A novel hybrid neural learning algorithm using simulated annealing and quasisecant method

JOHN YEARWOOD\textsuperscript{1} ADIL BAGIROV\textsuperscript{1} SATTAR SEIFOLLAHI\textsuperscript{1,2}

\textsuperscript{1}School of Science, Information Technology and Engineering, University of Ballarat, Victoria 3353, Australia
\textsuperscript{2} School of Civil, Environmental and Mining Engineering, University of Adelaide, South Australia 5005, Australia

Emails: j.yearwood@ballarat.edu.au, a.bagirov@ballarat.edu.au, s.seifollahi@ballarat.edu.au

Abstract

In this paper, we propose a hybrid learning algorithm for the single hidden layer feedforward neural networks (SLFNs) for data classification. The proposed hybrid algorithm is a two-phase learning algorithm and is based on the quasisecant and the simulated annealing methods. First, the weights between the hidden layer and the output layer nodes (output layer weights) are adjusted by the quasisecant algorithm. Then the simulated annealing is applied for global attribute weighting. The weights between the input layer and the hidden layer nodes are fixed in advance and are not included in the learning process. The proposed two-phase learning of the network is a novel idea and is different from that of the existing ones. The numerical results on some benchmark data sets are also reported and these results are promising.

Keywords: Classification, Quasisecant method, Simulated annealing, Attribute weighting

1 Introduction

Among different types of artificial neural network (ANN), single hidden layer feedforward neural networks (SLFNs) have become an extremely interesting topic of research because of their high learning ability and robustness. There are two main variations for SLFNs networks, those with additive hidden nodes and those with radial basis function (RBF) hidden nodes. Using both non-linear transfer functions provides the power of non-linearity to the network. Moreover, it has been proven that this class of networks is capable of universal approximation (Girosi et al. 1995).

A learning algorithm is at the heart of a neural based system. Learning can be considered as a weight updating rule of the ANN. Error Back Propagation (EBP), developed by Rumelhart et al., is probably the most cited learning algorithm. EBP is based on the gradient descent minimization method. Also, most of the neural learning algorithms depend on the gradient information of the error surface, which may not always be available or may be expensive to calculate. Moreover, the algorithm may easily be trapped in a local minima (Masters 1995).

One of the alternative learning techniques, that is more attractive, is the use of metaheuristics from global optimization. However, one of the problems though, with most of the global optimization methods is the time complexity of these methods, in particular for very large scale problems. For instance, finding the hidden layer weights in an SLFN, the weights between the input layer and the hidden layer nodes, can be handled by these methods, however when the application size or the number of hidden layer nodes is large, these methods need a further improvement in terms of the time complexity. Another alternative is the use of hybrid methods. The work of Ghosh et al., 1995, suggest different hybrid techniques, in which a local search method in conjunction with an evolutionary learning is used to update the hidden and output layer weights.

The ANN is one of the most popular algorithms used for data classification (Ghosh et al. 1995, Haykin 1999, Huang et al. 2006, Looney 1997). A neural network maps the input attribute vector to the network output consisting of classes. Most of the existing algorithms based on neural networks suffer from one or more drawbacks such as long training time, multiple local minima, high dependence on random initial weights and the need for tuning of parameters such as learning rate and momentum. In a classification process the contribution of attributes may be different. Giving weights to attributes may correct this imbalance and improve classification accuracy.

In this paper, we propose a new hybrid algorithm for the learning of the SLFN for classification. The proposed hybrid learning method is applied for updating the output layer weights and attribute weighting. It is based on the quasisecant algorithm (recently proposed by Bagirov and Ganjehlou, 2010) and the simulated annealing algorithm (Kirkpatrick et al. 1983). The simulated annealing is shown to be efficient for solving global optimization problems with box constraints.

