Artificial Neural Network as a Clinical decision Supporting tool to predict cardiovascular disease

Article · January 2009

CITATIONS
9

READS
78

4 authors, including:

Jayrani Cheeneebash
University of Mauritius
21 PUBLICATIONS **141** CITATIONS
SEE PROFILE

Smita Goorah
University of Mauritius
46 PUBLICATIONS **265** CITATIONS
SEE PROFILE

Trends in Applied Sciences Research 4 (1): 36-46, 2009
ISSN 1819-3579

Artificial Neural Network as a Clinical Decision-Supporting Tool to Predict Cardiovascular Disease

Beatrice Fidele, Jayrani Cheeneebash, Ashvin Gopaul and Smita S.D. Goorah
Faculty of Science, University of Mauritius, Réduit, Mauritius

Abstract: The aim of the study is to use artificial intelligence tools as a clinical decision support in assessing cardiovascular risk in patients. A two-layer neural network using the Levenberg-Marquardt algorithm and the resilient backpropagation have been used in the proposed artificial neural network. It has been shown how this network is efficient in predicting cardiovascular risk in individual patients by using the Long Beach dataset. The use of this new network seems to better address the prediction of cardiovascular disease at an individual level.

Key words: Artificial neural networks, Levenberg-Marquardt algorithm, resilient backpropagation, diagnosis of cardio-vascular problems

INTRODUCTION

Analyzing data for clinical decision support is a task of great importance to help save time of both patients and doctors and to minimize the risk of making wrong diagnoses. With the advent of new technology and the development of a broad variety of computerized analytical techniques, this process is being done by computers rather than by human intervention. In the past a number of algorithms for cardiovascular risk assessment have been proposed to the medical community (Wilson *et al*., 1998; Assmann *et al*., 2002; Mennotti *et al*., 2000; Mennotti *et al*., 2002; Conroy *et al*., 2003). These algorithms that were used were mainly based from statistical analyses performed on longitudinal study cohorts. There are some drawbacks in considering the algorithms presented by classical statistical approach mainly in dealing with nonlinear and complex data that arise in analyzing the heart disease data set (Enzo, 2006). The drawback is the inability to capture the disease complexity and the process dynamics. In this study we use a data-mining technique, namely Artificial Neural Network (ANN) to diagnose whether a patient has heart disease. The analysis will help as clinical decision support, that is, it will confirm the presence or absence of heart disease for a patient. The novelty in this study is the new network that is presented using the Levenberg-Marquardt Algorithm and the resilient backpropagation. The analysis shows that our ANN is fast and reliable in predicting. Since, ANNs are able to handle a very high number of variables at the same time and can furthermore capture the nonlinearity in a data set; it gives a great advantage over classical statistical techniques. ANNs are more concerned about the actual number of variables rather about their nature. Due to the particular mathematical infrastructure, ANNs can handle large amounts of variables, which constitute the basis for developing recursive algorithms. In our model we first check whether the variables in the data set are highly correlated and this is done by a principal component analysis. The output is then fed into the neural network.

MATERIALS AND METHODS

Due to rapid innovation of computer technology, we use the ANN technique which is a powerful tool for non-linear modeling compared to the classical ARIMA model. Two main advantages are that

Corrresponding Author: Jayrani Cheeneebash, Faculty of Science, University of Mauritius, Réduit, Mauritius

