A neural network meta-model for an ore crushing plant

by

Neil Schoonbee

26191068

Submitted in partial fulfillment of the requirements for

the degree of

BACHELORS OF INDUSTRIAL ENGINEERING

In the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT AND INFORMATION TECHNOLOGY

UNIVERSITY OF

PRETORIA

October 2010

Executive Summary

A data set obtained from Ite-Consultlink was used to test the capabilities and limitations of various neural networks. The different neural networks were identified and researched. The basic neural networks theory together with an overview of how these networks train was also researched. These networks were tested using a trial version of a program called Neurosolutions. The ANN's were then compared using a matrix format.

Neural networks were identified by Ite-Consultlink as a possible means for time and/or computational savings when used in conjunction with their traditional simulation modeling techniques. Output accuracy, and training times were among the requirements used to obtain suitable network performance indicators with which assess the performance of the networks.

Most of the networks provided good results on the data set whilst two of the network types did not obtain a suitable accuracy in its outputs. All the networks were compared using a relative scoring system, and a recommendation was made accordingly.

Contents

Table of figures

Glossary of terms and abbreviations

ANN– Artificial neural network.

Approximation – how well a network can determine an output

Axon - processing element in Neurosolutions

Epoch – one training iteration

Exemplars – data sets

Generalization – how well a network predicts new data

GFF – Generalized feed-forward network

LM – Levenberg Marquardt learning rule

LR – Linear regression network

MLP – Multi-layer perceptron

MSE – Mean Square error

PE – Processing Element

RBF – Radial basis function

PCA- Principal component analysis

1. Introduction

This chapter opens with an overview of the case study, which includes information concerning the company as well as the simulation model. Artificial Neural Networks are introduced followed by a brief overview of the software package that was used for this project.

1.1 Case Study: G2 model of an iron ore crusher plant

An iron ore crusher plant was used as a case study to study the various network types. The various networks were examined according to the requirements identified by Ite-Consultlink.

1.1.1 Ite-Consultlink

Ite-Consultlink specializes in modeling operational environments with the goal of locating problem areas, such as bottlenecks, and predicting the outcomes of future decisions (Ite Consulting, 2007). The company's clients often have very complex models, which can entail lengthy runtimes, and considerable computational requirements. In the case of this project's model the simulation is run for an entire year. Consequently a single run can take days to complete. It is in Ite-Consultlink's best interests to provide their clients with model results expediently and rapidly.

1.1.2 The G2 Gensym model

The current simulation model is built in G2 Gensym. G2 Gensym is the software package used for simulation and modeling by Ite-Consultlink. G2 uses a rule based engine that creates real-time expert system applications that can be used for optimization and improvement of production and processes (Gensym, 2009).

A sub-system of the Sishen Iron Ore crusher plant is the subject of this project. The sub-system is the tertiary crusher plant. The G2 model calculates the iron ore throughput based on various parameters.

The Tertiary crusher plant consists of 4 parallel tertiary subsystems. The sub systems feed into two parallel lines. These feed into a single line whereby the ore exits the system. It was assumed that for the purposes of the simulation model, these 4 subsystems are identical. Thus it was decided that one of the subsystems would be isolated and studied for the case study.

