Industrial process steam consumption prediction through an Artificial Neural Networks (ANNs) approach

FITSUM BEKELE TILAHUN
Renewable Energy Systems
ITT, TH Köln (Cologne University of Applied Science)
Betzdorfer Strasse 2, 50679 Köln
GERMANY
ftsebeek@gmail.com www.amu.edu.et

RAMCHANDRA BHANDARI
Renewable Energy Systems
ITT, TH Köln (Cologne University of Applied Science)
Betzdorfer Strasse 2, 50679 Köln
GERMANY
ramchandra.bhandari@th-koeln.de www.th-koeln.de

MENEGESHA MAMO
Electrical & Computer Engineering
Addis Ababa University Institute of Technology (AAIT)
King George VI Street, Addis Ababa
ETHIOPIA
menegesha.mamo@aau.edu.et www.aait.edu.et

Abstract: - Current research studies have demonstrated the capability of Artificial Neural Networks (ANNs) in learning to generalize for solving complex industrial problems. However, hardly few such studies have been conducted to investigate if these ANNs are also effective in identifying energy use patterns in industrial processes. In this research work a resilient gradient descent variant of a multilayer neural network (MLP) is developed for determining steam consumption patterns as a function of production rate in textile factory. The model is tested using real-time data from each steam-consuming machine's daily production and a meter reading of an electrical steam boiler. Parts of these data (85%) were randomly selected in order to train the network. The remaining data were used to test the performance of the trained network. The result obtained showed an acceptable error performance index of magnitude around 0.0674. The model also gave a correlation coefficient (R) between the estimated and target values as 0. 9781. Thus the proposed neural network can be used as a valuable tool as an energy use approximator in industrial production processes. Moreover, with the availability of more training data, an increased prediction capability can be achieved.

Key-Words: - Artificial Neural Networks (ANNs), multilayer neural network (MLP), resilient gradient descent industrial processes, steam consumption prediction.

1 Introduction

One common as well as important Artificial Neural Networks (ANNs) application that finds itself in much practical use is function approximation. Function approximation range from determining realizable feedback function that relates measured outputs to control input in control systems to finding a function that correlated past values of an input signal to output in adaptive filtering. Lately, Artificial Neural Networks (ANNs) have been used extensively in finding underling functional relation of engineering processes. This pertains to the ability of ANNs to predict or solve non-linear problems with high degree of accuracy given enough data to learn from. A wide variety of ANNs have been used with varying configuration that suits the specific requirements of an application.

A widely used and efficient ANN function approximation is the MLP (multi-layer perceptron) networks based on the BP (back-propagation) learning algorithm. Though researches are still contributing to know more about these ANNS, several studies have exemplified the back propagation learning algorithm as the forerunner among the Multi-layer perceptron algorithms [1-3]. The accuracy and convergence speed of these MLPs usually depend on the neural network architectural configuration as well as choice of tuneable parameters during the implementation stage. In previous studies, researchers have used some techniques to solve real applications using these algorithms. However hardly any examples of industrial processes energy consumption prediction from production process were done. This paper is an attempt to answer this question by implementing one of the most powerful ANNs-MLP while trying to consider issues relating to their practical application.

Realization of perceptron concept by Rosenblatt in 1958 was the hallmark of ANNs. The perceptron unit is an individual processing unit that accepts weighted input and produces a rule based threshold output. MLP is a feed-forward ANN that is implemented by customizing these fundamental units. This customization introduced addition of layers of neurons and a nonlinear transfer function [2, 3].

2 Problem Formulation

Energy consumption determination is perhaps the first crucial element in demand side energy management (DSM). Additionally, in integration process of renewables such as solar plant in industries, knowledge of the load is a necessary requirement. To achieve this, direct measurements of generation and consumption can be done, otherwise known as an energy audit. This method is costly and might mean persistent measurements under different industrial production conditions. Another way is to get the industrial processes average energy consumption from a manufacturer's specification. This method, even though simple, is not usually practically usable. This is because, it does not take into account the energy utilization under changing scenarios such as a not nominal operation, changing input parameters in production processes, and changing behavior of machines through its life cycle. The last method, which is proposed in this study, is to use ANNs to predict energy use patterns under real-time changing production processes. This however, requires a substantial data and several model configuration trials in order to generalize well.

This work is part of a larger project called "Control and Optimization of a Large-Scale Solar Plant in Ethiopian Textile Industry". The aim of the project is primarily a smooth integration of an economically realizable solar plant for existing steam boiler's feed water. During the course of this project, determining the thermal energy demand was deemed necessary for optimal sizing and operation of the solar plant. This task was difficult to achieve due to the absence of fund to do energy auditing. Neither was average thermal energy use determination possible since the factory was very old and no known specifications are available. These combined factors lead to the idea of predicting energy use patterns from other available related data through an ANNs. These related data are daily production from steam consuming machines and a KWh meter reading of a boiler.