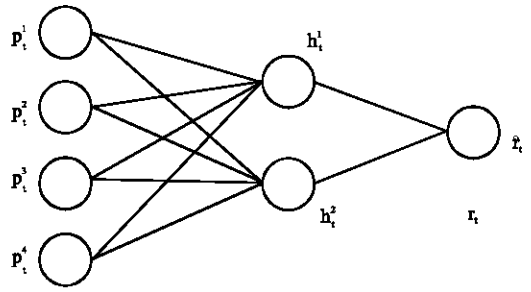

Fig. 1: An artificial neural network model

ANN has the ability to learn a complex nonlinear relationship with limited prior knowledge of the system structure and that it can perform inferences for an unknown combination of input variables. The ANN method imitates the way by which the brain processes information. Given an input vector $p = (p_1, p_2, ..., p_n)^T$, the network produces an output vector $r = (\hat{r}_1, \hat{r}_2, ..., \hat{r}_m)^T$ where, n indicates the number of inputs and m the number of output units. A neural network is typically organized into several layers of nodes. The first layer is the input layer, the number of nodes corresponding to the number of variables and the last layer is the output. The input and output layer can be separated by one or more hidden layers. The nodes in adjacent layers are fully connected. Each neuron receives information from the preceding layer and transmits to the following layer only. The neuron then performs a weighted summation of its inputs; if the sum passes a threshold the neuron transmits, otherwise it remains inactive. An example of a fully connected ANN model with one hidden layer is shown in Fig. 1, where, $p_t^i, i = 1, 2, 3, 4$, are the inputs at time t, $h_t^j, j = 1, 2$, are the hidden outputs. The variables r_t and $\hat{r}_t$ are the actual and ANN model output, respectively. The vector p represents the input to the ANN model where p_t^i is the level of activity at the ith input. Associated with the vector is a series of weight vectors $W_j = (w_{1j}, w_{2j}, ..., w_{nj})$ so that, w_{ij} represents the strength of the connection between the input p_t^i and the unit b_j. There may also be the input bias ϑ_j modulated with the weight w_{0j} associated with the inputs. The total input of the node b_j is the dot product between vectors p and w_j less the weighted bias. It is then passed through a nonlinear activation function to produce the output value of processing unit b_j defined as:

$$b_j = f\left(\sum_{i=1}^{n} x^i w_{ij} - w_{0j}\vartheta_j\right) = f(X_j)$$

The activation function introduces a degree of nonlinearity to the model and prevents the output from reaching very large values that can paralyze ANN models and inhibit training. In this study we choose $f(x) = (e^x - e^{-x})/(e^x + e^{-x})$ as the activation function. The modeling process begins by assigning random values to the weights. The output value of the processing unit is passed on to the output layer. If the output is optimal, the process is halted. Else the weights are adjusted by using an appropriate algorithm, which we choose as the back propagation algorithm for our work. The process continues until an optimal solution is found that is when the output or optimisation error, which is the difference between the actual value and the ANN model output, is minimized.

Here propose a two-layer network based on two algorithms namely the Levenberg-Marquardt algorithm and the resilient backpropagation method. We, first describes the Levenberg-Marquardt algorithm, which is a combination of the method of steepest descent and the Gauss-Newton method. We next study the resilient backpropagation (Riedmiller, 1994) algorithm, which performs supervised batch learning in multi-layer networks.

Levenberg-Marquardt Method (LM)

The Levenberg-Marquardt method is an iterative technique, which locates the minimum of a function expressed as the sum of squares of non-linear real-valued functions. Whenever the current solution is far from the current one, the algorithm behaves like a steepest descent method: slow but with the guarantee to converge. On the other hand, if the current solution is close to the correct one, it becomes a Gauss-Newton method. We next give a short description of the steepest descent and the Gauss-Newton method.

Gradient Descent Method

Gradient descent is an optimisation algorithm. To find a local minimum of a function, the steps are taken proportional to the negative of the gradient (or the approximate gradient) of the function at the current point. Gradient descent is based on the following observation: if the real-valued function F(x) is defined and differentiable in a neighborhood of a point a, then F(x) decreases fastest if one goes from a in the opposite direction of the gradient of F at a denoted by $\nabla F(a)$. It follows that if $b = a-\gamma\nabla F(a)$, for $\gamma>0$, then $F(b)\leq F(a)$. With this observation in mind, one starts with a guess x_0 for a local minimum of F and considers the sequence x_0, x_1, x_2, such that $x_{n+1} = x_n- \gamma\nabla F(x_n)$, $n\geq 0$.

We have $F(x_0)\geq F(x_1)\geq F(x_2)\geq \ldots$, so that the sequence (x_n) converges to the desired local minimum. The value of the step size γ is allowed to change at each iteration.

Gauss-Newton Algorithm

The Gauss-Newton Algorithm is used to solve non-linear least squares problems. It is a modification of the Newton's method that does not use second derivatives. Given m functions f_1, f_2, f_m of n parameters $p_1, p_2,\ldots, p_n$ with $m\geq n$, the aim is to minimize the sum

$$S(p) = \sum_{i=1}^{m}(f_i(p))^2,$$

where, $p = (p_1, p_2, \ldots, p_n)$.

