Agricultural Economics Research Review
Vol. 21 January-June 2008 pp 5-10

Artificial Neural Network Methodology for Modelling and Forecasting Maize Crop Yield

Rama Krishna Singh and Prajneshu*
Biometrics Division, Indian Agricultural Statistics Research Institute (ICAR), New Delhi - 110 012

Abstract

A particular type of "Artificial neural network (ANN)", viz. Multilayered feedforward artificial neural network (MLFANN) has been described. To train such a network, two types of learning algorithms, namely Gradient descent algorithm (GDA) and Conjugate gradient descent algorithm (CGDA), have been discussed. The methodology has been illustrated by considering maize crop yield data as response variable and total human labour, farm power, fertilizer consumption, and pesticide consumption as predictors. The data have been taken from a recently concluded National Agricultural Technology Project of Division of Agricultural Economics, I.A.R.I., New Delhi. To train the neural network, relevant computer programs have been written in MATLAB software package using Neural network toolbox. It has been found that a three-layered MLFANN with (11,16) units in the two hidden layers performs best in terms of having minimum mean square errors (MSE) for training, validation, and test sets. Superiority of this MLFANN over multiple linear regression (MLR) analysis has also been demonstrated for the maize data considered in the study. It is hoped that, in future, research workers would start applying not only MLFANN but also some of the other more advanced ANN models, like 'Radial basis function neural network', and 'Generalized regression neural network' in their studies.

Introduction

Multiple linear regression (MLR) modelling is a very powerful technique and is widely used to estimate linear relationship between response variable and predictors. Its main limitation is that it is useful only when the underlying relation between response and predictor variables is assumed to be "linear". However, in a realistic situation, this assumption is rarely satisfied. Also, if there are several predictors, it is well nigh impossible to have an idea of the underlying non-linear functional relationship between response and predictor variables. Fortunately, to handle such a situation, an extremely versatile approach of "Artificial neural networks" (ANNs) is developing rapidly. Cheng and Titterington (1994) have reviewed the ANN methodology from a statistical perspective, while Warner and Misra (1996) have laid emphasis on the understanding of ANN as a statistical tool.

A distinguishing feature of ANNs that makes them valuable and attractive for a statistical task is that, as opposed to traditional model-based methods, ANNs are data-driven self-adaptive methods in that there are a few a-priori assumptions about the models for problems under study. This modelling approach with ability to learn from experience is very useful in many practical problems since it is often easier to have data than to have good theoretical guesses about the underlying laws governing the systems from which data are generated. Recently, Zhang (2007) has discussed various pitfalls in the ANN modelling work, which must be avoided.

*Author for correspondence, Email: prajneshu@yahoo.co.in

This paper has been drawn from Ph.D. thesis of first author under the guidance of second author, submitted to Indian Agricultural Research Institute, New Delhi, in 2007.

Most widely used ANN is multilayered feedforward artificial neural network (MLFANN); and this paper aims to thoroughly discuss its various aspects. As an illustration, the methodology has been applied for modelling and forecasting of maize crop yield on the basis of four predictor variables, viz. total human labour, farm power, fertilizer consumption, and pesticide consumption, by taking a part of data from Singh *et al.* (2004). MLFANN with zero, one, and two hidden layers have been considered. Optimum numbers of hidden layers as well as optimum numbers of units in each hidden layer have been found by computing MSEs.

Methodology

Preliminaries of ANN

ANN can be considered as an interconnected assembly of simple processing elements (or units/ nodes/neurons). The processing ability of network is stored in the inter-unit connection strengths or weights obtained by a process of learning from a set of training patterns. A typical ANN consists of one input layer, one output layer and hidden layers. Each layer can have several units whose output is a function of weighted sum of their inputs. Input into a node is a weighted sum of outputs from nodes connected to it. Thus, net input into a node is given by Equation (1):

$$Netinput_i = \sum\left(w_{ij} * output_j\right) + u_i \qquad \ldots(1)$$

where, w_{ij} are weights connecting neuron *j* to neuron *i*; *output$_j$* is the output from the unit *j*; and u_i is a threshold for neuron *i*. The threshold-term is baseline input to a node in the absence of any other inputs. If weight w_{ij} is negative, it is termed '*inhibitory*' because it decreases net input, otherwise it is called '*excitatory*'.

