Neural, Parallel & Scientific Computations, Volume 10, 2002 (157-178)

Intelligent Weather Monitoring Systems Using Connectionist Models

Imran Maqsood*, Muhammad Riaz Khan♣, Ajith Abraham+

*Environmental Systems Engineering Program, Faculty of Engineering, University of Regina, Regina, Saskatchewan S4S 0A2, Canada, E-mail: maqsoodi@uregina.ca

♣Partner Technologies Incorporated, 1155 Park Street, Regina, Saskatchewan S4N 4Y8, Canada, E-mail: khan@pti.sk.ca

+Faculty of Information Technology, School of Business Systems, Monash University, Clayton 3800, Australia, E-mail: ajith.abraham@ieee.org

Abstract

This paper presents a comparative study of different neural network models for forecasting the weather of Vancouver, British Columbia, Canada. For developing the models, we used one year's data comprising of daily maximum and minimum temperature, and wind-speed. We used Multi-Layered Perceptron (MLP) and an Elman Recurrent Neural Network (ERNN), which were trained using the one-step-secant and Levenberg-Marquardt algorithms. To ensure the effectiveness of neurocomputing techniques, we also tested the different connectionist models using a different training and test data set. Our goal is to develop an accurate and reliable predictive model for weather analysis. Radial Basis Function Network (RBFN) exhibits a good universal approximation capability and high learning convergence rate of weights in the hidden and output layers. Experimental results obtained have shown RBFN produced the most accurate forecast model as compared to ERNN and MLP networks.

Key words: Weather forecasting, multi-layered perceptron, Elman recurrent neural network, radial basis function network, Levenberg-Marquardt algorithm, one-step-secant algorithm.

1. Introduction

Weather forecasts provide critical information about future weather. There are various techniques involved in weather forecasting, from relatively simple observation of the sky to highly complex computerized mathematical models. Weather prediction could be one day/one week or a few months ahead [2][18][24]. The accuracy of weather forecasts however, falls significantly beyond a week. Weather forecasting remains a complex business, due to its chaotic and unpredictable nature [13][15]. It remains a process that is neither wholly science nor wholly art. It is known that persons with little or no formal training can develop considerable forecasting skill [1][9]. For example, farmers often are quite capable of making their own short-term forecasts of those meteorological factors that directly influence their livelihood, and a similar statement can be made about pilots, fishermen, mountain climbers, etc. Weather phenomena, usually of a complex nature, have a direct impact on the safety and/or economic stability of such persons. Accurate weather forecast models are important to third world countries, where the entire agriculture depends upon weather [24]. It is thus a major concern to identify any trends for weather parameters to deviate from its periodicity, which would disrupt the economy of the country. This fear has been aggravated due to threat by the global warming and green house effect. The impact of extreme weather phenomena on society is growing more and more costly, causing infrastructure damage, injury and the loss of life. Several artificial intelligence techniques have been used in the past for modeling chaotic behavior of weather [5][6][8][13][14][15][17][19][24]. In recent times, much research has been carried out on the application of artificial intelligence techniques to the weather forecasting problem. Quantitative forecasting is based on extracting patterns from observed past events and extrapolating them into the future, one should expect artificial neural networks (ANN) to be good candidates for this task. In fact, ANN are very well suited for it, for at least two reasons. First, it has been formally demonstrated that ANN are able to approximate

numerically any continuous function to the desired accuracy [26][27]. In this sense, ANN may be seen as multivariate, nonlinear and nonparametric methods [28], and they should be expected to model complex nonlinear relationships much better than the traditional linear models that still form the core of the forecaster's methodology. Secondly, ANN are data-driven methods, in the sense that it is not necessary for the researcher to postulate tentative models and then estimate their parameters. Given a sample of input and output vectors, ANN are able to automatically map the relationship between them; they "learn" this relationship, and store this learning into their parameters. As these two characteristics suggest, ANN should prove to be particularly useful when one has a large amount of data, but little a priori knowledge about the laws that govern the system that generated the data. Several of them use simple feed-forward neural network training methods using backpropagation algorithm. One aim of this research is to develop an accurate weather forecast model so as to minimize the impact of extreme summer and winter weather. To improve the learning capability of the connectionist paradigms, we used second order error information using the one-step-secant and the Levenberg-Marquardt approaches. We also used a radial basis function network, which is also a well-established technique for function approximation [23].

The Radial Basis Function (RBF) network is a popular alternative to the Multi-Layered Perceptron (MLP), which, although is not as well suited to larger applications, can offer advantages over the MLP in some applications [16][21][23]. In Recurrent Neural Networks (RNN), the temporal nature of the data is taken into account. MLP networks are capable of modeling non-linearity and the only way to adapt MLP networks to temporal data is to provide the entire time-series to the network as input at each training cycle. This not only requires networks of immense size, which in turn require a great deal of processing power and time to converge, but also limits the network to fixed-length time series [4][7]. In Section 2 and 3 we present some theoretical background of MLP and RNN followed by learning algorithms in Section 4. RBF networks and an experimentation setup are presented in Sections 5 and 6, respectively, and some conclusions are drawn towards the end.

2. Multi-Layered Perceptron (MLP) Networks

Typical MLP network is arranged in layers of neurons (nodes), where every neuron in a layer computes the sum of its inputs and passes this sum through a nonlinear function (an activation function) as its output. Each neuron has only one output, but this output is multiplied by a weighting factor if it is to be used as an input to another neuron (in a next higher layer). There are no connections among neurons in the same layer. Figure 1 shows a 3-layered MLP network used for weather forecasting.

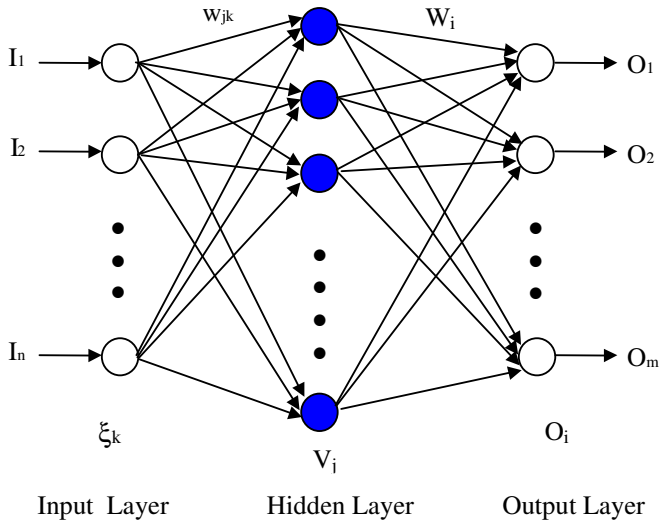


Figure 1: Architecture of multi-layered perceptron network.

The input, hidden and output layers are denoted by *k*, *j*, and *i*, respectively. For a given pattern, μ, a neuron *j* in the hidden layer receives, from a neuron *k* in the input layer, a net input signal as follows:

$$x_j^\mu = \sum_k w_{jk}\xi_k^\mu + b_k \tag{1}$$

where ξ_k^μ is the input signal fed to neuron k in the input layer, and w_{jk} is the connection strength between neuron *j* in the hidden layer and neuron *k* in the input layer, while b_k is a bias connected to the input layer. The bias is an additional input to a neuron that serves to normalize its output, and normally has a constant activation of 1 [10].

The output V_j produced by the *j*th neuron in the hidden layer is related to the activation value for that neuron, by a transfer function $g_h(x)$, which is given as:

$$V_j^\mu = g_h\left(x_j^\mu\right) = g_h\left(\sum_k w_{jk}\xi_k^\mu + b_k\right) \tag{2}$$

A neuron *i* in the output layer receives this signal from the neuron *j* in the hidden layer as input, and similarly produces an output, O_i^μ, which is related to the activation value for that neuron, by a transfer function $g_o(x)$, which is assumed to be the same form of $g_h(x)$ in this paper.

$$O_i^\mu = g_o\left(x_i^\mu\right) = g_o\left(\sum_j W_{ij}V_j^\mu + b_j\right) \tag{3}$$

The biases in the equations, b_k and b_j, can be omitted as they can be considered as an extra input of unit value connected to all units in the network. The final net output for an input pattern μ, $(\mu = 1, \ldots, p)$ can be then described by

$$O_i^\mu = g_o\left(\sum W_{ij}\, g_h\left(\sum w_{jk}\xi_k^\mu\right)\right) \tag{4}$$

Activation functions for the hidden layers are needed to introduce nonlinearity into the network. Without nonlinearity, hidden layers would not make networks more powerful than just simple perceptrons (which do not have any hidden layers, only input and output layers). A composition of linear functions is again a linear function. It is the nonlinearity (i.e., the capability to represent nonlinear functions) that makes MLP networks so powerful. For backpropagation learning, however, it must be differentiable and saturating at both extremes. Sigmoid functions such as the logistic and hyperbolic tangent functions, and the Gaussian function are the most common choices. If the transfer functions were chosen to be linear, then the network would become identical to a linear filter [12].

