Neurocomputing 248 (2017) 120–126

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

AdaBoost-based artificial neural network learning

Mirza M. Baig [a], Mian.M. Awais [a,*], El-Sayed M. El-Alfy [b]

[a] School of Science and Engineering (SSE), Lahore University of Management Sciences (LUMS), Lahore 54792, Pakistan
[b] Information and Computer Science Department, College of Computer Sciences and Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

ARTICLE INFO

Article history:
Received 27 June 2016
Revised 2 December 2016
Accepted 6 February 2017
Available online 10 March 2017

Keywords:
Artificial neural network
Boostron
Perceptron
Ensemble learning
AdaBoost

ABSTRACT

A boosting-based method of learning a feed-forward artificial neural network (ANN) with a single layer of hidden neurons and a single output neuron is presented. Initially, an algorithm called Boostron is described that learns a single-layer perceptron using AdaBoost and decision stumps. It is then extended to learn weights of a neural network with a single hidden layer of linear neurons. Finally, a novel method is introduced to incorporate non-linear activation functions in artificial neural network learning. The proposed method uses series representation to approximate non-linearity of activation functions, learns the coefficients of nonlinear terms by AdaBoost. It adapts the network parameters by a layer-wise iterative traversal of neurons and an appropriate reduction of the problem. A detailed performances comparison of various neural network models learned the proposed methods and those learned using the least mean squared learning (LMS) and the resilient back-propagation **(RPROP)** is provided in this paper. Several favorable results are reported for 17 synthetic and real-world datasets with different degrees of difficulties for both binary and multi-class problems.

1. Introduction

The single-layer Perceptron of Rosenblatt [1], as shown in Fig. 1, is a simple mathematical model for classification of patterns. It takes a vector $\bar{x} = [x_0, x_1, x_2, \ldots, x_m]$ of features as input and computes its class by calculating a dot product of $\bar{x}$ with an internally stored weight vector, $\bar{W} = [w_0, w_1, w_2, \ldots, w_m]$. Most commonly, the input component x_0 is permanently set to -1 with weight w_0 representing the magnitude of external bias. The output of a perceptron is computed using some non-linear activation function such as *sign* and can be written as:

$$\hat{y} = sign\left(\bar{W}.\bar{x}^T\right) = sign\left(\sum_{i=0}^{m} w_i.x_i\right) \quad (1)$$

In supervised learning settings, the main aim of a neural network learning algorithm is to deduce an optimal set of synaptic weights from the provided input-output pairs of vectors specifying a desired functional relationship to be modeled. For a neural networks similar to a single-layer perceptron (i.e inputs are directly connected to the output units), a simple learning rule that iteratively adjusts the connection weights so as to minimize the difference between desired and obtained outputs works well. For example, the well-studied perceptron learning algorithm initializes the weight vector to zeros and iteratively modifies these weights for each misclassified training example $(\bar{x}_i, y_i)$ using the perceptron learning rule:

$$\bar{W}_{new} = \bar{W}_{old} + \eta.(y_i - \hat{y}_i).\bar{x}_i \quad (2)$$

where η is a pre-specified constant known as learning rate, y_i is the desired output and $\hat{y}_i$ is the estimated output.

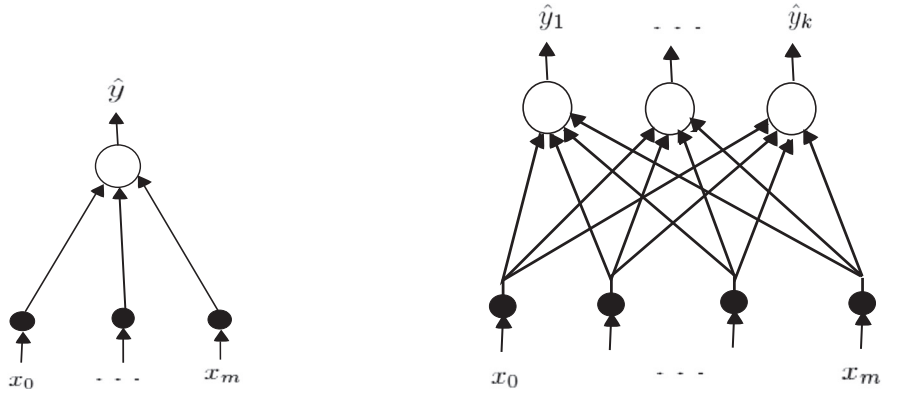
For more complicated networks consisting of several interconnected perceptrons, such as the network shown in Fig. 2, the weight adjustment of hidden neurons posed the main research challenge. However, since the emergence of back-propagation algorithm [2], a number of different learning algorithms have been proposed to adapt the synaptic weights [3–8]. Typically, these methods use an iterative weight update rule to learn an optimal network structure from the training examples by minimizing an appropriate cost function. For example, the back-propagation algorithm [2] uses the following weight update rule to minimize a measure of mean squared error using gradient of error function w.r.t. the weights:

$$\bar{W}_{new} = \bar{W}_{old} - \eta.\frac{\partial E(W)}{\partial W} \quad (3)$$

The gradients for output neurons are computed from the definition of error function whereas the gradients for hidden neurons are computed by propagation of gradients from the output

* Corresponding author.
E-mail addresses: mubasher.baig@nu.edu.pk (M.M. Baig), awais@lums.edu.pk (Mian.M. Awais), alfy@kfupm.edu.sa (E.M. El-Alfy).

http://dx.doi.org/10.1016/j.neucom.2017.02.077
0925-2312/



(a) Single-layer Perceptron with one output (b) Single-layer Perceptron with k outputs

Fig. 1. Typical structure of a single-layer perceptron.