Each unit takes its net input and applies an *activation function* to it. For example, output of the *j* th unit, also called *activation value* of the unit, is $g\left(\sum w_{ji} x_i\right)$, where $g(\,.\,)$ is activation function and x_i is output of the *i*th unit connected to unit *j*. Two important activation functions commonly used are :

(i) Pureline:

$$g(netinput) = constant\,.\,(netinput) \qquad \ldots(2)$$

(ii) Sigmoidal:

$$g(netinput) = 1/\left[1 + exp(-\,netinput)\right] \qquad \ldots(3)$$

With no hidden units, an ANN can classify only linearly separable problems (ones for which possible output values can be separated by global hyperplanes). However, it has been shown by Cybenko (1989) that with one hidden layer, an ANN can describe any continuous function (if there are enough hidden units), and that with two hidden layers, it can describe any function.

The weights in an ANN, similar to coefficients in a regression model, are adjusted to solve the problem presented to ANN. *Learning* or *training* is used to describe the process of finding values of these weights. Two types of learning with ANN are: *Supervised* and *Unsupervised* learning. The supervised learning occurs when there is a known target value associated with each input in the training set. Output of ANN is compared with a target value, and this difference is used to train ANN (alter the weights). The unsupervised learning is needed when training data lack target output values corresponding to input patterns. ANNs discussed so far are constructed with layers of units, and thus are termed *Multilayered* ANNs. A layer of units in such an ANN is composed of units that perform similar tasks.

Multilayered Feedforward Artificial Neural Network (MLFANN)

An MLFANN is one in which units in one layer are connected only to units in the next layer, and not to units in a preceding layer or units in the same layer. An MLFANN can have a number of hidden layers with a variable number of hidden units per layer. When counting layers, it is a common practice not to count input layer because it does not perform any computation, but simply passes data onto the next layer. So, an MLFANN with an input layer, one hidden layer, and an output layer is termed as a two-layered MLFANN.

The MLFANN is the most popular network architecture. It is the type of network in which units are arranged in a layered feedforward topology. The network thus has a simple interpretation as a form of input-output model, with weights and thresholds (biases) as free parameters of the model. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity.

Neural networks are constructed by learning from repeated presentation of inputs (the *x*'s) and outputs (the *y*'s) and adjusting internal parameters so as to minimize error between fitted and desired *y*'s. Neural network can be seen as a general way to parameterize data through arbitrary non-linear functions from space of predictor variables to the space of response variables. The utility and flexibility of neural network arise from the application of learning algorithms that allow the network to construct correct weights, and hence, the desired function, for a given set of observations.

Learning Algorithms

As the input–output vectors are presented to the network, a learning algorithm adjusts connection weights until the system converges on a function that correctly reproduces the output. Optimal connection weights may be obtained by using gradient descent algorithm or conjugate gradient descent algorithm with a view to minimizing sum of the squared error functions of the network output.

Gradient Descent Algorithm (GDA)

To optimize weights, the objective function to be minimized is generally taken as a sum of squared errors defined by Equation (4):

$$E = 0.5\sum\sum\left(y_{pk} - Y_{pk}\right)^2 \quad \ldots(4)$$

where, subscript *p* refers to patterns (observations) with a total of *n* patterns, subscript *k* to output unit with a total of *O* output units, and *y* and *Y* are observed and estimated responses, respectively.

As input units simply pass information to the hidden units, input into the j^{th} hidden unit, h_{pj}, is given by Equation (5):

$$h_{pj} = \sum w_{ji} x_{pi} \quad \ldots(5)$$

Here, w_{ji} is the weight from input unit *i* to hidden unit *j*, and x_{pi} is value of the *i*th input for pattern *p*. The *j*th unit applies an activation function say, sigmoid function, given by Equation (3) to its net input and outputs:

$$v_{pj} = g\left(h_{pj}\right) = 1/[1 + exp(-h_{pj})] \quad \ldots(6)$$

Similarly, output unit *k* receives a net input of

$$f_{pk} = \sum W_{kj} v_{pj} \quad \ldots(7)$$

Here, represents weight from the hidden unit *j* to output *k*. The unit then outputs quantity expressed by relation (8):