The steepness of the logistic sigmoid can be modified by a slope parameter σ. The more general sigmoid function (with range between 0 and 1) is given by

$$g(x) = g_h(x) = g_o(x) = \frac{1}{1 + exp(-\sigma x)} \tag{5}$$

with its derivative as

$$g'(x) = \sigma g(x)[1 - g(x)] \tag{6}$$

The slope may be determined such that the sigmoid function achieves a particular desired value for a given value of *x*.

The training of a network by backpropagation [11] involves three stages: the feedforward of the input training pattern, the calculation and backpropagation of the associated error, and the adjustment of the weights. After training, application of the net involves only the computations of the feedforward phase. In order to train the network, input is shown to the net together with the corresponding known output, and if there exists a relation between the input, ξ_k^μ, and the output, O_i^μ, the net learns by adjusting the weights until an optimum set of weights that minimizes the network error is found and the network then converges.

The network error, *E*, which is defined as the sum of the individual errors over a number of samples, is given by

$$E = \frac{1}{2}\sum_{\mu=1}^{p}\sum_{i=1}^{N_{out}}\left(T_i^\mu - O_i^\mu\right)^2 \tag{7}$$

where T_i^μ is the target or desired output $\left(i = 1, \ldots, N_{out}\right)$. Substituting Equation 4 for the net output, O_i^μ, then

$$E = \frac{1}{2}\sum_{\mu=1}^{p}\sum_{i=1}^{N_{out}}\left[T_i^\mu - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk}\xi_k^\mu\right)\right)\right]^2 \tag{8}$$

At the completion of a pass through the entire data set, all the nodes change their weights based on the accumulated derivatives of the error with respect to each weight. These weights change move the weights in such a direction that the error declines most quickly. The standard learning algorithm, which updates the weights can be expressed by the gradient descent rule, which means that each weight, say, w_{pq}, changes by an amount Δw_{pq}, which is proportional to the gradient of the error *E* at the present location.

For the hidden-to-output connections, the gradient descent rule gives the change in weight as

$$W_{ij}^{new} = W_{ij}^{old} + \Delta W_{ij} \tag{9}$$

where

$$\Delta W_{ij} = -\eta\frac{\partial E}{\partial W_{ij}} \tag{10}$$

and η stands for the learning rate. Similarly, the weight change for the input-to-hidden connections, is given by

$$w_{jk}^{new} = w_{jk}^{old} + \Delta w_{jk} \tag{11}$$

where

$$\Delta w_{jk} = -\eta\frac{\partial E}{\partial w_{jk}} = -\eta\sum_\mu \frac{\partial E}{\partial V_j^\mu}\frac{\partial V_j^\mu}{\partial w_{jk}} \tag{12}$$

3. Elman Recurrent Neural Networks (ERNN)

ERNN (also known as partially recurrent neural network) are a subclass of recurrent networks [3][7]. They are multilayer perceptron networks augmented with one or more additional context layers storing output values of one of the layers delayed by one step and used for activating this or some other layer in the next time step, as shown in Figure 2. The Elman network has context units, which store delayed hidden layer values and present these as additional inputs to the network. The Elman network can learn sequences that cannot be learned with other recurrent neural network e.g. with Jordan recurrent neural network (which is a similar architecture with a context layer fed by the output layer) since networks with only output memory

cannot recall inputs that are not reflected in the output. Several training algorithms for calculation of error gradient in general recurrent networks exist.

3.1. Training of Recurrent Neural Networks

In deriving a gradient-based update rule for recurrent networks, we now make network connectivity very unconstrained. Suppose that we have a set of input units, $I = \{x_k(t), 0 < k < m\}$, and a set of other units, $U = \{y_k(t), 0 < k < n\}$, which can be either hidden or output units. To index an arbitrary unit in the network, we can use

$$z_k(t) = \begin{cases} x_k(t) & if \ \ k \in I \\ y_k(t) & if \ \ k \in U \end{cases} \tag{13}$$

Let W be the weight matrix with n rows and $n+m$ columns, where $w_{i,j}$ is the weight to unit i (which is in U) from unit j (which is in I or U). The units compute their activations by first computing the weighted sum of their inputs:

$$net_k(t) = \sum_{I \in U \cup I} w_{kI} z_I(t) \tag{14}$$

where the only new element in the formula is the introduction of the temporal index t. The units then compute some non-linear function of their net input

$$y_k(t+1) = f_k(net_k(t)) \tag{15}$$

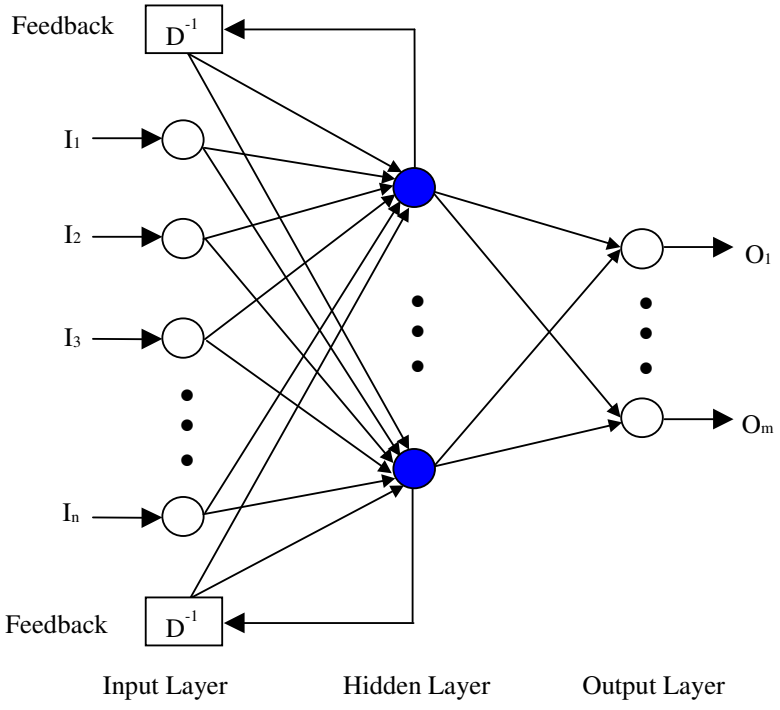


Figure 2: Schematic diagram of 3-layered Elman recurrent neural network.

Usually, both hidden and output units have nonlinear activation functions. Note that external input at time *t* does not influence the output of any unit until time *t+1*. The network is thus a discrete dynamical system.

Some of the units in *U* are output units, for which a target is defined. A target may not be defined for every single input however. For example, if we are presenting a string to the network to be classified as either grammatical or ungrammatical, we may provide a target only for the last symbol in the string. In defining an error over the outputs, therefore, we need to make the error time dependent too, so that it can be undefined (or zero) for an output unit for which no target exists at present. Let *T(t)* be the set of indices *k* in *U* for which there exists a target value $d_k(t)$ at time *t*. We are forced to use the notation d_k instead of *t* here, as *t* now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

and define the error function for a single time step as

$$E(\tau) = \frac{1}{2}\sum_{k \in U} [e_k(\tau)]^2 \tag{17}$$

To minimize the error function is the sum of this error over all past steps of the network

$$E_{total}(t_0, t_1) = \sum_{\tau=t_{0+1}}^{t_1} E(\tau) \tag{18}$$

As the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla w E_{total}(t_0, t+1) = \nabla w E_{total}(t_0, t) + \nabla w E(t+1) \tag{19}$$

As a time series is presented to the network, the values of the gradient can be accumulated, or equivalently, of the weight changes. Thus to keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}} \tag{20}$$

After the network has been presented with the whole series, each weight w_{ij} can be changed by $\sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t)$.