We consider a two-phase learning algorithm. In the first phase of the algorithm, the output layer weights are found by the least square approach using the quasisecant method. In the second phase, the attribute weights are determined using the simulated annealing. The learning procedure here is different from that of existing ones, for instance the extreme learning machine (ELM) algorithm proposed by Huang et al. in 2006. In the ELM algorithm, the hidden layer parameters (hidden layer weights and biases) are randomly assigned and the only unknown parameters that need to be determined are the output linear weights, which can be handled using a least square method. However, the similarity between the
The proposed algorithm and the ELM algorithm is that in both algorithms the hidden parameters are first decided by using random numbers and they are not adjusted during learning process. This process reduces the number of unknown parameters in the network and hence it mitigates the complexity of the problem.

The rest of this paper is organized as follows. Section 2 gives a brief review on the simulated annealing method and then the quasisecant method. In Section 3, we propose a hybrid method, which is based on the optimization methods introduced in the previous section. In Section 4, practical test results of the proposed algorithm by employing some test problems from literature are illustrated. We conclude the paper in section 5 followed by a few directions for future work.

2 Methods used in hybrid algorithm

2.1 The simulated annealing method

The simulated annealing procedure derives its name from the physical process of “annealing” or cooling of heated metals in which many final crystalline configurations which correspond to different energy states are possible depending upon the rate of the associated cooling process. According to (Kirkpatrick et al. 1983), this procedure can be traced to Metropolis who originally attempted to simulate the behavior of an ensemble of atoms in equilibrium at a given temperature. Metropolis constructed a mathematical model of the behavior of such a system that contained a method for minimizing the total energy of the system.

It is a fact that the atoms of a molten metal when cooled to a freezing temperature will tend to assume relative positions in a lattice in such a way as to minimize the potential energy of their mutual forces. Because of the huge number of atoms and resulting possible lattice arrangements (a combinatorial problem), the final derived state will typically correspond to only a local optimum and not a global one. Computationally, simulated annealing has been devised as a general optimization methodology and it can find global minima of general Lipschitz functions. It is not required that the function is smooth, that is, continuously differentiable.

The simulated annealing method consists of two main iterations: outer and inner iterations. In outer iterations the temperature \( T \) is updated. In order to do so we take any initial value \( T_0 \) for temperature and a number \( \alpha \in (0, 1) \) and use the following schedule for temperature: \( T_{k+1} = \alpha T_k, k = 0, 1, 2, \ldots \). In inner iterations we update the solution. Unlike many other optimization algorithms the simulated annealing method may accept not only downhill moves but also uphill moves. In order to generate a new solution in the inner iteration we randomly generate a new point \( x \) and also a uniformly distributed random number \( p \in [0, 1] \). Then we calculate the following number:

\[
\beta = \min (1, \exp((f_{\text{best}} - f(x))/T))
\]

If \( p \leq \beta \) then we accept \( x \) as a new solution, otherwise we repeat inner iterations. \( f_{\text{best}} \) is the best function value obtained by the simulated annealing. The number of inner iterations is restricted by some number provided by the user. If this number is reached then we go to implement the outer iteration.

The simulated annealing is a stochastic method and it can deal with both discrete and continuous variables. It is shown to be efficient for solving global optimization problems with box constraints.

2.2 The quasisecant method

The quasisecant method was introduced in (Bagirov and Ganjehlou 2010). It is a local method for solving nonsmooth, including nonconvex optimization problems. In general, this method is applicable for solving the following unconstrained minimization problem:

\[
\text{minimize } f(x)
\]

where \( x \in \mathbb{R}^n \) and the objective function \( f \) is assumed to be locally Lipschitz. Formally, quasisecants are defined as follows. Let \( S_1 = \{ x \in \mathbb{R}^n : \| x \| = 1 \} \) be the unit sphere. A vector \( v \in \mathbb{R}^n \) is called a quasisecant of the function \( f \) at the point \( x \) in the direction \( g \in S \) with the length \( h > 0 \) if

\[
f(x + h g) - f(x) \leq h \langle v, g \rangle.
\]