The Gauss-Newton algorithm is an iterative procedure and hence an initial guess for the parameter vector p has to be provided which is denoted as p^0. Subsequent guesses p^k are then obtained by the recurrence relation given by $p^{k+1} = p^k-(J_f(p^k)^TJ_f(p^k))^{-1} J_f(p^k)^T f(p^k)$, where $f = (f_1, f_2, \ldots, f_m)$ and $J_f(p)$ is the Jacobian of f at p. The matrix inverse is never computed explicitly in practice. Instead, we use $p^{k+1} = p^k+\delta^k$ and we find the update δ^k by solving the linear system $J_f(p^k)^T J_f(p^k)\delta^k = -J_f(p^k)^T f(p^k)$.

The Levenberg-Marquardt Algorithm

We assume f is a functional relation, which maps a parameter vector $p\in R^m$ to an estimated measurement vector $\hat{x} = f(p)$, $\hat{x}\in R^n$. An initial parameter estimate p^0 and a measured vector x are provided and the aim is to find the vector p^+ that will best satisfy the function relation f, that is, a function that minimises the squared distance $\varepsilon^T\varepsilon$ with $\varepsilon = x - \hat{x}$. The Levenberg-Marquardt algorithm (Lourakis Manolis, 2005) is an iterative algorithm, which is based on linear approximation to the neighborhood of p.

The method, initiated at the starting point p_0, produces a series of vectors $p_1, p_2, \ldots$ that converge towards a local minimiser p^+ for f. At each iteration, we have to look for the δ_p that minimises the quantity $\|x-f(p+\delta_p)\|\approx\|x-f(p)-J\,\delta_p\|=\|\varepsilon-J\,\delta_p\|$. The sought δ_p is therefore, the solution to a linear least-squares problem: we reach the minimum when $J\,\delta_p-\varepsilon$ is orthogonal to the column space of J. This leads

to $J^T(J\ \delta_p-\varepsilon) = 0$), which yields the Gauss-Newton step δ_p as the solution of the so-called normal equation:

$$J^T J\delta_p = J^T\varepsilon \tag{1}$$

Ignoring the second derivative terms, the matrix J^TJ in the left hand side of Eq. 1 is the approximate. Hessian, that is, an approximation to the matrix of second order derivatives. However, the Levenberg-Marquardt method actually solves a slight variation of Eq. 1, known as the augmented normal equations:

$$N\delta_p = J^T\varepsilon \tag{2}$$

where the off-diagonal elements of N are identical to the corresponding elements of J^TJ and the diagonal elements are given by $N = J^TJ+\mu I$ for some $\mu>0$ and where I is the identity matrix. This strategy of altering the diagonal elements of J^TJ is called damping and μ is called the damping term.

The algorithm, which is given in Appendix A, terminates when at least one of the following conditions is met:

- The magnitude of the gradient of $\varepsilon^T\varepsilon$, that is, $J^T\varepsilon$ in the right-hand side of Eq. 1, drops below a threshold ε_1
- The relative change in the magnitude of δ_p drops below a threshold ε_2
- The error $\varepsilon^T\varepsilon$ drops below a threshold ε_3
- A maximum number of iterations k_{max} is completed

Resilient Backpropagation Algorithm (RP)

Resilient backpropagation (Riedmiller, 1994) is a local adaptive learning scheme, which performs supervised batch learning in multilayer networks. The basic principle is to eliminate the harmful influence of the size of the partial derivative on the weight step. Consequently, consider only the sign of the derivative to indicate the direction of the weight update. The size of the weight change is exclusively determined by a weight-specific, so-called update value $\Delta_{ij}^{(t)}$:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, \text{if } \dfrac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \Delta_{ij}^{(t)}, \text{if } \dfrac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

Where:

$$\frac{\partial E^{(t)}}{\partial w_{ij}}$$

denotes the summed gradient information over all patterns of the data set (batch learning).