An algorithm can be required that computes

$$-\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} \frac{\partial E(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \tag{21}$$

at each time step *t*. Since $e_k(t)$ is known at all times (the difference between targets and outputs), it is only required to find a way to compute the second factor $\partial y_k(t) / \partial w_{ij}$. From Equations 14 and 15, we obtain:

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k'(net_k(t)) \left[\sum_{I \in U \cup I} w_{kI} \frac{\partial z_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right] \tag{22}$$

where δ_{ik} is the Kronecker delta

$$\delta_{ik} = \begin{cases} 1 & if\ i = k \\ 0 & otherwise \end{cases} \tag{23}$$

As input signals do not depend on the weights in the network,

$$\frac{\partial z_I(t)}{\partial w_{ij}} = 0\ for\ l \in I \tag{24}$$

Equation 24 becomes:

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k^{'}(net_k(t))\left[\sum_{I \in U} w_{kI}\frac{\partial y_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t)\right] \tag{25}$$

This is a recursive equation. That is, if the value of the left hand side for time $t = 0$ is known, we can compute the value for time 1, and use that value to compute the value at time 2, etc. By assuming that starting state ($t = 0$) is independent of the weights, then

$$\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0 \tag{26}$$

These equations hold for all $k \in U, i \in U\ and\ j \in U \cup I$. It is also required to define the values

$$p_{ij}^k(t) = \frac{\partial y_k(t)}{\partial w_{ij}} \tag{27}$$

for every time step t and all appropriate i, j and k. Now start with the initial condition

$$p_{ij}^k(t_0) = 0 \tag{28}$$

and compute at each time step

$$p_{ij}^k(t+1) = f_k^{'}(net_k(t))\left[\sum_{I \in U} w_{kI}\, p_{ij}^I(t) + \delta_{ik} z_j(t)\right] \tag{29}$$

The algorithm then consists of computing the quantities $p_{ij}^k(t)$ at each time step t, using equations 27 and 28, and then using the differences between targets and actual outputs to compute weight changes

$$\Delta w_{ij}(t) = \mu \sum_{k \in U} e_k(t)\, p_{ij}^k(t) \tag{30}$$

and the overall correction to be applied to w_{ij} is given by

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t) \tag{31}$$

Artificial neural networks (ANNs) were designed to mimic the characteristics of the biological neurons in the human brain and nervous system [10]. The network "learns" by adjusting the interconnections (called

weights) between layers. When the network is adequately trained, it is able to generalize relevant output for a set of input data. Learning typically occurs by example through training, where the training algorithm iteratively adjusts the connection weights (synapses). We investigated two learning algorithms for training of MLP and ERNN techniques, namely one-step-secant and the Levenberg Marquardt approaches.

4. Learning Algorithms for MLP and ERNN

4.1. One-Step-Secant Algorithm (OSS)

Quasi-Newton method involves generating a sequence of matrices $G^{(k)}$ that represents increasingly accurate approximations to the inverse Hessian (H^{-1}). Using only the first derivative information of E [11], the updated expression is as follows:

$$G^{(k+1)} = G^{(k)} + \frac{pp^T}{p^T v} - \frac{(G^{(k)}v)v^T G^{(k)}}{v^T G^{(k)} v} + (v^T G^{(k)} v)uu^T \tag{32}$$

where

$$p = w^{(k+1)} - w^{(k)}\ , v = g^{(k+1)} - g^{(k)},\ u = \frac{p}{p^T v} - \frac{G^{(k)}v}{v^T G^{(k)} v} \tag{33}$$

and T represents transpose of a matrix. The problem with this approach is the requirement of computation and storage of the approximate Hessian matrix for every iteration. The One-Step-Secant (OSS) is an approach to bridge the gap between the conjugate gradient algorithm and the quasi-Newton (secant) approach. The OSS approach doesn't store the complete Hessian matrix; it assumes that at each iteration the previous Hessian was the identity matrix. This also has the advantage that the new search direction can be calculated without computing a matrix inverse.

4.2. Levenberg-Marquardt (LM) Algorithm

Levenberg-Marquardt (LM) algorithm is a variation of Newton's method that was designed for minimizing functions that are sums of squares of other nonlinear functions [11]. An important feature of the LM algorithm is its speed (as compatible as Newton's method) and the convergence of steepest descent.

Newton's method for optimizing a performance index F (x) is given by

$$x_{k+1} = x_k - A_k^{-1} g_k \tag{34}$$

where $A_k = \nabla^2 F(x)\big|_{x=x_k}$ is the Hessian matrix (second derivatives) of the performance index at the current values of the weights / biases, and $g_k = \nabla F(x)\big| x = x_k$ the gradient can be written in matrix form

$$\nabla F(x) = 2J^T(x)v(x) \tag{35}$$

where J is the Jacobian matrix. We can approximate the Hessian matrix as

$$\nabla^2 F(x) = 2J^T(x)J(x) \tag{36}$$

substituting (35) and (36) into (34) will give the Guass-Newton method

$$x_{k+1} = x_k - [J^T(x_k)J(x_k)]^{-1} J^T(x_k)v(x_k) \tag{37}$$

One problem with the gaussian method is that the matrix $H = J^T J$ may not be invertible. This can be overcome by using $G = H + \mu I$ modification to the approximate Hessian Matrix. Therefore the eigenvectors of G are the same as the eigenvectors of H. The LM algorithm is given by

$$x_{k+1} = x_k - [J^T(x_k)J(x_k) + \mu_k I]^{-1} J^T(x_k)v(x_k) \tag{38}$$

This algorithm has the very useful feature that as μ_k is increased it approaches the steepest descent algorithm with small learning rate and while μ_k is decreased to zero the algorithm becomes Gauss-Newton. The key step of the LM algorithm is the computation of the Jacobian matrix, which is created by computing the derivatives of the errors.

5. Radial Basis Function Network (RBFN)

RBFN network consists of 3-layers: input layer, hidden layer, and output layer, as shown in Figure 3. The neurons in hidden layer are of local response to its input and known as RBF neurons, while the neurons of the output layer only sum their inputs and are called linear neurons [22].

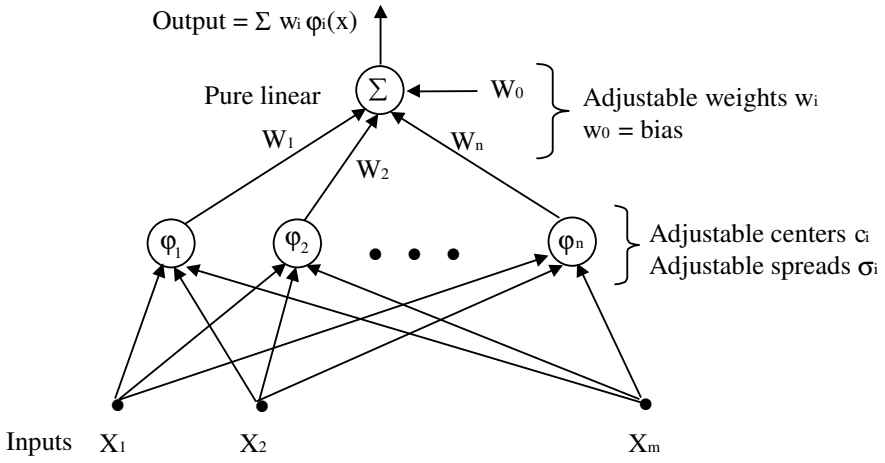


Figure 3: Architecture of radial basis function network.

The RBFN of Figure 3 is normally used to approximate an unknown continuous function $\phi : R^n \to R^n$, which can be described by the affine mapping:

$$u(x) = BK(x,\ p) \tag{39}$$

where $B = [\alpha_{ij}]$ is a $n \times M$ weight matrix of the output layer, $K(x,\ p)$ is the kernel function vector of the RBFN, which consists of the locally receptive functions. Usually, $K(x,p)$ takes one of the forms such as $K(r) = r$ (linear) or $K(r) = \exp(-r^2/2)$ (Gaussian), where r is scaled radius $\|x - \mu_i\| / \sigma_i$. For convenience of notation, we let $p = (\mu_1, ..., \mu_M, \sigma_1, ..., \sigma_M)$ be the parameter vector of the kernel, and further denote $\theta = (B,\ p)$ as the parameter set of the RBFN. For the proposed model, we choose the conventional Gaussian kernel as the activation function of RBF neurons as it displays several desirable properties from the viewpoint of interpolation and regularization theory [12]. Then the kernel function vector $K(x,\ p)$ can be further expressed as

$$K(x,p) = \left[1, exp\left(-(x-\mu_1)^T(x-\mu_1)/\sigma_1^2\right), ..., exp\left(-(x-\mu_M)^T(x-\mu_M)/\sigma_M^2\right)\right]^T \quad (40)$$

Here we let the first component of $K(x,\ p)$ be one for taking the bias into account.

It is well known that neural network training can result in producing weights in undesirable local minima of the criterion function. This problem is particularly serious in recurrent neural networks as well as for MLP with highly nonlinear activation functions, because of their highly nonlinear structure, and it gets worse as the network size increases. This difficulty has motivated many researchers to search for a structure where the output dependence on network weights is less nonlinear. The RBF network has a linear dependence on the output layer weights, and the nonlinearity is introduced only by the cost function for training, which helps to address the problem of local minima. Additionally, this network is inherently well suited for weather prediction, because it naturally uses unsupervised learning to cluster the input data [20][21][23].

5.1. Training of RBF Network

There are two basic methods to train an RBFN in the context of neural networks. One is to jointly optimize all parameters of the network similarly to the training of the MLP. This method usually results in good quality of approximation but also has some drawbacks such as a large amount of computation and a large number of adjustable parameters. Another method is to divide the learning of an RBFN into two steps. The first step is to select all the centers μ in terms of an unsupervised clustering algorithm such as the *K*-means algorithm proposed by Linde et al. (denoted as the LBG algorithm) [25], and choose the radii σ by the *k*-nearest neighbor rule. The second step is to update the weights *B* of the output layer, while keeping the μ and σ fixed. The two-step algorithm has fast convergence rate and small computational burden.