$$Y_{pk} = g\left(f_{pk}\right) = 1/[1 + exp(-f_{pk})] \quad \ldots(8)$$

Equations (5) to (8) demonstrate that objective function given by Equation (4) is a function of unknown weights w_{ji} and W_{ji}. So, we evaluate partial W_{kj} derivative of objective function with respect to weights, and then move weights in a direction down the slope, continuing until error function no longer decreases. Mathematically, this can be expressed by Equation (9):

$$\Delta W_{kj} = -\eta\, \partial E / \partial W_{kj} \quad \ldots(9)$$

The η term is known as *learning rate* and simply scales step size. Substituting Equations (5) to (8) in Equation(4) and expanding Equation (9) using chain rule, we get:

$$\partial Y_{pk} / \partial f_{pk} = g'\left(f_{pk}\right) = Y_{pk}\left(1 - Y_{pk}\right) \quad \ldots(10)$$

and

$$\partial f_{pk} / \partial W_{kj} = v_{pj} \quad \ldots(11)$$

Substituting these results back in Equation (9), change in weights from the hidden units to output units is given by Equation (12):

$$\Delta W_{kj} = -\eta[(-1)y_{pk} - Y_{pk})]Y_{pk}\left(1 - Y_{pk}\right)v_{pj} \quad \ldots(12)$$

Weights are updated as

$$W_{kj}(t+1) = W_{kj}(t) + \Delta W_{kj} \qquad \ldots(13)$$

Similarly, calculations for weights from inputs to the hidden units can be carried out as given in Warner and Misra (1996). Finally, the algorithms, following similar lines as given by Hertz *et al.* (1991) are as follows:

(i) Initialize the weights to small random values. This puts the output of each unit around 0.5.

(ii) Choose a pattern *p* and propagate it forward. This yields values for v_{pj} and Y_{pk}, the outputs from the hidden layer and output layer.

(iii) Compute the output errors:
$$\delta_{pk} = (y_{pk} - Y_{pk})g'(f_{pk})$$

(iv) Compute the hidden layer errors:
$$\psi_{pj} = \sum \delta_{pk} W_{kj} v_{pi}(1 - v_{pj})$$

(v) To update the weights, compute:
$\Delta W_{kj} = \eta \delta_{pk} v_{pj}$ and $\Delta w_{ji} = \eta \psi_{pj} i_{pi}$

Repeat the steps for each pattern.

Conjugate Gradient Descent Algorithm (CGDA)

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

In most of the training algorithms, a learning rate is used to determine length of weight update (step size). In CGDA, step size is adjusted at every iteration. A search is made along conjugate gradient direction to determine step size, which minimizes performance function along that line. As CGDA requires only a little more storage than GDA, this is often a good choice for networks with a large number of weights (greater than 100).

An Illustration

Singh *et al.* (2004) have carried out a study dealing with various aspects of the maize crop. In the present illustration, part of the data from the state of Uttar Pradesh covering 170 farmers, for whom complete data were available, has been considered: Specifically, response variable taken was 'maize crop yield', while four predictors were: total human labour (Rs/ha), farm power (Rs/ha), fertilizer consumption (kg/ha) and pesticide consumption (Rs/ha).

Neural Network Toolbox in MATLAB® (2006) available at IASRI, New Delhi, was employed to train the MLFANN. Before training, input and target values were pre-processed using suitable scaling, so that they fell within a specified range. Available 170 observations were divided into three subsets: (i) First sub-set was training set comprising 130 observations, which was used for computing gradient and updating the network weight and biases, (ii) Second sub-set of 30 observations comprised validation set, and (iii) Test set comprised the remaining 10 observations.

MLFANN was trained using both GDA and CGDA. Several possibilities were tried. When no layer was taken as the hidden layer, only input and output layers were used. Activation function employed was "Pureline". When one hidden layer was considered, activation function between input layer and hidden layer was taken as "Sigmoidal", while that between hidden layer and output layer, "Pureline" activation function was used. Further, with two hidden layers, Sigmoidal activation functions in the hidden layer and a linear transfer function in the output layer were used. The purpose of doing so was that if the last layer of a MLFANN had sigmoid neurons, then the outputs of the network were limited to a small range because of the "squashing" property of sigmoid function. If linear output neuron were used, the network output could take on any value.