The second step of resilient backpropagation learning is to determine the new update-value $\Delta_{ij}(t)$. This procedure is based on a sign-dependent adaptation process.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} \times \Delta_{ij}^{(t-1)}, \text{ if } \dfrac{\partial E^{(t-1)}}{\partial w_{ij}} \times \dfrac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ \eta^{-} \times \Delta_{ij}^{(t-1)}, \text{ if } \dfrac{\partial E^{(t-1)}}{\partial w_{ij}} \times \dfrac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)}, \text{ otherwise,} \\ \text{where} 0 < \eta^{-} < 1 < \eta^{+}. \end{cases} \quad (4)$$

The adaptation rule works as follows: whenever the partial derivative of the corresponding weight w_{ij} changes in sign indicating that the previous update was too large and the algorithm has jumped over a local minimum, the update-value $\Delta_{ij}^{(t)}$ is decreased by the factor η^{-}. If the sign of the derivative remains unchanged, the update-value is slightly increased in order to accelerate convergence in shallow regions. Moreover, if there is a change in sign, there should be no adaptation in the succeeding learning step. In practice, this can be achieved by setting

$$\frac{\partial E^{(t-1)}}{\partial w_{ij}} = 0$$

in the above adaptation rule.

The resilient backpropagation algorithm follows the principle of batch learning or learning by epoch since it tries to adapt its learning process to the topology of the error function. Weight-update and adaptation are performed after the gradient of the whole pattern set is computed.

RESULTS AND DISCUSSION

The heart disease data set problem is a pattern recognition problem. The objective of the network is to decide if an individual has cardio pathologies, based on personal data (age, sex) and the results of medical examinations (e.g., blood pressure, cholesterol, maximum heart rate, etc). The data was obtained from the Long Beach data set (more specifically Cleveland Clinic Foundation data set [Heart Disease Databases]) with fourteen attributes (including the target) and 303 records and the network has been implemented in Matlab 7.0. Table 1 shown an extract of the data set.

Table 1: An extract of the data set showing the records of the patients

Age	Sex	Chest pain type	Blood pressure	Cholesterol	Fasting blood sugar <120	Resting ECG	Max. heart rate	Angina	Peak	Slope	Colored vessels	Thal	Class
60	1	4	130	206	0	2	132	1	2.4	2	2	7	0
49	1	2	130	266	0	0	171	0	0.6	1	0	3	1
64	1	1	110	211	0	2	144	1	1.8	2	0	3	1
63	1	4	130	254	0	2	147	0	1.4	2	1	7	0
53	1	4	140	203	1	2	155	1	3.1	3	0	7	0
58	0	1	150	283	1	2	162	0	1.0	1	0	3	1
58	1	2	120	284	0	2	160	0	1.8	2	0	3	0
58	1	3	132	224	0	2	173	0	3.2	1	2	7	0
63	1	1	145	233	1	2	150	0	2.3	3	0	6	1
67	1	4	160	286	0	2	108	1	1.5	2	3	3	0
67	1	4	120	229	0	2	129	1	2.6	2	2	7	0
37	1	3	130	250	0	0	187	0	3.5	3	0	3	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	1
56	1	2	120	236	0	0	178	0	0.8	1	0	3	1
62	0	4	140	268	0	2	160	0	3.6	3	2	3	0

Before training, it is useful to scale the input and targets so that they always fall within a specified range. One approach is to normalise the mean and standard deviation of the training set. This procedure is implemented in the function prestd in Matlab. It normalises the inputs and targets so that, they will have zero mean and unity standard deviation.

Principal Component Analysis

In some situations, when the input vector is large, the components of the vectors are highly correlated (redundant). It is useful in this situation to reduce the dimension of the input vectors and one way is to perform a principal component analysis. The aim of the principal component analysis is to eliminate those components that contribute the least to the variation in the data set.

The matrix ptrans contains the transformed input vectors. The matrix transMat contains the principal component transformation matrix. After the network has been trained, the matrix should be used to transform any future inputs that are applied to the network. We must note that pn * transMat = ptrans.

For the data set we find that all components account up to 99% of the variation.

