Predictive Models of Buildings Energy Consumption

Yaser Imad Mohammed Al-amin

M.Sc. in Electronics and Telecommunications Engineering

Master thesis dissertation supervised by:

Professor Doctor Antonio RUANO

2015

Predictive Models of Buildings Energy Consumption

Yaser Imad Mohammed Al-amin

M.Sc. in Electronics and Telecommunications Engineering

Master thesis dissertation supervised by:

Professor Doctor Antonio RUANO

2015

Statement of Originality

Predictive Models of Buildings Energy Consumption

Statement of authorship: The work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text. The material has not been submitted, either in whole or in part, for a degree at this or any other university.

Candidate:

(Yaser Imad Mohammed Al-Amin)

Contents

List of Figures

List of Tables

Acknowledgements

To my supervisor, Professor Doctor Antonio RUANO, for all the help on the definition of the object of study, the requirement of methodology and accuracy, tireless scientific guidance for the critical review, comments, clarifications, opinions and suggestions, for letting indication of relevant literature, the accessibility, warmth and friendliness shown by the confidence that has always given me and for following along this journey.

To all the Professors who followed my academic journey, for what they taught me and mainly by research methodology that instilled me.

To my mother and father for the solid education given till my youth, what provided the continuity of studies till this Masters. To all my family members, in particular to my sister for the support during this master.

To all, I reiterate my appreciation and my eternal gratitude.

Resumo

O rápido crescimento do consumo energético no mundo levanta preocupações sobre as dificuldades de fornecimento, a exaustão dos recursos energéticos e os pesados impactos ambientais (desgaste da camada de ozono, aquecimento global, alterações climáticas, etc.), assim como os efeitos económicos. Devido a esta razão, eficiência de energia em edifícios é o principal objectivo das políticas de hoje em dia, a nível regional, national e international.

Previsões do consumo energético da rede elétrica de um edifício é interessante para investigadores de muitas areas, e existem muitos algoritmos que permitem a previsão energética para diferentes horizontes. Redes neuronais artificiais de função de base radial (RBF ANN) são uma das abordagens mais testadas com resultados mais satisfatórios.

O uso de um algoritmo genético multi-objectivo para optimizar o desempenho da ANN mostrou melhoramentos fascinantes nos últimos anos.

Este trabalho procura a introdução modelos preditivos para consumo energético em edifícios. A disposição do trabalho foi dividido em varias partes, nomeadamente redes neuronais de perceptron multicamadas(MLPs) e de função de base radial (RBFs) treinadas e testadas para previsão, seguido pela aplicação de um MOGA de forma a obter um óptimo numero de neurónios para a rede de função de base radial e para as entradas.

Palavras-chave: Consumo energético do edifício, Redes Neuronais Artificiais, perceptron multi-camadas, funções de base radial, Algoritmo genético multi-objectivo.

Abstract

The rapidly growing world energy use has already raised concerns about supply difficulties, exhaustion of energy resources and heavy environmental impacts (ozone layer depletion, global warming, climate change, etc.), as well as economical effects. For this reason, efficient energy in buildings is today's a prime objective for energy policy at regional, national and international levels.

Buildings electrical energy consumption forecasting is interesting for researchers from many areas, and there are many algorithms to predict the energy, over different horizons. Radial Basis Function Artificial Neural Network (RBF ANN) are one of the most tested approaches with satisfactory results.

Optimizing the performance of the ANN, by multi-objective genetic algorithm (MOGA), have shown a fascinating improvement in the last years.

This work aims to introduce predictive models for buildings energy consumption. The work setup has been divided into several parts, namely MultiLayer Perceptrons(MLPs), and Radial Basis Function (RBFs) neural networks trained and tested for prediction, then MOGA applied to obtain the optimum inputs and number of neurons for the RBFs NN.

Keywords: Building energy consumption, Artificial Neural Networks, Multi-layer perceptrons, Radial Basis Functions, Multi-Objective Genetic Algorithm.

Chapter 1

Introduction

Electricity consumption and its economical costs have been increasing significantly over the years, as the demand has increased rapidly, and as electricity cannot be stored and conserved efficiently and easily. For these reasons, methods that enable the forecasting of energy consumed, and their applications, has become a actively researched problem.

In order to minimize the waste of energy, researchers from various fields of knowledge have joined efforts and created ways to improve the agreement between the production and consumption of energy. Over the time, this task became harder because of the increase in electricity consumption, derived from the increase in electric and electronic equipment usage, the global warming and the proliferation of the microgeneration of energy. Furthermore, another factor that hampers the achievement of results closer to reality is the fact that in many tested approaches, the variables that contribute to the variations of the electricity consumption like temperature, humidity, season of the year, weekdays, holidays and others, are not taken into account. And the introduction and massification of the use of new electric and electronic technologies, has made this task even harder.

Some studies take these variables into account, but many studies do not, like in [3], therefore obliterating the chances of improving the results. These studies of forecasting are in intersect of the power companies, as well as facilities.

1.1 Electricity load demand

To effectively accommodate the use of solar powered systems for the decrease of electricity bill of the University of Algarve, Electrical Demand forecasting is a necessary tool to effectively measure the benefits of applying solar power installation, in term the of the energy bill payed by University. This is not an easy task because the electricity demand has been increasing over the years and because electricity consumption patterns vary with many factors including time. Therefore, one of the goals is to get forecasts very close to the reality in order to have an efficient supply system.

To make on-line adaptation for forecasting model, "a one step" and "multi steps" (over a prediction horizon) accurate model should be available and implemented.

Non-linear Auto-Regressive with eXogenous (NARX) inputs model structures are usually considered, In this thesis, only one eXogenous input was considered, which tries to encode the occurrence of events perturbing the daily and weekly patterns of electricity consumption [4].

1.2 Proposal of dissertation

The aim of this MSc work is to design predictive models, to forecast over a prediction horizon, the energy consumption of one of the campi of Algarve University, the Penha campus. The data of energy consumption for the Campus are available, an existing Multi Objective Genetic Algorithm (MOGA) [4] is used for designing the predictive models, and the approach used in a contract with Rede Eletrica Nacional(REN) for the prediction of the Portuguese Energy Consumption is used as well [3] [5].

1.3 Objective of the thesis

To design a model, for forecasting the electrical load demand of the Penha campus as it is needed in future solar installation projects of the university.

The historical data for one year is taken, with interval of fifteen minutes, and they were be used to train neural networks that predicts one day ahead of ELD.

The methodology uses one-step-ahead ELD predictive models using computational intelligence methodologies, which are then iterated in a multi-step fashion to obtain the required prediction consumption over the horizon needed. Two approaches are used: first by training several ANN with different topologies, and selecting the best model from the topologies considered; secondly by using MOGA to design the model.
The implementation of the predictive model using artificial neural networks, namely multilayer perceptron (MLP), and Radial Bais Function (RBF), is consider in the first step. By designing and training, MLPs and RBFs, that can be used for prediction of one day horizon. In the subsequent phase, the model structure (number of neurons and input terms) has been evolved using MOGA, by considering as inputs the lags of the power target, as well as one extra input to code the day type.

1.4 Layout

This dissertation is divided into seven chapters. This chapter (chapter 1) is the Introduction to the thesis topic. The second chapter presents the state-of-the-art of ELD forecasting using ANNs, then chapter 3 will show the methodologies employed in this dissertation regarding neural network implementation and optimization. Subsequently, experiments details will be explained in chapter 4, 5 and 6. Chapter 7 will discuss the results; and the conclusion and possible future work will be drawn on the final chapter.
The chapters are then split as:

- Chapter 1: Introduction
- Chapter 2: The state of the art of related work
- Chapter 3: Methodologies
- Chapter 4: MLP Experimental work
- Chapter 5: RBF Experimental work
- Chapter 6: Applying multi-objective genetic algorithm

- Chapter 7: Results
- Chapter 8: Conclusion and future directions

Chapter 2

The state of the art and related work

2.1 Power Forecasting

Knowing the next step ahead makes major rule in your decision, forecasting is very important in many aspects of our life. Power forecasting has received a great deal of attention due to its importance for planning and operations of any power system.

The use of ELD forecasting conquered a big interest from power grid companies, smaller power systems like solar system [6], wind power system [7], and researchers from different areas. The forecasting depends on many variant factors, in different systems, which makes this task very hard.

In recent years, significant emphasis on the energy savings can be observed. Out of the overall primary energy consumption, up to 40% is consumed in the building sector and more than half of this energy is spent on HVAC (Heating, Ventilation and Air Conditioning) systems [8]. Historical data may be of extreme importance in demand forecasting and its preparation is an important aspect. Over the past decades, numerous investigations have been conducted to improve the accuracy of ELD forecasting.

Demand forecasting is concerned with the prediction of hourly, daily, weekly or annual values of consumption and peak demands [9]. These forecasts are categorized in general as short term , medium-term and long-term forecasting depending on the prediction horizon [10].

The forecasting step time depends on the interval of the sampling data, which along with the size of area covered, are important for accuracy of the forecasting [11], and of-course this

resolution aspect will have different effects on the applications.

In short-term load forecasting, generally, weather conditions (particularly temperature in different months of the year) have significant influence on loads [12], and in long-term forecasts economic factors play an important role [13].

Different techniques are used to forecast the energy consuming, like least squares support vector machines LMS-VM [14], self-organizing maps SOM [15], ARMA [16] and ARIMX [17], data mining techniques [18], and hybrids of NN and fuzzy logic, known as Adaptive Network-based Fuzzy Inference System (ANFIS) [7].

Energy estimation models [19] can be classified as:

- Engineering:

  Forward

  Calibrated

- Statistic:

  Regression

  Intelligent:

  - Genetic Algorithm
  - Neural Network
  - Support Vectors Machine

- Hybrid

Building energy consumption prediction is often needed in the evaluation of building performance, optimization of building operations, fault detection and diagnosis and demand side management for smart grid.

In this thesis, MLPs and RBFs ANNs, are applied for predicting the ELD, and MOGA is used for selection of inputs and topology determination.

Meteorological data effect directly on the forecasting, and could be important input to the ANN system, although some researchers have showed that some cases that could be done without these data [3] [20] [21] [2] .

The forecasts prediction horizon are categorized in general [22] as:

- short-term (few hours to a few weeks)
- medium-term (few weeks to a few months and even up to a few years)
- long-term forecasting (from 5 to 25 years)

Artificial neural networks are widely accepted as a technology offering an alternative way to tackle complex and ill-defined problems. They can learn from examples, are fault tolerant in the sense that they are able to handle noisy and incomplete data, and are able to deal with nonlinear problems, and, once trained, can perform prediction and generalization at high speed.
In this work involve two type of ANN: MLPs and RBFs.

2.2 MLP

The Multilayer Perceptron (MLP) is a standard feed forward artificial neural network [23]. Due to its excellent nonlinear matching and generalization abilities, the MLP has been generally applied in several engineering tasks. MLP is used for forecasting alone [24] or in conjunction with other techniques such as hybrid systems [25].
MLP used in several topic related applications, for example, global irradiation forecasting [24], and for solar system power produce forecasting [26].

2.3 RBF

Radial bais function neural network is also applied to many application, like solar or wind power system [9], or fault section estimation system [27], as well as in ELD [3] [5], off-line or on-line [4].

2.4 MOGA

To solve a problem of two vague notions, a formal problem definition for each notion is needed. On one hand the concept of best ANN mapping requires the definition of best. On the other, the sentence considering the application at hands implies that the problem will be solved by taking the application into account. In fact the two notions are related as it seems appropriate to define what is a best ANN.

Considering the application at hand, select d ($d_m < d < d_M$) input features from the set F, a suitable number of neurons n ($n_m < n < n_M$), and compute the ANN parameter vector w, such that the best ANN mapping, given below, is obtained [2].

$$y_k = g(x_k; w)$$

ANN parameters : Includes their computation by means of a training algorithm. (The quality measures should reflect how well did the training stage performed and how good is the mapping obtained by the parameters computed).

ANN structure : Relates to the network topology. Includes the selection of suitable inputs and an appropriate number of neurons. (The quality measures should tell how fit is the ANN structure for the application at hands) [5].

Multi-objective genetic algorithm is used for several optimization problems. In the scope of this thesis it is used in the selection of the topology and the inputs feature of the ANN, following the works described in [28] [5] [29].

Chapter 3

Methodologies

In this chapter the main concepts implemented in this master thesis will be addressed, and will focus on: multilayer perceptrons (MLP) and radial basis function (RBF) ANNs applied to ELD; additionally a simple background about genetic algorithm (GA) and multi-objective genetic algorithm (MOGA) will be given, and how it could be used to select parameters for ANN. A brief explanation about training algorithms, such as Levenberg-Marquardt(LM), k-mean algorithm, and other concepts which were used in this work will be introduced.

3.1 Artificial Neural Networks

In machine learning, and cognitive science, artificial neural networks (ANNs) are a family of statistical learning models inspired by biological neural networks (the central nervous systems of animals, in particular the brain), and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which send messages to each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning from historical data.

ANNs are used in many disciplines as neuroscience, mathematics, statistic, physics, computer science and engineering to solve forecasting, optimization, pattern recognition, associative memory and other problems.

Neural networks are defined as a massively parallel distributed processor made up of simple

processing units to store experimental knowledge and making it available for use [30].

ANNs can be seen as weighted directed graphs where the neurons are nodes and the directed edges (with weights in each) are connections between input and output neurons. They are used for non-linear mapping between the input data X and the output vector y in order to model relations or detect patterns between them.

Neural networks (NN) are black-box models, meaning that both their parameters and structure need to be determined from data. It is clear that the artificial neural networks design problem could be separated in two distinct sub-problems, each effecting different aspects of the design: ANN parameters and ANN structure.

In the applications where the goal is to create a system that generalizes well in unseen examples, the problem of over-training appears. There are two schools of thoughts for avoiding this problem: The first is to use cross-validation and similar techniques to check for the presence of over-training and optimally select hyper-parameters such as to minimize the generalization error. The second is to use some form of regularization. This is a concept that emerges naturally in a probabilistic framework, where the regularization can be performed by selecting a larger prior probability over simpler models; but also in statistical learning theory, where the goal is to minimize over two quantities: the 'empirical risk' and the 'structural risk', which roughly corresponds to the error over the training set and the predicted error in unseen data due to over-fitting [1].

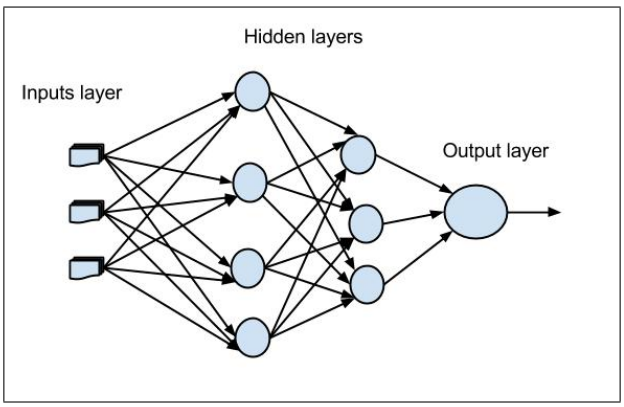


Figure 3.1: Illustration of Neural Network structure [1]

The class of feed-forward ANNs include, among others, radial basis function (RBF) networks, multi-layer perceptrons (MLPs), B-spline networks, wavelet networks, and some types of

neuro-fuzzy networks. Importantly, a common topology of these architectures share the property of parameter separation, i.e., they can be regarded as a non-linear/linear topology because one or more hidden layers of non-linear neurons are followed by a linear combination of neuron outputs to produce the network overall result. It is commonly accepted that gradient-based algorithms, in particular the Levenberg-Marquardt (LM) algorithm are the best methods for training ANNs.