Here \( \langle v, g \rangle \) is the inner product of vectors \( v \) and \( g \) in \( \mathbb{R}^n \). The above inequality is called a quasisecant inequality. Quasisecants provide overestimation to the function \( f \) in some neighborhood of a point \( x \). There are many vectors \( v \) satisfying the quasisecant inequality. We consider only those which provide approximation to the function. Subgradient-related quasisecants introduced in (Bagirov and Ganjehlou 2010) provide such approximations and they converge to tangents of the graph of the function \( f \). Any quasisecant is defined with respect to a given direction \( g \in S_1 \) and with given length \( h > 0 \). The choice of \( h \) allows one to compute descent directions with different lengths. Therefore, one can compute descent directions even from some shallow local minimizers using quasisecants. This observation makes the quasisecant method applicable to nonconvex problems and compute a “deep local minimizers”. On the other hand, the quasisecant method uses a bundle of quasisecants at a given point to compute descent directions which makes it similar to the well-known bundle methods in nonsmooth optimization (Frangioni 2002, luksan and Vlcek 1998). Therefore, it is applicable to solve nonsmooth optimization problems. Results presented in (Bagirov and Ganjehlou 2010) demonstrate that the quasisecant method is efficient and robust method for solving nonsmooth, nonconvex optimization problems.

A brief description of the quasisecant method is followed. For more and detailed description of the method, the definitions and also the settings of the algorithms, see (Bagirov and Ganjehlou 2010).

Algorithm 1 Computation of the descent direction.

**Step 1.** Choose any \( g^1 \in S_1 \) and compute a quasisecant \( v^1 = v(x, g^1, h) \) in the direction \( g^1 \). Set \( V_1(x) = \{ v^1 \} \) and \( k = 1 \).

**Step 2.** Compute \( \| v^k \| \leq \min \{ \| v \| : v \in \text{co } V_k(x) \} \).

If \( \| v^k \| \leq \delta \), then stop. Otherwise go to Step 3.

**Step 3.** Compute the search direction by \( g^{k+1} = -\| v^k \|^{-1} v^k \).

**Step 4.** If...
\[ f(x + hg^{k+1}) - f(x) \leq -c_1 h\|v^k\|, \]  
then stop. Otherwise go to Step 5.

Step 5. Compute a quasisecant \( v^{k+1} = v(x, g^{k+1}, h) \) in the direction \( g^{k+1} \), construct the set \( V_{k+1}(x) = \text{co} \{ V_k(x) \cup \{v^{k+1}\} \} \), set \( k = k + 1 \) and go to Step 2.

Algorithm 2 The quasisecant method for finding \((h, \delta)-\)stationary points.

Step 1. Choose any starting point \( x^0 \in \mathbb{R}^n \) and set \( k = 0 \).

Step 2. Apply Algorithm 1 for the computation of the descent direction at \( x = x^k \) for given \( \delta > 0 \) and \( c_1 \in (0, 1) \). This algorithm terminates after a finite number of iterations \( m > 0 \). As a result, we get the set \( V_m(x^k) \) and an element \( v^k \) such that

\[
\|v^k\|^2 = \min \{ \|v\|^2 : v \in V_m(x^k) \}.
\]

Furthermore, either \( \|v^k\| \leq \delta \) or for the search direction \( g^k = -\|v^k\|^{-1} v^k \)

\[
f(x^k + hg^k) - f(x^k) \leq -c_1 h\|v^k\|.
\]  
(4)

Step 3. If

\[
\|v^k\| \leq \delta
\]
then stop. Otherwise go to Step 4.

Step 4. Compute \( x^{k+1} = x^k + \sigma_h g^k \), where \( \sigma_h \) is defined as follows

\[
\sigma_h = \text{arg max} \{ \sigma > 0 : f(\pi) - f(x^k) \leq -c_2 \sigma \|v^k\| \},
\]

where \( \pi = x^k + \sigma_h g^k \). Set \( k = k + 1 \) and go to Step 2.