Training Session

The next step is to divide the data up into training, validation and test subsets. One fourth of the data has been taken for the validation set, one fourth for the test set and one half for the training set. Now we can create a network and train it. For this purpose, here used a two-layer feed-forward network, with tan-sigmoid transfer function in the hidden layer and linear transfer function in the output layer. First, we start with the resilient backpropagation algorithm:

```
net=newff(minmax(p),[50,1],{'logsig','purelin'},'trainrp');
net.trainParam.show=2;
net.trainParam.epochs=100;
net.trainParam.goal=0;
net.trainParam.lr=0.01;
[net,tr]=train(net,ptr,ttr,[],[],val,test);
```

The performance of the network during the training is shown in Table 2. The training stopped after 22 iterations because the validation error increased. The training, validation and test errors have been plotted to check the progress of the training. The network's performance is shown in Fig. 2 and 3. The result here is reasonable because the test set error and the validation set error have almost similar characteristics and it does not appear that any significant over-fitting has occurred.

Table 2: Performance of the resilient backpropagation method

Epoch No./100	MSE	Gradient 1e-006
0	27.3780	44.66290
2	2.27457	5.50831
4	0.77268	2.05134
6	0.357213	1.00170
8	0.21155	0.14890
10	0.14548	0.511365
12	0.105335	0.115695
14	0.08044771	0.212329
16	0.0631842	0.0995071
18	0.0505347	0.0846935
20	0.0414915	0.102431
22	0.0346487	0.0729705

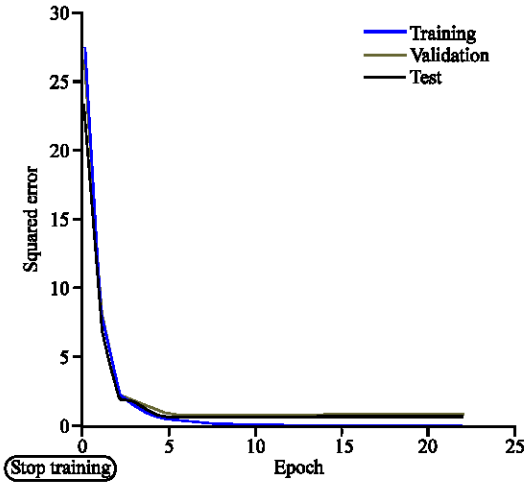

Fig. 2: Minimum square error per epoch with RP algorithm

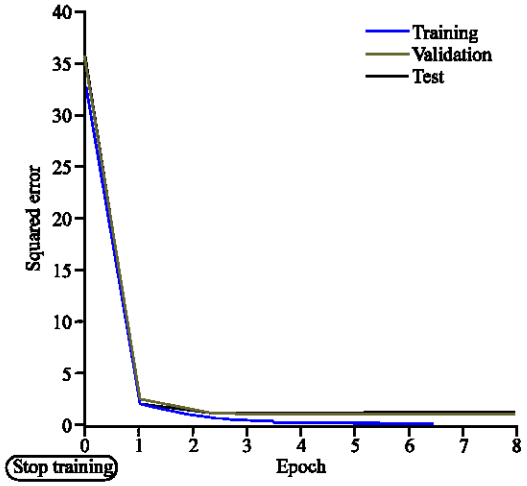

Fig. 3: Minimum square error per epoch using the LM algorithm

Now, we train the same set of data using the Levenberg-Marquardt algorithm. The same commands have been used with the same parameters except for the training function's parameter, which has been set to 'trainlm' (Table 1).

The training stopped after eight iterations since the minimum gradient had been reached (Fig. 2, 3).

For each approach, the technique for calculating the weight changes, so as to minimize the mean square error between the targeted output and the network's output, have been analyzed. When applied to our data, for the resilient backpropagation algorithm, the training started with a mean square error of 27.378 and it reached a minimum of 0.03465 after 22 epochs as shown in Table 2. On the other hand, in Table 3, we find that for the Levenberg-Marquardt algorithm, the training started with a mean square error of 32.7958 and it attained a minimum of $1.18597\times10^{-0.29}$ after eight epochs. With these levels of error, we can consider the results as being reliable.