Performance of the trained network can be measured by mean square on the training, validation and test sets, but it is often useful to investigate the network response in more details. One option is to perform a regression analysis between network response and the corresponding targets. A large number of networks were trained, and the correlation coefficient values obtained between network output and target values for training data have been reported in Table 1. It was noticed that the MLFANN model was able to approach the "best" possible fit quite closely. We used MSE for network performance evaluation. In Table 2, MSEs for training as well as validation sets have been summarized for both GDA and CGDA.

The computer programs were written in MATLAB to train MLFANN using the two training

Table 1. Correlation coefficients between output and target yield values

Number of neurons in hidden layers	Training methods	
	GDA	CGDA
(i) No hidden layer	0.437	0.639
(ii) One hidden layer		
8	0.705	0.811
12	0.691	0.847
15	0.698	0.874
(ii) Two hidden layers		
(5, 10)	0.738	0.793
(8,13)	0.773	0.916
(11,16)	0.808	0.933

Table 2. MSE for training and validation data using both learning algorithms

Number of neurons in hidden layers		Training methods	
		GDA	CGDA
(i) No hidden layer	Training	69.01	69.01
	Validation	41.40	41.38
(ii) One hidden layer			
8	Training	55.61	33.00
	Validation	29.98	0.013
12	Training	54.54	32.46
	Validation	35.68	0.003
15	Training	54.78	22.86
	Validation	14.73	0.003
(ii) Two hidden layers			
(5, 10)	Training	52.88	43.05
	Validation	14.35	0.22
(8,13)	Training	46.72	18.68
	Validation	17.92	0.004
(11,16)	Training	40.32	12.94
	Validation	20.39	0.003

Table 3. Performance of MLFANN (11-16-1) model for test data

Observation number	Predicted values	Actual values
161	15.23	17.75
162	39.23	38.21
163	27.49	28.75
164	34.96	32.02
165	40.38	39.94
166	39.48	38.97
167	21.46	26.46
168	18.50	13.75
169	37.56	39.34
170	48.46	52.50

algorithms and may be obtained from the first author on request. As CGDA is a faster learning method than GDBP, therefore, the number of epochs used to train the MLFANN using CGDA was less than that for GDA. The MSEs for best trained MLFANN (11-16-1) using CGDA and for traditionally used MLR were computed as 12.94 and 69.01, respectively; thereby clearly demonstrating superiority of MLFANN (11-16-1) over MLR for data under consideration. Finally, for the test data comprising 10 observations, predicted values of response variable using MLFANN (11-16-1) model along with actual values were obtained and have been reported in Table 3. Evidently, predicted and actual values are quite close. Thus, it could be concluded that artificial neural network methodology is successful in describing the given data.

Conclusions

The potential of artificial neural network methodology has been highlighted for successfully tackling the realistic situation in which exact non-linear functional relationship between response variable and a set of predictors is not known. Although ANNs may not be able to provide the same level of insight as many statistical models do, it is not correct to treat them as "black boxes". In fact, one active area of research in ANN is 'understanding the effect of predictors on response variable'. It is hoped that, in future, research workers would start applying not only MLFANN but also some of the other more advanced ANN models, like 'Radial basis function neural network', and 'Generalized regression neural network' in their studies.

Acknowledgements

The authors are grateful to the referee for valuable comments and thank Dr A.K. Vasisht for providing the data.

References

Cheng, B. and Titterington, D. M. (1994). Neural networks: A review from a statistical perspective. *Statistical Science,* **9:** 2-54.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, **2:** 303-14.

Hertz, J., Krogh, A. and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation.* Reading, MA: Addison-Wesley.

Matlab® (2006). *Neural Network Toolbox User's Guide.* USA: The Math Works, Inc.

Singh, R.P., Kumar, R., Singh, B.B., Awasthi, P.K., Atibudhi, H.N., Chahal, S.S., Varghese, K.A., Singh, R.K. and Maurya, S.P. (2004). *Technological Change and Production Performance in Irrigated Maize-based Agro-ecosystem: The Interplay of Economic, Technological and Institutional Factors.* N.A.T.P.(PSR-61). New Delhi: Division of Agricultural Economics, IARI, Research Report 2004-01, pp. 1-107.

Warner, B. and Misra, M. (1996). Understanding neural networks as statistical tools. *American Statistician,* **50:** 284-93.

Zhang, G. P. (2007). Avoiding pitfalls in neural network research. *IEEE Transactions on Systems, Man and Cybernetics— Part C: Applications and Reviews*, **37:** 3-16.