Let \( \{h_k\}, \{\delta_k\} \) be sequences such that \( h_k \to 0 \) and \( \delta_k \to 0 \) as \( k \to \infty \).

Algorithm 3 The quasisecant method.

Step 1. Choose any starting point \( x^0 \in \mathbb{R}^n \), and set \( k = 0 \).

Step 2. If \( \partial f(x^k) \neq 0 \), then stop. \( \partial f \) stands for subdifferential of \( f \).

Step 3. Apply Algorithm 2 starting from the point \( x^k \) for \( h = h_k \) and \( \delta = \delta_k \). This algorithm terminates after finite many iterations \( M > 0 \), and as a result, it computes \((h_k, \delta_k)-\)stationary point \( x^{k+1} \).

Step 4. Set \( k = k + 1 \) and go to Step 2.

It is proved in (Bagirov and Ganjehlou 2010) that under mild assumptions the sequence of points generated by the quasisecant method converges to stationary point of locally lipschitz continuous functions.

3 The proposed hybrid algorithm

3.1 The network formulation

Let us assume a data set \( D = \{ (\mathbf{x}_s, y_s) | s = 1, \ldots, N \} \) of \( N \) arbitrary samples, where \( \mathbf{x}_s \) is an \( n \)-dimensional vector of decision-making attributes, \( \mathbf{x}_s = [x_{s1}, x_{s2}, \ldots, x_{sn}]^T \), and \( y_s \) is the desired output corresponding to the input \( \mathbf{x}_s \), then the output of an SLFN, with \( H \) number of additive nodes, can be mathematically modeled as:

\[
\sum_{j=1}^{H} \beta_{j}s_j(\mathbf{x}_s) = o_s, \quad s = 1, \ldots, N
\]  
(6)

\( \beta_j, j = 1, \ldots, H, \) is the weight vector connecting the \( j \)th hidden node to the output nodes, \( o_s \) is the network response to the \( sth \) sample and \( \phi_j \) is the \( j \)th node in the hidden layer.

Let us consider sigmoid nodes in the hidden layer, then the function \( \phi_j \) can be written in the following form:

\[
\phi_j(\mathbf{x}_s) = \phi(w_j, \mathbf{x}_s + w_{0j}) = \frac{1}{1 + \exp(-w_j, \mathbf{x}_s - w_{0j})}
\]

where \( w_j = [w_{1j}, w_{2j}, \ldots, w_{nj}]^T \) and \( w_{0j} \) are the learning parameters of the \( j \)th hidden node, \( w_j, \mathbf{x}_s \) denotes the inner product of vectors \( w_j \) and \( \mathbf{x}_s \) in \( \mathbb{R}^n \).

If an SLFN with \( H \) number of hidden nodes can approximate these \( N \) samples with zero error, meaning that \( \sum_{s=1}^{N} \|y_s - o_s\| = 0 \), it then implies that there exist \( \beta_j, w_j \) such that:

\[
\Phi \beta = Y
\]  
(7)

where

\[
\Phi = \begin{bmatrix}
\phi_1(\mathbf{x}_1) & \cdots & \phi_{H}(\mathbf{x}_1) \\
\vdots & \ddots & \vdots \\
\phi_1(\mathbf{x}_N) & \cdots & \phi_{H}(\mathbf{x}_N)
\end{bmatrix}_{N \times H},
\]

\[
\beta = [\beta_1, \ldots, \beta_H]^T
\]

and

\[
Y = [y_1, y_2, \ldots, y_N]^T.
\]