The proposed research work employs the well-known BP algorithm for a multilayer feedforward neural network. Figure 1 depicts the methodology used. The work has taken in to consideration all issues pertinent in practical implementation of these ANNs. To this end a Matlab scrip code was written that incorporates all the above mentioned issues and arrived at acceptable performance during run-time.

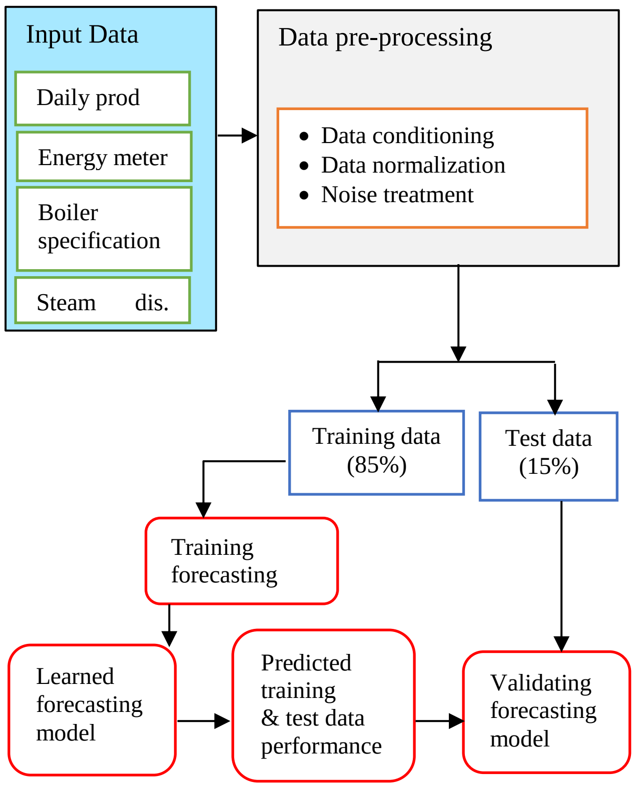


Figure 1 proposed steam consumption prediction

2.1 Fundamentals of Artificial Neural Networks (ANNs)

ANNs are defined as a collection of processing units with networks for interaction with each other through a weighted interconnection [3]. The whole aim of these networks is to replicate, in a rather simplified manner, the workings of a human biological central nervous system. The performance of these ANNs depends, in a not clearly defined manner, on the number, interconnection and interaction of these constituent units.

The aforementioned units are known as neurons. These neurons receive and give input signals to all other units of which they are connected.

A neuron model is shown in Figure 2. The output strength from the neuron is determined from the function f, which itself depend on the value of weight (W) and bias (b) associated with each interconnection. The implementation process begins when an input is presented to the network and propagated through the network as an output by the transfer function otherwise known as activation function. For MLP this process goes on from neuron to neuron and layer by layer through the output layer that process and gives the final value.

In MLP the training is implemented by examples prior to their usage as a useful network. This training attempts to iteratively adjust connection weights and biases using a known training data. To facilitate this training, the outputs from the network are compared to the target examples, which are known as the error performance index (PI). This error is compared and propagated back through the network to adjust weights and biases until an acceptable PI is achieved.

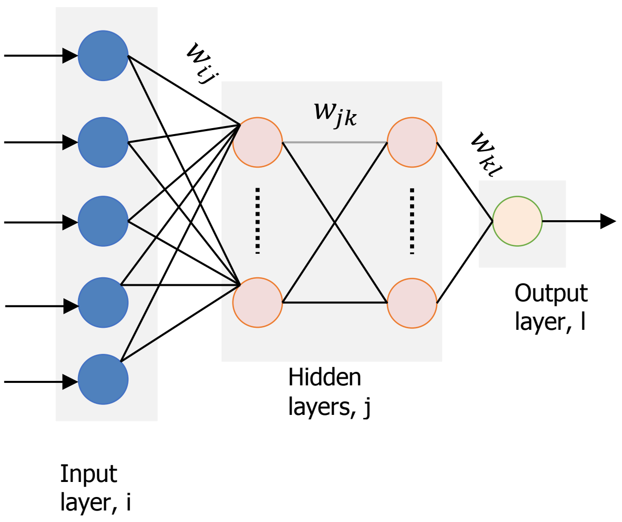


Figure 2 Multi-layer neural network model

The final stage of the network implementation involves fixing the adaptive weights and biases using the last values of the training stage. The network then computes the output directly to give an estimated value for the inputs.

2.1. The MLP Architecture and the Back Propagation (BP) Algorithm

The three-layer MLP network with the associated notation is depicted in Fig. 2.

For MLP the result from preceding layer feeds the following layer which is denoted by

$$a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1})$$
$$for\ m = 0,1, \dots M-1 \quad (1)$$

Where M is the number of layers in the network.

The neurons in the first layer accept network inputs:

$$a^0 = p \qquad (2)$$

The outputs of the network in the final layer are taken as outputs:

$$a = a^M \qquad (3)$$

The target and input to the network are:

$$\{P_1, t_1\}, \{P_2, t_2\}, \dots, \{P_Q, t_Q\} \qquad (4)$$

Where P_Q and t_Q are input and target for the network respectively.