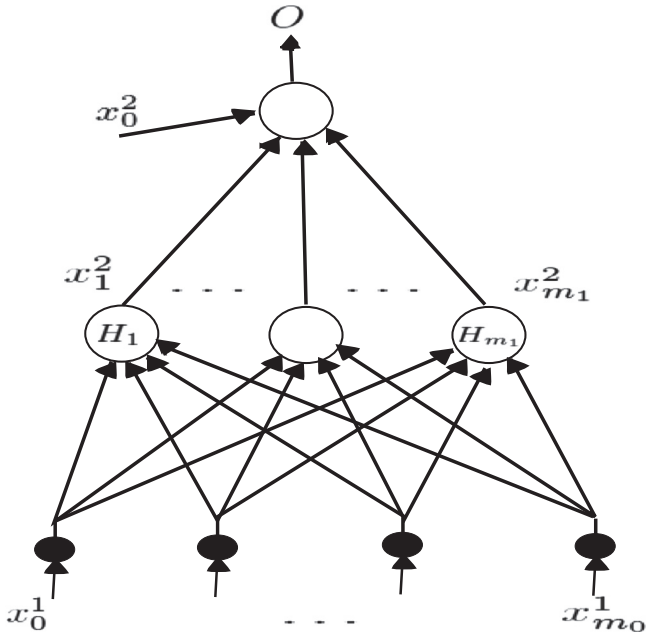


Fig. 2. Feed-forward neural network with a single-hidden layer and a single output unit.

to hidden neurons. Since its introduction, the gradient-based back-propagation learning algorithm has been extensively used to learn models for a very diverse class of learning problems [9–11]

The gradient-based back-propagation algorithm and its variants have a manually selected parameter (η: learning rate) which is trick to work with as for smaller values of η the convergence of the algorithm becomes very slow whereas for larger values the algorithm might become unstable. The problems of converging to a local minimum and over-fitting are also amongst the well-known issues with gradient descend based learning algorithms. Although the mean-squared error minimized by the gradient-based algorithm is suitable for regression problems, it might not work well for classification tasks because most of the natural measures of classification accuracy are typically non-smooth.

AdaBoost [12] is one of the most successful ensemble learning algorithm that iteratively selects several classifier instances by maintaining an adaptive weight distribution over the training examples. AdaBoost forms a linear combination of selected classifier instances to create an overall ensemble. AdaBoost-based ensembles rarely over-fit a solution even if a large number of base classifier instances are used [13] and it minimizes an exponential loss function by fitting a stage-wise additive model [14]. As the minimization of classification error implies an optimization of a non-smooth, non-differentiable cost function which can be best approximated by an exponential loss [15], AdaBoost therefore performs extremely well over a wide range of classification problems.

Motivated by these facts, this paper presents an AdaBoost-based learning method to learn a non-linear feed-forward artificial neural network with a single hidden layer and a single output neuron. The proposed method uses a novel approximation approach to represent any given non-linear function using its' series representation and then uses it to introduce non-linearity into boosting-based ANN learning without introducing significant modification in the method of learning an optimal linear feed-forward ANN. The proposed method consists of three components: (i) a boosting-based perceptron learning algorithm, called Boostron [16], that learns a perceptron without a hidden layer of neurons, (i) an extension of the basic Boostron algorithm to learn a single output feed-forward network of linear neurons [17] and finally (iii) a method of using series representation of the activation function to introduce non-linearity in the neurons which is the main focus of this paper.

The remainder of the paper is organized as follows. Section 2 presents a detailed description of the three components of our proposed method. Details of the experimental settings and the corresponding results are presented in Section 3. Finally, the main findings and are concluded and some limitations of proposed method are highlighted along with the presentation of possible future directions.

2. AdaBoost based neural network learning

This section begins with a brief review of Boostron that transforms the problem of perceptron learning into that of learning a boosting-based ensemble. An extension of Boostron to learn a **linear, feed-forward** perceptron network with a single hidden layer and a single output neuron is then presented. This discussion is succeeded by the introduction of a novel approach that uses the series representation of a given non-linear function to introduce non-linearity into boosting-based learning of an artificial neural network.

2.1. Boostron: boosting based perceptron learning

The AdaBoost algorithm [12,18] is one of the most commonly used ensemble learning algorithms that can be used to construct

a highly accurate classifier ensemble from a moderately accurate learning algorithm [19]. It takes a number of labeled training examples as input and iteratively selects T classifiers by modifying a weight distribution maintained on the training examples. AdaBoost uses a linear combination of the selected classifiers, h_t, to form the ensemble using:

$$H(\bar{x}) = sign\left(\sum_{t=1}^{T} \alpha_t . h_t(\bar{x})\right) \tag{4}$$

where T is the number of base learner instances in the ensemble and α_t is the weight of classifier instance h_t.