The system (7) is a linear system with respect to the output weights of the network. It can be estimated as the least square problem:

\[
f(\beta) = \sum_{s=1}^{s} \left[ \sum_{k=1}^{K} (y_{ks} - o_{ks})^2 \right]
\]  
(8)

where \( y_{kp} \) is the desired value of the \( k \)th output and the \( sth \) sample, \( o_{kp} \) is the actual value of the \( k \)th output and the \( sth \) sample, \( S \) is the number of samples, and \( K \) is the number of the network outputs. Equation (8), also, can be written in the following matrix form.

\[
f(\beta) = E^TE
\]

where \( E \) is an \( K \times S \) matrix with entries \( e_{ks} = y_{ks} - o_{ks} \), \( k = 1, \ldots, K \), \( s = 1, \ldots, S \).
1 and “1” for samples from class 2, the classifier determines the class label of the input vector $X_s$ as

$$\hat{Y}(X_s) = \text{sign}(a_i) = \text{sign}(\sum_{j=1}^{H} \beta_j \phi_j(X_s)) \quad (9)$$

where the scaler $a_i$ is the network output corresponding to the input vector $X_s$.

### 3.2 Attribute weighting

Good attribute weighting can eliminate the effects of noisy or irrelevant attributes. There are some attribute weighting methods in the literature (Ozsen and Gunes 2009, Wu and Cai 2011). In this section, we propose weights for attributes, in which each attribute has its own power as a weight. The idea of our weighting method is the same as the work of Wu and Cai, for attribute weighting in attribute weighted Naive bayes (AWN). More precisely, assuming the attributes are independent conditionally to the class variable, the AWNB classifier assigns to each sample $X_s$ the class value having the highest conditional probability as:

$$\hat{Y}(X_s) = \arg \max_{y \in Y} P(y) \prod_{i=1}^{n} P(x_{si}|y)^{\xi_i} \quad (10)$$

where $x_{si}$ is the value of the $i$th attribute, $P(y)$ is known as the apriori probability of the class, $P(x_{si}|y)$ are conditional attribute-value probabilities and $\xi_i$ is the weight for the $i$th attribute. In this work, similar to the attribute weighting of Wu and Cai, we consider the weights for each attribute. Then they are updated by a global optimization procedure using the simulated annealing method in the second phase of our algorithm.

If we consider weights for attributes as powers, then, the output of the hidden layer nodes, $\phi$, can be formulated as:

$$\phi_j(X_s) = \phi(\sum_{i=1}^{n} w_{ij} x_{si}^{\xi_i} + w_{0j}), \quad j = 1, \cdots, H \quad (11)$$

where $\xi_i$ is the weight for the $i$th attribute. So, the cost function of (8), by considering these attribute weights, can be considered as a function of two sets of variables, that is, the output layer weights and the weights for attributes. More precisely, the purpose here is minimizing the cost function with respect to the sets of variables so that the attribute weights lie in a hyperbox $[a, b]$. Therefore, the minimization problem of (8) can be reformulated as:

$$\text{minimize} \ f(\beta; \xi) \quad (12)$$

subject to

$$\xi \in [a, b], \quad a, b \in R_{+}^n. \quad (13)$$

Lower bound $a_i$ and upper bound $b_i, i = 1, \cdots, n, \text{ith}$ components of $a$ and $b$ corresponding to the $i$th attribute, are positive and $a_i = \ell$ and $b_i = \xi_i + u$. $\ell$ and $u$ are constants; here we set them 0.1 and 1, respectively.

### 3.3 The learning algorithm

A new heuristic hybrid algorithm for the learning of the SLFN is considered for classification. The proposed algorithm is a new two-phase learning algorithm and differs from that of the existing ones. In the first phase of the algorithm, the output layer weights are adjusted by a local optimization, which here is the quasiscient method. Then, a global optimization method, which here is the simulated annealing, is applied to find the attribute weights. The weights and biases between the input layer and the hidden layer nodes, the hidden layer weights, are fixed in advance and they are not updated during the learning process. The algorithm proceeds until an improvement in the objective function value occurs and a pre-specified number of iteration is not reached. The hybrid algorithm proceeds as follows:

#### Algorithm 4 Hybrid method.

1. **Step 1.** Fix all the hidden layer weights with random numbers uniformly distributed from $[0, 1]$ and initialize the attribute weights, $\xi_i = 1, i = 1, \cdots, n$. Choose any starting point $\beta^0 \in R^H$ and set $a^0 = a, b^0 = b, k := 0$.