The MLPs, and RBF will be considered here, due to it relative simplicity, and their good performance.

3.1.1 Multilayer perceptrons

MLPs are the most widely known type of ANNs. It has been shown that they considered universal approximators [31], both with one or more hidden layer [32] [33].

MLPs have input, hidden, and output layers, each could contain several neurons with typically sigmoid functions. The neural output is typically a weighted sum of the activation of the neurons in the last hidden layer, which signifies that the activation function of the neurons in the output layer is linear.

Typically, learning algorithms require the calculation of the network's derivatives with respect to its parameters and this is not practical when using the original Perceptron. An alternative is to replace the sign function of the perceptron with a smoother and differentiable non-linearity, such as a sigmoid or a hyperbolic tangent function. This modified Perceptron is then applied as processing units in the Multilayer Perceptron which is a feedforward multilayer ANN. A simple structure of MLP ANN is shown in figure 3.2.

MLPs refers to the kind of feedforward artificial network consisting of a set of sensory units (source nodes or source neurons) that constitute of the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. Neurons in any layer of the network are connected to all the neurons in the previous layer through parameters commonly called weights. The input signal propagates through the network in a forward direction, from left to right and on a layer-by-layer basis.

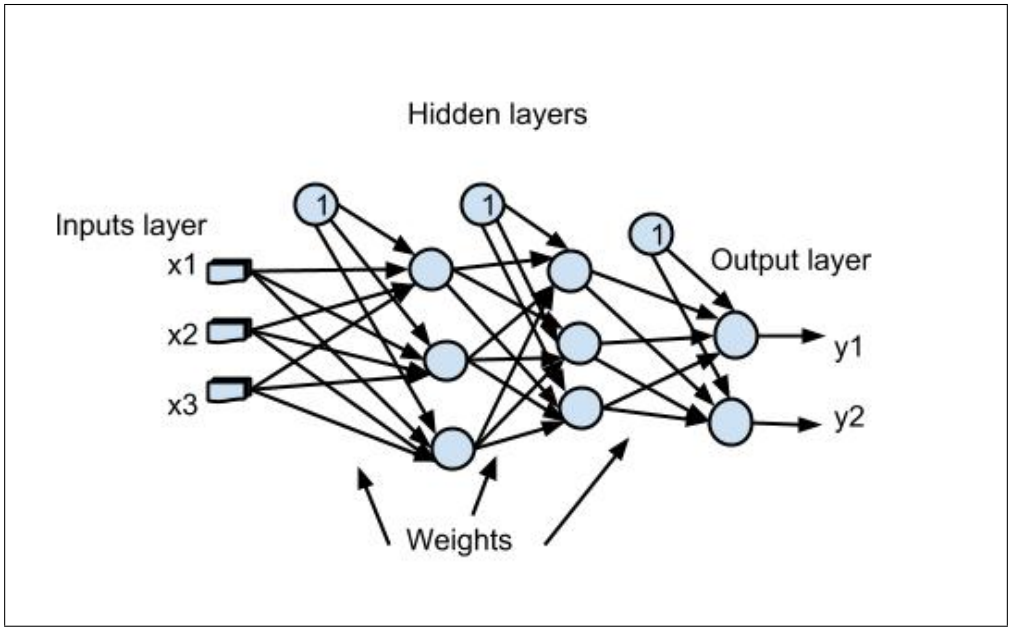


Figure 3.2: Structure of a Multi-Layer Perceptron Neural Network [1]

3.1.2 Radial Basis Functions

Like the MLP, the RBF is a feed-forward neural network. RBFs were first introduced in the framework of ANNs by Broomhead and Lowe [34], where they were used as functional approximators and for data modeling. Since then, due to the faster learning associated with these neural networks (relatively to MLPs) they were more and more used in different applications.

RBFs are a three-layer network, where the first layer is composed by buffers (or input layer), the hidden layer consists of a certain number of neurons whose design parameters are called the centers and the spreads, and whose activation function describe a radial distance between the inputs and the centers.

The output layer is just a linear combiner.

The modeling capabilities of this network are determined by the shape of the radial function, the number and placement of the centers, and the width (spread) of the function. Several different choices of functions are available for the function, like radial linear function, radial cubic function, Gaussian function, thin plate spline function, multi-quadratic function, inverse multi-quadratic function, shifted logarithm function can be used. Gaussian function

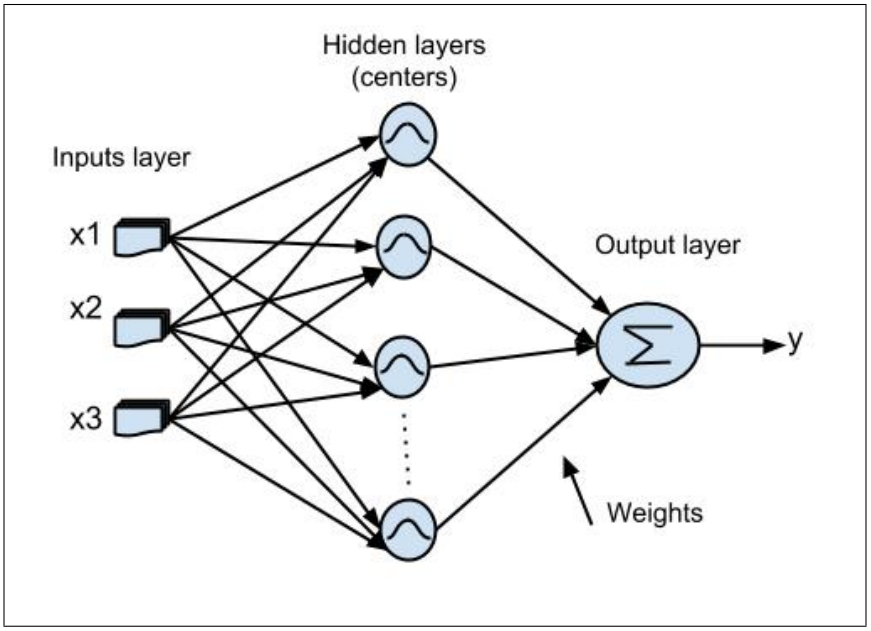


Figure 3.3: Structure of a Radial Basis Function Neural Network [1]

are the most used one [1]:

$$f_i(x) = e^{\left(-\frac{||c_i - x||_2^2}{2\sigma_i^2}\right)} \tag{3.1}$$

And can be written as a product of univariate Gaussian functions:

$$f_i(x) = \prod_{j=1}^{n} e^{\left(-\frac{||c_i - x||_2^2}{2\sigma_i^2}\right)} \tag{3.2}$$

The output of an RBF can be expressed as:

$$y = \sum_{i=1}^{p} w_i f_i \cdot (||c_i - x||_2) \tag{3.3}$$

Where w_i is the weight associated with the i^{th} hidden layer node .

3.1.2.1 RBF parameters

Later in section 3.1.4, the output weights act as a linear combiner, and can be determined in just one iteration. The learning problem is therefore associated with the determination of

the centers and spreads of the hidden neurons. Training methods can be divided into four classes:

(1) *Selected centers at random* : one way of determining the centers is to choose them randomly, within the range of the training set. In this case, the Gaussian standard deviation is usually taken as:

$$\sigma = \frac{d_{max}}{\sqrt{2m_1}} \tag{3.4}$$

where d_max is the maximum distance between centers and m_1 is the number of centers. The linear weights are computed as a least-squares solution.

(2) *Supervised selection of centers and spreads*: The linear weights,the centers, and the spread of the basis functions are computed using derivative-based algorithms. This way, the weight vector, w, the center matrix, C, and the spread vector, σ , are all parameters to be determined by one of the training methods.

(3) *Regularization*: this method comes from regularization theory.

(4) *Self-organized selection of centers*: The first methods of selection need large training set for a satisfactory level of performance. An hybrid learning process [1] consists of:

(a)Self-organized learning step - Find the centers of the radial basis functions.

(b)Supervised learning step - Find linear output weights.

The K-mean algorithm is normally used to find the centers. To understand this algorithm, let m denote the number of patterns in the training set, and n the number of radial basis functions.

The k-means clustering algorithm is presented below:

Algorithm 1 k-means clustering

1: Initialization - Choose random values for the centers; they must be all different

2: **while** on **do**

3: * Sampling - Find a sample vector $x(j)$ from the input matrix

4: * Similarity matching - Find the center closest to $x(j)$. Let its index be $k(x)$:

$$k(x) = argmin_i ||x(j) - c_i||_2 \quad , \qquad i = 1, ..., n$$

5: * Updating - Adjust the centers of the radial basis functions according to:

$$c_i[j+1] = \begin{cases} c_i[j] + \eta(x(k) - c_i) & , \quad i = k(x) \\ c_i[j] & , \quad \text{otherwise} \end{cases}$$

6: * $j = j + 1$

7: **end while**

This algorithm continues until there is no change in the centers. η is an update parameter between 0, and 1.

Algorithm 1 depend on initial values of the center, so the k-means adaptive clustering [1] can be used instead.

For the computation of the spreads there are several alternatives:

(a) The empirical standard deviation: if n denotes the number of patterns assigned to cluster i, we can use:

$$\sigma = \sum_{j=1}^{n} \sqrt{\frac{||C_{j,i} - x_j||^2}{n}} \tag{3.5}$$

(b) The k-nearest neighbours heuristic: considering the k (a user-defined percentage of the total number of centers) centers nearest the center C_i, the spread associated with this center is:

$$\sigma = \frac{\sum_{j=1}^{k} ||C_i - C_j||}{k\sqrt{2}} \tag{3.6}$$

(c)Nearest neighbour: Assume that k is the nearest neighbour of center i. Then the spread is:

$$\sigma = Q\frac{||C_k - C_i||}{\sqrt{2}} \tag{3.7}$$

Where Q is an application-defined parameter.

(d) Maximum distance between patterns: Assume that m is the total number of patterns. Then the spread, which is the same through all the centers, is:

$$\sigma = \frac{max_{j,i=1...m}||x_i - x_j||}{2\sqrt{2}} \tag{3.8}$$

3.1.3 Training

Some derivative-based training algorithms for MLP and RBF, will be explained in this section. The text in the next sections up to Section 3.2 are reproduced, with permission, from reference [1].

3.1.3.1 LMS Algorithm

The basic block of the NN is the neuron, which has inputs, an activation function and output. To make learning possible an error signal must be computed, the error is the different between the real output (target) and the computed output. the LMS algorithm replaces, in the original Perceptron learning rule, the output by the net input, as shown in figure 3.4 , the rule is :

$$w[k+1] = w[k] + \alpha(t[k] - net[k])x[k] \tag{3.9}$$

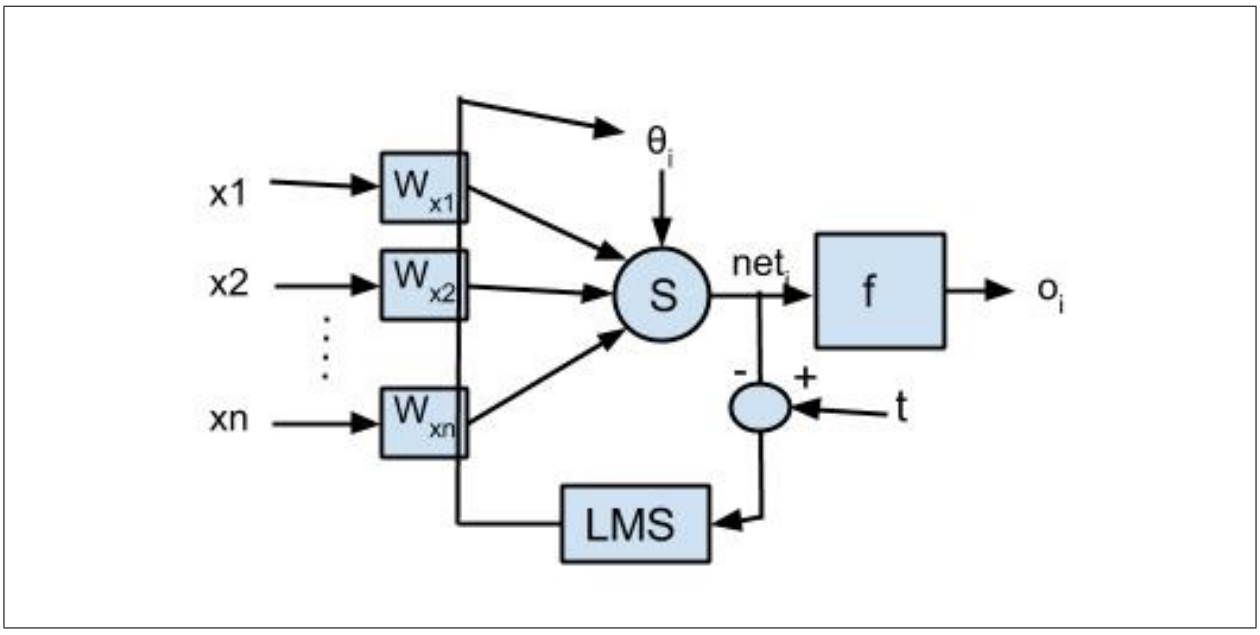


Figure 3.4: Structure of LMS algorithm [1]

The error function in the training set (composed of N patterns), is defined as:

$$E = \frac{1}{2}e^T e = \frac{1}{2}\sum_{i-1}^{N} e^2[i] \tag{3.10}$$

Where:

$$e[k] = t[k] - y[k] \tag{3.11}$$

For simplicity, assume that the error is defined at the output of a linear combiner, and denote this value by $y[.]$.

To find the value of the weight that corresponds to the minimum value of the error function, compute the gradient of the function, and perform a step in the opposite direction. This is called the method of ***steepest descent***.

The gradient of the error is denoted as:

$$g = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ . \\ . \\ . \\ \frac{\partial E}{\partial w_n} \end{bmatrix} \tag{3.12}$$

And the LMS rule can be given as:

$$w[k+1] = w[k] - \eta g[k] \tag{3.13}$$

By substituting 3.10 and 3.12:

$$g = \begin{bmatrix} \frac{1}{2}\frac{\partial \sum_{i-1}^{N} e^2[i]}{\partial w_1} = \frac{1}{2}\sum_{i-1}^{N} \frac{\partial e^2[i]}{\partial w_1} \\ . \\ . \\ . \\ \frac{1}{2}\frac{\partial \sum_{i-1}^{N} e^2[i]}{\partial w_n} = \frac{1}{2}\sum_{i-1}^{N} \frac{\partial e^2[i]}{\partial w_n} \end{bmatrix} \tag{3.14}$$

Then by applying the chain rule, we obtain:

$$\frac{\partial e^2[j]}{\partial w_i} = \frac{\partial e^2[j]}{\partial y}\frac{\partial y}{\partial w_i} = -2e[j]x_i[j] \tag{3.15}$$

Now by replacing this equation in 3.14 :

$$g = \begin{bmatrix} \sum_{i-1}^{N} e[i]\frac{\partial y}{\partial w_i} = \sum_{i-1}^{N} e[i]x_1[i] \\ . \\ . \\ . \\ \sum_{i-1}^{N} e[i]\frac{\partial y}{\partial w_p} = \sum_{i-1}^{N} e[i]x_1[i] \end{bmatrix} = -(e^T J)^T \tag{3.16}$$

Where e is the error vector, and J is the Jacobean matrix(the matrix of the derivatives of the output vector with respect to the weights).

$$e = \begin{bmatrix} e[1]. \\ . \\ . \\ e[N] \end{bmatrix} \tag{3.17}$$

$$J = \begin{bmatrix} \frac{\partial y[1]}{\partial w_1} & \dots & \frac{\partial y[1]}{\partial w_n} \\ & . & \\ & . & \\ & . & \\ \frac{\partial y[N]}{\partial w_1} & \dots & \frac{\partial y[N]}{\partial w_n} \end{bmatrix} \tag{3.18}$$

In this case, as the output is just a linear combination of the inputs, the Jacobean is just the input matrix X:

$$J = \begin{bmatrix} x[1]. \\ . \\ . \\ x[N] \end{bmatrix} \tag{3.19}$$

To have the steepest descent update, replace 3.16 in 3.13

$$w[k+1] = w[k] + \eta e^T[k]X \tag{3.20}$$

Note that in the last equation all the errors must be computed for all the training set before the weights are updated, in each iteration of the algorithm. This is called batch update or

epoch mode. If, on the other hand, in each pass through all the training set, an instantaneous version of the gradient is computed for each pattern, and the weights are updated pattern by pattern, this is called pattern mode, and we have the Widrow-Hoff algorithm.