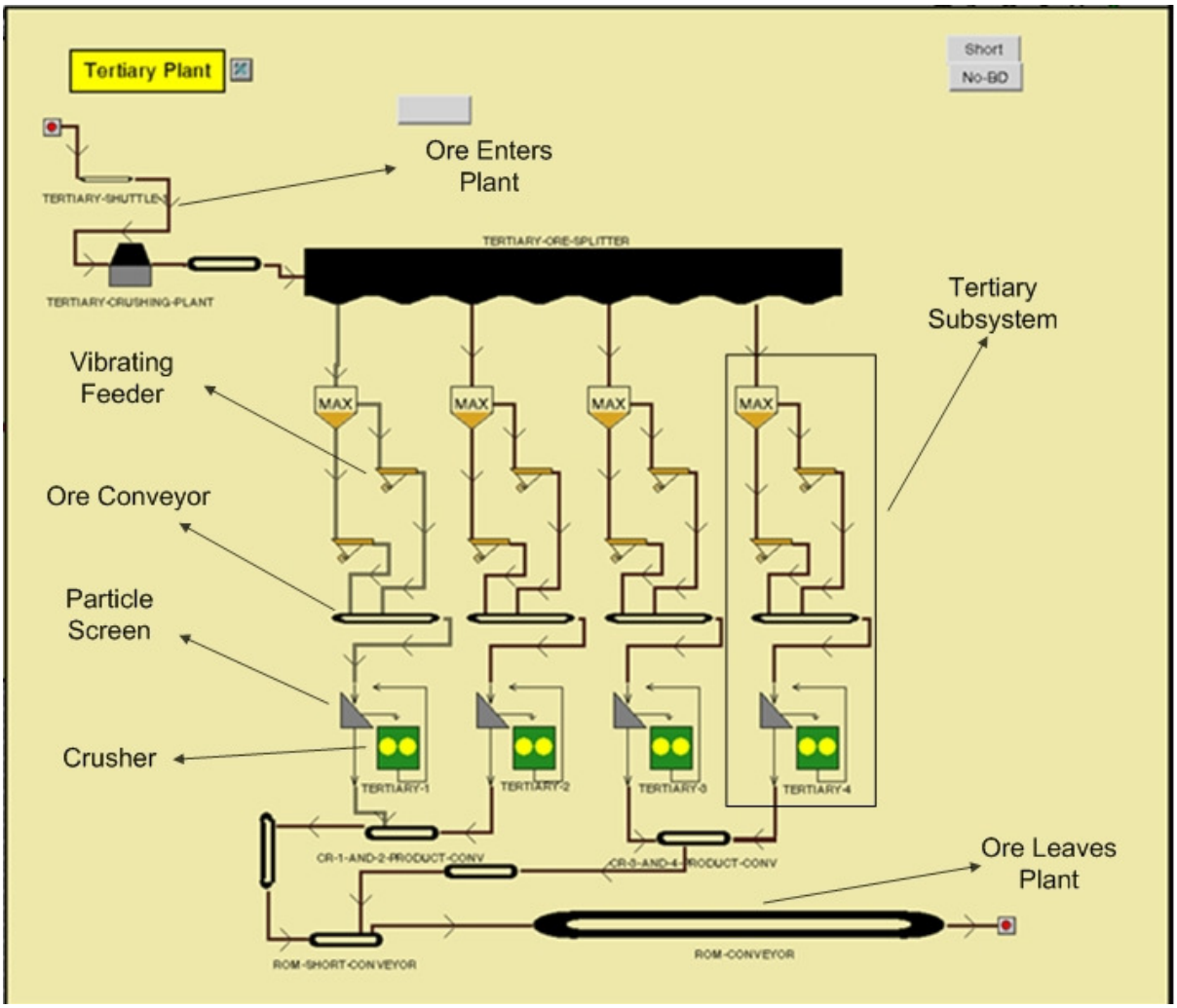

Figure 1: Tertiary plant

1.2 Artificial Neural Networks: What are they, and what aren't they?

An ANN is a set of processing elements (PE's) called neurons, these elements communicate with other elements in the network and process local data received from the data-set or other processing elements (Svozil, Kvasnicka, & Pospichal, 1997). The output of a neuron becomes the system output or input for another neuron. These neurons are connected with synaptic links. Most ANN's neurons are grouped in layers, where each layer's neurons are fully connected with each neuron in a successive layer. ANN's are used for a wide variety of applications but are most commonly used for pattern classification, which is what is required for this project.

If an ANN is mentioned today, some ability of the system to learn is usually implied (Karayiannis & Venetsanopoulos, 1993). Every processing element and synaptic link has

certain parameters which can include a weight and bias value. These values can be altered using a training algorithm and this is how a network learns.

A neural network derives its name from the fact that both the human brain, and the ANN has individual parts or ‘neurons’ that have some processing function. The link between cerebral functioning of the brain and artificial neural networks however, is tenuous (Willis, Montague, & Peel, ,1995). Neural networks have now simply become a problem solving tool. This project will deal exclusively with networks that receives input, and uses the correct output (which is available to the network) to alter its weight to obtain the desired accuracy.

Although neural network are efficient approximators, they have limitations, do not always work and even a working network may get stuck on a local minima or maxima.

1.3 Neurosolutions

The software that was used to build, train and test the models is called Neurosolutions, and was developed by a company called Neurodimension, who specialize in Neural Network software. The Program uses a graphical user interface and object-oriented approach to facilitate ease-of-use.

Neurosolutions can be used to build various types of network, and is not limited to pattern classification networks. Neurosolutions also allows for the processing element type, learning rule and the network topology to be selected.

Having decided to test the feasibility of ANN’s and identifying the software tool for the project, the problem needs to be succinctly defined and the scope of the project needs to be identified.

