</td>
<td>-0.2625</td>
<td>0.1600</td>
<td>0.9400</td>
</tr>
<tr>
<td>$D_{10}$</td>
<td>0.4906</td>
<td>0.2934</td>
<td>0.5980</td>
<td>0.0863</td>
<td>0.1200</td>
<td>0.8600</td>
</tr>
<tr>
<td>$\sigma_h$</td>
<td>2.3926</td>
<td>1.3572</td>
<td>0.5670</td>
<td>0.2310</td>
<td>0.1304</td>
<td>5.4400</td>
</tr>
</tbody>
</table>
weights were selected after fifty trials. Here also the hidden node numbers were determined using trial-error method. The variables $D_R$, $\gamma_s$, $D_{60}$, $D_{10}$ and $\sigma_v$ were given to networks as inputs to find the output, lateral stress ($\sigma_h$). The networks were trained using LM, CGF and RB algorithms. Table 7 shows the training results of each network. As can be seen from the table, the LM has the smallest time (6 s) for training and the lowest training performances (MARE = 2.29 kg/cm², MSE = 0.0033 kg²/cm⁴). The testing results of the networks are given in Table 8. The optimum hidden layer node number was found as 13.
Table 7—Training results of each algorithm

<table>
<thead>
<tr>
<th>Algorithm used for training</th>
<th>CPU time (s)</th>
<th>Number of epochs</th>
<th>MARE (kg/cm²)</th>
<th>MSE (kg²/cm⁴)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>6</td>
<td>50</td>
<td>2.29</td>
<td>0.0033</td>
</tr>
<tr>
<td>CGF</td>
<td>28</td>
<td>554</td>
<td>2.98</td>
<td>0.0035</td>
</tr>
<tr>
<td>RB</td>
<td>35</td>
<td>2000</td>
<td>2.94</td>
<td>0.0037</td>
</tr>
</tbody>
</table>

Table 8—Testing results of each algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Nodes in hidden layer</th>
<th>MARE (kg/cm²)</th>
<th>MSE (kg²/cm⁴)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>7</td>
<td>4.13</td>
<td>0.0123</td>
</tr>
<tr>
<td>CGF</td>
<td>13</td>
<td>4.27</td>
<td>0.0120</td>
</tr>
<tr>
<td>RB</td>
<td>7</td>
<td>4.13</td>
<td>0.0118</td>
</tr>
</tbody>
</table>

Table 9—Statistics of the ANOVA and t-test

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>F-Statistic</th>
<th>Resultant Significance Level</th>
<th>t-Statistic</th>
<th>Resultant Significance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>0.081</td>
<td>0.776</td>
<td>0.285</td>
<td>0.388</td>
</tr>
<tr>
<td>CGF</td>
<td>0.062</td>
<td>0.804</td>
<td>0.249</td>
<td>0.401</td>
</tr>
<tr>
<td>RB</td>
<td>0.0781</td>
<td>0.780</td>
<td>0.279</td>
<td>0.390</td>
</tr>
</tbody>
</table>

for CGF algorithm and 7 for RB and LM algorithms. Unlike the training period, the RB has the lowest MSE (0.0118 kg²/cm⁴). The LM and the RB algorithms have the same MARE (4.13 kg/cm²) that is lower than that of the CGF algorithm.

The testing results are shown in Fig. 3. As can be seen from the figure all the networks estimates are close to corresponding observed values. It can be seen from the best-fit straight line equations and $R^2$ values that the LM and RB algorithms have the performances close to the each other and slightly better than those of CGF. This is also confirmed by the MARE values in Table 8. The CGF has the 84 estimates lower than 10% relative error in the testing period while the RB and LM have the 82 and 83 estimates lower than 10% error, respectively. However, while the CGF and RB have 63 and 61 estimates lower than 5% relative error, the LM has the 65 estimates lower than 5% error. The statistics of the ANOVA and t-test are given in Table 9. When considering these test results, the CGF algorithm seems to be more robust than the other algorithms in estimation of lateral stress. The RB also seems to be better than the LM.

Conclusions

This study indicated that the forecasting of streamflow and the estimation of lateral stress, are possible through the use of LM-, CGF- and RB-based neural networks. The algorithm of Levenberg-Marquardt takes a small fraction of the time taken by the other algorithms for training of the network. The RB algorithm was also found to be faster than the CGF algorithm in training period. Although the LM had the best performance criteria in training, the testing results of the RB were found to be better than those of the LM in both case studies. This shows the consistency of the RB algorithm. Based on the ANOVA and $t$-tests, it is found that the RB and CGF algorithms are the most robust in stream-flow forecasting and lateral stress estimation, respectively. It is very difficult to know which training algorithm will perform the best for a given problem. It depends on many factors, including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network.

References