2. **Phase 1.** Updating output layer weights

   **Step 2.** Apply the quasiscient method to find a stationary point for the output weights of the network starting from the point $\beta^k$.

3. **Phase 2.** Updating attribute weights

   **Step 3.** Update a hyperbox $[a^{k+1}, b^{k+1}] \subset [a, b]$, where $a$ and $b$ are positive, i.e. $a, b \in R_{+}$.

   **Step 4.** Apply the simulated annealing method to find the attribute weights in order to globally minimize the function value in the hyperbox $[a^{k+1}, b^{k+1}]$.

   **Step 5.** Let $\xi$ be obtained by minimizing of the function $f$ using the simulated annealing on $[a^{k+1}, b^{k+1}]$. If $f(\beta^k; \xi) < f(\beta^k; \xi^k)$ then set $\xi^k = \xi$ and go to Phase 1 (Step 2), otherwise the algorithm terminates.

### 4 Experiments

#### 4.1 Data collections

We use 10 real world data sets to test the proposed algorithm. The first nine data sets used in these experiments can be found in the UCI repository of machine learning databases (Asuncion and Newman 2007), and the last data set is downloadable on the tools page of LIBSVM (Chang and Lin 2001). These data sets have been analyzed more frequently by the current data mining approaches. Another reason for selecting of these data sets were that conventional approaches have analyzed them with variable success. Table 1 shows the brief description of the data sets used.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Attributes</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>10</td>
<td>683</td>
</tr>
<tr>
<td>Congres Voting</td>
<td>16</td>
<td>435</td>
</tr>
<tr>
<td>Credit Approval</td>
<td>14</td>
<td>690</td>
</tr>
<tr>
<td>Diabetes</td>
<td>8</td>
<td>768</td>
</tr>
<tr>
<td>Haberman’s Survival</td>
<td>3</td>
<td>306</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>34</td>
<td>351</td>
</tr>
<tr>
<td>Liver Disorders</td>
<td>5</td>
<td>345</td>
</tr>
<tr>
<td>Phomeone CR</td>
<td>5</td>
<td>5404</td>
</tr>
<tr>
<td>Statlog (Heart)</td>
<td>13</td>
<td>270</td>
</tr>
<tr>
<td>Synguided</td>
<td>21</td>
<td>1284</td>
</tr>
</tbody>
</table>

Table 1: Brief description of the experimental data sets
4.2 Results and discussion

We consider a two-phase learning algorithm, in which the output layer weights are found by the least square approach using the quasi-secant method and then the attribute weights are determined using the simulated annealing. The hidden layer weights of the networks are fixed in advance. The proposed algorithm terminates until no improvement in the objective function value occurs and/or a prespecified number of iteration has not been reached.

In the following table, “ELM” stands for the network with the ELM learning algorithm, proposed by Huang et al. in 2006. In the ELM, the parameters of hidden layer nodes, such as weights and biases between the input and the hidden layer nodes, are randomly assigned. Therefore, the only parameters which need to be determined are the output layer weights. These weights are found by the least square method using the Moore-Penrose generalized inverse (Huang et al. 2006, Rao and Mitra 1971). The activation functions used in the hidden layer are sigmoid functions.