2. Scope Definition

This chapter expounds on the problem faced by Ite-Consultlink, defines the scope of this project and identifies the constraints of the project.

2.1 Problem Statement

Whilst running complex simulation models, outputs within the G2 environment depend on various parameters that are recalculated each time a state change occurs. The G2 model represents the physical layouts, processes and system rules of the plant. Consequently run times can be lengthy. Ite-Consultlink wants to know if some parts of the current model can be substituted by a neural network, so that the run times could be reduced.

Neural networks are known to be universal approximators, with a few advantages over other modeling approaches:

- The network does not need information about the actual system being modeled.
- A properly chosen network can retrain itself if a state in the system changes.
- A good network should be able to make intra- or even to some extent extrapolations from a training data set.

Ite-Consultlink wants to test these and other possible advantages that ANN's might offer, in their current modeling environment.

2.2 Constraints

Due to financial constraints a validation version of Neurosolutions, that provides limited functionality, is used for the project. The software does not allow one to save the network parameters, or export the network for further use.

For the purposes of testing and comparing the networks, the program is suitable, but Ite-Consultlink will have to purchase a license to implement an ANN.

2.3 Scope Statement

Neural networks will be trained on a fully representative data-set to determine whether the networks can accurately approximate data it has been trained with. Having been trained with a percentage of the data, the network will be tested on data it has not been trained with[1]. Ideally the network should be able to find a suitable output for any possible combination of input values. This entails that the network should be able to make interpolations on data that isn't in the given data-set.

Various function approximating network types will be studied and defined. The case-study data will be processed by each of these networks. Various parameters will be changed, and the results measured. The networks are then compared to each other and the best one is recommended for use by Ite-Consultlink.

[1] See Section 4.2 Data Preparation.

3. Requirement Analysis

Ite-Consultlink is looking to test the feasibility of neural networks. The first factor that determines whether a network is feasible is accuracy, but this is not the only requirement. A good neural network should be able to train quickly, and provide accurate outputs for data it has not been trained with, and data it has been trained with. A networks ability to predict the output of unseen data is referred to as generalization, and its ability to attain low errors with training data is referred to as approximation.

Ite-Consultlink's requirements for the ANN are a low run time, low training time and network accuracy. The following indicators were identified: **Elapsed run time (in seconds)**, **# runs before predetermined accuracy is reached**, **non-training data error** and **training data error**. The indicators are all directly linked to a requirement as shown below:

Performance Indicator	Related Requirement	Indicator effect
Elapsed run time (in seconds)	Low run time, low training time	Determines how quickly a network completes a run. Provides a means to compare networks in terms of computational requirements.
# runs before predetermined accuracy is reached	Low training time	A lower training time will save time in initial training, but will also impact positively when state changes occur in the G2 model. This indicator indicates how quickly a network converges to the correct outputs.
Non-training data error	Network accuracy	Will determine how well a network can interpolate. Determines how well a network generalizes.
Training data error	Network accuracy	Provides a means to compare networks, and determines how well a network approximates.

Table 1: Relation between performance indicators and requirements

Table 1 links each of Ite-Consultlink's three requirements with a measurable performance indicator. These performance indicators provide a way to quantify performance of an ANN

with regards to the requirements. They also provide a means to compare different ANN's to each other.

Ite-Consultlink had a specific requirement with regard to network accuracy:

> The percentage error shall be at most 0.1.

The specific requirement apply to the data that a network has been trained with, and data that requires interpolation.

Thus the project requirements have been converted into measurable indicators. The indicators provide a quantifiable measurement for the testing of the ANN's in order to determine how well an ANN meets the requirements.

4. Data Analysis & preparation

The chapter opens with a brief explanation of the tertiary crusher sub system followed by an explanation of the data generated by the G2 Gensym model of the sub system. The data preparation necessary for Neurosolutions and indeed ANN's in general is also discussed in 4.2.

4.1 Data Analysis

As can be seen in the figure 2, each subsystem has 2 parallel lines were the ore passes through a vibrating feeder. These two lines feed into a single line onto the conveyor component. The ore then passes into a screen that has an underflow and an overflow of ore. The screen's underflow has a finer particle distribution, and the coarser overflow enters the crusher component. The crusher component re-circulates its output into the screen. The screen's underflow exits the subsystem.