The performance of the network is judged by the mean square error given as

$$F(X) = (t(k) - a(k))^T(t(k) - a(k)) = e(k)^T e(k) \qquad (5)$$

Using the steepest descent algorithm, a formulation for recursive learning of the network is given as

$$W_{i,j}^m(k+1) = W_{i,j}^m(k) - \alpha \frac{\partial F}{\partial W_{i,j}^m} \qquad (6)$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{\partial F}{\partial b_i^m} \qquad (7)$$

Where α is the learning rate.

Since the above error function does not have an explicit relation for the weights in the hidden layer, use of chain rule for derivatives manipulation. The chain rule for a function f with explicit variable n, the derivative for the implicit variable w could be found

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \; x \; \frac{dn(w)}{dw} \qquad (8)$$

$$\frac{\partial F}{\partial w_{i,j}^m} = \frac{\partial F}{\partial n_i^m} \; x \; \frac{\partial n_i^m}{\partial w_{i,j}^m} \qquad (9)$$

$$\frac{\partial F}{\partial b_i^m} = \frac{\partial F}{\partial n_i^m} \; x \; \frac{\partial n_i^m}{\partial b_i^m} \qquad (10)$$

Calculation of the second part of the above equations is now straightforward because there is a simple relation between the net input to layer m and the weights and bias in that layer:

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m \, a_j^{m-1} + b_i^m \qquad (11)$$

Thus

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \frac{\partial n_i^m}{\partial b_i^m} = 1 \qquad (12)$$

Let’s define

$$S_i^m = \frac{\partial F}{\partial n_i^m} \qquad (13)$$

Where S_i^m is the sensitivity i.e. the sensitivity of *F* that is associated with variation in the *i*th element of the net input layer *m*. Employing this definition results in a simpler form for equations (9) and (10) which is:

$$\frac{\partial F}{\partial w_{i,j}^m} = S_i^m a_j^{m-1} \qquad (14)$$

$$\frac{\partial F}{\partial b_i^m} = S_i^m \qquad (15)$$

Thus the steepest descent algorithm can be generalized as

$$W_{i,j}^m(k+1) = W_{i,j}^m(k) - \alpha S_i^m a_j^{m-1} \qquad (16)$$

$$b_i^m(k+1) = b_i^m(k) - \alpha S_i^m \qquad (17)$$

The condensed matrix representation is given by:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - \alpha S^m (\boldsymbol{a}^{m-1})^T \qquad (18)$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - \alpha S^m \qquad (19)$$

Where:

$$S^m = \frac{\partial F}{\partial n^m} = \begin{bmatrix} \frac{\partial F}{\partial n_1^m} \\ \frac{\partial F}{\partial n_2^m} \\ \vdots \\ . \\ \frac{\partial F}{\partial n_{S^m}^m} \end{bmatrix} \qquad (20)$$

Here also the sensitivities S^m will be computed using the chain rule. This computation of sensitivities which are determined from previous layers gave the name backpropagation to the algorithm.

Let’s now define the Jacobian matrix for backpropagation of the sensitivities:

$$\frac{\partial \boldsymbol{n}^{m+1}}{\partial \boldsymbol{n}^m} = \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_1^m} & \cdots \frac{\partial n_1^{m+1}}{\partial n_1^m} \\ \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_1^m} & \cdots \frac{\partial n_1^{m+1}}{\partial n_1^m} \\ . & . & . \\ . & . & . \\ . & . & . \\ \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_1^m} \end{bmatrix} \qquad (21)$$

Now let’s take the i, j element of the above matrix:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial\left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right)}{\partial n_j^m}$$

$$= w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

$$= w_{i,j}^{m+1} \dot{f}^m(n_j^m) \qquad (22)$$

where

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m} \qquad (23)$$

Thus, the Jacobian matrix is given as:

$$\frac{\partial n^{m+1}}{\partial n^m} = W^{m+1} \dot{F}^m(n^m) \qquad (24)$$

Where:

$$\dot{F}^m(n^m) = \begin{bmatrix} \dot{f}^m(n_j^m) & 0 & \ldots 0 \\ 0 & \dot{f}^m(n_j^m) & \ldots 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 0 & 0 & \dot{f}^m(n_j^m) \end{bmatrix} \quad (25)$$

Finally using the chain rule the sensitivities can be given as:

$$S^m = \frac{\partial F}{\partial n^m} = \left(\frac{\partial n^{m+1}}{\partial n^m}\right)^T \frac{\partial F}{\partial n^{m+1}}$$
$$= \dot{F}^m(n^m)(W^{m+1})^T \frac{\partial F}{\partial n^{m+1}}$$
$$= \dot{F}^m(n^m)(W^{m+1})^T s^{m+1} \quad (26)$$