As we have already seen the gradient vector can be computed as:

$$g^T = -e^T J \tag{3.21}$$

Where J is the Jacobean matrix.

Following this approach, it is advisable to compute J in a partitioned fashion, reflecting the topology of the MLP:

$$J = \begin{bmatrix} J^{(q-1)} & \dots & J^{(1)} \end{bmatrix} \tag{3.22}$$

$J^{(z)}$ denotes the derivatives of the output vector of the MLP with respect to the weight vector $w^{(z)}$. The chain rule can be employed to compute $J^{(z)}$. And J computed starting from the output layer towards the input layer.

Jacobean matrix compute using the following algorithm [1]:

Algorithm 2 How to Computation J_i

1: $z = q$ ▷ q : number of layers
2: $d = [f'(net)_i^{(q)}]$
3: **while** $z > 1$ **do**
4: **for** $j \leftarrow 1$ to k_z **do** ▷ k the no. of neurons (withou input layer)
5: $J_{i,j}^{(z-1)} = d_j[O_i^{(z-1)} \quad 1]$
6: **end for**
7: **if** $z > 2$ **then**
8: $d = d(W^{(z-1)})^T$
9: $d = d \times f'((net)_i^{(z-1)})$
10: **end if**
11: $z = z - 1$
12: **end while**

To compute the gradient vector using this algorithm 2, in conjunction with equation 3.21, the error vector, and consequently the output vector, must be available and to do that a recall operation must be executed [1]:

Algorithm 3 Recall Operation

1: $z = 1$
2: **while** $z < q$ **do**
3: $\quad Net_i^{(z+1)} = [O_i^{(z)} \quad 1]W^{(z)}$
4: $\quad O_i^{(z+1)} = f^{(z+1)}(Net^{(z+1)})$
5: $\quad z = z + 1$
6: **end while**

Using algorithms 2, and 3 the gradient vector can be obtained as :

Algorithm 4 Computation of g

1: $g^T = 0$
2: $i = 1$
3: **while** $i \leq m$ **do**
4: $\quad$ ... compute $O_i^{(q)}$ using algorithms 3
5: $\quad e_i = t_i - O_i^{(q)}$
6: $\quad$ ... compute J_i using algorithms 2
7: $\quad g^T = g^T - e_i J_i$
8: **end while**

This last algorithm computes the gradient vector, with the same computational complexity as the error back-propagation algorithm [35]. However, it does not share one of its advantages because not all the computations are performed locally in the neurons. This property can be provided by slightly modifying algorithm 4. By doing that, we obtain the gradient vector as computed by the BP algorithm.

3.1.3.2 The Error Back-Propagation Algorithm

The aim of training the NN is to find the values of the weights and biases of the network that minimize the sum of the square of the errors between the target and the actual output, as shown in equation 3.10.

The original error back-propagation (BP) algorithm implements a steepest descent method. In each iteration, the weights of the NN are updated by a fixed percentage in the negative

gradient direction.

Several modifications of this algorithm have been proposed. One of them is to perform the update of the weights each time a pattern is presented. The reasoning behind pattern mode update is that, if η is small, the departure from true gradient descent will be small and the algorithm will carry out a very close approximation to gradient descent in sum-squared error [34]. Another modification, also introduced by Rumelhart and the PDP group [34], is the inclusion of a portion of the last weight change, called the *momentum term*, in the weights update equation:

$$w[k+1] = w[k] - \eta g[k] + \alpha(w[k] - w[k-1]) \tag{3.23}$$

The use of this term would, in principle, allow the use of a faster learning rate, without leading to oscillations. Several other modifications to the BP algorithm were proposed. For example, Jacobs proposed an adaptive method for obtaining the learning rate parameter [1].

Two disadvantage for BP:

(a) The error back-propagation algorithm is not a reliable algorithm; the training procedure can diverge;

(b) The convergence rate obtained depends on the learning rate used and on the characteristics of the training data.

3.1.3.3 Alternatives to the error back-propagation algorithm

In this section alternatives to the error back-propagation are given, to eliminated the limitation of the BP, and the unreliability, by incorporating a line search algorithm, by using equation 3.23 and :

- compute a search direction $p[k] = g[k]$
- compute a step length: $\alpha[k] = \eta$
- update the weights: $w[k+1] = w[k] + \alpha[k]p[k]$

The second step could then be modified, to compute, in each iteration, a step length $\alpha[k]$

$$\Omega(w[k] + \alpha[k]p[k]) < \Omega(w[k]) \tag{3.24}$$

To be verified in every iteration.

Still the error back-propagation algorithm is far from attractive. One way to express how the two different updates appear is to consider that two different approximations are used for Ω , in 3.13, a first-order approximation of Ω is assumed:

$$\Omega(w[k] + p[k]) \approx \Omega(w[k]) + g^T[k]p[k] \tag{3.25}$$

It can be shown that the normalized (Euclidean norm) $p[k]$ that minimizes 3.13 is:

$$p[k] = -g[k] \tag{3.26}$$

And that is the steepest-descent direction.

When the gradient can then be expressed as:

$$g^l = -A^T t + (A^T A)w \tag{3.27}$$

Where A denotes the input matrix, augmented by a column vector of ones to account for the threshold. It has a unique solution when it obtain to zero:

$$\hat{\mathrm{w}} = (A^T A)^{-1} A^T t \tag{3.28}$$

In last equation, appears when a second order approximation is assumed for Ω

$$\Omega(w[k]p[k]) \approx ()\Omega(w[k]) + g^T[k]p[k] + \frac{1}{2}p^T[k]G[k]p[k] \tag{3.29}$$

The matrix $G[k]$ denotes the matrix of the second order derivatives of Ω at the k^th iteration. And this matrix is called the Hessian matrix of Ω.

The Hessian matrix has a special structure when the function to minimize is, as in the case in consideration, a sum of the square of the errors. For the nonlinear case:

$$G[k] = J^T[k]J[k] - Q[k] \tag{3.30}$$

where $Q[k]$ is:

$$Q[k] = \sum_{i=1}^{m} e_i[k]G_i[k] \tag{3.31}$$

This serious limitation, together with the high computational costs of the method (the second order derivatives are needed, and G must be inverted) stimulated the development of alternatives to Newtons method,called the Quasi-Newton methods. Other methods that exploit this structure are the Gauss-Newton and the Levenberg-Marquardt methods.

Quasi-Newton method employs the observed behaviour of Ω and g to build up curvature information, in order to make an approximation of G (or of $H = G^{-1}$) using an appropriate updating technique. The update formula used possess the property of hereditary positive definiteness, i.e., if $G[k]$ is positive definite, so is $G[k+1]$.

The basis of the Gauss-Newton (GN) method lies in dropping the use of second order derivatives in the Hessian, so that 3.30 is approximated as:

$$G[k] \approx J^T[k]J[k] \tag{3.32}$$

3.1.3.4 Levenberg-Marquardt method

Levenberg and Marquardt were the first to suggest this type of method in the context of non-linear least-squares optimization. Many different algorithms of this type have subsequently been suggested.

This method which has global convergence property, and overcomes the problems arising when $Q[k]$ is significant [1].

The search direction for this method:

$$(J^T[k]J[k] + \upsilon[k]I)_{p_{LM}}[k] = -J^T[k]e[k] \tag{3.33}$$

Where the scalar υ controls both the magnitude and the direction of $p[k]$. When υ is zero, $p[k]$ is identical to the Gauss-Newton direction. As υ tends to infinity, $p[k]$ tends to a vector of zeros, and a steepest descent direction.

To introduce the algorithm actually employed [36], one way of envisaging the approximation of the Hessian employed in LM methods is that this method, at every k^th iteration, consider a linear model for generating the data:

$$o^{(nl)}[k] = J[k]w[k] \tag{3.34}$$

Using this, the predicted error vector, after taking a step $p[k]$ is:

$$e^p[k] = e[k] - J[k]p[k] \tag{3.35}$$

So the predicted reduction of Ω :

$$\Delta\Omega^p[k] = \Omega(w[k]) - \frac{e^p[k]^T(e^p[k])}{2} \tag{3.36}$$

Actual reduction is given by:

$$\Delta\Omega[k] = \Omega(w[k]) - \Omega(w[k] + p[k]) \tag{3.37}$$

And to measures the accuracy to which the quadratic function approximates the actual function, in the sense that the closer that $r[k]$ in equation 3.38 is to unity, the better the agreement.

$$r[k] = \frac{\Delta\Omega[k]}{\Delta\Omega^p[k]} \tag{3.38}$$

Then the LM algorithm can be stated, for iteration k:

Algorithm 5 Levenberg-Marquardt algorithm

1: obtain $J[k]$ using algorithm 2, and $e[k]$ using algorithm 3
2: compute p[k] using equation 3.33 ;
3: ... IF $(J^T[k]J[k] + \upsilon[k]I)$is not positive definite then ($\upsilon[k] = 4\upsilon[k]$ and repeat)
4: evaluate $\Omega(w[k] + p[k])$ and hence $r[k]$
5: **if** $r[k] < 0.25$ **then**
6: $\quad \upsilon[k+1] = 4\upsilon[k]$
7: **else if** $r[k] < 0.75$ **then**
8: $\quad \upsilon[k+1] = \frac{\upsilon[k]}{2}$
9: **else**
10: $\quad \upsilon[k+1] = \upsilon[k]$
11: **end if**
12: **if** $r[k] \leq 0$ **then**
13: $\quad w[k+1] = w[k]$
14: **else**
15: $\quad w[k+1] = w[k] + p[k]$
16: **end if**

The algorithm is usually initiated with $\upsilon[1] = 1$ and is not very sensitive to the change of the parameters 0.25, 0.75, etc. It is usually agreed that the Levenberg-Marquardt method is the best method for nonlinear least-squares problems [36]. This was confirmed in this thesis, as it is compared to BP algorithm in a number of training examples performed.
Quasi-Newton method achieves a much better rate of convergence than the BP algorithm, but the LM algorithm is undoubtedly the best of all [1].

3.1.4 A new learning criterion

The only difference between this topology and the standard one lies in the activation function of the output neuron, which is linear. This simple fact, can be exploited to decrease the number of iterations needed for convergence [1].
If the weights divide into two parts, u the weights that connect to the output neuron, and v for all the other weights. Using this convention, the output vector can be given as:

$$o^{(q)} = [O^{(q-1)} \quad 1]u \tag{3.39}$$

If last equation used with equation 3.10, the criterion will be:

$$\Omega = \frac{||t - [O^{(q-1)} \quad 1]u||_2^2}{2} \tag{3.40}$$

The nonlinear relation between v and Ω appear only by $O^{(q-1)}$, The optimum value of the linear parameters is therefore conditional upon the value taken by the nonlinear variables.

$$\hat{\mathrm{u}}(v) = ([O^{(q-1)}(v) \quad 1])^{+}t \tag{3.41}$$

Denoting $[O^{(q-1)} \quad 1]$ matrix by A,creating therefore a new criterion :

$$\Psi = \frac{||t - AA^{+}t||_2^2}{2} = \frac{||P_{A_\perp}t||_2^2}{2} \tag{3.42}$$

$P_{A_\perp}$is the orthogonal projection matrix to the complementary space spanned by the columns of A, where the dependence of A on v has been omitted.
This new criterion (eq. 3.42) depends only on the nonlinear weights, and although different from the standard criterion (eq. 3.40), their minimum are the same.

Instead of determining the optimum of equation 3.40, first minimize equation 3.42, and then, using v in equation 3.41, to obtain the complete optimal weight vector w.
The main advantage of using this new criterion:

- Reducing the dimensionality of the problem
- Faster convergence of the training algorithm

For that, a new criterion will be noticed in :

- A better convergence rate is normally observed
- The initial value is usually much smaller, for the same initial value of the nonlinear parameters.

For Training a derivative of equation 3.42 will be needed [1].
For the error back-propagation algorithm or the quasi-Newton with this new criterion, the gradient of Ψ given as:

$$\begin{bmatrix} 0 \\ g_\Psi \end{bmatrix} = g_\Omega|_{u=\hat{u}} = - \begin{bmatrix} \left(O^{(q-1)}\right)^T \\ 1 \\ J_\Psi^T \end{bmatrix} e_\Psi \tag{3.43}$$

Where: $g_\Omega|_{u=\hat{u}}$,J_Ψ, and e_Ψ, denote respectively the gradient, the partition of the Jacobean matrix associated with the weights$v$ and the error vector of Ω obtained when the values of linear weights are their conditional optimal values:

$$J_\Psi = (A)_v A^+ t \tag{3.44}$$

$$e_\Psi = P_{A_\perp} t \tag{3.45}$$

To compute g_Ψ:

- compute $O^{(q-1)}$ using algorithm 3
- obtain $\hat{u}$
- replace $\hat{u}$ in the linear parameters, and complete algorithm 3

- compute g_Ψ using algorithm 4

For Gauss-Newton or Levenberg-Marquardt methods are employed with this new formulation, then the Jacobean of equation 3.42 must be obtained. Three different Jacobean matrices have been proposed: the first was introduced by Golub and Pereyra , who were also the first to introduce the reformulated criterion. To reduce the computational complexity of Golub and Pereyras approach, Kaufman proposed a simpler Jacobean matrix. Ruano et al. proposed to employ the Jacobean matrix J_Ψ (in equation 3.44), which further reduces the computational complexity of each training iteration [1].

- Compute $O^{(q-1)}$ using algorithm 3
- Obtain $\hat{u}$
- Replace $\hat{u}$ in the linear parameters, and complete algorithm 3
- Compute g_Ψ using algorithm 2

3.1.5 Initial weight values

A good (close to the optimum) initial value of the design variables is important for the decrease of the number of iterations towards convergence. There are no guidelines for determining good initial parameter, it is a common practice to employ random values, or zero values, as the initial parameter values.
The computation of the Jacobean matrix involves three terms:
(1) Derivative of the neuron output to its *net* input. But if the neuron *net* input is outside its active region the corresponding derivative is very small. This affects all the weights related with the neurons below the one considered. Therefore, all the columns associated with those neurons will have a small norm, which makes them more linear dependent on the column associated with the output bias.
(2) Derivative of the neuron net input to its inputs: this turns out to be the transpose of the partial weight matrix. The value of the weights should not be too big or too small. Focusing on the last layer, assuming that it is linear, as the layer inputs are limited between 0 and 1,

this means that the target data should lie between the range. So, it is convenient to scale the target data to this range.
(3) derivative of the neuron net input to the weights, which is related with the neuron inputs. The layer inputs should also lie between a range.