A single-node decision trees, commonly referred to as a decision stump, has been frequently used as base classifiers in AdaBoost [18,20] and it makes a decision based on only one of the feature values. For real-valued features a decision stump can be specified by an **if-then-else** rule of the form:

if $x_i \leq U$ **then**
  Class = +ve/−ve
else
  Class = −ve/+ve
end if

For an instance vector $\bar{x} \in R^m$ the above decision stump can be converted into an equivalent classifier by using the inner product defined on R^m. For example, in the case where a $+1$ label is assigned if $x_i \leq U$ and a -1 label otherwise, we can create a classifier equivalent to the decision stump as:

$$s(\bar{x}_i) = -\left(\bar{w}.\bar{x}^T - U\right) \tag{5}$$

In Eq. (5), all components of the weight vector $\bar{w} = [w_1, w_2, ..., w_m]$ are 0 except one of the components w_j. The sign of $s(\bar{x})$ is the classification decision and its magnitude can be regarded as the confidence of prediction. Such a classifier can be represented as a single dot product by representing the instance $\bar{x}$ and the vector $\bar{w}$ using the homogeneous coordinates:

$$s(\bar{x}) = \bar{W}.\bar{X}^T \tag{6}$$

where the vector $\bar{X} = [-1, x_1, x_2, \ldots, x_m]$ is obtained from the instance $\bar{x}$ by adding a -1 as the (1)st component and the vector $\bar{W} = -[U, w_1, w_2, ..., w_m]$ is obtained from $\bar{w}$ by adding U as the (1)st component. Boostron [16] uses this homogeneous representation of a decision stump along with AdaBoost to form a boosted classifier of the form

$$H(\bar{x}) = sign\left(\sum_{t=1}^{T} \alpha_t . h_t(\bar{X})\right) = sign\left(\sum_{t=1}^{T} \alpha_t . \left(\bar{W}_t . \bar{X}^T\right)\right) \tag{7}$$

which by using simple arithmetic manipulation can be written as,

$$H(\bar{x}) = sign\left(\bar{W}.\bar{X}^T\right) \tag{8}$$

where the $(m+1)-$dimensional vector, $\bar{W} = \sum_{t=1}^{T}\left(\alpha_t.\bar{W}_t\right) = [w_0, w_1, \ldots, w_m]$, is the weighted sum of the selected decision stump weight vectors.

The classifier given by Eq. (8) is equivalent to a perceptron as given in Eq. (1). Hence, the new representation of decision stumps using homogeneous coordinates when used as base classifier in AdaBoost learns a linear perceptron. An extension of the Boostron learning algorithm is described in the following subsections to learn the parameters of a linear ANN with a single hidden layer and a single output neuron. The proposed extension uses a transformed set of examples and a layer-wise iterative traversal of neurons in order to tune the connection weights of an ANN.

2.2. Beyond a single-node perceptron learning

To present the proposed method, it is assumed that the inputs of a neuron at layer l are denoted by $x_0^l, x_1^l, ..., x_k^l, \ldots, x_m^l$ respectively where x_0^l is permanently set to -1 and and the corresponding connection weight represents the bias term. In this notation, the superscript denotes the layer number and the subscript denotes the input feature number where m is the total number of neurons in the *previous* layer (i.e. layer $l-1$). The corresponding weights of the jth neuron at layer l are denoted by $w_{j0}^l, w_{j1}^l, \ldots, w_{jk}^l, ..., w_{jm}^l$ where the weight of kth input from the previous layer to jth neuron in the present layer is denoted by w_{jk}^l for $k \in \{1, ..., m\}$.

A two-layer feed-forward neural network with a set of m_0 input neurons $\{I_1, \ldots, I_{m_0}\}$ at layer 0, m_1 hidden neurons $\{H_1, \ldots, H_{m_1}\}$ at layer 1 and a single output neuron O at layer 2 is shown in Fig. 2. If f^l denotes the activation function used at layer l then the output, O, of the neural network shown in Fig. 2 is computed as follows:

$$O = f^2\left(\sum_{k=0}^{m_1} w_{1k}^2 . x_k^2\right) \tag{9}$$

where f^2 denotes the activation function used at layer 2, x_k^2 is the output of the hidden neuron H_k for k =1 to m_1, and x_0^2 is the external bias.

Since each neuron in a single hidden-layer neural network is either an output or a hidden neuron therefore the proposed algorithm uses two reductions:

- Learning an output neuron is reduced to that of perceptron learning.
- Learning a hidden neuron is reduced to that of perceptron learning.

These reductions are iteratively used to learn weights of each neuron in a given neural network. The details of each is explained in the following subsections.