We used a two-step learning algorithm to speed up the learning process of the RBFN. The selection of the centers and radii of RBF neurons can be done naturally in an unsupervised manner, which makes this structure intrinsically well suited for weather prediction. As a result, we adopt below a self-organized learning algorithm for selection of the centers and radii of the RBF in the hidden layer, and a stochastic gradient descent of the contrast function for updating the weights in the output layer. For the selection of the centers of the hidden units, we may use the standard *k*-means clustering algorithm [21]. This algorithm classifies an input vector *x* by assigning it the label most frequently represented among the *k*-nearest neighbor samples. Specifically, it places the centers of RBF neurons in only those regions of the input space, where significant data are present. Let $p_k(n),\ k = 1,..., K$ denotes the centers of RBF neurons at iteration *n*. Then the best matching (winning) center $\hat{k}(x)$ at iteration *n* using the minimum distance Euclidean criterion can be found as follows:

$$\hat{k}(x) = \arg\min_k \|x - p_k(n)\|,\ k = 1,..., K \quad (41)$$

The update rule for the locations of the centers are given by

$$p_k(n+1) = \begin{cases} p_k(n) + \eta_s[x - p_k(n)],\ k = \hat{k}(x) \\ p_k(n),\quad otherwise \end{cases} \quad (42)$$

where η_s denotes the learning rate in interval [0, 1], Once the centers and radii are established, we can make use of the minimization of the contrast function to update the weights of the RBFN.

6. Experimental Setup

Before data are ready to be used as input to the neural networks, they were subjected to some form of pre-processing, which usually intends to make the forecasting problem more manageable. Pre-processing is required to reduce the dimension of the input vector, so as to avoid the "curse of dimensionality" (the exponential growth in the complexity of the problem that results from an increase in the number of dimensions). Pre-processing is also required to clean the data, by removing outliers, missing values or any irregularities, since neural networks are sensitive to such defective data [29][30]. In the normal case, architecture of the connectionist models is determined after a time-consuming trial-and-error procedure. To circumvent this disadvantage, we use a more systematic way of finding good architectures. A sequential network construction [25] is employed to select an appropriate number of hidden neurons for each of the connectionist model considered. First, a network with a small number of hidden neurons is trained. Then a new neuron is added with randomly initialized weights and the network is retrained with changes limited to these new weights. Next all weights in the network are retrained. This procedure was repeated until the number of hidden neurons reaches a preset limit and then substantially reduces the training time in comparison with time needed for training of new networks from scratch. More importantly, it creates a nested set of networks having a monotonously decreasing training error and provides some continuity in the model space, which makes a prediction risk minimum more easily noticeable.

The concept of forecasting-model consists of: (a) proper model selection of the technique that matches with the local requirements, (b) calculation and update of model parameters, which includes the determination of the network parameters and selection of the method to update the constants values as the circumstance varies (seasonal changes), (c) evaluation of the model performance to validate the model using historical data, also the final validation to use the model in real life conditions. The evaluation terms includes accuracy, ease of use and bad/anomalous data detection, (d) update/modification of the model, if the performance is not satisfactory. Due to sudden variation in weather parameters, the model becomes obsolete and inaccurate. Thus, model performance and accuracy should be evaluated continuously [25]. Sometimes, periodic update of parameters or change of model structure is also required. We used the weather data from 01 September 2000 to 31 August 2001 for analyzing the connectionist models. For MLP and ERNN, we used the first eleven months data for training the networks and the dataset from (01-31) August 2001 for testing the trained models. We also used the dataset (01-30) April 2001 for testing and the remaining for training of RBFN, MLP and ERNN networks. We used this method to ensure that there is no bias on the training and test datasets. We used a Pentium-III, 1GHz processor with 256 MB RAM and all the experiments were simulated using MATLAB. The following steps were taken before starting the training process:

- The error level was set to a relatively small value (10^{-4}) that could be decreased to a smaller level, but the results show satisfactory prediction of the required outputs. Also setting the training accuracy to a higher level will take a much longer training time.
- The hidden neurons were varied (10-80) and the optimal number for each network were then decided as mentioned previously by changing the network design and running the training process several times until a good performance was obtained.
- When the network faces local minima (false wells), new ones to escape from such false wells replace the whole set of network weights and thresholds. Actually, a random number generator was used to assign the initial values of weights and thresholds with a small bias as a difference between each weight connecting two neurons together since similar weights for different connections may lead to a network that will never learn.

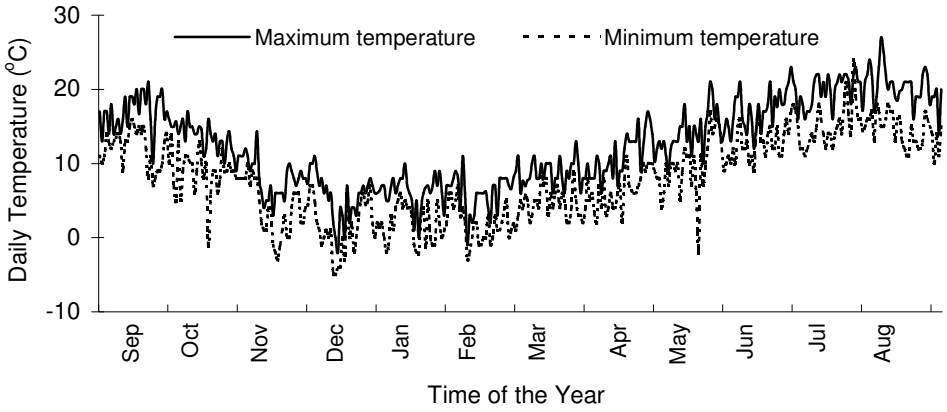


Figure 4: Actual daily minimum and maximum temperatures for the year 2001 in B.C, Canada.

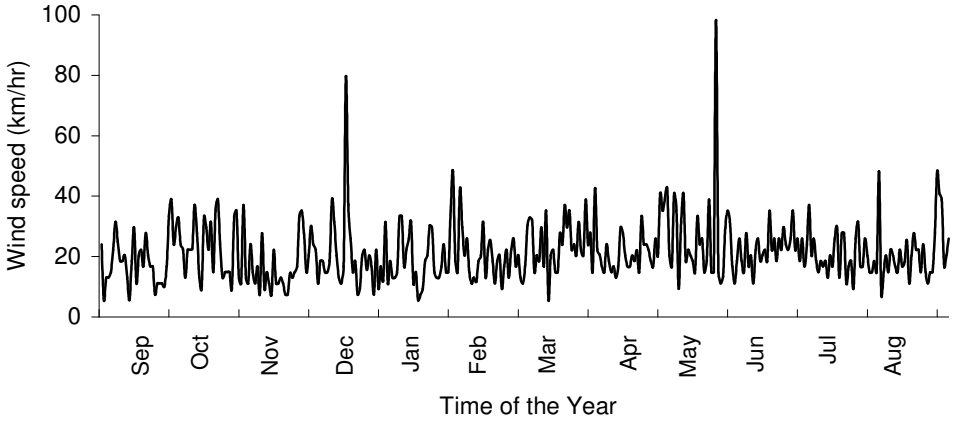


Figure 5: Actual daily wind-speed for the year 2001 in B.C., Canada.

6.1. Weather Parameters

6.1.1. Temperature

Our initial analysis of the data has shown that the most important weather parameters are the minimum and maximum temperature variables. These variables also represent a strong correlation with other weather-parameters. Temperature, in general, can be measured to a higher degree of accuracy relative to any of the other weather variables. Anyhow, forecasting temperature requires the consideration of many factors; day or night, clear or cloudy skies, windy or calm, or will there be any precipitation? An error in judgment on even one of these factors may cause forecasted temperature to be off by as much as 20 degrees. Historical temperature data recorded by a weather station at the prominent meteorological center of the Vancouver, British Columbia is used for the analysis. The maximum and minimum temperatures for period of one year (2000-2001) are plotted in Figure 4.

Space heating and cooling are the human response to how hot and how cold it 'feels'. Temperature in this case is only one of the contributors to such human response. Factors such as wind-speed in the winter and

humidity in the summer have to be accounted for when describing how cold or hot it 'feels'. For that purpose, the weather department developed a measure of how cold the air feels in the winter or how hot the air feels in the summer. These two new measurements are called the *wind-chill* and the *heat index*, respectively. Meteorologists use wind-chill equations to calculate the rate at which exposed human skin loses body heat.

6.1.2. Wind-Speed

It is significant in winter, if the temperature is low, which is roughly below 4.4 °C. When temperature is below freezing point, wind is a major factor determining the cooling rate. Figure 5 presents the wind-speed recorded in Vancouver, British Colombia for the year 2001.