These sensitivities are propagated backward layer by layer till the input layer as:

$$S^M \rightarrow S^{M-1} \rightarrow \cdots \rightarrow S^2 \rightarrow S^1$$

2.2.1 Resilient Gradient algorithm

Although the BP algorithm is the best among the MLP networks, in its basic form it has two major limitations-long learning time and possibility of local minima [1, 3-5]. Thus a variant of the basic BP algorithm known resilient gradient method which is known to remove these drawbacks is utilized. [4, 5]

In this algorithm, only the sign of derivative is used to determine the weight update value. The implementation of this algorithm follows the following rule:

a) If the partial derivative of the corresponding weight has the same sign for the two consecutive iterations, the weight update is increased by a factor say, ɳ+ otherwise
b) the weight update value is decreased by a factor ɳ- else
c) if the derivative is zero, then the weight update value remains same.
d) However, if the weight continues to change in the same direction for several iterations, the weight is increased by its update value otherwise the update value is reduced.

2.3 Implementation of BP Algorithm for steam -consumption prediction

The diagram in Figure 4 depicts the ANNs training procedure followed. This procedure is a continuous iterative process starting from data collection and preprocessing stage to achieve more efficient neural network training. While at this first step, the data were partitioned into training and testing sets. Following this, selection of suitable network type and architecture (e.g., number of hidden layers, number of nodes in these layers) were done. Then choice of appropriate training algorithm from the multitude of available paradigms were carried out to handles the task. Finally, once the ANNs is trained, analysis to determine the network performance was done. This last stage has dealt with some practical issues with the data, the network architecture, and the training algorithm. The whole procedure is then iterated until an acceptable performance is achieved.

2.3.1 Pre-Training Steps

The pre-training steps comprises three separate tasks namely data collection, data Preprocessing, and choice of Network type and architecture.

2.3.1.1 Data Collection

Input data which are actual daily production from all steam consuming machines were collected for the year 2016 in Bahir Dar textile factory. Parts of these data are shown in Figure 3 for first week of August 2016. Further, daily total steam production from an electrical boiler (Collins Walker) was used as an output Data. The existing steam electrical boiler with its specification is given in Table 1. Meter readings for the same year and day as the input data were also recorded. Figure 2, depicts these meter readings for the same days of August 2016.

2.3.1.2 Data pre-processing

The aim of this step is to lay a conducive ground for better network training. Though several data pre-processing steps exit in the literature, this work used feature extraction, normalization, and handling of missing data.

The available data for the ANNs output are meter reading of an electrical boiler. These data show the total electrical energy (KWh) consumed by the boiler. To make these data useful a manipulation to get the total steam delivered at the premises of the steam-consuming machines is done. The procedure is explained as follows:

The total steam delivered at the steam-consuming machines is given by

$$S_B = S_M + S_{loss} \quad (28)$$

Where S_M is the steam delivered, S_B is the total boiler steam produced and S_{loss} is the steam transmission loss

The total daily steam produced by the boiler can be determined from

$$S_B = bx\frac{B_{KWh}}{B_{KW}} \qquad (29)$$

Table 1: Production rate vs boiler meter reading

Where B_{KWh} is the daily electrical energy consumed by boiler, B_{KW} is the rated boiler power that relates to boiler steam production b in Kg as given in boiler specification Table 1.

Table 1 Boiler specification

Specification Description	Specification Value
Name and Type	COLLINE, electrical boiler
Permissible & working pres.	13 bar, 10.3 bar
Design & Max Steam temp.	190°C, 184°C
Rated steam output	3348Kg/hr./boiler
Power consumption	2106KW/boiler

The steam loss could range from 5-20% of the steam produced [6]. In the current model, a stochastic representation of this loss as a uniform distribution of the minimum and maximum values was used. This was done to reduce the uncertainty of quantifying the steam loss in the several varying steam distribution networks.

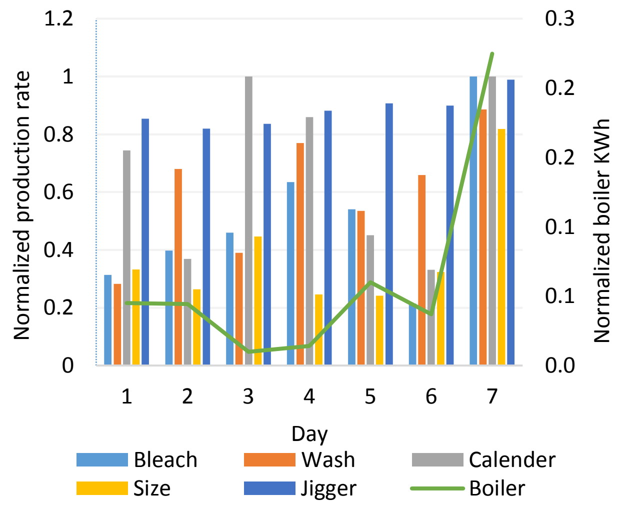


Figure 3 Daily production rates from steam consuming machine, 1st week, 2006

It is reported in [7-8] that rescaling or normalization of training data improves the learning and convergence of a network. The normalization procedure used in this work aims to adjust the data so that they have a specified mean and variance — typically 0 and 1. This can be done with the transformation

$$D_n = \frac{D - D_{min}}{D_{max} - D_{min}} \qquad (30)$$

where D_{min} is the minimum of the input vectors in the data set, and D_{max} is the maximum value. Practically what this normalization does is to shift zero of the scale and normalize the standard deviation of the data. Also shuffling of these data were done to decrease the effect of learning of the network for similar sets of data at the expense of another.