2.2.1. Learning weights of output neuron

The problem of learning weights of an output neuron is reduced into that of of learning a perceptron by transforming the training examples. Each training example $(\bar{x}_i, y_i)$ is transformed into a new training example $(\bar{x}_i^2, y_i)$ by computing the output of hidden layer neurons. For instance, in the neural network of Fig. 2 with m_1 hidden neurons in a single hidden-layer each training instance $\bar{x}_i \in R^{m_0}$ is mapped to a new training instance $\bar{x}_i^2 \in R^{m_1}$ by using the hidden-layer neurons. In this mapping each component of the mapped instance $\bar{x}_i^2 \in R^{m_1}$ corresponds to exactly one of the hidden neuron outputs. After mapping the examples into R^{m_1}, the Boostron algorithm, as described earlier, can be directly used to learn the weights of output neuron using the transformed training examples $(\bar{x}_i^2, y_i), i = 1 \ldots N$.

2.2.2. Learning weights of hidden neuron

To learn the weights, $\{w_{j0}^1, w_{j1}^1, \ldots, w_{jm0}^1\}$, of the j^{th} hidden neuron H_j while keeping the rest of the network fixed, Eq. (9) is written as:

$$O = f^2\left(w_{1j}^2 . x_j^2 + \sum_{k=0, k\neq j}^{m_1} w_{1k}^2 . x_k^2\right) \tag{10}$$

Here, the term x_j^2 is the output of the hidden neuron H_j and can be written as a combination of the inputs to layer 1 and the weights

of the neuron H_j as:

$$x_j^2 = f^1\left(\sum_{i=0}^{m_0} w_{ji}^1.x_i^1\right) \tag{11}$$

Substituting this value of x_j^2 in Eq. (10) gives:

$$O = f^2\left(w_{1j}^2.f^1\left(\sum_{i=0}^{m_0} w_{ji}^1.x_i^1\right) + \sum_{k=0,k\neq j}^{m_1} w_{1k}^2.x_k^2\right) \tag{12}$$

When both activation functions, f^1 and f^2, are linear, the above equation can be written as:

$$O = w_{1j}^2.\sum_{i=0}^{m_0} w_{ji}^1.f^2\left(f^1\left(x_i^1\right)\right) + f^2\left(\sum_{k=0,k\neq j}^{m1} w_{1k}^2.x_k^2\right) \tag{13}$$

If $C = f_1^2\left(\sum_{k=0,k\neq j}^{m_1} w_{1k}^2.x_k^2\right)$ denotes the output contribution of all hidden neurons other than the neuron H_j and $\mathbf{X_i^1} = f^2\left(f^1\left(x_i^1\right)\right)$ denotes the inputs transformed using the activation functions, Eq. (13) can be written as:

$$O = w_{1j}^2.\sum_{i=0}^{m_0} w_{ji}^1.\mathbf{X_i^1} + C \tag{14}$$

A method of learning the weights of the hidden neuron H_j can be obtained by ignoring effect of the constant term C and the effect of magnitude of scale term w_{1j}^2 on the overall output by rewriting Eq. (14) as:

$$O = sgn(w_{1j}^2).\sum_{i=0}^{m_0} w_{1i}^1.\mathbf{X_i^1} \tag{15}$$

As the form of this equation is exactly equivalent to the computation of a perceptron, therefore we can use the Boostron algorithm to learn the required weights, $w_{ji}^1, i = 0, 1, \ldots, m_0$, of a hidden neuron.

Algorithm 1 uses the above reductions and outlines a method of iterating over the neurons of a linear feed-forward neural network to learn its weights. The algorithm randomly initializes all weights in the interval (0, 1) and assigns a randomly selected subset of features to each hidden-layer neuron so that the hidden neuron uses only these features to compute its output. This random assignment of overlapping feature subsets to neurons causes each hidden neuron to use a different segment of the feature space for learning. After this initialization, the algorithm iterates between the hidden layer and the output layer neurons in order to learn the complete neural network.

At the hidden layer, the algorithm iterates over the hidden neurons and computes their weights at steps 5–6 by using the transformed training examples computed at step 4. Weights of each hidden neuron are computed using the Boostron algorithm while keeping the weights of all remaining neurons fixed. These hidden neuron weights are then used to transform the training examples $(\bar{x}_i, y_i), i = 1 \ldots N$ into new training examples $(\bar{x}_i^2, y_i), i = 1 \ldots N$ which are subsequently used to learn the output neuron using Boostron. This whole process is repeated a number of times specified by the input parameter P.

This method of learning a feed-forward neural network works only if all the activation functions used by neurons in a given neural network are linear. Without the linearity assumption, the transformation used by the hidden neuron breaks and the method is no more applicable. Since the extended network outputs a sum of linear classifiers, the resultant decision boundary is still a hyperplane. To mitigate this limitation of the existing technique a series based solution to represent non-linearity is introduced so that the method described earlier can be directly used to learn a non-linear feed-forward network. A detailed discussion of this novel function approximation approach using a series based solution is given below.

Algorithm 1: Algorithm to learn a linear feed-forward ANN using AdaBoost.