3.1.6 Termination criteria

When to stop training?. Early stopping can be used to overcome the over-trained, this approach is employed if the training set is divided in two data sets: the training and test data set.
As training progresses, the model tends to approximate the data better. But sometime it starts memorizing the peculiarities of the training data, eventually becoming over-trained and losing the capability of generalizing to new data samples. By using early stopping, the performance of the model is evaluated on the test data and training stops at a point where the performance in the test set deteriorates [1].
To achieve the early stopping, three criterion needed. First criterion emphasizes the need for the convergence of the parameters to its optima:

$$||w[k-1] - w[k]||_2 < \sqrt{\tau}(1 + ||z[k]||_2) \tag{3.46}$$

A second criterion is meant to assess the proximity of $\Psi[k]$ to the expected minimal value:

$$\Omega[k+1] - \Omega[k] < \beta[k] \tag{3.47}$$

Where $\beta = \tau(1 + |\Omega[k]|)$ and τ is a measure of the desired number of correct digits in the objective function.
The final termination criterion tests the gradient convergence to zero:

$$||g[k]||_2 = \sqrt[3]{\tau}(1 + |\Omega[k]|) \tag{3.48}$$

When New criterion is employed, Ω and z should be replaced by Ψ and v, in the above equations.

3.2 Genetic Algorithm

GA is one class of Evolutionary Algorithms (EAs) that benefits from a set of procedures and operators inspired on the process of natural evolution and on the notion of survival of the fittest.

GA starts with an initial population of individuals, the initial generation, which are then evaluated and manipulated to compute the population of individuals composing the next generation. Hopefully, after a sufficient number of generations the population has evolved achieving a satisfactory approximation to the Pareto front.

This method can be applied to solve the selection for the models problem of ANN mentioned before.

3.2.1 GA properties and parameters

GA has certain rules of evolving, and finding the solution for the problem, each solution represented by individual, GA starts with a initial set of individuals, the set of individuals called a population, were the first population generates a new one, by GA operations, and the fittest individuals (best solutions) will survive. After some amount of generations, a good solution will appear.

3.2.1.1 Individual representation

Each individual has a structure representation, a chromosome, to encode the proposed solution (ANN topology and inputs). In the following the general class of feed-forward ANNs having one hidden layer of neurons is considered.

The chromosome has the number of neurons n, from $n \in [n_{min}; n_{Max}]$ neurons, and the indices $D = [d_1 ... d_p]$, where $d \in [d_{min}; d_{Max}]$ of available input features. For each candidate model, the network parameters are determined using the LM algorithm, minimizing the new criterion.

Features f_l in the columns of F, could be any input available, like lags of modelled variable, or of any other extra inputs.

$$
\begin{aligned}
y(t+1) = & g(y(t), y(t-1) \quad .... \quad y(t-T_y), \\
& v_1(t), v_1(t-1), \quad .... \quad v_1(t-T_{v_1}), \\
& ....., \\
& v_o(t), v_o(t-1), \quad .... \quad v_o(t-T_{v_0})),
\end{aligned}
\tag{3.49}
$$

3.2.1.2 Fitness

To select the best individual in the population, a fitness function is required, which measures how good these individuals are, by selecting the fittest individuals for mating, a better population will be generated.

The fitness in ANN could be the complexity of the model, as well as the RMSE of the training set or testing set , also it could be the forecasting values, by adding each forecasting steps together.

$$
RMSE = \sqrt{\frac{\sum_1^n e^2}{n}} \tag{3.50}
$$

Where e is the error, and RMSE shows the error for the set, so it can be a good fitness function.

3.2.1.3 Selection

Selection uses Baker's stochastic universal sampling (SUS) algorithm [37], which is optimal in terms of bias and spread. It minimizes the stochastic errors associated with roulette-wheel selection, and, thus, the genetic drift.

The selective pressure (defined as the ratio between the number of offspring expected by the best individual in the population and the average number of offsprings per individual) imposed by rank-based fitness assignment is constant and can be easily set. In a fixed-size population, the average number of offspring per individual is one, and, therefore, the number of offspring expected by the best individual equals the selective pressure [38].

3.2.1.4 Recombination

Once the parents have been selected from the population, they are paired up and recombined with given probability. after that exchange the parts between these parents individuals, to get the offset individuals, this operation called recombination or crossover. Figure 3.5 shows this operation.

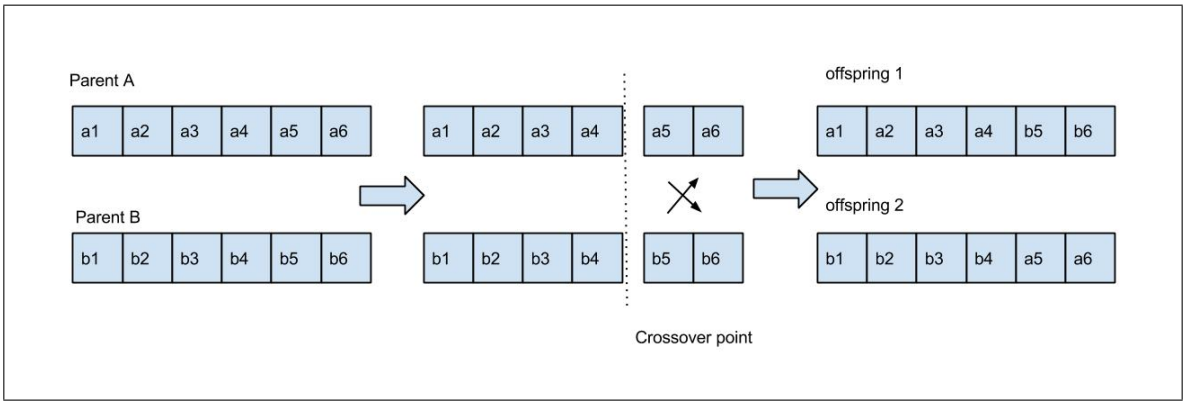


Figure 3.5: Crossover operation [2]

3.2.1.5 Mutation

It has been found that, despite its simplicity, mutation is an important, active search operator. Independent bit mutation is characterized by a single parameter, the bit mutation rate, or probability, the setting of which depends on the selective pressure and chromosome length. Care must be taken in order to guarantee the boundary conditions for number of neurons, and numbers of inputs.

3.2.2 Multi-Objective Genetic Algorithm

Most engineering problems are characterized by several objectives to be optimized.
As in GAs, individuals in MOGA are represented as chromosomes, a population produced in each generation. Each of these individuals have a fitness (rank) value for evaluation. After GA operations on the individuals to get a new generation, with possibly better solutions, then repeated until having an acceptable solution. Figure 3.6 illustrates the flow of MOGA operation.

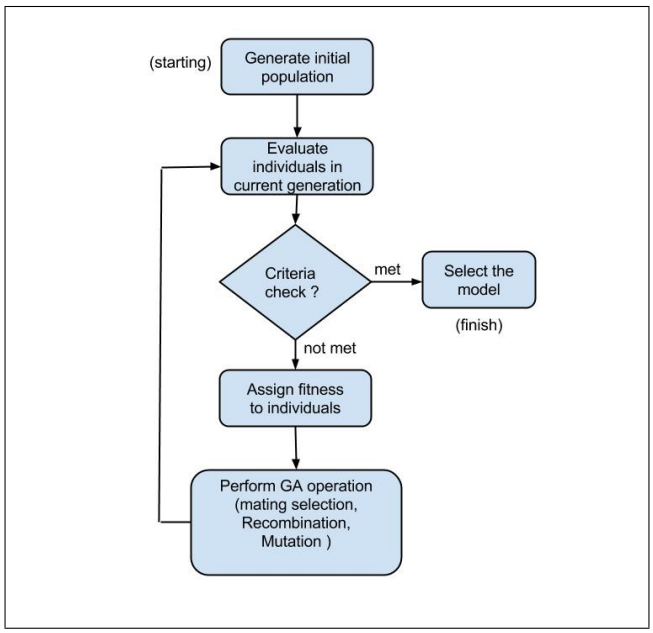


Figure 3.6: Flow of MOGA operation (modifed from [2])

The solution, in multi-objective problems, is the set of Pareto points in search space (or design space) that define the Pareto front in the space of objectives. This means that the ANN model designer has to select one particular ANN by examining trade-offs in the objectives of the Pareto front.

3.2.2.1 Rank and fitness assignment

As opposed to the single objective case, the ranking of a population in the multi-objective case is not unique.

Fitness is understood here as the number of offspring an individual is expected to produce through selection. It differs from individual utility, which reflects the result of the decision making process. The selection process determines which individuals actually influence the production of the next generation and is, therefore, a part of the search strategy.

The fitness assigned to individuals with the same multi-objective rank is averaged, and fitness shared within each rank before selection takes place. One way of ranking the individuals, is to take the individuals in the Pareto front, assigned as rank 0, remove them from the population, and find the next Pareto front, assigned rank 1, and remove them from the population, and so on [39].

The traditional rank-based fitness assignment is only slightly modified, as follows:

- Sorting population according to rank.
- Assign fitness by interpolating from the best individual (rank = 0) to the worst (rank = max) according to some function, usually linear or exponential, but possibly of other type.
- Average the fitness assigned to individuals with the same rank.

By doing so, all of individuals in the Pareto front are sampled at the same rate while keeping the global population fitness constant.
Rank-based fitness assignment is independent from objective scaling.

3.2.2.2 Constraints and priorities

Beside the Pareto front, if constraints are used in the problem formation, better results could be in the direction of objective priority on these constraints.
Constraints can often be seen as hard objectives, which need to be satisfied before the optimization of the remaining, soft, objectives takes place.
Constraints usually fall into one of two different categories [39]:
Domain constraints: express the domain of definition of the objective function. In control systems, closed-loop system stability is an example of a domain constraint, because most performance measures are not defined for unstable systems.
Preference constraints: impose further restrictions on the solution of the problem according to knowledge at a higher level. A given stability margin, for example, expresses a preference of the designer.
Beside constraints, priorities could help to reach the goals faster. The specification of goals and priorities can accommodate a whole variety of constrained and/or multi-objective problem formulations. Goal and priority information is often naturally available from the problem formulation, although not necessarily in a strict sense. Therefore, the interpretation of such information should take its partial character into account. This can be accomplished by allowing different objectives to be given the same priority, and by avoiding using measures of

the distance to the goals, which inevitably depend on the scale in which the objective values are presented.

Chapter 4

MLP Experimental Work

MLP ANNs are used in the first experimental work. Before that, the data needed for this experiment must be prepared. The first section of this chapter will explain the data that been used.

4.1 Data preparation

The available data for the power consumption for the campus are in Kilo-Watt (KW), and it were measured each 15 minutes for one year (2014). There for there are 96 points for each day, and over 35 thousands point for the year. Some samples were removed in situation where the electric power off.

In MATLAB, this data was loaded from EXCEL sheets, and saved as one vector for all 35038 points. This vector is the time series that has been used; from this data a target power and inputs matrix of the lags, have been created.

From the available data set it is possible to notice that the energy consumption in working days is higher, while during holidays and at night is lower, and it is almost the some level (constant), due to running of the basic equipment. In fact at night it increased a little bit due to the use of lighting, It is also noticeable that energy consumption differs with seasons, mainly because of air conditioning usage. Among all the affecting factors, is the holidays break seem to be the most important.

For this part of the thesis work (MLPs and also for RBFs later), the inputs are:

- Two lags of power.
- A day type code; this input represents the day type [5], for that particular point of time.

There are a lot of variable affecting the energy consumption. In this work, only the day type is taken into account. Using the university calendar, for 2013-2014 and 2014-2015 years, and a holidays calendar, table 4.1 shows the representation of holidays, and normal working days, as well as some other days categorized, for a different behavior of energy in these days noticed from the data. Figure 4.2(a) shows a plot of the entire data set, figure 4.2(b) shows a week period, and figures 4.2(c),4.2(d), and 4.2(e) shows examples of a 24 hours period.

Type of days	Code
Bank holidays , Sundays , new year eve , new year day	0
Saturdays	0.1
Academic holidays, summer holiday	0.5
Days before holidays (like Fridays)	0.9
Normal working days	1

Table 4.1: Code for days classification

The methodology that had been used is to divide the data into three sets: Training, generalization, and validation sets. For training the NN obviously the training set had been used, and to pervert over-training a generalization set had been used. After training is finished the validation set is used to validate the model, and to select the best models for one step prediction. The three data sets define, training, testing, and validation data sets, were taken as around 10% of the data points (exactly 3 000 points each). All the data sets points were taken randomly from the data, care has been taken not to over lap the data.
Final check for the models is made, by prediction of 96 steps ahead, using the whole data available in this check. Figure 4.1 shows the data parts divisions.

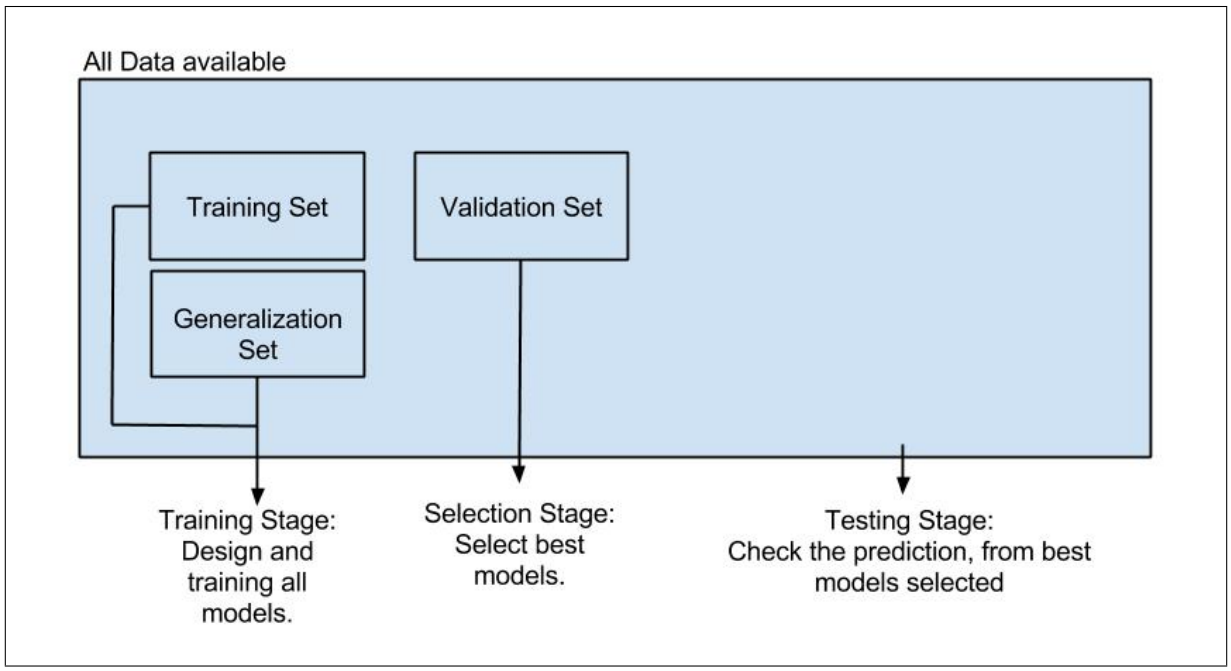


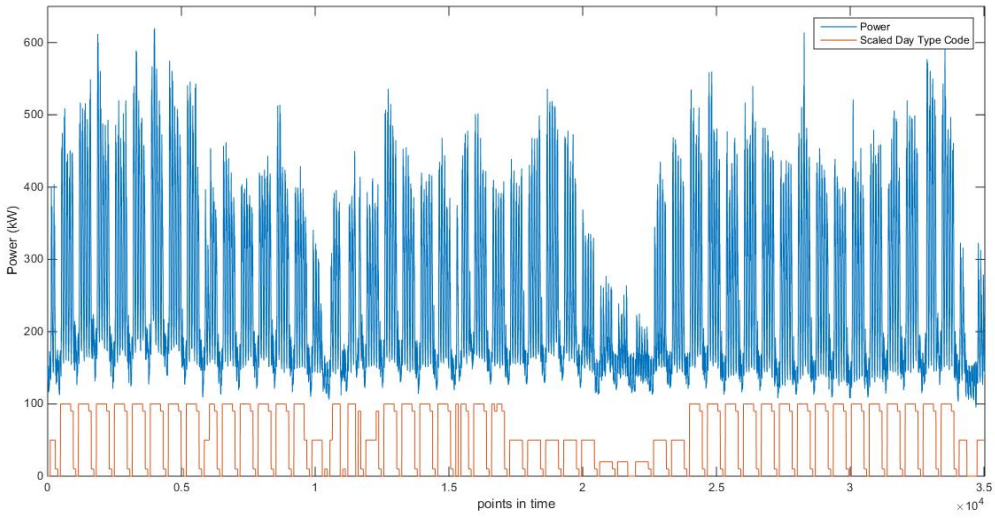
Figure 4.1: Division the data into subsets