6.2. Discussions and Test Results

MLP and ERNN networks trained by backpropagation algorithm are prone to overfit sample data because of the large number of parameters that must be estimated. Overfitting usually means estimating a model that fits the data so well that it ends by including some of the error randomness in its structure, and then produces poor forecasts. In MLP network, this may come about for two reasons: because the model was overtrained, or because it was too complex. Overtraining can be avoided by using cross-validation. The sample set is split into a training set and validation set. The ANN parameters are estimated on the training set, and the performance of the model is tested on the validation set. When this performance starts to deteriorate (which means the ANN is overfitting the training data), the iterations are stopped, and the last set of parameters to be computed is used to produce the forecasts. Another way is by using regularization techniques. This involves modifying the cost function to be minimized, by adding to it a term that penalizes for the complexity of the model. This term might, e.g., penalize for the excessive curvature in the model by considering the second derivatives of the output with respect to the inputs. Relatively simple and smooth models usually forecast better than complex ones. Overfitted ANN may assume very complex forms, with pronounced curvature, since they attempt to track down every single data point in the training sets; their second derivatives are, therefore, very large and the regularization term grows with respect to the error term. Keeping the total error low means keeping the model simple. Overfitting, however, may also be a consequence of over-parameterization, i.e., of the excessive complexity of the model. The problem is very common in backpropagation algorithm-based models; since they are often used as black-box devices, the users are sometimes tempted to add to them a large number of variables and neurons, without taking into account the number of parameters to be estimated. Many methods have been suggested to prune the ANN, i.e., to reduce the number of its weights, either by shedding some of the hidden neurons, or by eliminating some of the connections [31]. However, the adequate rate between the number of sample points required for training and the number of weights in the network has not yet been clearly defined; it is difficult to establish, theoretically, how many parameters are too many, for a given sample size.

Table 1 summarizes the architecture of the different network models used in this study. The training convergence of MLP and ERNN are illustrated in Figures 6 (a) and (b), respectively. RBFN took approximately 2 seconds to construct the network.

The optimal network is the one that should have the lowest error on test set and reasonable learning time. All the obtained results were compared and evaluated by the Maximum Absolute Percentage Error (MAP), Root Mean Square Error (RMSE), and Mean Absolute Deviation (MAD). Test results (August 2001) for the actual versus predicted minimum/maximum temperature and wind speed using MLP and ERNN with OSS and LM approaches are plotted in Figures 7, 8 and 9, respectively. Relative percentage error for minimum/maximum temperature and wind-speed is also plotted in Figures 10, 11 and 12, respectively. Empirical results are depicted in Tables 2 through 4.

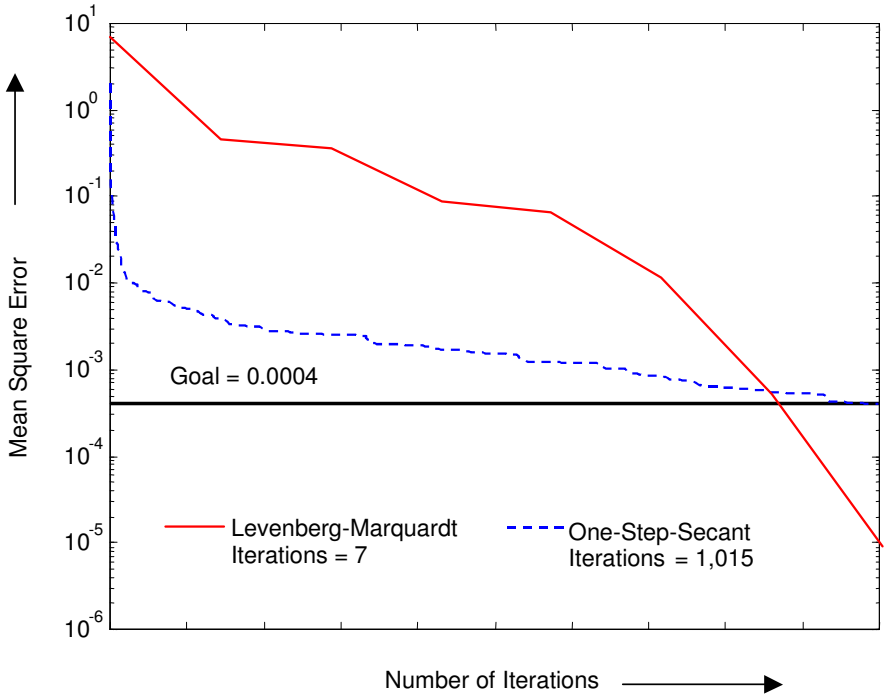


Figure 6 (a): Convergence of the LM and OSS training algorithms using MLP network.

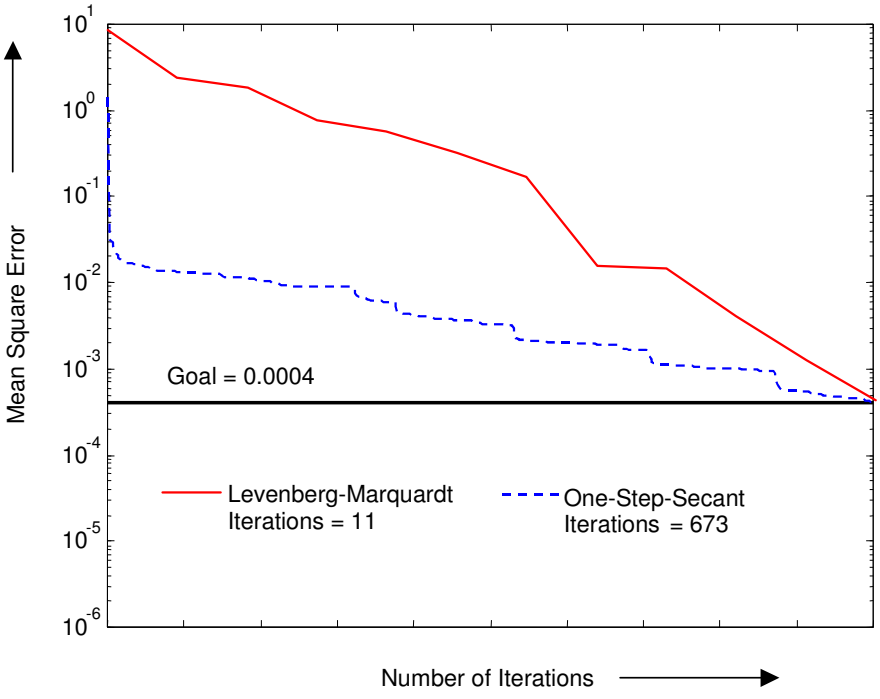


Figure 6 (b): Convergence of the LM and OSS training algorithms using ERNN.

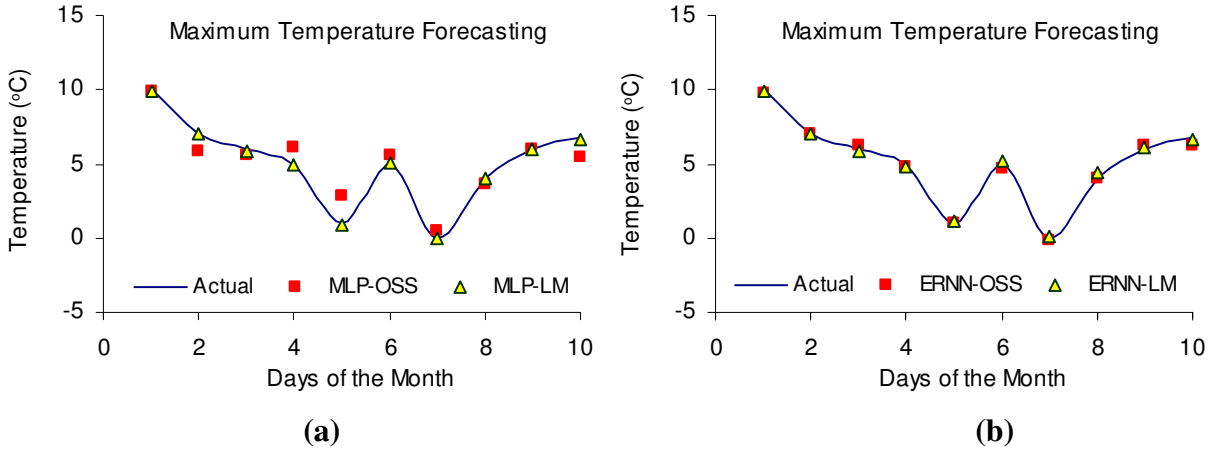


Figure 7: Comparison of actual and forecasted maximum temperature using OSS and LM approaches **(a)** MLP network and **(b)** ERNN.