Require: Training examples $(\bar{x}_1, y_1) \ldots (\bar{x}_N, y_N)$ where
$\bar{x}_i$ is a training instance and $y_i \in \{-1, +1\}$ the corresponding class label
P is the number of iterations over ANN layers

1: Randomly initialize all weights in the range (0, 1)
2: Randomly assign features to each hidden neuron.
3: **for** j = 1 **to** P **do**
4: Compute transformed training examples $\left(\bar{X}_i, y_i\right), i = 1, 2, \ldots, N$where $\bar{X}_i = [X_0^1, X_1^1, \ldots, X_{m_0}^1]$ and $X_i^1 = f^2\left(f^1\left(x_i^1\right)\right)$
5: **for** each hidden layer neuron H_j **do**
6: Use the Boostron algorithm and the transformed training examples, $\left(\bar{X}_i, y_i\right)$, to learn the weights $w_{j_k}^1$ of H_j where $k = 0, 1, \ldots, m_0$
7: **end for**
8: Compute transformed training examples $\left(\bar{X}_i^2, y_i\right), i = 1, 2, \ldots, N$
where $\bar{X}_k^2 = [x_0^2, x_1^2, \ldots, x_{m_1}^2]$
9: Use the Boostron algorithm and the training examples $\left(\bar{X}_i^2, y_i\right)$, to learn the weights $w_0^2, w_1^2, \ldots, w_{m_1}^2$, of the output neuron O_1
10: **end for**
11: Output the learned ANN weights.

2.3. Incorporating non-linearity into neural network learning

Any smooth infinitely differentiable function can be approximated at any point by a series using one of the several well-studied methods including Taylor/Maclaurin series , Chebyshev polynomial approximation or Minimax approximation [21–23]. Most of the commonly used activation functions, including the *sigmoid* and *tanh*, are differentiable and hence can be approximated by using a series representation. If $Y = W.X$ denotes the result of dot product being carried out inside a neuron before the application of activation function the series representation would typically involve computation of powers Y^k for $k = 1, 2, \ldots$. For example, Maclaurin series representation of $tanh(Y)$ is given by:

$$tanh(Y) = Y - \frac{1}{3}Y^3 + \frac{2}{15}Y^5 - \frac{17}{315}Y^7 \ldots$$

and the sigmoid function is given by:

$$\frac{1}{1+e^{-Y}} = \frac{1}{2} + \frac{1}{4}Y - \frac{1}{48}Y^3 + \frac{1}{480}Y^5 - \frac{17}{80640}Y^7 \ldots$$

In general a power series representation of a function can be written as

$$f(x) = \sum_k \alpha_k.Y^k \tag{16}$$

Such a series representation specifies a fixed coefficient α_k of Y^k such that these coefficients can be used to approximate the value of the function with arbitrary accuracy at any given point Y. The idea of representing an activation function using a series has been extensively used in the past.

To derive the proposed method of attaining effect of non-linear activation function into the extended Boostron algorithm, we use the value of $Y = \sum_{i=1}^{m} w_i.x_i$ in Eq. (16) and consider only the first K powers of the resulting equation to obtain an estimate of $f(x)$ containing $\beta = m + m^2 + \cdots + m^K$ terms

$$f(x) = \alpha_1.x_1 + \cdots + \alpha_m.x_m + \alpha_{m+1}.{x_1}^2 + \alpha_{m+2}.x_1.x_2 + \cdots + \alpha_\beta.{x_m}^\beta \tag{17}$$

Therefore, to introduce non-linearity we extend the inputs by computing all products of degree less than or equal to β and use these extended inputs to learn weights of the perceptron in Algorithm 1 with the identity activation function. From implementation point of view, the extended inputs can either be computed in a global way by extending all examples before step 2 in the Algorithm 1 or in a local way within each hidden neuron by extending its inputs just before step 6 of Algorithm 1. In the global way, non-linearity is incorporated by adding non-linear products as features in all the examples whereas extending the dataset in local way incorporates non-linearity inside the hidden neurons by extending the inputs internally. The uses of fixed weights for non-linear terms is an important difference between the series representation of a given activation function and the method described above. Rather than having fixed weights of non-linear terms, the proposed method uses all products of degree up to β and uses the boosting algorithm to compute their weights.

2.4. Multiclass learning

The algorithm for learning a feed-forward ANN, as presented above, can only be used with networks having a single output neuron and working as binary classifiers. Several simple methods for reducing a multiclass learning problem into a set of problems involving binary classification are in common use. Such methods include the binary encoding of classes using the error-correcting codes [24], the all-pairs approach of Hastie and Tibshirani [25] and the simple approach of one-versus-remaining coding of classes. For each bit in the binary code of classes, a binary classifier is trained and the outputs of all binary classifiers are combined, e.g. using Hamming distance, to produce a final multiclass classifier. Results reported in this paper have been obtained using one-versus-remaining coding of classes (+1 for the class and −1 for the remaining classes). This method reduces a k-class classification problem into k binary-classification problems.

3. Experimental settings and results

A detailed empirical comparison of boosting-based methods of learning a given neural network with that of the corresponding neural network learning algorithms is given in this section. A set of experiments has been carried out to compare the performance of a single neuron trained using Boostron algorithm with that of a neural network trained using perceptron learning rule. Another set of experiments has been conducted to compare the linear multilayer perceptron learning algorithm described in Section 2 with the corresponding network trained using back-propagation learning algorithm [2]. Further, a set of experiments compares the performance of a non-linear neural network trained using the proposed approach with a corresponding non-linear network trained using a combination of sigmoid and linear activation functions. Finally, the last set of experiments compare the performance of the proposed method with the neural network trained by Levenberg–Marquardt (LMA) algorithm [26] for two larger imbalanced datasets.