4.2 Design and train the MLP NN

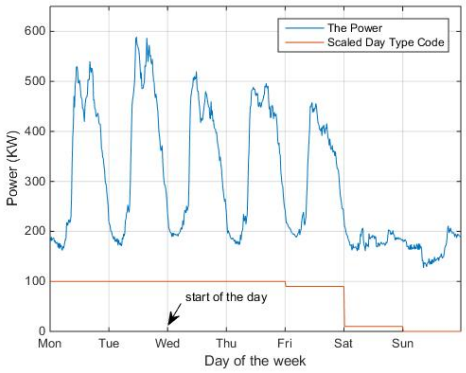
Using MATLAB code available in the lab (written by Prof Antonio Ruano), to train the MLPs network, using training and generalization data sets for early stopping, the topology used to run the network, and parameters, like resolution parameter (τ) and criterion type, is executed in a loop, applied to most combinations as possible, with simple modification to the training code. The options that have been used are:

- Topology : two hidden layers with number of neurons as [4 4],[4 8], [2 4], [4 2],and [2 2].
- Resolution parameter (τ) : 0.05, 0.01, 0.005, 0.001 0.0005, and 0.0001.
- Criterion Type: Standard criterion, and new criterion.
- LM algorithm is used for training.

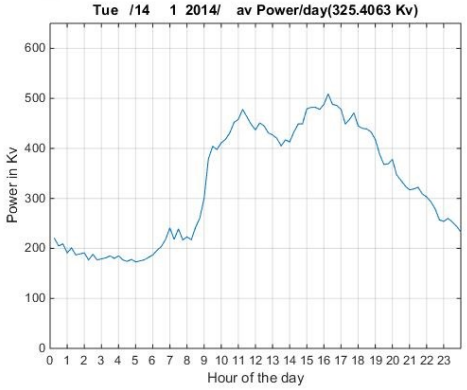
Then executing the models in the validation set, to select the best models according to the lowest value of root mean square error (RMSE) of the validation set (ϵ_v). These models are used for the prediction of one day (96 steps ahead).



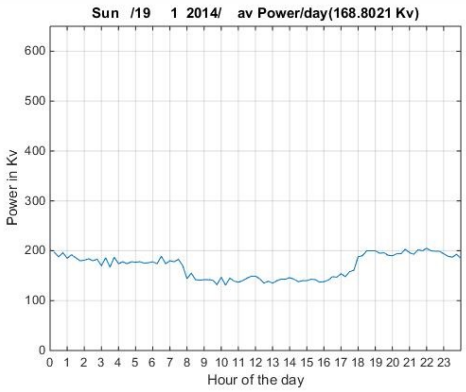
(a) The power and day type code for all year



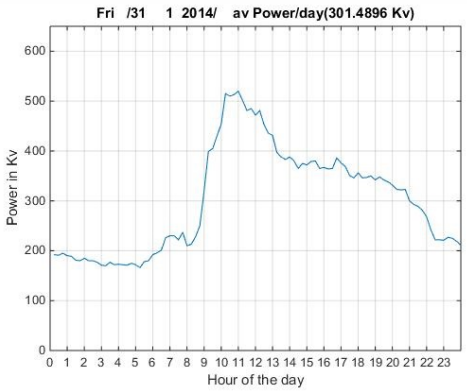
(b) The power and day type code for one week



(c) The power for one Day (working day)



(d) The power for one Day (holiday)



(e) The power for one Day (Friday day)

Figure 4.2: The data of the power consumed in 2014, with examples of some days, and a week

4.3 Results for the MLP

The best models obtained for the MLP Design and Training stage, are shown in table 4.2, where best five models from the validation set (considering one step ahead error) are highlighted. These five models are executed for 96 steps prediction (one day ahead), and the results for that prediction are shown in figure 4.3, for the three best prediction models.

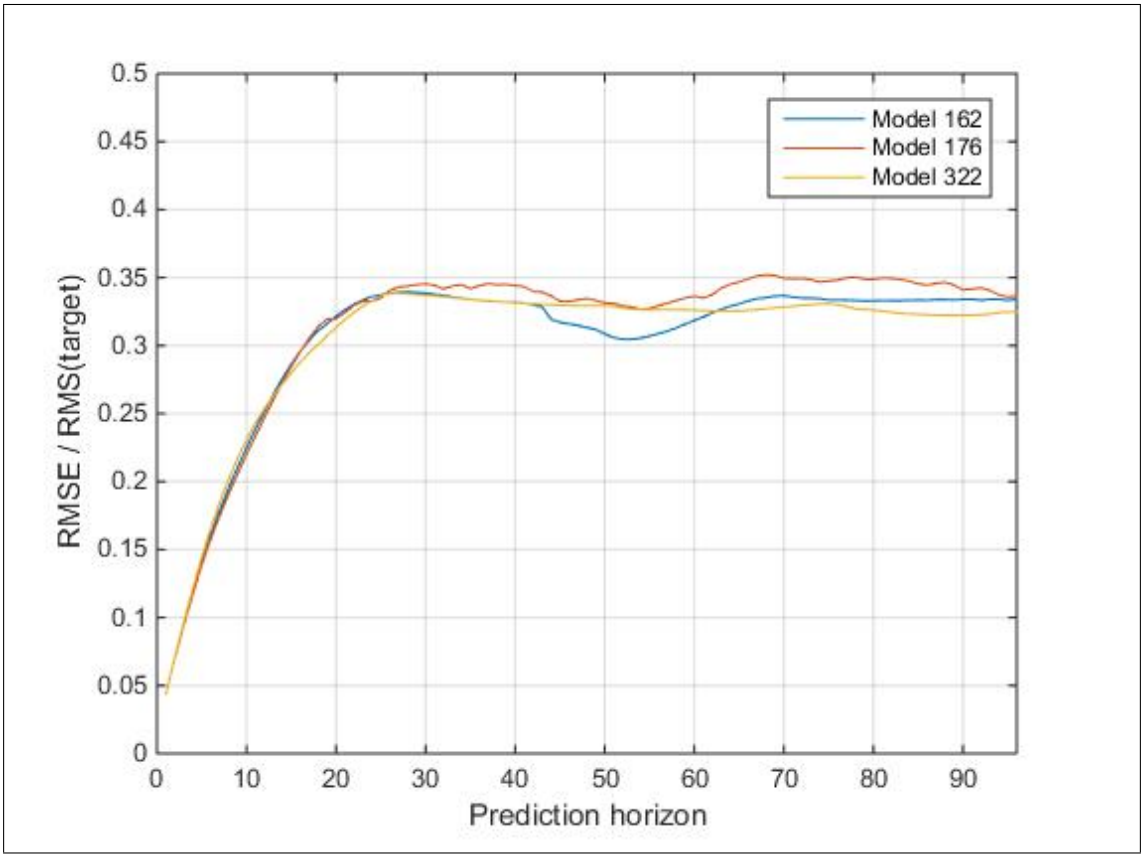


Figure 4.3: Performance of best three MLP models obtained

ID	Topology	Complexity	ϵ_t	$\frac{\epsilon_t}{RMS(t)}$	ϵ_g	$\frac{\epsilon_g}{RMS(g)}$	ϵ_v	$\frac{\epsilon_v}{RMS(v)}$
127	[4 4 1]	36	12.0765	0.0426	12.7766	0.0448	12.4308	0.0440
130	[4 4 1]	36	12.0907	0.0426	12.7139	0.0446	12.4615	0.0441
132	[4 4 1]	36	12.1075	0.0427	12.7409	0.0447	12.4110	0.0440
162	[4 4 1]	36	12.0044	0.0423	12.8131	0.0450	12.4070	0.0439
170	[4 4 1]	36	12.0304	0.0424	12.7173	0.0446	12.4309	0.0440
173	[4 4 1]	36	12.0254	0.0424	12.7410	0.0447	12.4687	0.0442
176	[4 4 1]	36	12.0116	0.0423	12.6617	0.0444	12.4038	0.0439
177	[4 4 1]	36	12.1155	0.0427	12.6983	0.0446	12.4613	0.0441
213	[4 4 1]	36	11.8542	0.0418	12.8447	0.0451	12.4504	0.0441
217	[4 4 1]	36	12.2289	0.0431	12.7660	0.0448	12.4383	0.0441
220	[4 4 1]	36	12.1635	0.0429	12.7171	0.0446	12.4414	0.0441
322	[4 8 1]	56	12.1502	0.0428	12.7540	0.0448	12.4054	0.0439
364	[4 8 1]	56	12.0176	0.0423	12.7731	0.0448	12.4193	0.0440
370	[4 8 1]	56	12.0499	0.0425	12.7587	0.0448	12.4600	0.0441
376	[4 8 1]	56	12.1457	0.0428	12.7597	0.0448	12.4273	0.0440
377	[4 8 1]	56	12.1046	0.0427	12.7499	0.0447	12.4258	0.0440
402	[4 8 1]	56	12.1127	0.0427	12.7304	0.0447	12.4139	0.0440
403	[4 8 1]	56	12.0986	0.0426	12.7585	0.0448	12.4654	0.0441
412	[4 8 1]	56	12.0375	0.0424	12.7532	0.0448	12.3537	0.0438
414	[4 8 1]	56	12.0224	0.0424	12.8690	0.0452	12.3990	0.0439
417	[4 8 1]	56	12.1887	0.0429	12.7464	0.0447	12.4660	0.0442
420	[4 8 1]	56	11.9531	0.0421	12.7876	0.0449	12.4069	0.0439
446	[4 8 1]	56	11.9235	0.0420	12.8742	0.0452	12.4496	0.0441
454	[4 8 1]	56	12.1939	0.0430	12.6972	0.0446	12.4582	0.0441
698	[2 4 1]	20	12.1282	0.0427	12.6674	0.0445	12.4425	0.0441
854	[4 2 1]	26	12.1414	0.0428	12.7455	0.0447	12.4615	0.0441
940	[4 2 1]	26	12.1394	0.0428	12.7052	0.0446	12.4527	0.0441

Table 4.2: Best models from MLP training

Chapter 5

RBF Experimental Work

To make an equal comparison, the data sets used for the RBFs is the exact data sets used and prepared for MLP, as explained in section 4.1.

5.1 Design and train the RBF NN

Designing RBF ANNs results from the achievement of the following steps:

- The determination of coordinates of the center.
- The determination of weights applied to the radial basis functions output.
- And the determination of the width of each radial basis function.

MATLAB code available in the lab to train the RBF NN has been used, (with some modifications). The loop for the training will go through these parameters:

- Number of neurons of the hidden layer(centers): 3, 6 ,9 , 12, and 15.
- Training type : Steepest descent, or Levenberg-Marquardt.
- Criterion type : New criterion, or standard criterion.
- Learning rate : 0.05, 0.01, 0.005, or 0.001 in case of steepest descent.

A clustering algorithm is used to initialized the centers.

5.2 Results for the RBF

The best models obtained from RBF training stage, are shown in table 5.1, where best five models are highlighted. These five models had been run for 96 steps prediction, but their performances were unsatisfactory. More models from the table had been run for 96 steps prediction, and three of them are shown in figure 5.1. The models that perform better in one step-ahead prediction, are different from the ones that perform better for 96 steps ahead.

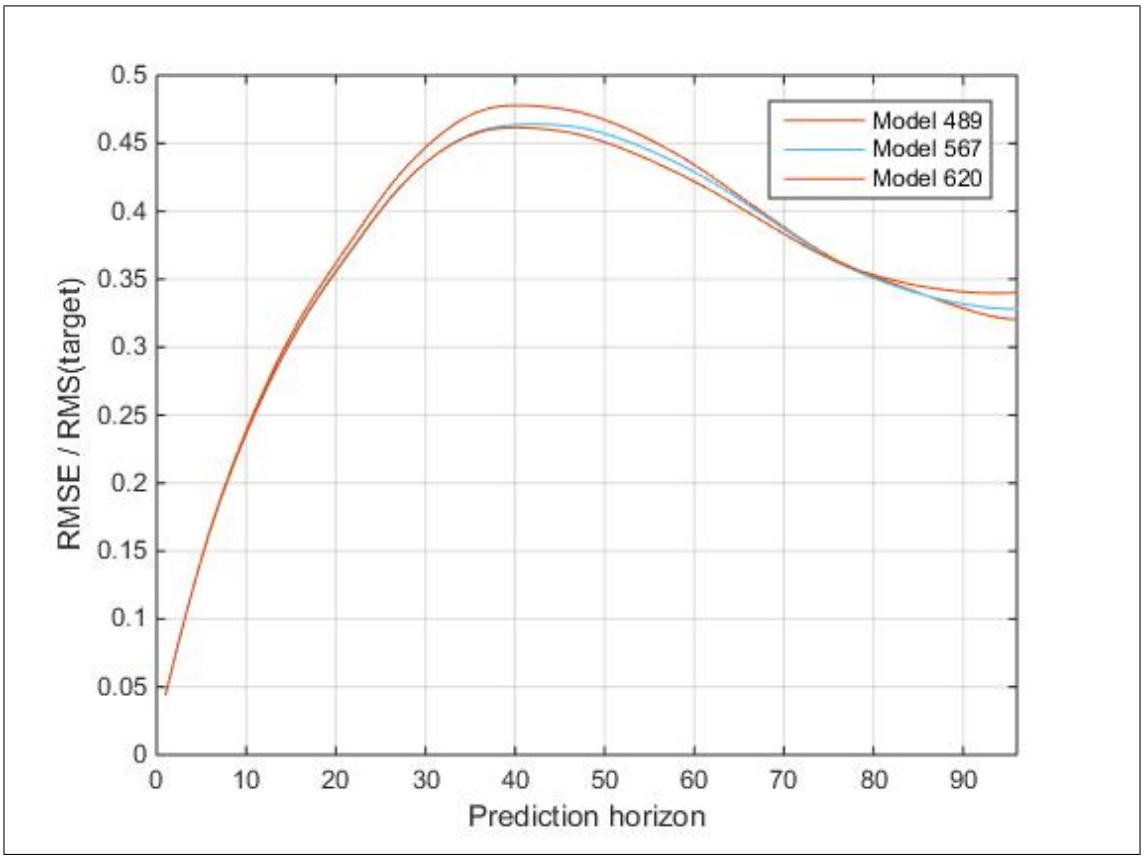


Figure 5.1: Performance of best three RBF models obtained

ID	Centers	Complexity	ϵ_t	$\frac{\epsilon_t}{RMS(t)}$	ϵ_g	$\frac{\epsilon_g}{RMS(g)}$	ϵ_v	$\frac{\epsilon_v}{RMS(v)}$
330	6	24	12.3682	0.0436	12.7642	0.0448	12.4793	0.0442
489	6	24	12.3164	0.0434	12.7312	0.0447	12.4911	0.0442
538	6	24	12.3413	0.0435	12.7529	0.0448	12.4981	0.0443
546	6	24	12.2989	0.0433	12.7313	0.0447	12.4898	0.0442
547	6	24	12.2995	0.0433	12.7271	0.0447	12.4848	0.0442
567	6	24	12.2918	0.0433	12.7286	0.0447	12.4975	0.0443
589	6	24	12.2973	0.0433	12.7321	0.0447	12.4974	0.0443
608	6	24	12.3173	0.0434	12.7464	0.0447	12.4991	0.0443
620	6	24	12.3056	0.0434	12.7328	0.0447	12.4963	0.0443
959	9	36	12.3207	0.0434	12.7253	0.0447	12.4923	0.0442
1208	12	48	12.2469	0.0432	12.7391	0.0447	12.4999	0.0443
1389	15	60	12.2281	0.0431	12.7355	0.0447	12.4900	0.0442
1406	15	60	12.1955	0.0430	12.7513	0.0447	12.4870	0.0442
1410	15	60	12.1728	0.0429	12.7377	0.0447	12.4766	0.0442
1427	15	60	12.1806	0.0429	12.7623	0.0448	12.4859	0.0442
1429	15	60	12.1692	0.0429	12.7537	0.0448	12.4964	0.0443
1447	15	60	12.2095	0.0430	12.7623	0.0448	12.4950	0.0443
1526	15	60	12.1237	0.0427	12.7802	0.0449	12.4790	0.0442
1546	15	60	12.1843	0.0429	12.7443	0.0447	12.4730	0.0442
1568	15	60	12.2155	0.0430	12.7226	0.0446	12.4794	0.0442

Table 5.1: Best models from RBF training

Chapter 6

Applying multi-objective genetic algorithm

The MOGA [40] has been used in this work in order to evolve suitable RBF NN for ELD predictive models. It was used specifically to select the number of neurons and inputs of models, so that the model performance is maximized. The software was written in Python 2.7, and run on 6 servers (1 master and 5 slaves). The master handles the GA operations, and the slaves handle the ANN training.