Because of limited data, we just can't afford to simply throw out missing data. Rather, two strategies were used depending on whether the missing data was from input or output. When there was a missing input data, a flag to know this data (either a 1 or 0) were set and a replacement of this missing component with the average values of the input data were carried out. Instead when a missing data was present at the output a modification of the error performance was done in such a way that, for this particular data the performance calculation was skipped to nullify its contribution to learning process.

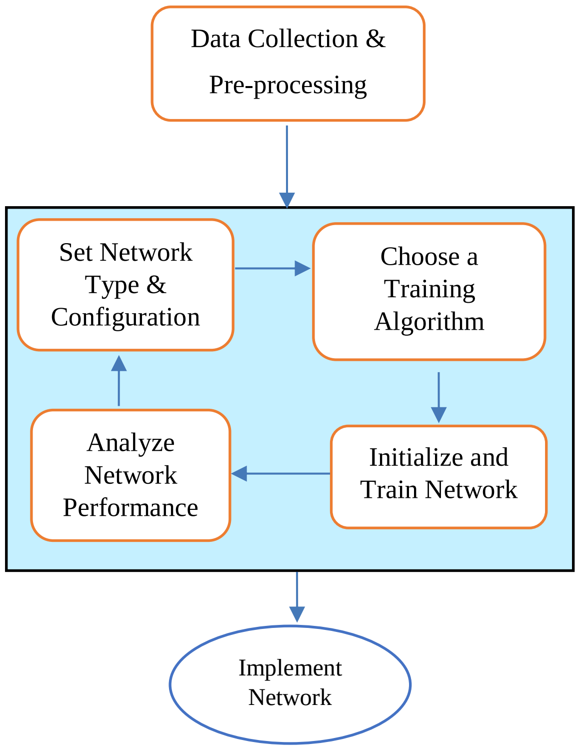

Data Collection & Pre-processing
Set Network Type & Configuration
Choose a Training Algorithm
Initialize and Train Network
Analyze Network Performance
Implement Network

Figure 4 implementation procedure of the proposed neural network prediction model

Finally, the collected data was divided in to two sets: training, and testing. The training set made up

85% of the full data set, with testing making up the remaining 15% each. Caution to make each of these sets representative of the full data set — that the test sets cover the same region of the input space as the training set were considered. For this, selections of each set from the full data set were done.

2.3.2 Choice of Network Architecture

The universally accepted network architecture for fitting problems is the multilayer perceptron [1-3]. It was shown in [3] that this standard neural configuration uses tansig function in the hidden layers, and linear function in the output layer. This is because the former function produces outputs (which are inputs to the next layer) that are centered near zero, whereas the later function always produces positive outputs.

The choice of the optimum number of hidden units depends on many factors whose interactions are not easy to understand. These factors are amount of training data, number of input and output units, the level of generalization requirement from the network, type of transfer function and the training algorithm [9]. Conflicting trends are observed when the number of hidden units vary i.e. too few leads to under-fitting while too many results in over-fitting and slow learning process. However, it is highly unlikely to use more than two hidden layers for a standard function approximation problem [3].

To fix the number of neuron in the hidden layer, different authors suggest a rule-of thumb from their experiences. In [10] it is given as

$$n = \sqrt{n_i + n_o} + a \qquad (31)$$

Where n is the number of hidden neurons, n_i and n_o are number of neurons in input and output and a is a constant between 1 and 10.

Another work [11] suggested to use

$$N_h = N_p x \sqrt{(N_i + N_o)} \qquad (32)$$

Where N_h is hidden neuron numbers, N_p is number of training samples, $N_i \& N_o$ are input and output neurons.

The authors strongly believe that the best way is to try multiple runs for a range of different hidden layers with different neurons in each layer and observe the network performance. For the current work, two hidden layers with ten neurons in each layer achieved the set performance criterion.

2.3.3 Weight Initialization

ANNs weights should be initialized with small random values. Since the BP algorithm work on the weights in a similar fashion, initializing these weights alike will eventually make all units learn in the same way [12-14]. Similarly, these small random values will result in network output that corresponds to highest weight update [13]. In this work, effort has been made to make the performance of the final trained neural network independent of the choice of initial weight values. For that several runs of the network for different initial weight values were performed that has resulted in similar performance.

2.3.4 Choice of Training Algorithm

For multilayer networks to perform function approximation, the resilient gradient descent training algorithm provides a guaranteed performance minimization of the error function with relatively fast convergence rate [4, 14, 16]. In this work, this algorithm was tested to check its validity for the task at hand.