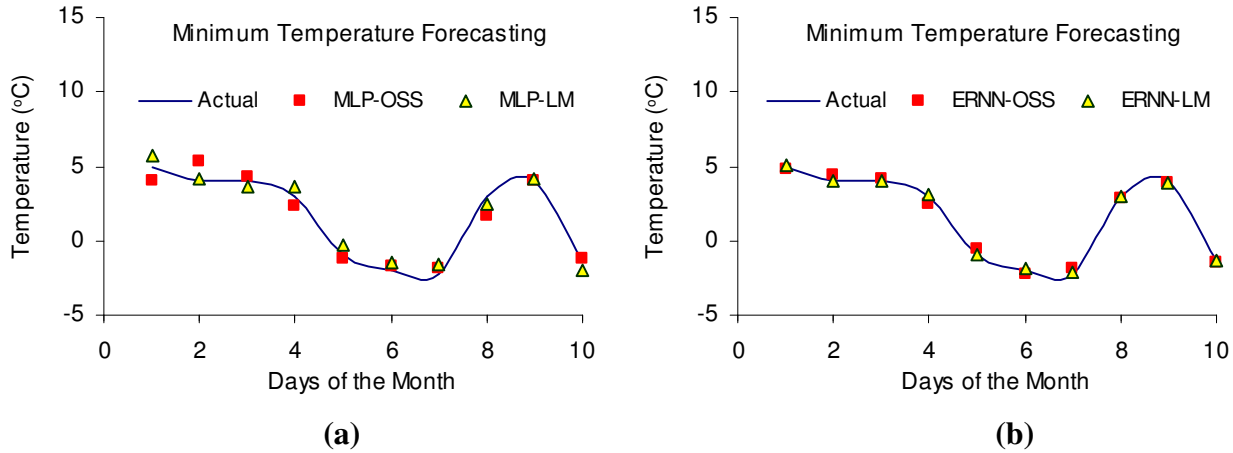


Figure 8: Comparison of actual and forecasted minimum temperature using OSS and LM approaches **(a)** MLP network and **(b)** ERNN.

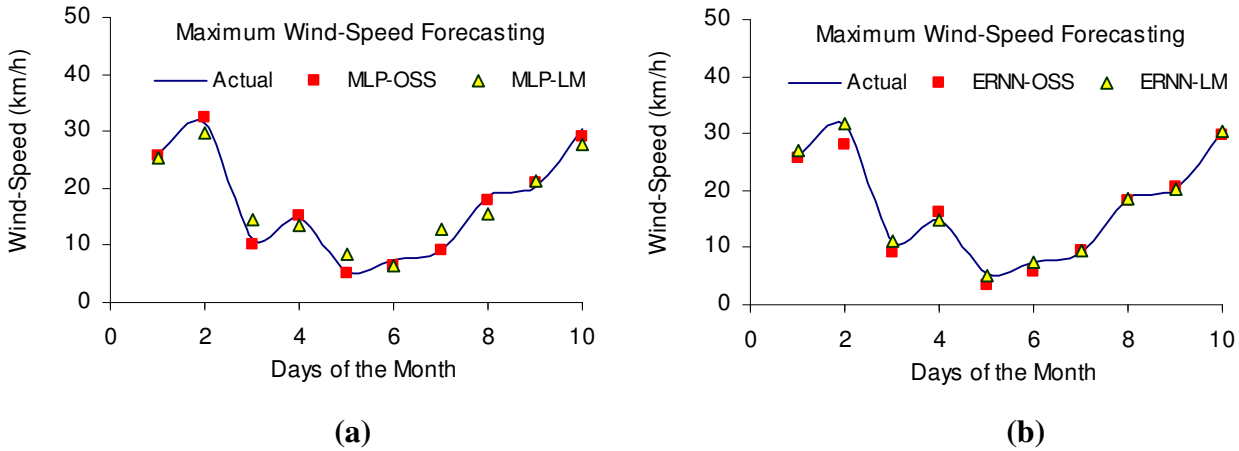
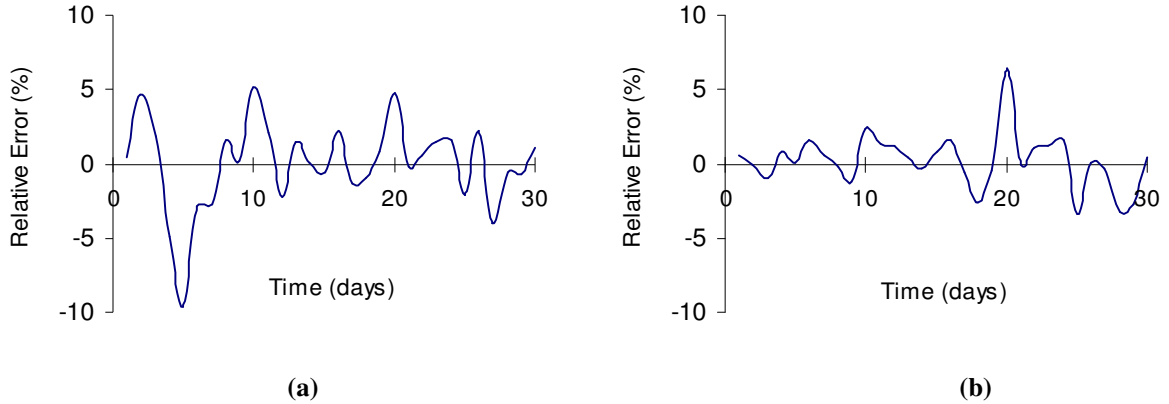
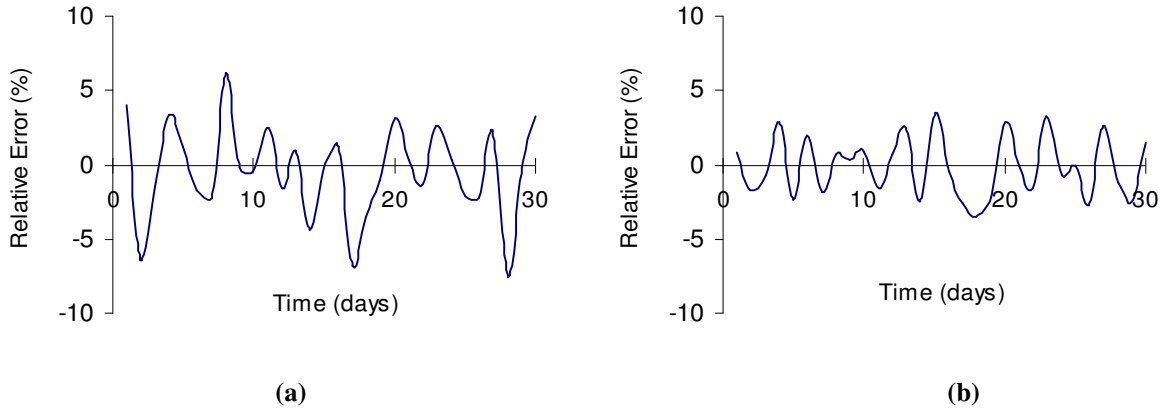


Figure 9: Comparison of actual and forecasted wind-speed using OSS and LM approaches **(a)** MLP network and **(b)** ERNN.



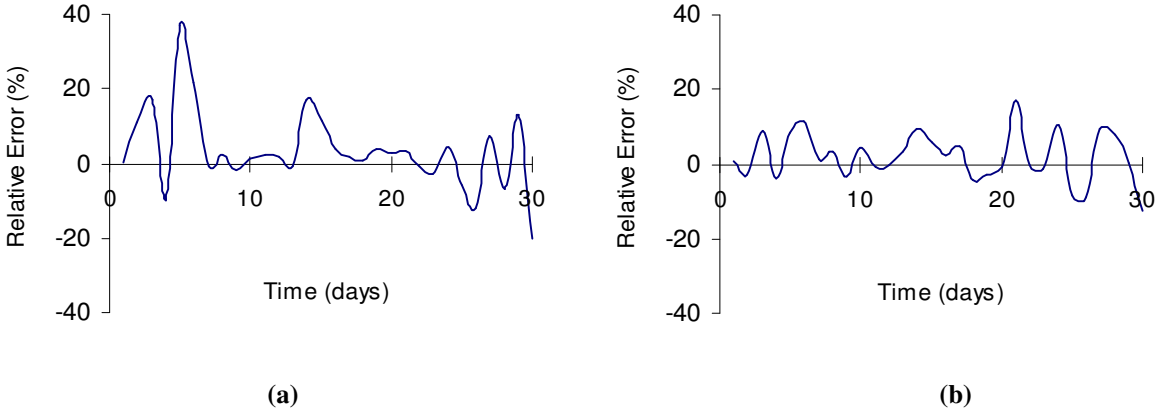
(a) **(b)**

Figure 10: Relative percentage error between actual and forecasted maximum daily temperature **(a)** MLP network and **(b)** ERNN.