6.1 Data preparation

The data was prepared to have available a set of 142 input. The RMSE of the training and generalization,as well as complexity of model are minimized [3] [5] [4].
The data sets is the same used for the MLP and RBF experiments, the only difference is adding of more lags as:

- 96 lags (24 hours).
- 7 day lags of the same time stamp(7 points for each day).

And also extra one input for the day type code. Figure 6.1 illustrates the power lags used for MOGA.

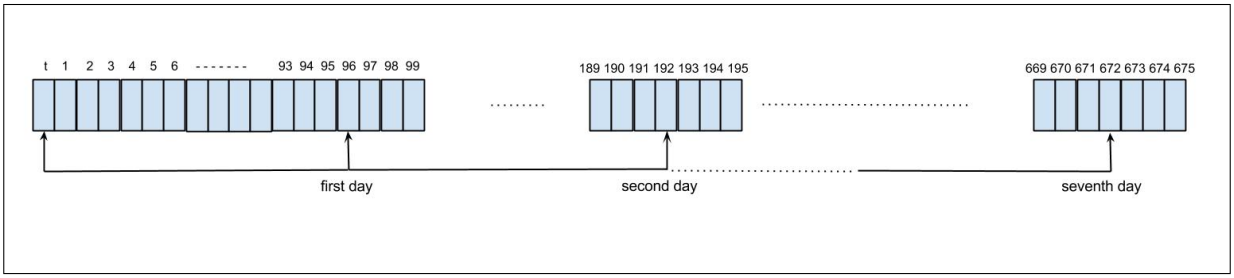


Figure 6.1: The power lags prepared for MOGA

6.2 Applying MOGA

The parameters used in MOGA are:

- Model Types selected for RBF:

  Numbers of neurons is 2 to 30.

  Centers Selected using adaptive k-mean algorithm.

  Number of inputs terms is 1 to 30.

  Early stopping criterion (with maximum 100 iterations).

- Objectives:

  minimizing RMS for training error ϵ_t, with constraint of 11 KW.

  minimizing RMS for testing error ϵ_g, with constraint of 11 KW.

  Model complexity minimizing$o(\mu)$, with constraint of 300.

- GA parameters :

  Number of generations: 100.

  Population size: 100.

  Proportion of random emigrant is 0.1

  Selection pressure is 2

  Crossover rate is 0.7

Many different experiments were done, before the final parameters were set, specially to select the constraints.

6.3 MOGA results

After MOGA has finished execution, results 158 non-dominated models. Their RMSE results for trainingϵ_t, generalizationϵ_g, validationϵ_v, and model complexity $o(\mu)$ are shown in table 6.1, and table 6.2 showns the features for the best 5 models, which have better RMSE for validation ϵ_v.

The complexity of the model is calculated as:

$$o(\mu) = Numbersofinputs \quad \times \quad Numbersofneurons \quad + \quad Numbersofspreads$$

Figure 6.2, show the prediction of best three models, for 96 steps ahead.

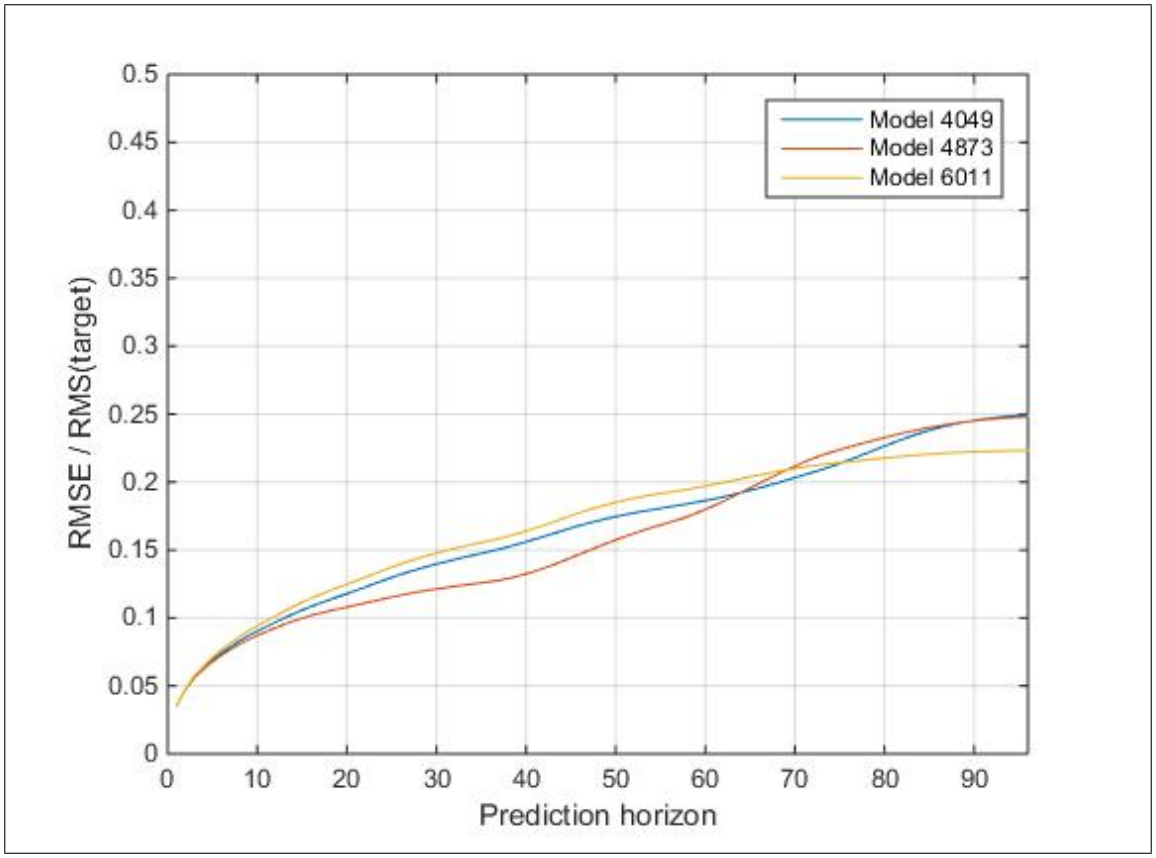


Figure 6.2: Performance of best three models from MOGA

ID	Centers	input	$o(\mu)$	ϵ_t	$\frac{\epsilon_t}{RMS(t)}$	ϵ_g	$\frac{\epsilon_g}{RMS(g)}$	ϵ_v	$\frac{\epsilon_v}{RMS(v)}$
6267	3	23	72	9.5934	0.0338	10.1928	0.0358	10.0624	0.0356
7958	6	24	150	9.0379	0.0318	10.1929	0.0358	10.0574	0.0356
4955	4	20	84	9.5702	0.0337	10.1522	0.0356	10.0566	0.0356
5265	9	22	207	8.7445	0.0308	10.2552	0.0360	10.0824	0.0357
5071	4	23	96	9.3423	0.0329	10.2635	0.0360	10.0892	0.0357
7970	6	22	138	9.0438	0.0319	10.2246	0.0359	10.0500	0.0356
8039	9	27	252	8.6519	0.0305	10.2924	0.0361	10.0732	0.0357
6324	6	21	132	9.1013	0.0321	10.0952	0.0354	10.0665	0.0357
9814	6	30	186	8.8814	0.0313	10.2962	0.0361	10.0866	0.0357
4714	4	23	96	9.3814	0.0331	10.1230	0.0355	10.0530	0.0356
9847	6	28	174	9.0429	0.0319	10.1699	0.0357	10.0818	0.0357
7320	3	21	66	9.6547	0.0340	10.2426	0.0359	10.0775	0.0357
6011	3	24	75	9.6052	0.0338	10.1371	0.0356	10.0481	0.0356
4873	9	25	234	8.6767	0.0306	10.4099	0.0365	9.9874	0.0354
6664	3	22	69	9.6288	0.0339	10.1831	0.0357	10.0212	0.0355
4049	4	25	104	9.3127	0.0328	10.0450	0.0353	10.0396	0.0356
4168	4	24	100	9.3002	0.0328	10.2268	0.0359	10.0953	0.0358
6983	4	25	104	9.2755	0.0327	10.3115	0.0362	10.0804	0.0357
8036	6	26	162	8.9231	0.0314	10.3255	0.0362	10.0738	0.0357
7905	6	27	168	9.0998	0.0321	10.1121	0.0355	10.0573	0.0356

Table 6.1: Best models resulting from the MOGA run

ID	Generation	Centers	Inputs
7970	79	6	1, 3, 19, 26, 42, 61, 72, 95, 96, 97, 99, 104, 116, 118, 121, 126, 131, 132, 135, 136, 137, and 140
6011	60	3	1, 2, 19, 23, 48, 82, 84, 88, 95, 96, 97, 99, 103, 105, 112, 116, 120, 123, 125, 131, 132, 135, 137, and 140
4873	48	9	1, 2, 3, 12, 21, 23, 52, 53, 62, 79, 80, 96, 97, 99, 102, 103, 105, 116, 120, 131, 132, 134, 136, 137, and 140
6664	66	3	1, 2, 19, 29, 52, 74, 80, 88, 95, 96, 97, 103, 105, 116, 120, 123, 125, 127, 131, 132, 137, and 140
4049	40	4	1, 2, 19, 23, 26, 53, 79, 88, 96, 97, 99, 102, 104, 105, 114, 116, 117, 120, 123, 126, 131, 132, 134, 137, and 140

Table 6.2: Features for best models from MOGA run

6.4 Applying MOGA with prediction horizon object

To use MOGA with prediction ahead as another object with ϵ_t, ϵ_g, and complexity , a time sequence data set is needed for both the power lags and day time code, so MOGA can generate the lags inputs needed to make the prediction.

The additional set is taken from the data as a two months sequence, from mid of April to mid of June. The data sets used are shown in figure 6.3. With the exception for the time sequence data set, the other data are exactly the same sets from the previous experiment.

With the same parameters and objectives of the previous experiment a new objective (RMSE of the months sequence over the prediction horizon), is set as a constraint of 85 kW. The prediction horizon considered is 48 steps.

6.5 Results for MOGA with prediction horizon object

Table 6.3 shows the best models considering the validation set, for one step-ahead. Using the prediction objective, there were 166 models in Pareto front set. Table 6.4 shows the features for the best 5 models performance, on the validation set ϵ_v.

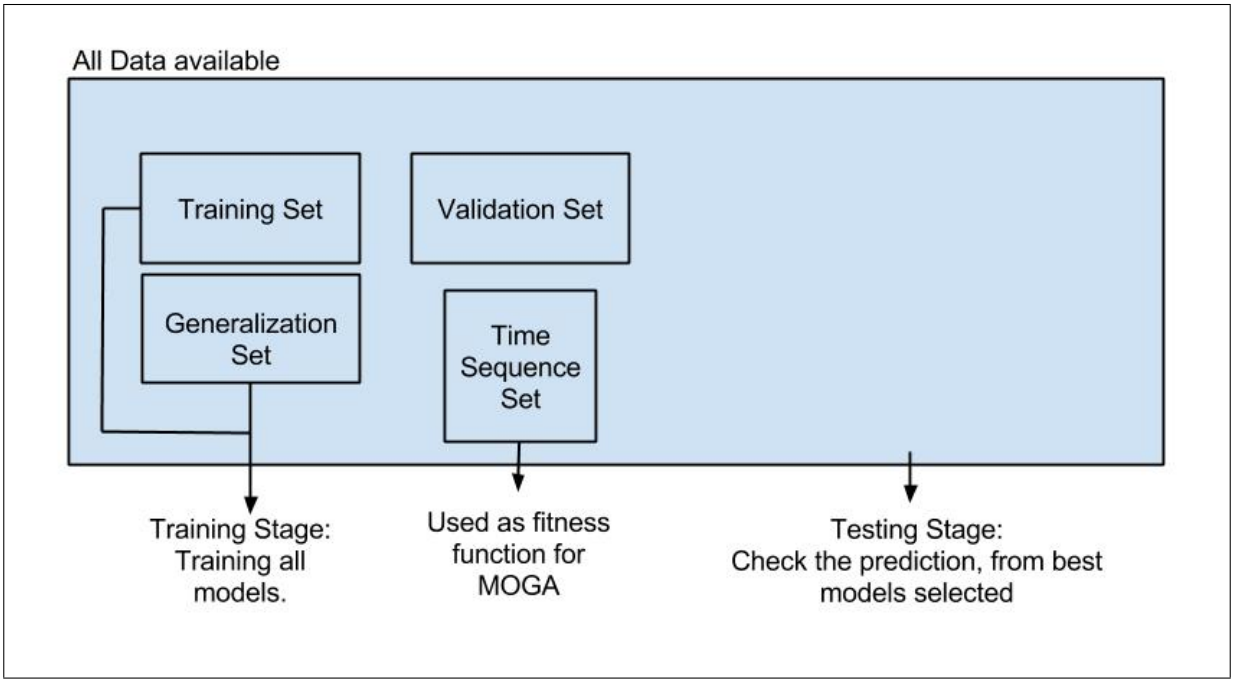


Figure 6.3: Data sets used in MOGA with prediction ahead object

A prediction for a 96 prediction horizon was done using the whole data set, and the performance of best three models are shown in figure 6.4.