All the experimental work reported in this paper has been performed on eleven multiclass learning problems and six binary classification datasets mostly from the UCI machine learning repository [27]. These datasets include three simulated binary classification problems: the two-norm, three-norm and ring-norm taken from the work of Leo Breiman [28]. They also include two intrusion detection problems including the dataset adapted from the KDD-Cup 1999 intrusion detection challenge [29] and a recent intrusion detection dataset presented by [30]. The later two datasets pose an interesting learning problem as they have a large number of non-uniformly distributed classes.

The main characteristics of all the datasets including instance space dimension, training/test set sizes and the number of classes are summarized in Table 1. Both synthetic and real-world learning problems of varying complexity are included in these datasets. The datasets cover a wide variety of classification problems including a very small lung cancer dataset that has only 32 training examples, and larger datasets including the Waveform recognition dataset and intrusion detection datasets.

Estimates of error rates has been obtained using 10-fold cross validation for datasets without explicit division into training and test sets. For such learning problems, the paired t-test as described by Dietterich [31] has been used to measure the significance of difference between the performance of algorithms. For 10 fold cross-validation, a test value larger than 2.228 indicates a significant difference between the two algorithms. Whenever a learning problem provided an explicit division into training and test sets, the complete training set has been used to train the network and the complete test set has been used to estimate the test error rate. For such learning problems the statistical significance of difference in the performance of algorithms is measured using McNemar's test [32]. In case of McNemar's test, a value of less than 3.841459 indicates that with 95% confidence the null hypothesis is correct and therefore a value larger than 3.841459 means a significant difference between the two algorithms [31].

For each learning problem, the reported results have been obtained for an ANN having 15 neurons in a single hidden layer and one output neuron. Results for the back-propagation learning algorithms have been obtained by using sigmoid activation function in the hidden layer with 1000 epochs used to train the network. Since non-linear terms increase exponentially fast when larger powers of Y are included in Eq. (16), only the quadratic terms have been used to incorporate non-linearity of activation function in all the reported experiments. While learning a classifier to handle more than two classes, a separate neural network has been trained for each class using one-versus-remaining encoding of classes, therefore k different neural networks are created for a k-class learning problem. For an instance x, the class corresponding to the neural network producing the highest positive output is predicted as the class of x.

3.1. Results and discussions

The first set of results, shown in Table 2, compares the performance of Boostron with perceptron learning rule for nine classification tasks described in Table 1. The last column of the table lists values of statistical test used to compare the two algorithms. A value in bold font indicate a significant difference between the two algorithm. These results show a clear superiority of Boostron over simple perceptron learning rule for adapting weights of a single layer network of linear neurons. It has been found that the performance of Boostron is better than that of perceptron learning rule for eight out of nine datasets and for the four learning problems including: Spambase, Iris, Forest-fire and Glass Boostron learned a significantly improved decision boundary. However, in case of ring-norm dataset the perceptron learning rule converged to a significantly better classifier.

Table 3 provides a comparison of the extended Boostron algorithm with the back-propagation algorithm with linear activation function for nine classification tasks. The error rates of extended Boostron and back-propagation learning algorithm are similar for seven of the tasks and hence the proposed method is comparable to the state-of-the-art back-propagation learning algorithm. However Boostron converged to a significantly better decision surface for the smaller Lung-cancer dataset whereas the back-propagation algorithm found a significantly improved decision boundary for the Glass identification dataset.

Table 1
Description of datasets.

Dataset name	Dataset dimension	Training instances	Test instances	Total classes	Error estimate
Balance scale	4	625		2	Cross validation
Spambase	57	4601		2	Cross validation
Two norm	20	2000	2000	2	Training/test
Three norm	20	1000	2000	2	Training/test
Ring norm	20	2479	2442	2	Training/test
Ionosphere	31	569		2	Cross validation
Iris	4	150		3	Cross validation
Forest fire	5	500		4	Cross validation
Glass	10	214		7	Cross validation
Vowels	10	528	372	11	Cross validation
Wine	13	178		3	Cross validation
Waveform	21	5000		3	Cross validation
Segmentation	20	210	2100	8	Training/test
Yeast	8	980	504	10	Training/test
Lung cancer	56	32		3	Cross validation
KDD-CUP 99	41	494021		23	Cross validation
UNSW-NB15	49	175341	82332	10	Training/test

Table 2
Test error rate comparison of Boostron vs. perceptron.

Dataset name	Boostron algorithm	Perceptron learning	Difference significance
Balance scale	**7.32**	9.73	0.32
Spambase	**24.67**	39.43	**8.46**
Two norm	**2.65**	5.0	0.21
Three norm	**29.05**	35.7	1.15
Ring norm	46.71	**31.58**	**2.41**
Iris	**7.66**	40.27	**6.43**
Forest fire	**17.73**	24.24	**2.68**
Glass	**18.82**	39.52	**5.84**
Lung cancer	26.67	**18.33**	1.1

Table 3
Test error rate comparison of extended Boostron vs linear back-propagation.