2.3.5 Stopping Criteria

For the majority of practical neural networks, the training error never converges identically to zero. As a result, other criteria for deciding when to stop the training is generally considered. There are several methods reported in the literature such as stooping when the performance index reaches a certain level, setting a high training iteration number, training for a fixed iteration then restarting the training with initial weights from previous training and stopping when the gradient of the performance index is sufficiently low [17-18]. For this work a stopping criterion when either the performance index is met or when a large number of iteration reached is implemented for the simple reason it met the practical requirement of the task.

2.3.6 Post-Training Analysis

Prior to concluding the work, analysis of the trained network to see if the training was successful is necessary. A powerful method of doing this is to do curve fitting for regression between the trained network outputs and the corresponding targets [3]. For that, we fit a linear function of the form

$$a_q = mt_q + c + \varepsilon_q \qquad (33)$$

where m & c are the slope & offset, respectively, of the linear function, $t_q$is a target value, a_q is a trained network output, and ε_q is the residual error of the regression.

The terms in the regression can be computed as follows:

$$\hat{m} = \frac{\sum_{q=1}^{Q}(t_q - \bar{t})(a_q - \bar{a})}{\sum_{q=1}^{Q}(t_q - \bar{t})^2} \qquad (34)$$

$$\hat{c} = \bar{a} - \hat{m}\bar{t} \qquad (35)$$

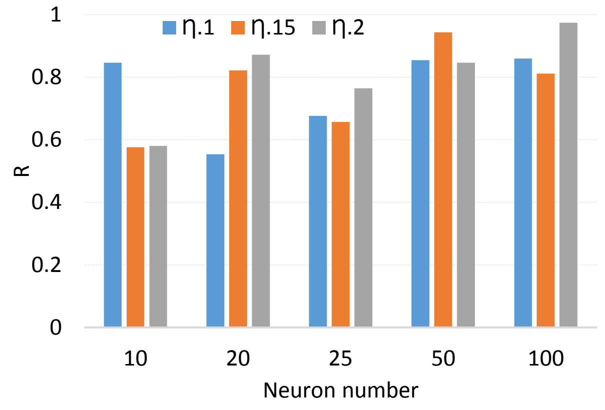
Where,

$$\bar{a} = \frac{1}{Q}\sum_{q=1}^{Q} a_q, \qquad \bar{t} = \frac{1}{Q}\sum_{q=1}^{Q} t_q$$

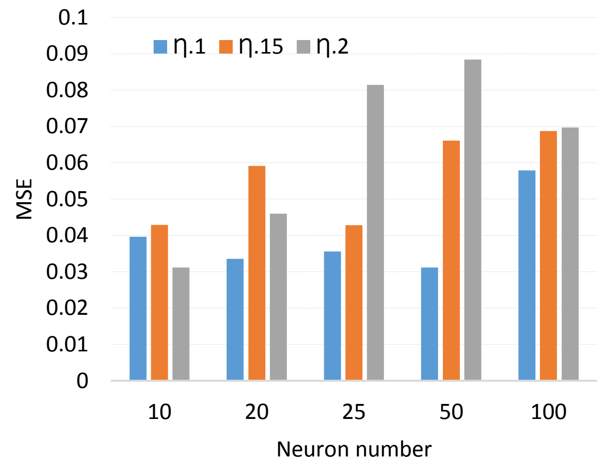
A plot of this fitting to gauge the performance of the proposed ANNs is discussed in the results section.

3 Results and Discussions

A Matlab script file for the implementation of resilient gradient variant of the BP algorithm were written. This code was run for different learning rates and varying number of hidden neurons. The regression coefficient (R) and Mean Square Error (MSE) were compared. As can be seen from Figure 5 the resilient gradient method shows superior performance as the complexity of the neural network increase.



(a)



(b)

Figure 5 effect of learning rate and neuron number variation on (a) correlation and; (b) mean square error

Next we will consider performance of the best resilient configuration. Figure 6 shows the regression analysis where the solid line represents the linear regression, the thin dotted line represents the perfect match, and the circles represent the data points. From this figure it is possible to see that the match is good, although not perfect. There are few points that seem to diverge from the regressed line. This might rise due to the presence of an incorrect data point, or because the data is far from other training points. The latter is the case here since the data used is not representative of all input space. Analysis of the scatter plot as shown in Figure 7 clearly shows the case.

Addition of points that span the whole data space will improve the generalization capability of the proposed neural network. Additionally, the correlation coefficient between the estimated and target values, which is the R value was computed. Generally, the R value varies from –1 to 1, however it is should be closer to 1 for prediction applications of BP algorithm. R=1 means all of the data points lie exactly on the regression line & R=-1 means they are randomly scattered away from the regression line. For this case as can be seen from Figure 6, the data does not fall exactly on the regression line, but the variation is very small.