Notation “RBFN” represents the RBF network, i.e. a three-layer feed-forward neural network with RBF kernels in the hidden layer nodes (Abdel at al. 2008, Haykin 1999, Golbabai et al. 2009). In the RBF network, all the hidden layer weights are assigned one oner and other parameters in the hidden layer (the centers and widths of the RBFs) can often be pre-fixed. Here, in “RBFN”, the centers of the RBFs are chosen randomly from the training samples and the jth width, \( \sigma_j \), is defined as the distance between the jth center, \( X_j \), and its nearest center (Abdel at al. 2008):

\[
\sigma_j = \gamma \min_{k \neq j} \{ \| X_j - X_k \| : k = 1, \ldots, H, k \neq j \}
\]

where \( \gamma \) has to be set heuristically (the suggested value is \( \gamma = 1.5 \)). The only unknown output weights, in “RBFN”, are found by the least square method using the Moore-Penrose generalized inverse. Here, the multiquadratic function is used in the hidden layer nodes which is of the form:

\[
\phi(\|X - X_j\|) = (\|X - X_j\|^2 + \sigma_j^2)^{1/2},
\]

where \( \|X-X_j\| \) is the Euclidean distance between the input vector \( X \) and the center \( X_j \).

“QSM” represents the network with considering the first phase of the algorithm, i.e. the quasi-secant method for updating the output layer weights. The attribute weights are not considered in this case. “HYBRID” stands for the network with the proposed hybrid algorithm, in which in the first phase the output layer weights and in the second phase the attribute weights are updated. Here, we used sigmoid function in the hidden layer nodes of the networks “QSM” and “HYBRID”.

Table 2 shows the average accuracy results of the networks “ELM”, “RBFN”, “QSM” and “HYBRID” described above; using 15 nodes in the hidden layer. The first column of the table shows the data set used in our experiments and the subsequent columns show the average accuracies (in percentage) of 5 independent runs. Also, in each run, we use 5-fold cross validation method with random orders in partitioning training and test data sets to have more reliable results. More precisely, each fold contained 20% of the data set randomly selected (without replacement).

According to the results in Table 2, the proposed hybrid method, “HYBRID”, outperforms the “ELM” and “RBFN”; improving almost 4.9%, 2.8% on the averages, respectively. Also, the numerical results show that the quasi-secant, “QSM”, method works well, and it can be a good alternative to update the weights in ANNs. Attribute weighting could not remarkably improve the results obtained by the first phase of the algorithm; improving almost 1.3% on the average. Of course, it should be noted that we repeat the algorithm twice and after that we terminate the algorithm. Moreover, the results obtained by the RBF kernels, i.e. “RBFN”, are more accurate than those of “ELM”; improving almost 2% on the average.

We have coded the “QSM” and “HYBRID” in Fortran and “ELM” and “RBFN” in Matlab. The time complexity of the networks are not included in this work due to different programming languages, however it is clear that the time complexity of the proposed hybrid method is more than the others, which is followed by the “QSM”. Also, it is note that the time complexity of the methods including the proposed methods are less than the standard BP network since the hidden weights in all these networks are fixed in advance.

5 Conclusion

In this paper, we proposed a classifier based on single hidden layer feed-forward neural networks (SLFNs). A novel two-phase learning algorithm was used for learning of the SLFNs. The proposed algorithm uses the quasi-secant and the simulated annealing algorithms. In the first phase, the output layer weights are updated by the quasi-secant method, then the simulated annealing is utilized for attribute weighting. The hidden layer weights are fixed in advance and are not included in the learning process. The proposed algorithm is different from that of the existing ones and also the ELM algorithm. The proposed algorithm is a new general learning idea and also can be extended to multiclass. It is noted that other alternative optimization methods can be used in the hybrid algorithm to update the proposed weights.

We carried out a number of experiments on some different data sets obtained from the UCI repository and also from tools page of LIBSVM. The numerical results show that the proposed algorithm has positive effects on the network performance. If the number of hidden nodes increases beyond a certain threshold, the network performance may degrade because of the network complexity and overlapping. It will not be as bad as other traditional networks since their complexity increases due to increasing the number of the hidden layer weights. Research concerning the complexity is not included in this work and it is future challenging work.
References