Dataset name	Extended Boostron	Back propagation	Difference significance
Balance scale	4.82	**4.79**	0.01
Spambase	**10.58**	11.45	0.13
Two norm	2.15	**2.05**	0.03
Three norm	18.25	**17.4**	0.21
Ring norm	**22.29**	24.53	0.24
Iris	**5.33**	7.33	0.11
Forest fire	**13.6**	18.4	1.1
Glass	18.39	**10.32**	**2.31**
Lung cancer	**26.67**	41.47	**2.79**

Table 4
Boosting based ANN learning vs back-propagation (LMA algorithm).

	Proposed extended Boostron		Back-propagation LMA algorithm		
Dataset name	Training error	Test error	Training error	Test error	Difference significance
Balance scale	0.07	**0.32**	0.48	3.48	**3.8**
Two norm	3.3	**3.7**	0.55	4.05	0.4
Ionosphere	7.59	**11.78**	0	14	0.8
Wine	27.82	29.38	1.5	**15.75**	**4.3**
Waveform	13.94	**25.39**	4.51	29.03	**6.84**
Segmentation	25.48	**32.19**	22.38	36.48	**2.45**
Yeast	30.2	**44.25**	11.02	60.12	**22.5**
Lung cancer	0	**13.33**	0	35	**3.33**

Table 5
Percentage test error rate comparison.

Dataset name	Proposed extended Boostron	RPROP	LMA
KDD-CUP 99	24.76	33.32	29.21
UNSW-NB15	42.15	51.75	53.12

A comparison of Extended Boostron with the simple Boostron can also be made from Tables 2 and 3. This comparison makes sense as the extended multilayer version of Boostron also learns a linear classifier like the single layer version of the algorithm. It is apparent from this comparison that the extended Boostron learns a significantly improved linear decision-boundary.

The next set of results, shown in Table 4, compares the performance of boosting-based ANN learning algorithm with the standard back-propagation(LMA) learning algorithm. These results have been obtained by using the global way of incorporating non-linear activation function as proposed in Section 2. For the back-propagation algorithm, sigmoid activation function has been used at the single hidden layer while the linear activation function has been used at the output layer.

The last set of results, shown in Table 5, presents the test error rate of proposed method, Levenberg–Marquardt Algorithm of learning an ANN [26], and the RPROP (resilient back-propagation) method [33]. In this experiment the hidden layer consisted of 20 neurons and the output layer contained a single neuron for the three learning algorithms. The artificial neural network classifier trained using the proposed method has better test error rate than the two networks trained using the MATLAB implementation of the two remaining algorithms.

It is apparent from the presented results that the introduction of non-linearity has resulted in a significantly improved decision surface achieved by the proposed method. Secondly, the presented results indicate that the boosting-based neural network learning converged to a significantly better decision surface than the default LMS learning algorithm for most of the classification tasks. It is also interesting to note that although the boosting-based method had a relatively higher training error rate for most of the learning tasks, it had lower test error rates. The last set of results also compared the proposed method with resilient back-propagation learning algorithm and once again the proposed method outperformed the remaining two algorithms for this large scale and highly skewed learning task.

Since the proposed method introduces non-linearity into ANN learning by computing all products of inputs up to a certain degree and uses these as extended inputs features. As the number of additional features introduced grow exponentially with the num-

ber of product terms used therefore the proposed method requires larger training time as compared to the standard learning methods. During the experiments it has been observed that the number of extended inputs becomes intractably large even for a moderate number of input feature terms and hence requires large amount of additional training time. However, this difficulty might be handled by devising a parallel version of decision stump learning and a method of simultaneously updating neuron weights in the hidden layer.

4. Conclusion

This paper described a boosting based method, called Boostron, for adapting weights of a single neuron followed by an extension of the method for learning a linear feed-forward ANN and finally a series based solution to incorporate non-linearity in ANN learning. Boostron uses AdaBoost along with a new representation of decision stumps to learn a linear classifier whereas its extension for learning a linear ANN with a hidden layer of neurons uses a layer-wise traversal of neurons and two reductions to learn the synoptic weights. For each neuron it can be considered a greedy method that minimizes an exponential cost function typically associated with AdaBoost. The method proposed for introducing non-linearity into ANN learning uses products of features as extended inputs to each hidden neuron.

The proposed methods have been empirically tested and compared to the corresponding learning algorithms for several standard binary and multiclass classification tasks with varying degrees of complexity. Datasets used in our experiments included both synthetic as well as real-world learning problems and the reported results reveled the superiority of proposed method over the gradient based back-propagation algorithm for several learning tasks.