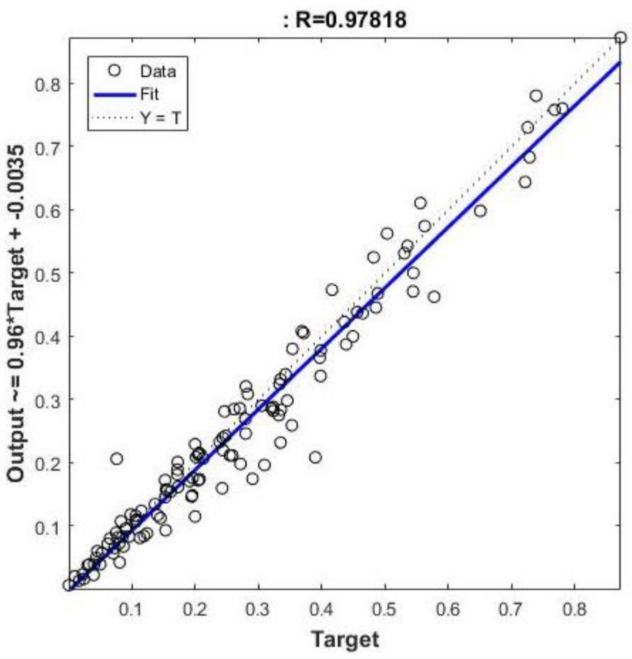


Figure 6 Correlation factor for the optimum neural model

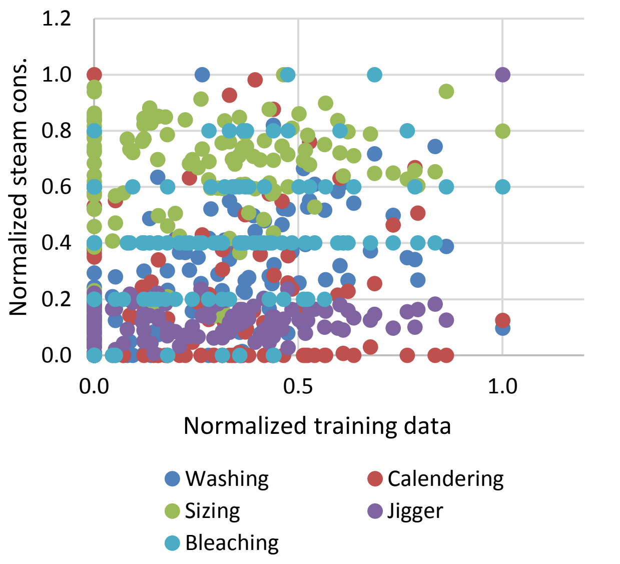


Figure 7 scatter plot of the training data and the steam consumption

The MSE values for the best network configuration are given in Figure 8. As can be seen from this figure, the error overshoot at the start of the training and subsequently receded to a stable lower value. As stated in section 2, the stopping criterion was based on the mean square error that minimized the actual target and output of the network. Although the neural model achieved a relatively minimum values around 300 iteration, further increment was done to get better result with the correlation coefficient. This trade-off is considered acceptable since the overall neural model error is significantly low about 0.0674.

Figure 9 gives the final trained neural network output after a test data was presented. The variation of neural output is due to variation of daily production rates of steam consuming machines.

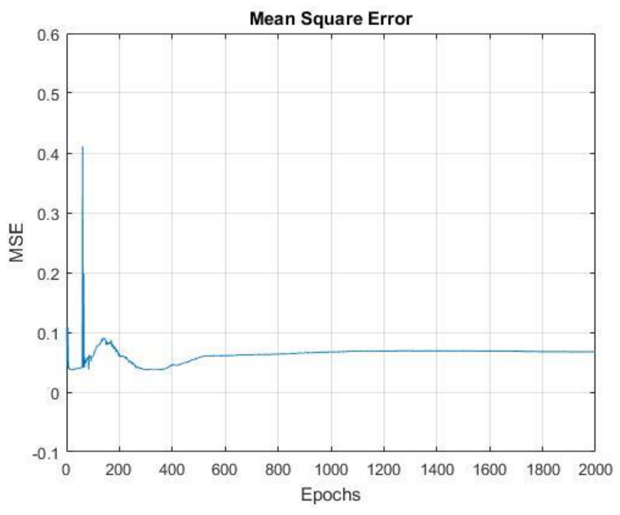


Figure 8 MSE of the optimized neural model

Table 2 summarizes the final result i.e. the steam consumption rate of each textile machine. For this the average production value of the machine is presented to the network as input. The output value is given in a range because of the random stochastic nature of steam loss and weight initialization used.