ID	Centers	input	$o(\mu)$	ϵ_t	$\frac{\epsilon_t}{RMS(t)}$	ϵ_g	$\frac{\epsilon_g}{RMS(g)}$	ϵ_v	$\frac{\epsilon_v}{RMS(v)}$	$\sum \epsilon_p h$
8869	9	24	225	9.0748	0.0320	10.4605	0.0367	10.3005	0.0365	1622.2876
9624	4	21	88	10.0703	0.0355	10.7150	0.0376	10.5634	0.0374	1732.9768
9942	7	21	154	9.2578	0.0326	10.4391	0.0366	10.4161	0.0369	1617.2121
1665	2	25	52	10.0138	0.0353	10.5946	0.0372	10.3995	0.0368	1513.2488
5338	9	19	180	9.0671	0.0319	10.4835	0.0368	10.4417	0.0370	1584.5808
8140	4	23	96	9.8814	0.0348	10.4319	0.0366	10.4768	0.0371	1472.3280
9816	9	25	234	8.9351	0.0315	10.4108	0.0365	10.4144	0.0369	1715.8687
9856	7	23	168	9.2425	0.0326	10.4008	0.0365	10.3106	0.0365	1457.1846
5061	9	24	225	9.0323	0.0318	10.4121	0.0365	10.2638	0.0364	1700.4319
9879	7	24	175	9.3700	0.0330	10.3732	0.0364	10.5169	0.0372	1582.1001
7080	4	18	76	10.1679	0.0358	10.8216	0.0380	10.7086	0.0379	1541.8714
9539	9	20	189	9.0209	0.0318	10.4897	0.0368	10.4933	0.0372	1689.8243
9045	7	19	140	9.4420	0.0333	10.5032	0.0369	10.3523	0.0367	1473.6278
9928	9	22	207	9.0911	0.0320	10.3518	0.0363	10.2918	0.0365	1521.2612
9963	7	25	182	9.1927	0.0324	10.2875	0.0361	10.3320	0.0366	1529.4953
9982	9	23	216	8.9358	0.0315	10.4791	0.0368	10.5405	0.0373	1607.6294
9210	3	21	66	10.0308	0.0353	10.7135	0.0376	10.5159	0.0372	1396.7245
9036	9	20	189	8.9665	0.0316	10.7806	0.0378	10.4363	0.0370	1588.6442
7345	9	22	207	8.9628	0.0316	10.6296	0.0373	10.4159	0.0369	1696.3115
9702	9	19	180	9.1571	0.0323	10.8787	0.0382	10.6390	0.0377	1518.9623

Table 6.3: Best model results from MOGA run with perdition objective

ID	Generation	Centers	Inputs
1665	16	2	1, 16, 18, 22, 29, 39, 45, 46, 47, 62, 69, 78, 87, 92, 94, 95, 96, 97, 99, 117, 123, 131, 132, 138, and 140
8140	81	4	1, 2, 4, 15, 18, 23, 30, 34, 49, 58, 62, 64, 82, 95, 99, 117, 118, 123, 126, 128, 131, 134, and 142
9856	98	7	1, 2, 6, 13, 30, 40, 42, 54, 57, 60, 93, 96, 99, 117, 118, 121, 122, 123, 127, 129, 130, 133, and 142
9045	90	7	1, 2, 40, 45, 71, 74, 79, 89, 95, 99, 116, 117, 118, 121, 122, 127, 130, 133, and 142
9210	92	3	1, 3, 10, 16, 17, 21, 31, 56, 64, 69, 82, 85, 95, 97, 114, 122, 127, 131, 132, 136, and 142

Table 6.4: Features for best models from MOGA run with prediction objective

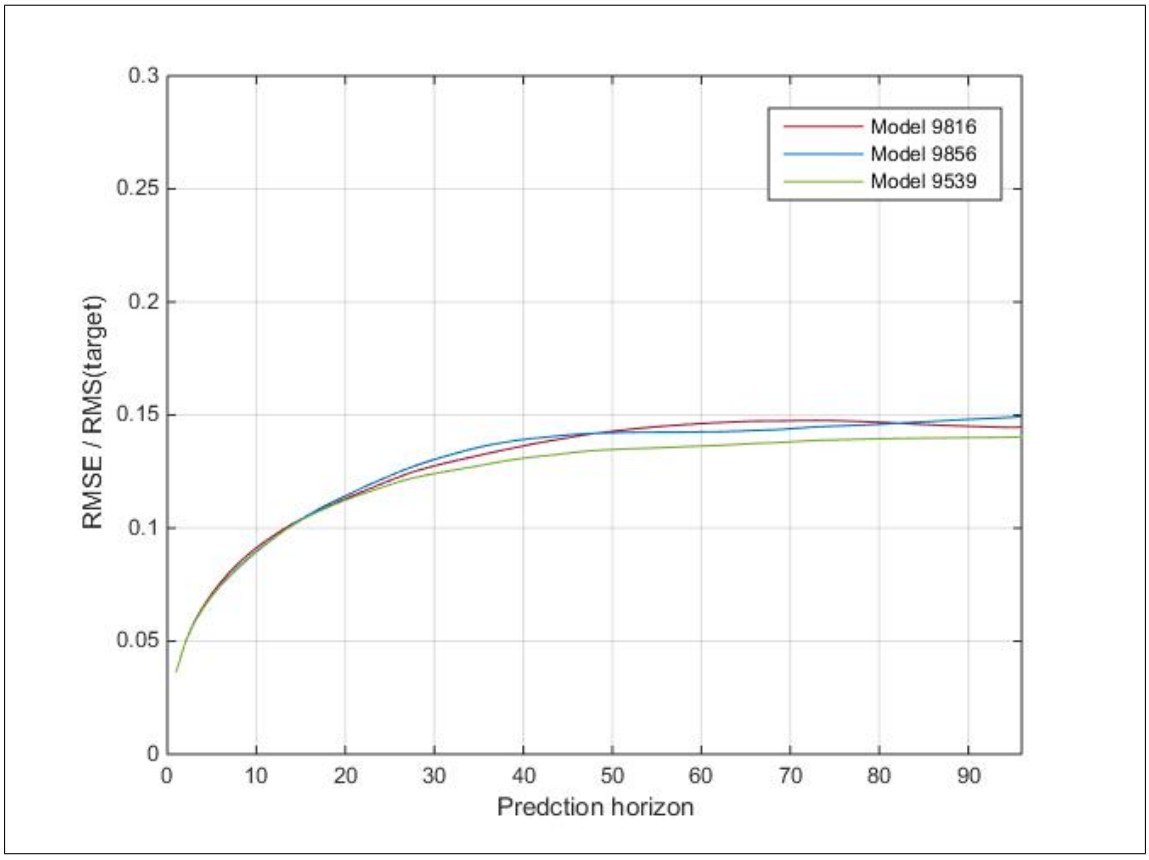


Figure 6.4: Performance of best three models from MOGA with prediction objective

A further comparison between MOGA with prediction objective, and without using it, is shown in figure 6.5.

The models with prediction objective have better performance, with smaller RMSE for validation set compared with models of the same complexity obtained without the prediction objective.

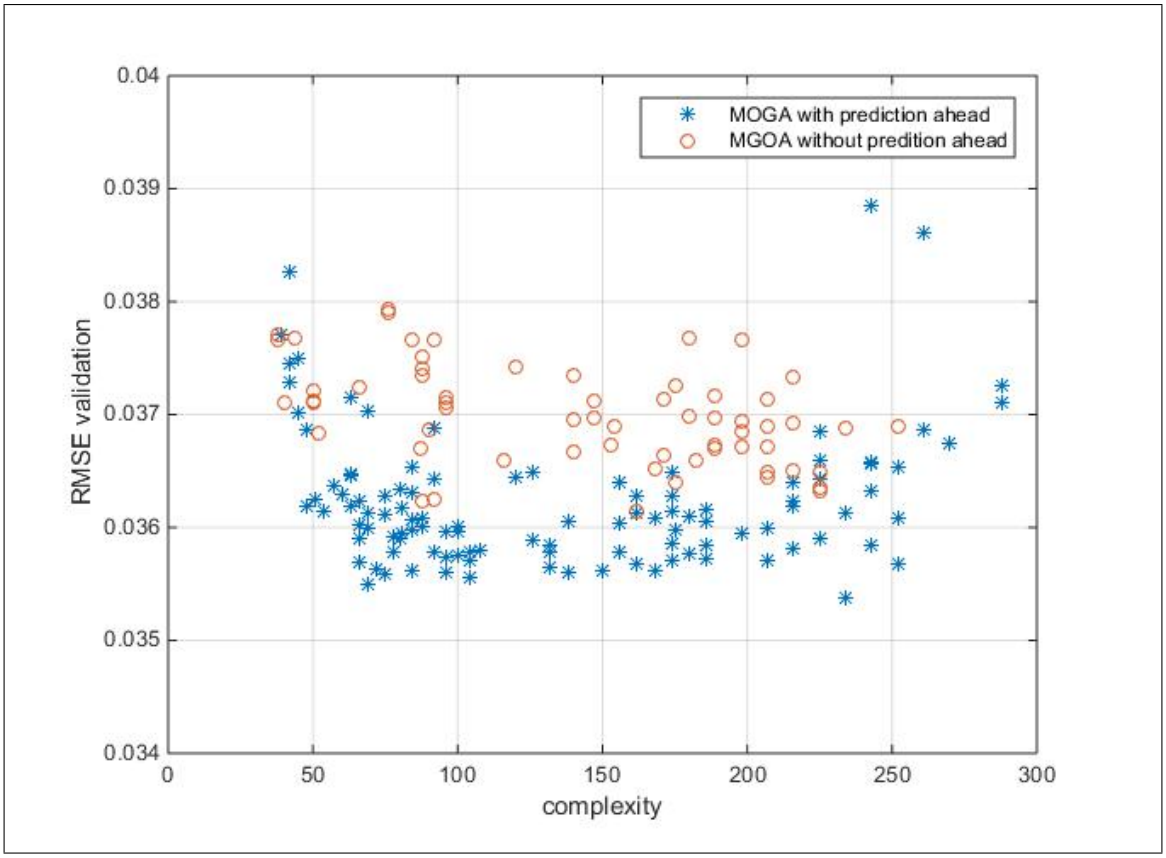


Figure 6.5: Complexity versus RMSE of validation for preferable set models

Chapter 7

Results

The results of each experimental part are shown on the related chapter. In this chapter a comparison between the obtained results is presented, and additional results for the best models are shown.

Table 7.1, show the results for the best 5 models, considering one-step ahead error on the validation set, for the different experiment. Table 7.2, and figure 7.1, show the results of the best three models, considering a 96 step ahead prediction (one day prediction), illustrating its 1st step prediction, 96 steps over the validation set, and 96 steps prediction computed over the whole data (one year).

The last figures 7.2(a), 7.2(b), and 7.2(c) show the one-step-ahead prediction, for best models, from MLP training, RBF training, and model evolved MOGA, with and without prediction objective.

ID	Topology	Input	Complexity	ϵ_t	$\frac{\epsilon_t}{RMS(t)}$	ϵ_g	$\frac{\epsilon_g}{RMS(g)}$	ϵ_v	$\frac{\epsilon_v}{RMS(v)}$
MLP									
162	[4 4 1]	3	36	12.0044	0.0423	12.8131	0.0450	12.4070	0.0439
176	[4 4 1]	3	36	12.0116	0.0423	12.6617	0.0444	12.4038	0.0439
322	[4 8 1]	3	56	12.1502	0.0428	12.7540	0.0448	12.4054	0.0439
412	[4 8 1]	3	56	12.0375	0.0424	12.7532	0.0448	12.3537	0.0438
414	[4 8 1]	3	56	12.0224	0.0424	12.8690	0.0452	12.3990	0.0439
RBF									
330	6	3	24	12.3682	0.0436	12.7642	0.0448	12.4793	0.0442
1410	15	3	60	12.1728	0.0429	12.7377	0.0447	12.4766	0.0442
1526	15	3	60	12.1237	0.0427	12.7802	0.0449	12.4790	0.0442
1546	15	3	60	12.1843	0.0429	12.7443	0.0447	12.4730	0.0442
1568	15	3	60	12.2155	0.0430	12.7226	0.0446	12.4794	0.0442
RBF using MOGA									
7970	6	22	138	9.0438	0.0319	10.2246	0.0359	10.0500	0.0356
6011	3	24	75	9.6052	0.0338	10.1371	0.0356	10.0481	0.0356
4873	9	25	234	8.6767	0.0306	10.4099	0.0365	9.9874	0.0354
6664	3	22	69	9.6288	0.0339	10.1831	0.0357	10.0212	0.0355
4049	4	25	104	9.3127	0.0328	10.0450	0.0353	10.0396	0.0356
RBF using MOGA with prediction objective									
1665	2	25	52	10.0138	0.0353	10.5946	0.0372	10.3995	0.0368
8140	4	23	96	9.8814	0.0348	10.4319	0.0366	10.4768	0.0371
9856	7	23	168	9.2425	0.0326	10.4008	0.0365	10.3106	0.0365
9045	7	19	140	9.4420	0.0333	10.5032	0.0369	10.3523	0.0367
9210	3	21	66	10.0308	0.0353	10.7135	0.0376	10.5159	0.0372

Table 7.1: Best 5 models, considering the RMSE prediction (1 step-ahead) of validation set, for all experiments

ID	Topology	Input	Complexity	RMSE(1 step)	RMSE(96 step)	$\sum$ RMSE (96 steps)
MLP						
162	[4 4 1]	3	36	12.5457	94.6637	8293.6
176	[4 4 1]	3	36	12.4535	95.2815	8552.2
322	[4 8 1]	3	56	12.5113	92.0609	8289
RBF						
489	6	3	24	12.5441	90.7668	10131
567	6	3	24	12.5379	93.0052	9994.2
620	6	3	24	12.5410	96.4129	9981.8
RBF using MOGA						
6011	3	24	75	10.1389	63.2932	4585.5
4873	9	25	234	10.1239	70.3314	4349.7
4049	4	25	104	10.0693	70.8194	4531.6
RBF using MOGA with prediction objective						
9816	9	25	234	10.3221	40.9932	3515
9856	7	23	168	10.3335	42.3062	3516.1
9539	9	20	189	10.3271	39.7712	3369.6

Table 7.2: Best 5 models, considering RMSE prediction of 96 steps, for whole data, for all experiments