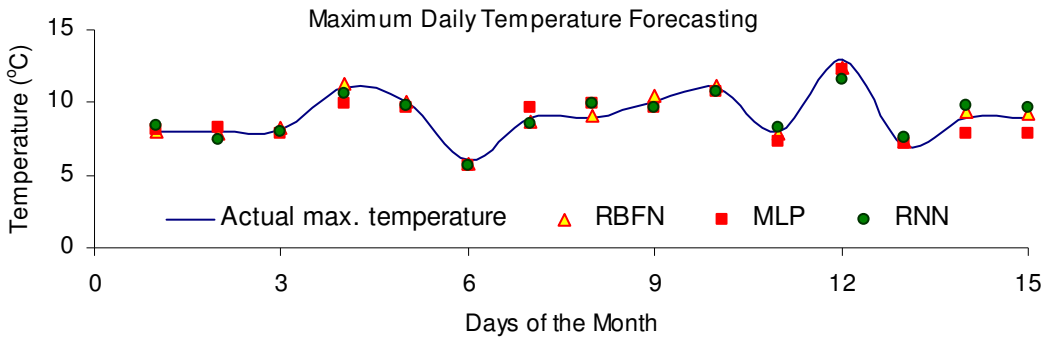
(a) **(b)**

Figure 11: Relative percentage error between actual and forecasted minimum daily temperature **(a)** MLP and **(b)** ERNN.

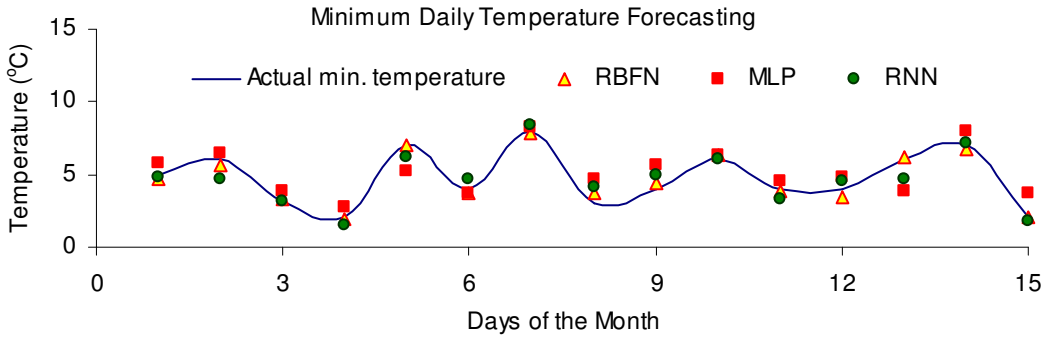


(a) **(b)**

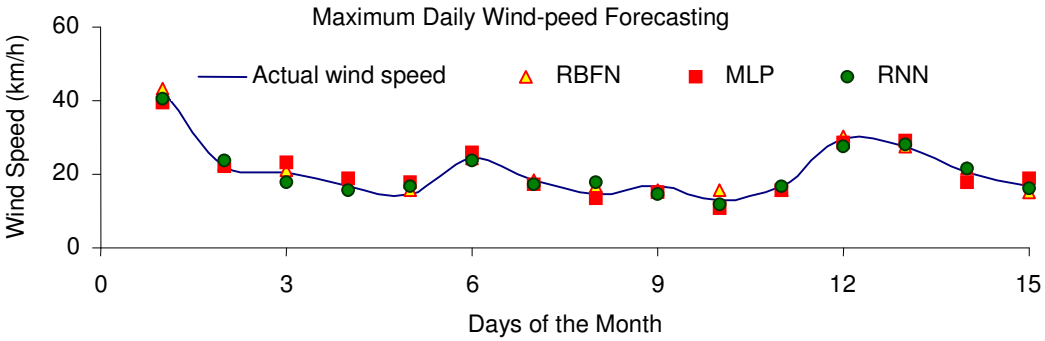
Figure 12: Relative percentage error between actual and forecasted wind-speed **(a)** MLP network and **(b)** ERNN.



(a)



(b)



(c)

Figure 13: Comparison among three neural networks techniques for 15-day ahead forecasting of **(a)** maximum daily temperature, **(b)** minimum daily temperature, **(c)** maximum daily wind-speed.

Table 1: Comparison of training of connectionist models.

Network model	Number of hidden neurons	Number of hidden layers	Activation function used in hidden layer	Activation function used in output layer
MLP	45	1	Log-sigmoid	Pure linear
ERNN	45	1	Tan-sigmoid	Pure linear
RBFN	180	2	Gaussian function	Pure linear

Test results (April 2001) for the actual versus predicted minimum/maximum temperature and wind-speed using MLP (OSS) and ERNN (OSS) and RBFN are illustrated in Figures 13 (a), (b) and (c). Empirical results are depicted in Table 5. Comparison of mean absolute percentage error (MAPE) using RBFN, MLP and ERNN techniques for three forecasted weather parameters is shown in Figure 14.

In this paper, the assessment of the forecasting performance of the trained networks was done by various forecasting errors. If training is successful, the network will be able to generalize, resulting in a high accuracy in the forecasting of unknown patters (provided the training data is sufficient representative of the forecasting situation). Various forecasting error measures between the actual and forecasted weather parameters are defined, however the most commonly adopted by weather forecasters are used here as listed below:

$$MAD = \frac{\sum_{i=1}^{N} \left| P_{actual,i} - P_{predicted,i} \right|}{N} \tag{44}$$

$$MAPE = \frac{\sum_{i=1}^{N} \frac{\left| P_{actual,i} - P_{predicted,i} \right|}{P_{actual,i}}}{N} \times 100 \tag{45}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \left(P_{actual,i} - P_{predicted,i} \right)^2}{N}} \tag{46}$$

$$MAP = \max\left(\frac{\left| P_{actual,i} - P_{predicted,i} \right|}{P_{predicted,i}} \times 100 \right) \tag{47}$$

where P_{actual} and $P_{predicted}$ are the actual and forecasted weather parameter, respectively, and N is the number of days in the data set.

Table 2: Performance of MLP and ERNN for peak temperature forecast.

Performance evaluation parameters	**MLP Network**		**ERNN**	
(maximum temperature)	**OSS**	**LM**	**OSS**	**LM**
Mean absolute percentage error (MAPE)	0.0170	0.0087	0.0165	0.0048
Root mean square error (RMSE)	0.0200	0.0099	0.0199	0.0067
Mean absolute deviation (MAD)	0.8175	0.4217	0.7944	0.2445
Correlation coefficient	0.96474	0.9998	0.9457	0.9826
Training time (minutes)	0.4	30	1.8	30
Number of iterations (epochs)	850	7	1135	10

Table 3: Performance of MLP and ERNN for minimum temperature forecast.

Performance evaluation parameters	**MLP Network**		**ERNN**	
(minimum temperature)	**OSS**	**LM**	**OSS**	**LM**
Mean absolute percentage error (MAPE)	0.0221	0.0202	0.0182	0.0030
Root mean square error (RMSE)	0.0199	0.0199	0.0199	0.0031
Mean absolute deviation (MAD)	0.7651	0.8411	0.7231	0.1213
Correlation coefficient	0.9657	0.9940	0.9826	0.9998
Training time (minutes)	0.3	1	0.3	7
Number of iterations (epochs)	1015	7	673	11

Table 4: Performance of MLP and ERNN for wind-speed forecast.

Performance evaluation parameters (wind-speed)	**MLP Network**		**ERNN**	
	OSS	**LM**	**OSS**	**LM**
Mean absolute percentage error (MAPE)	0.0896	0.0770	0.0873	0.0333
Root mean square error (RMSE)	0.1989	0.0162	0.0199	0.0074
Mean absolute deviation (MAD)	0.8297	0.6754	0.7618	0.3126
Correlation coefficient	0.9714	0.9974	0.9886	0.9995
Training time (minutes)	0.3	1	0.5	8
Number of iterations (epochs)	851	8	1208	12

Table 5: Performance of MLP / ERNN / RBFN.

Model	**Performance Evaluation Parameters**	**Maximum Temperature**	**Minimum Temperature**	**Wind Speed**
RBFN	MAP	3.821	3.622	4.135
	MAD	0.420	1.220	0.880
	Correlation Coefficient	0.987	0.947	0.978
MLP	MAP	6.782	6.048	6.298
	MAD	1.851	1.898	1.291
	Correlation Coefficient	0.943	0.978	0.972
RNN	MAP	5.802	5.518	5.658
	MAD	0.920	0.464	0.613
	Correlation Coefficient	0.946	0.965	0.979

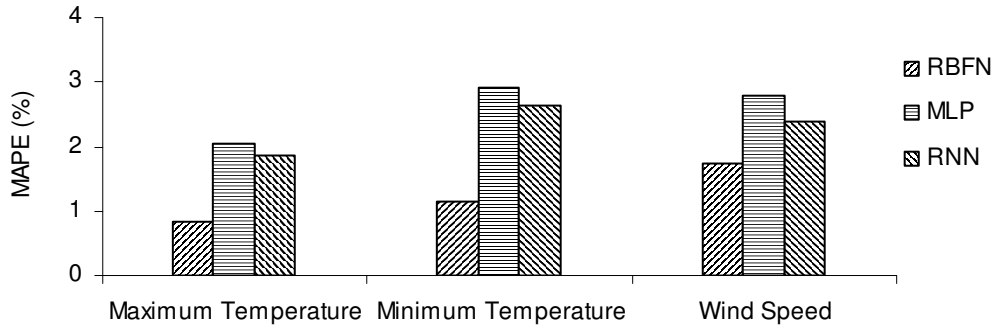


Figure 14: Comparison of mean absolute percentage error (MAPE) using RBFN, MLP and ERNN techniques for three forecasted weather parameters.