Table 3: Performance of the Levenberg-Marquardt method

Epoch/100	MSE	Gradient/1e-010
0	32.7958	4012.23
2	0.55615	96.2396
4	0.013799	19.6271
6	3.04231e-008	0.00704829
8	1.18597e-029	4.95044e-013

Table 4: Network's output and targets with the Levenberg-Marquardt and the resilient backpropagation algorithms

Target	Train LM	Difference	Train RP	Difference
0	0.20566	0.20566	-0.06423	-0.06423
1	0.92145	-0.07855	0.96938	-0.03062
1	0.94331	-0.05669	0.96354	-0.03646
0	-0.10144	-0.10144	-0.14915	-0.14915
0	0.020379	0.020379	-0.11029	-0.11029
1	**0.11824**	**-0.88176**	**0.0098**	**-0.9902**
0	0.25587	0.25587	0.19964	0.19964
0	0.22976	0.22976	0.44656	0.44656
1	0.92561	-0.07439	1.0008	0.0008
0	-0.15991	-0.15991	-0.34354	-0.34354
0	0.12718	0.12718	-0.09191	-0.09191
1	**-0.03765**	**-1.03765**	**0.0348**	**-0.9652**
1	1.2102	0.2102	1.017	0.017
1	0.65052	-0.34948	1.0795	0.0795
0	0.20316	0.20316	0.038164	0.038164
1	0.85242	-0.14758	0.52739	-0.47261
1	0.96297	-0.03703	0.7813	-0.2187
1	1.0578	0.0578	1.1283	0.1283
0	0.12602	0.12602	0.1933	0.1933
1	1.049	0.049	0.93916	-0.06084
1	0.99569	-0.00431	1.0108	0.0108
1	1.4273	0.4273	0.97587	-0.02413
1	0.97839	-0.02161	0.96026	-0.03974
1	0.8241	-0.1759	0.76021	-0.23979

Table 4 shows a sample of the network's output compared to the targeted output. When analyzing the network's output, we can see that it has not been limited to 0 and 1 only since we have used the linear transfer function in the output layer. In such a way, it can be analyze the severity of the disease. For example, a patient for whom the network's output is -0.8 is not likely to have heart disease, whereas, a network's output of 1.5 indicates the presence of the disease and it can be considered as being severe. Also, the cases where the doctor may have made wrong diagnoses are in bold.

CONCLUSIONS

In this study, we have considered two algorithms for training the artificial neural network for heart disease database. From this experiment we obtain a well trained network with a relatively good accuracy by using the LM method, which is faster than the RP method. Most of the diagnoses made by the cardiologists were accurate but still there were a few cases where the diagnosis of heart disease might have been wrong. Based on this analysis, cardiologists can reconsider the records of those patients to see if there has been any error in the diagnosis. The same approach may be used in other medical problems where prediction based on subsets of attributes is required on several target attributes at the same time.

APPENDIX A

We present the Levenberg-Marquardt algorithm.

Input: A vector function $f : R^m \rightarrow R^n$ with $n \geq m$, a measurement vector $x \in R^n$ and an initial parameter's estimate $p_0 \in R^m$
Output: A vector $p^+ \in R^m$ minimising $\|x-f(p)\|^2$
Algorithm:
$k = 0$; $v = 2$; $p = p_0$;
$A = J^T J$; $\varepsilon_p = x-f(p)$; $g = J^T \varepsilon_p$;
stop = $(\|g\|_\infty \leq \varepsilon_1)$; $\mu = \tau \times \max_{i=1,\ldots,m}(A_{ii})$;
while (not stop) and ($k<k_{max}$)
    $k = k+1$;
    repeat
        Solve $(A+\mu I)\delta_p = g$;
        If $(\|\delta_p\| \leq \varepsilon_2 \|p\|)$
            Stop = true;
        Else
            $P_{new} = p+\delta_p$;
            $\rho = (\|\varepsilon_p\|^2 - \|x - f(p_{new})\|^2)/(\delta_p^T(\mu\delta_p + g))$;
            if ($\rho>0$)
                $p = p_{new}$;
                $A = J^T J$; $\varepsilon_p = x - f(p)$; $g = J^T \varepsilon_p$;
                stop = $(\|g\|_\infty \leq \varepsilon_1)$ or $(\|\varepsilon_p\|^2 \leq \varepsilon_3)$;
                $\mu = \mu \times \max(\frac{1}{3}, 1-(2\rho-1)^3)$; $v = 2$;
            else
                $\mu = \mu \times v$; $v = 2 \times v$;
            endif
        endif
    until ($\rho>0$) or (stop)
endwhile
$p^+ = p$;