References

[1] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386.
[2] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back--propagating errors, Cognitive Model. 5 (3) (1988).
[3] E.M. Johansson, F.U. Dowla, D.M. Goodman, Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method, Int. J. Neural Syst. 2 (04) (1991) 291–301.
[4] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, in: Proceedings of IEEE International Conference on Neural Networks, 1993, pp. 586–591.
[5] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Netw. 5 (6) (1994) 989–993.
[6] W. Jin, Z.J. Li, L.S. Wei, H. Zhen, The improvements of bp neural network learning algorithm, in: Proceedings of 5th IEEE International Conference on Signal Processing, vol. 3, 2000, pp. 1647–1649.
[7] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of IEEE International Joint Conference on Neural Networks, vol. 2, 2004, pp. 985–990.
[8] D. Karaboga, B. Akay, C. Ozturk, Artificial Bee Colony (ABC) optimization algorithm for training feed-forward neural networks, in: Modeling Decisions for Artificial Intelligence, Springer, 2007, pp. 318–329.
[9] L. Wang, Y. Zeng, T. Chen, Back propagation neural network with adaptive differential evolution algorithm for time series forecasting, Expert Syst. Appl. 42 (2) (2015) 855–863.
[10] J. Yuan, S. Yu, Privacy preserving back-propagation neural network learning made practical with cloud computing, IEEE Trans. Parallel Distrib. Syst. 25 (1) (2014) 212–221.
[11] S. Munawar, M. Nosheen, H.A. Babri, Anomaly detection through NN hybrid learning with data transformation analysis, Int. J. Sci. Eng. Res. 3 (1) (2012) 1–6.
[12] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1) (1997) 119–139.
[13] A.J. Wyner, On Boosting and the Exponential Loss, in: AISTATS, 2003.
[14] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, Ann. Stat. 28 (2) (2000) 337–407.
[15] R.E. Schapire, Explaining AdaBoost, in: Empirical Inference, Springer, 2013, pp. 37–52.
[16] M.M. Baig, M.M. Awais, E.-S.M. El-Alfy, Boostron: boosting based perceptron learning, in: Neural Information Processing, in: Lecture Notes in Computer Science, vol. 8834, Springer, 2014, pp. 199–206.
[17] M.M. Baig, E.-S.M. El-Alfy, M.M. Awais, Learning Rule for Linear Multilayer Feedforward ANN by Boosted Decision Stumps, in: Neural Information Processing, Springer, 2015, pp. 345–353.
[18] R.E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Mach. Learn. 37 (3) (1999) 297–336.
[19] R.E. Schapire, Y. Freund, Boosting: Foundations and Algorithms, MIT press, 2012.
[20] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, 2001, pp. 313–321.
[21] E. Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons, 2010.
[22] D. Braess, W. Hackbusch, Approximation of 1/x by exponential sums in [1,∞), IMA J. Numer. Anal. 25 (4) (2005) 685–697.
[23] E.W. Cheney, G.G. Lorentz, Approximation theory III, Academic Press, 1980.
[24] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J. Artif. Intell. Res. 2 (1) (1995) 263–286.
[25] T. Hastie, R. Tibshirani, Classification by pairwise coupling, Ann. Stat. 26 (2) (1998) 451–471.
[26] H. Gavin, The Levenberg–Marquardt method for nonlinear least squares curve-fitting problems: Duke University 2011.
[27] M. Lichman, UCI machine learning repository, 2016. http://archive.ics.uci.edu/ml.
[28] L.B.J.F.R. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth International Group, 1984.
[29] (KDD Cup 1999 dataset for network-based intrusion detection systems.(1999) Available on: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)
[30] N. Moustafa, J. Slay, The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set, Information Secur. J.: Global Perspective (2016).
[31] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895–1923.
[32] B.S. Everitt, The Analysis of Contingency Tables, CRC Press, 1992.
[33] H. Braun, M. Riedmiller, RPROP: a fast adaptive learning algorithm, in: Proceedings of the International Symposium on Computer and Information Science VII, 1992.

Mubasher Baig PhD candidate at Department of Computer Science, SBASSE, LUMS.

Mian M Awais Professor, Department of Computer Science, SBASSE, LUMS Dr. Awais received his Ph.D. from Imperial College, University of London. Prior to joining LUMS, Dr Awais conducted European Union research and development projects for a UK based SME. His Ph.D. work related to the development of on-line models for parametric estimation of solid fuel-fired industrial boilers. Dr Awais has also conducted research work on a class of iterative methods pertinent to Krylov subspaces for optimization, such as the oblique projection and implicitly restarted model reduction methodologies.

El-Sayed M. El-Alfy is Associate Professor, College of Computer Sciences and Engineering, King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He has accumulated over 25 years of academic and industrial experience. His research interests include Intelligent Systems and Data Analytics, Pattern Recognition and Machine Learning, Networking and Information Security. He has been actively involved in many funded research projects and has published 130+ refereed journal and conference papers, book chapters, and technical reports on various research topics in his field. El-Alfy has founded and coordinated the Intelligent Systems Research Group (ISRG). He has co-translated four textbooks, and co-edited four books and two Journal Special Issues. He contributed in the organization of many world-class international conferences. He is senior member of IEEE; member of IEEE CIS, Egyptian Syndicate of Engineers. He served on the editorial boards of a number of international journals, including IEEE Transactions on Neural Networks and Learning Systems, International Journal on Trust Management in Computing and Communications, Journal of Emerging Technologies in Web Intelligence. He received several awards and participated to various academic committees and chaired the ABET/CAC Accreditation Committee in the Computer Science Department (KFUPM) in 2010.