The final stage is the validation of the proposed forecasting models. It is well known that goodness-of-fit are not enough to predict the actual performance of a method, so we test our models by examining our errors in samples other that the one used for parameter estimation (out-of-sample errors, as opposed to in-sample errors). The three strategies adopted for testing the applicability of an ANN in the present work are:

(a) to test the capability of an ANN to correctly predict the output for the given input set originally used to train the network (accuracy performance);
(b) to test the capability of the ANN to correctly predict the output for the given input sets that were not included in the training set (generalized performance); and
(c) to develop a neural network model which could be trained faster.

7. Conclusions

Neural networks have gained great popularity in time-series prediction because of their simplicity and robustness. The learning method is normally based on the descent gradient method – backpropagation algorithm. Backpropagation algorithm has two major drawbacks: the learning process is time-consuming and the performance is heavily dependent on the network parameters like learning rate, momentum and so on.

In this paper, we compared the performance of multi-layered perceptron (MLP) neural network, Elman recurrent neural network (ERNN) and radial basis functions network (RBFN). Compared to the MLP neural network, the ERNN could efficiently capture the dynamic behavior of the weather, resulting in a more compact and natural internal representation of the temporal information contained in the weather profile. ERNN took more training time but it is dependent on the training data size and the number of network parameters. It can be inferred that ERNN could yield more accurate results, if good data selection strategies, training paradigms, and network input and output representations are determined properly. Levenberg-Marquardt (LM) approach appears to be the best learning algorithm for mapping the different chaotic relationships. Due to the calculation of Jacobian matrix at each epoch, LM approach requires more memory and is computationally complex while compared to one-step-secant (OSS) algorithm.

On the other hand, RBFN gave the overall best results in terms of accuracy and fastest training time. Empirical results clearly demonstrate that radial basis function networks are much faster and more reliable for the weather forecasting problem considered. The proposed RBFN network can also overcome several limitations of the MLP and ERNN networks such as highly nonlinear weight update and slow-convergence rate. Since the RBFN has natural unsupervised learning characteristics and modular network structure, these properties make it a more effective candidate for fast and real-time weather forecasting.

Acknowledgements

The authors are grateful to the staff of the meteorological department, Vancouver, B.C., Canada for the useful discussions and providing the weather data used in this research work.

References

[1] Gedzelman, S.D., "Forecasting skill of beginners", *Bull. Am. Meteorol. Soc*. Vol. 59, pp. 1305-1309, 1978.

[2] Maqsood Imran, Khan Muhammad Riaz and Abraham Ajith, "Neuro-computing based Canadian weather analysis", *2nd International Workshop on Intelligent Systems Design and Applications ISDA-02*, Atlanta, USA, July 2002. (Forth coming)

[3] Elman J. L., "Distributed representations, simple recurrent networks and grammatical structure", *Machine Learning*, Vol. 7, No. 2/3, pp. 195-226, 1991.

[4] Moody J. and Utans J., "Architecture selection strategies for neural networks: Application to corporate bond rating prediction", *Neural Networks in the Capital Markets*, J. Wiley and Sons, 1994.

[5] Cholewo J. T. and Zurada M. J., "Neural network tools for stellar light prediction", *IEEE Aerospace Conference*, Vol. 3, pp. 514-422, USA, 1997.

[6] Neelakantan T.R. and Pundarikanthan N.V., "Neural network-based simulation-optimization model for reservoir operation", *Journal of Water Resources Planning and Management*, Vol. 126, No. 2, pp. 57-64, 2000.

[7] Elman J. L., "Finding structure in time", *Cognitive Science*, Vol. 14, pp. 179-211, 1990.

[8] Kugblenu S., Taguchi S. and Okuzawa T., "Prediction of the geomagnetic storm associated D_{st} index using an artificial neural network algorithm", *Earth Planets Space*, Vol. 51, pp. 307-313, Japan, 1999.

[9] Khan Muhammad Riaz and Ondrusek Cestmir, "Short-term load forecasting with multilayer perceptron and recurrent neural network", *Journal of Electrical Engineering*, Vol. 53, pp. 17-23, Slovak Republic, January-February 2002.

[10] Zurada J. M., *Introduction to Artificial Neural Systems*, West Publishing, 1992.

[11] Bishop C. M, *Neural Networks for pattern recognition*, Oxford Press, 1995.

[12] Hagan M. T., Demuth H.B. and Beale M.H., *Neural Network Design*, Boston, PWS Publishing, 1996.

[13] Kuligowski R. J., Barros A. P., "Localized precipitation forecasts from a numerical weather prediction model using artificial neural networks", *Weather and Forecasting,* Vol. 13, No. 4, pp.1194, 1998.
[14] Kuligowski R. J., Barros A. P., "Experiments in short-term precipitation forecasting using artificial neural networks", *Monthly Weather Review,* Vol. 126, No. 2, pp. 470, 1998.
[15] Moro Sancho Q. I., Alonso L. and Vivaracho C. E., "Application of neural networks to weather forecasting with local data", *Applied Informatics*, Vol. 68, 1994.
[16] Aussem A., Murtagh F. and Sarazin M., "Dynamical recurrent neural networks and pattern recognition methods for time series prediction, Application to Seeing and Temperature Forecasting in the Context of ESO's VLT Astronomical Weather Station", *Vistas in Astronomy,* Vol. 38, No. 3, pp. 357, 1994.
[17] Allen G. and Le Marshall J. F., "An evaluation of neural networks and discriminant analysis methods for application in operational rain forecasting", *Australian Meteorological Magazine*, Vol. 43, No. 1, pp.17-28, 1994.
[18] Doswell C. A., "Short range forecasting: Mesoscale Meteorology and Forecasting", Chapter 29, *American Meteor. Society*, pp. 689-719, 1986.
[19] Murphy A. H. and et al., "Probabilistic severe weather forecasting at NSSFC: An experiment and some preliminary results", 17th conference on Severe local storms, *American Meteor. Society*, pp. 74-78, 1993.
[20] Tan, Y., Wang, J. and Zurada, J. M., "Nonlinear blind source separation using a radial basis function network*", IEEE Transactions on Neural Networks*, Vol. 12, No. 1, pp. 124-134, 2001.
[21] Chen, S., MacLaughlin, S., and Mulgrew, B., "Complex-valued radial basis function network, Part I: Network architecture and learning algorithm", *Signal Processing*, Vol. 35, pp. 19-31, 1994.
[22] Orr, M. J., "Regularization in the selection of radial basis function centers", *Neural Computation*, Vol. 7, No. 3, pp. 606-623, 1995.
[23] Park, J. and Sandberg, I. W., "Universal approximation using radial basis function", *Neural Computation*, Vol. 3, No. 2, pp. 246-257, 1991.
[24] Abraham Ajith, Philip Sajith and Joseph Babu, "Will We Have a Wet Summer? Long-term Rain Forecasting Using Soft Computing Models", Modeling and Simulation 2001, In Proceedings of the 15th European Simulation Multi Conference, *Society for Computer Simulation International*, Prague, Czech Republic, pp. 1044-1048, 2001.
[25] Linde, Y., Buzo, A. and Gray, R., "An algorithm for vector quantizer design", *IEEE Transactions on Communications*, Vol. 28, pp. 84-95, 1980.
[26] Haykin, S., "Neural networks – A comprehensive foundation, 2nd edition, Upper Saddle River, NJ: Prentice Hall, 1999.
[27] Zhang, S. Patuwo, B.E. and Hu M.Y., "Forecasting with artificial neural networks: The state-of-the-art", *International Journal on Forecasting*, Vol. 14, pp. 35-62, 1998.
[28] Hippert, H.S. Pedreira, C.E. and Souza, R.C., "Neural networks for short-term load forecasting: A review and evaluation", *IEEE Transactions on Power Systems*, Vol. 16, No. 1, pp. 44-55, 2001.
[29] Kiartzis, S.J. Zoumas, C.E. Theocharis, J.B. Bakirtzis, A.G. and Petridis V., "Short-term load forecasting in an autonomous power system using artificial neural networks", *IEEE Transactions on Power Systems*, Vol. 12, No. 4, pp. 1591-1596, 1997.
[30] Piras, A. Germond, A. and Buchenel, B. Imhof, K. and Jaccard, Y., "Heterogeneous artificial neural network for short-term electrical load forecasting", *IEEE Transactions on Power Systems*, Vol. 11, No. 1, pp. 397-402, 1996.
[31] Reed, R., "Pruning algorithms – A survey", *IEEE Transactions on Neural Networks*, Vol. 4, No. 5, pp. 740-747, 1993.