Next, we consider the Resilient Backpropagation algorithm. The minimum (maximum) operator is supposed to deliver the minimum (maximum) of two numbers; the sign operator returns +1 if the argument is positive and -1 if the argument is negative and 0 otherwise.

$$\text{for all } i, j: \Delta_{ij}(t) = \Delta_0,$$

$$\text{for all } i, j: \frac{\partial E}{\partial w_{ij}}(t-1) = 0,$$

Repeat

Compute Gradient $\frac{\partial E}{\partial w}(t)$

For all weights and biases {

$$\text{If } \left(\frac{\partial E}{\partial w_{ij}}(t-1)\times\frac{\partial E}{\partial w_{ij}}(t) > 0\right) \text{ then } \{$$

$$\Delta_{ij}(t) = \text{minimum } (\Delta_{ij}(t\text{-}1)\times\eta^{+}, \Delta_{max})$$

$$\Delta w_{ij}(t) = -\text{ sign } \left(\frac{\partial E}{\partial w_{ij}}(t)\right)\times\Delta_{ij}(t)$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$$

$$\}$$

$$\text{else if } \left(\frac{\partial E}{\partial w_{ij}}(t-1)\times\frac{\partial E}{\partial w_{ij}}(t) < 0\right) \text{ then } \{$$

$$\Delta_{ij}(t) = \text{maximum } (\Delta_{ij}(t-1)\times\eta^{-}, \Delta_{min})$$

$$\frac{\partial E}{\partial w_{ij}}(t-1) = 0$$

$$\}$$

$$\text{else if } \left(\frac{\partial E}{\partial w_{ij}}(t-1)\times\frac{\partial E}{\partial w_{ij}}(t) = 0\right) \text{ then } \{$$

$$\Delta w_{ij}(t) = \text{-sign } \left(\frac{\partial E}{\partial w_{ij}}(t)\right)\times\Delta_{ij}(t)$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$$

$$\}$$

$$\}$$

Until convergence

The algorithm takes two parameters: the initial update-value Δ_0 and a limit for the maximum step size, Δ_{max}. All the update-values are set to an initial value Δ_0 when learning starts. Since, the size of the first weight step is directly determined by Δ_0, it should be chosen according to the initial values of the weights themselves, for example $\Delta_0 = 0.1$ (default setting). The choice of this value is rather uncritical since it is adapted as learning proceeds.

To prevent the weights from becoming too large, the maximum weight step determined by the update-value is limited. The upper bound is set by the second parameter Δ_{max} and its default value is set to $\Delta_{max} = 50.0$. The minimum step size is constantly fixed to $\Delta_{min} = 10^{-6}$.

REFERENCES

Assmann, G., P. Cullen and H. Schulte, 2002. Simple scoring scheme for calculating the risk of acute coronary events based on the 10 year follow up of the prospective cardiovascular Munster (PROCAM) study. Circulation, 105: 310-315.

Conroy, R.M., K. Pyorala, A.P. Fitzgerald, S. Sans and A. Menotti *et al*., 2003. SCORE project group. Estimation of ten-year risk of fatal cardiovascular disease in Europe: The SCORE project. Eur. Heart J., 24: 987-1003.

Enzo, G., 2006. How artificial intelligence tools can be used to assess individual patient risk in cardiovascular disease: Problems with the current methods. BMC Cardiovascular Disorders, 6: 20, 10.1186/1471-2261-6-20.

Lourakis Manolis, I.A., 2005. A brief description of the Levenberg-Marquardt algorithm implemented by levmar. http://www.ics.forth.gr/~lourakis/levmar/levmar.pdf.

Menotti, A., P.E. Puddu and M. Lanti, 2000. Comparison of the framingham risk function-based coronary chart with risk function from an Italian population study. Eur. Heart J., 21: 365-370.

Menotti, A., P.E. Puddu and M. Lanti, 2002. The estimate of cardiovascular risk. Theory, tools and problems. Ann. Ital. Med. Int., 17: 81-94.

Riedmiller, M., 1994. Rprop-Description and Implementation. http://citeseer.ist.psu.edu/riedmiller94rprop.html.

Wilson, P.W., R.B. D'Agostino D. Levy, A.M. Belanger, H. Silbershatz and W.B. Kannel, 1998. Prediction of coronary heart disease using risk factor categories. Circulation, 97: 1837-1847.