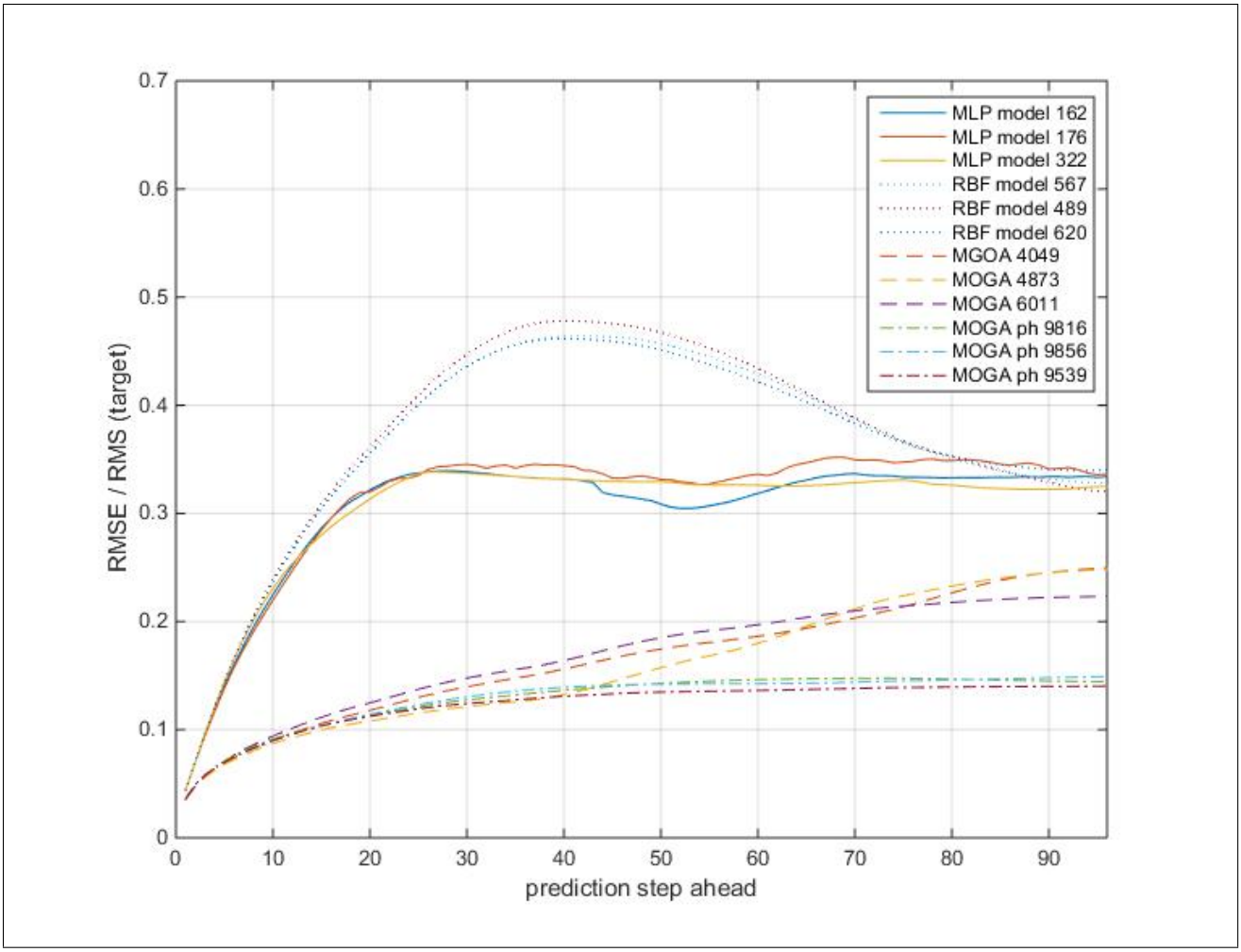
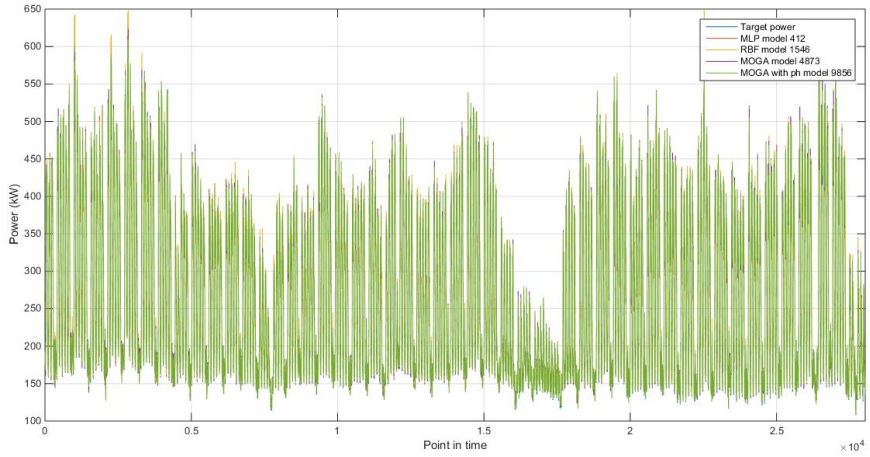
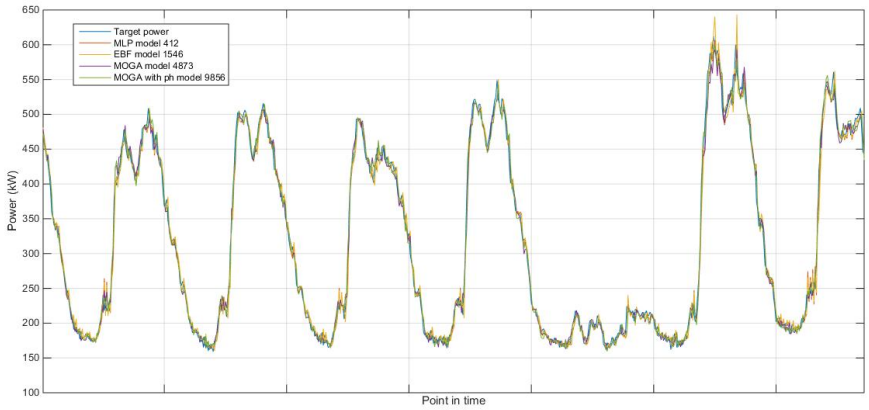


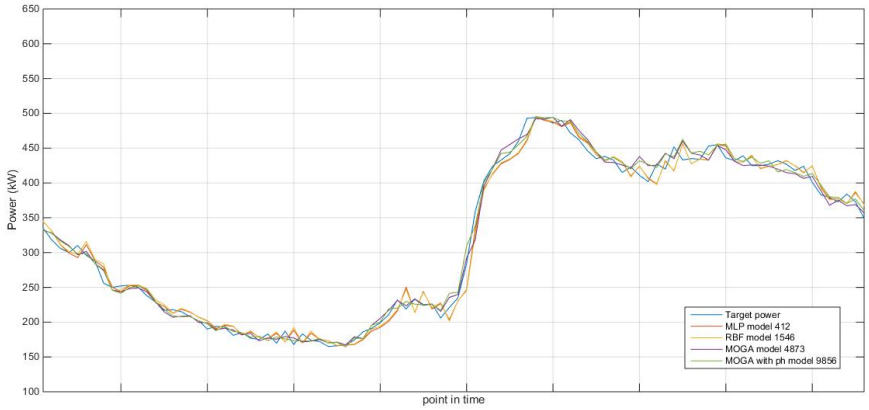
Figure 7.1: Performance of best three models from all experiments



(a) Target power and one-step-ahead prediction for whole year



(b) Target power and one-step-ahead prediction for one week



(c) Target power and one-step-ahead prediction for one day

Figure 7.2: Target power and one-step-ahead prediction, for best model, from each experiment

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this master dissertation, NN models are trained by the Levenberg-Marquardt algorithm using a modified training criterion, and the model structure (number of neurons and input terms) is evolved using a multi-objective genetic algorithm. The set of goals and objectives used in the model optimization reflect different requirements in the design: obtaining good generalisation ability, good balance between one- step-ahead prediction accuracy, multi steps-ahead prediction accuracy and model complexity.

The MLP and RBF experiments gave reasonable results for one step ahead prediction. The MOGA implementation to select the number of neurons, and the inputs for RBF NN, obtain models with better accuracy, and show more applicability for this problem, the ELD forecasting for University campus. These results show no more than 10 neurons were used in the NN models selected. In fact in the RBF experiment without using MOGA, also show that for the same inputs, more neurons will increase the one-step prediction accuracy, but for long prediction horizon less neurons preforms better. Also when MOGA was used with the prediction objective, the numbers of neurons stays low, and the number of input appears to be between 20 and 25.

When MOGA implemented, the most inputs selected were the 1st, 2nd, 96th, 97th (which are for one day period), 137th and 140th (which are for the week period), and when MOGA is implemented with the prediction objective, the most selected inputs were almost the same,

with additional the 142th input (which is the day type code input). This eXogenous input was beneficial to have better forecasting results.

8.2 Future Work

For future work, testing models should be recommended to takes into account temperature, precipitation amount, insolation duration, and humidity. The use of this meteorological variables that can be easily obtain from the local weather station as exogenous inputs, possibly improving the accuracy. Weather inputs, and other input like a distinguishes the day of the week and the occurrence and severity of holidays based on the day they occur, hour of day [10], with the day type as holiday or not, can be added [20], or used as one input [5], and other different inputs can be also used to improve this work.

Without using MOGA, MLP performs a little better than RBF, considering longer prediction horizons, although it almost the same in one-step ahead predictions, so using MOGA with MLP ANNs can be interesting.

Bibliography

[1] A. E. Ruano, *Artificial Neural Networks*. Centre for Intelligent Systems, University of Algarve, 2003.

[2] P. M. Ferreira and A. E. Ruano, "Evolutionary multiobjective neural network models identification: evolving task-optimised models," in *New Advances in Intelligent Signal Processing*. Springer, 2011, pp. 21–53.

[3] P. M. Ferreira, A. E. Ruano, R. Pestana, and L. Kóczy, "Evolving rbf predictive models to forecast the portuguese electricity consumption," in *Intelligent Control Systems and Signal Processing*, vol. 2, no. 1, 2009, pp. 414–419.

[4] P. M. Ferreira, A. E. Ruano, and R. Pestana, "Towards online operation of a RBF neural network model to forecast the Portuguese electricity consumption," *IEEE 7th International Symposium on Intelligent Signal Processing*, pp. 1–7, Sep. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6051697

[5] P. M. Ferreira, A. E. Ruano, and R. Pestana, "Improving the identification of rbf predictive models to forecast the portuguese electricity consumption," in *Control Methodologies and Technology for Energy Efficiency*, vol. 1, no. 1, 2010, pp. 208–213.

[6] S. X. Chen, H. B. Gooi, and M. Q. Wang, "Solar radiation forecast based on fuzzy logic and neural networks," *Renewable Energy*, vol. 60, pp. 195–201, 2013.

[7] H. M. I. Pousinho, V. M. F. Mendes, and J. P. S. Catalão, "Neuro-fuzzy approach to forecast wind power in portugal," in *International Conference on Renewable Energies and Power Quality (ICREPQ'10), Spain*, 2010, pp. 1–4.

[8] M. Pcolka, E. Zacekova, R. Robinett, S. Celikovsky, and M. Sebek, "Economical nonlinear model predictive control for building climate control," in *American Control Conference (ACC), 2014*. IEEE, 2014, pp. 418–423.

[9] Y. Ren, P. N. Suganthan, and N. Srikanth, "Ensemble methods for wind and solar power forecastinga state-of-the-art review," *Renewable and Sustainable Energy Reviews*, vol. 50, pp. 82–91, 2015.

[10] M. Q. Raza and A. Khosravi, "A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings," *Renewable and Sustainable Energy Reviews*, vol. 50, pp. 1352–1372, 2015.

[11] R. K. Jain, K. M. Smith, P. J. Culligan, and J. E. Taylor, "Forecasting energy consumption of multi-family residential buildings using support vector regression: Investigating the impact of temporal and spatial monitoring granularity on performance accuracy," *Applied Energy*, vol. 123, pp. 168–178, 2014.

[12] A. S. Ahmad, M. Y. Hassan, M. P. Abdullah, H. A. Rahman, F. Hussin, H. Abdullah, and R. Saidur, "A review on applications of ann and svm for building electrical energy consumption forecasting," *Renewable and Sustainable Energy Reviews*, vol. 33, pp. 102–109, 2014.

[13] H. R.J. and S. E., "Modelling long-term peak half-hourly electricity demand for south australia," *Report for Electricity Supply Industry Planning Council (SA)*, 2007.

[14] F. Kaytez, M. C. Taplamacioglu, E. Cam, and F. Hardalac, "Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines," *International Journal of Electrical Power & Energy Systems*, vol. 67, pp. 431–438, 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0142061514007637

[15] M. López, S. Valero, C. Senabre, J. Aparicio, and A. Gabaldon, "Application of som neural networks to short-term load forecasting: the spanish electricity market case study," *Electric Power Systems Research*, vol. 91, pp. 18–27, 2012.

[16] W. Xu, R. Gu, Y. Liu, and Y. Dai, "Forecasting energy consumption using a new gm–arma model based on hp filter: The case of guangdong province of china," *Economic Modelling*, vol. 45, pp. 127–135, 2015.

[17] Y. Li, Y. Su, and L. Shu, "An armax model for forecasting the power output of a grid connected photovoltaic system," *Renewable Energy*, vol. 66, pp. 78–89, 2014.

[18] C. Fan, F. Xiao, and S. Wang, "Development of prediction models for next-day building energy consumption and peak power demand using data mining techniques," *Applied Energy*, vol. 127, pp. 1–10, 2014.

[19] N. Fumo, "A review on the basics of building energy estimation," *Renewable and Sustainable Energy Reviews*, vol. 31, pp. 53–60, 2014.

[20] A. Bagnasco, F. Fresi, M. Saviozzi, F. Silvestro, and A. Vinci, "Electrical consumption forecasting in hospital facilities: An application case," *Energy and Buildings*, vol. 103, pp. 261–270, 2015.

[21] I. Esener, T. Yüksel, and M. Kurban, "Short-term load forecasting without meteorological data using ai based structures," *Turk J Electr Eng Comput Sci*, 2013.

[22] A. Azadeh, S. F. Ghaderi, S. Tarverdian, and M. Saberi, "Integration of artificial neural networks and genetic algorithm to predict electrical energy consumption," *Applied Mathematics and Computation*, vol. 186, no. 2, pp. 1731–1741, 2007.

[23] M. Rafei, S. E. Sorkhabi, and M. R. Mosavi, "Multi-objective optimization by means of multi-dimensional mlp neural networks," *Neural Network World*, vol. 24, no. 1, p. 31, 2014.

[24] C. Voyant, P. Randimbivololona, M. L. Nivet, C. Paoli, and M. Muselli, "Twenty four hours ahead global irradiation forecasting using multi-layer perceptron," *Meteorological Applications*, vol. 21, no. 3, pp. 644–655, 2014.

[25] H. Liu, H. Tian, and Y. Li, "Comparison of new hybrid feemd-mlp, feemd-anfis, wavelet packet-mlp and wavelet packet-anfis for wind speed predictions," *Energy Conversion and Management*, vol. 89, pp. 1–11, 2015.

[26] S. K. H. Chow, E. W. M. Lee, and D. H. W. Li, "Short-term prediction of photovoltaic energy generation by intelligent approach," *Energy and Buildings*, vol. 55, pp. 660–667, 2012.

[27] T. Bi, Z. Yan, F. Wen, Y. Ni, C. M. Shen, F. F. Wu, and Q. Yang, "On-line fault section estimation in power systems with radial basis function neural network," *International journal of electrical power & energy systems*, vol. 24, no. 4, pp. 321–328, 2002.

[28] L. Magnier and F. Haghighat, "Multiobjective optimization of building design using trnsys simulations, genetic algorithm, and artificial neural network," *Building and Environment*, vol. 45, no. 3, pp. 739–746, 2010.

[29] E. Asadi, M. G. da Silva, C. H. Antunes, L. Dias, and L. Glicksman, "Multi-objective optimization for building retrofit: A model using genetic algorithm and artificial neural network and an application," *Energy and Buildings*, vol. 81, pp. 444–456, 2014.

[30] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[31] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[32] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *Information Theory, IEEE Transactions on*, vol. 39, no. 3, pp. 930–945, 1993.

[33] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183–192, 1989.

[34] D. Broomhead and D. Lowe, "Multi-variable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, pp. 321–355.

[35] D. E. Rumelhart, J. L. McClelland, P. R. Group *et al.*, *Parallel distributed processing*. IEEE, 1988, vol. 1.

[36] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[37] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the second international conference on genetic algorithms*, 1987, pp. 14–21.

[38] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. ii. application example," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, no. 1, pp. 38–47, 1998.

[39] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, no. 1, pp. 26–37, 1998.

[40] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization," in *ICGA*, vol. 93. Citeseer, 1993, pp. 416–423.