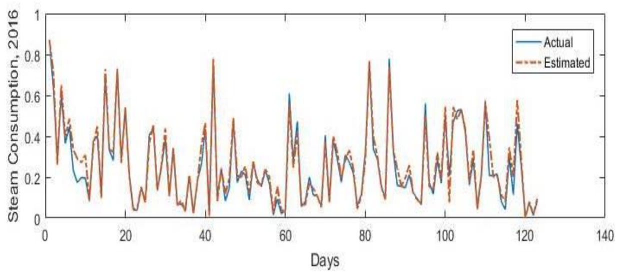


Figure 9 output of trained neural model using test data

Table 2 final steam consumption estimates

Industrial process	Steam consumption (kg/kg)
Bleaching	0.6-0.9
Washing	0.7-1.1
Calendaring	0.8-1.4
Jigger	1.2-4.5
Sizing	7.8-9.0

4 Conclusion

In this research paper, effort has been made to estimate industrial steam consumption form machines daily production rates and boiler meter reading. The neural algorithm used is explained in detail with the associated practical issues of implementation. Several simulations run was carried out in Matlab to arrive at an optimum neural configuration. Finally real textile factory data was used for training and test of this final optimized neural model.

From the simulation results of a Matlab code implementation, it was found out that the resilient gradient descent algorithm of an MLP is a valuable tool for function approximation such as energy use prediction. However, practical considerations that relates to pre-processing as well as selection of representative input data were found to be a prerequisite before implementation.

Moreover, it was found out that the number of layers and the amount of neurons in those layers has a direct influence on the accuracy of the network. From the experiment it was found out that two hidden layers and hundred neurons on those layers has resulted in best performance of the network. However, the number of neurons in a layer could be reduced with the availability of more data to train the network.

References:

[1] Yu-Rong Zeng, Yi Zeng, Beomjin Choi, Lin Wang. Multifactor-Influenced Energy Consumption Forecasting Using Enhanced Back- propagation Neural Network. Energy, 2017; 127:381-396.

[2] Uzlu E, Kankal M, Akpınar A, Dede T. Estimates of energy consumption in Turkey using neural networks with the teaching–learning-based optimization algorithm. Energy 2014; 75: 295-303.

[3] Martin T. Hagan, Howard B. Demuth. Neural Network Design 2nd Edtion, 2014.

[4] Alaa Ali Hameed, Bekir Karlik, Mohammad Shukri Salman. Back-propagation Algorithm with Variable Adaptive Momentum. Knowledge-Based Systems,2016; 114:79-87.

[5] C.G. Looney, "Advances in feedforward neural networks: demystifying knowledge acquiring black boxes", IEEE Transactions on Knowledge and Data Engineering, Volume: 8, Issue: 2,1996

[6] "Energy Audit of Bahir Dar Textile Share Company, Ethiopia", Bangalore: The Energy and Resources Institute; 53 pp., Project Report No. 2013IB22, 2014

[7] J. Sola, "Importance of input data normalization for the application of neural networks to complex industrial problems", IEEE Transactions on Nuclear Science, Volume: 44, Issue: 3, 1997

[8] Zhang Q., Sun S. Weighted Data Normalization Based on Eigenvalues for Artificial Neural Network Classification. In: Leung C.S., Lee M., Chan J.H. (eds) Neural Information Processing. ICONIP. Lecture Notes in Computer Science, Springer, 2009; 5863.

[9] N. Murata, S. Yoshizawa & S. Amari, "Network information criterion-determining the number of hidden units for an artificial neural network model", IEEE Transactions on Neural Networks, Volume: 5, Issue: 6, 1994

[10] Saduf Afzal, Mohd. Arif Wani "Comparative Study of Adaptive Learning Rate with Momentum and Resilient Back Propagation Algorithms for Neural Net Classifier Optimization"

[11] Wahed, M. A "Adaptive learning rate versus Resilient back propagation for numeral recognition" Journal of Al-Anbar University for Pure Science, 94-105,2008

[12] D. Erdogmus. Accurate initialization of neural network weights by backpropagation of the desired response. Proceedings of the International Joint Conference on Neural Networks, 2003

[13] Go J., Baek B., Lee C. Analyzing Weight Distribution of Feedforward Neural Networks and Efficient Weight Initialization. In: Fred A., Caelli T.M., Duin R.P.W., Campilho A.C., de Ridder D. (eds) Structural, Syntactic, and Statistical Pattern Recognition. Lecture Notes in Computer Science, Springer, 2004;3138

[14] Weipeng Cao, Xizhao Wang, Zhong Ming, Jinzhu Gao, A Review on Neural Networks with Random Weights, Neurocomputing, 2017; In Press, Corrected Proof — Note to users.

[15] E. Barnard, "Optimization for training neural nets," IEEE Trans. on Neural Networks, vol. 3, no. 2, pp. 232–240, 1992.

[16] T. P. Vogl, J. K. Mangis, A. K. Zigler, W. T. Zink and D. L. Alkon, "Accelerating the convergence of the backpropagation method," Biological Cybernetics., vol. 59, pp. 256–264, 1988.

[17] Liu, Y., Starzyk, J.A., Zhu, Z.,. Optimized approximation algorithm in neural networks without overfitting. IEEE Trans. Neural Networks 19 (6), 2008; 983–995.

[18] Masoud Yaghinin, Mohammad M. Khoshraftar, Mehdi Fallahi. A hybrid algorithm for artificial neural network training. Engineering Applications of Artificial Intelligence, 2013:26:293-301.