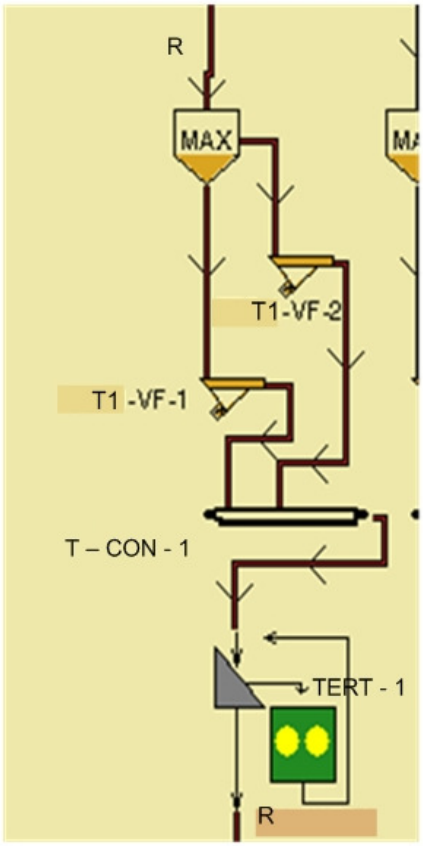

Figure 2: Tertiary Crusher plant sub-system

Table 2 is an extract of the data set, and illustrates the input and output columns.

Inputs						
Particle Distributions		Component States				Flow rate
Wc	Wm	T1-VF1	T1-VF2	T2-VF1	T-CON-1	R
0	0	1	1	1	1	980.358
0	0.067	1	1	1	1	979.418
0	0.133	1	1	1	1	978.48
0	0.2	1	1	1	1	977.544
0	0.267	1	1	1	1	976.609
0	0.333	1	1	1	1	975.676
0	0.4	1	1	1	1	974.745

Table 2: An Extract of the data set

W_c and W_m are inputs that represent the fraction of that particular batch of ore that consists of a coarse - or a medium particle distribution respectively[2]. The data that was generated from G2 for these two inputs are in discrete fractions of 10^{th}'s and 15^{th}'s. The data from these two sets were added together to create a larger set of data.

The remaining four inputs are binary numbers and give the states of the various components in the model. A 0 value indicates that the particular component is off and a 1 indicates that the component is on. T1-VF1 & T2-VF2 represent two parallel vibrating feeders, T – CON – 1 represents a conveyor and TERT – 1 represents the state of a crusher component.

R represents the flow rate of iron ore and in this model flow rate into the model is equal to flow rate out of the model.

Normalization of the input-output data must occur so that the output can occur within a 0-1 range (Hsu & Yang, 1995). The input values are already in the 0 and 1 range as they are fractions. Output values in the training set will be normalized according to the following equation:

$$R_n^{(m)} = \frac{R^{(m)} - R_{min}}{(R_{max} - R_{min})} \quad (1)$$

Here R_{min} and R_{max} are the minimum and maximum R values in the data set, and $R^{(m)}$ is the current R value prior to normalization.

[2] Note that W_f can always be inferred by subtracting W_c and W_m from 1.

The main performance indicator of a pattern classification network is how well it approximates the desired output. To establish a network's accuracy the Mean Squared error (MSE) is normally used, the MSE is defined as follows:

$$MSE = \frac{\sum_P \sum_O (x_k - \acute{\alpha}_k)2}{OP}$$

Where O represents the number of exemplars in the data set, P represents the number of output processing elements, x_k is the actual output, and $\acute{\alpha}_k$ is the desired output.

4.2 Data Preparation

Each row within the data set represents a set of inputs that are related to a single specific outcome. A row also represents a single exemplar within the data set. The binary inputs representing the states were initially added, but it was found that if one of the vibrating feeders were down (TER-1-VF-1 or TER-1-VF-2), the model returned the same R value for all particle size combinations. And if either the conveyor (T – CON – 1) or the crusher (TERT – 1) is down R becomes 0. Thus the component states were not included in the neural network data set. Only the rows (exemplars) were all component states were 1, was eventually added.

The final data set only included W_c, W_m and R. The rows of the dataset were then randomized. Neurosolutions normalizes the data automatically according to (1), so the output column was not normalized in excel.

The data was received in a spreadsheet format, and was converted to a csv. format to accommodate Neurosolutions. After the data was imported into Neurosolutions the data set had to be divided into three sub-sets. These sub-sets consist of **training data**, **validation data** and **testing data**. 55 % of the rows were tagged as training data, 15% as validation data and 30% as testing data.

4.2.1 Training Data

Training data relates to the input-output sets that are used to train the network's various parameters. In most cases training proceeds iteratively and the training data is continuously used to retrain the networks. The error between the actual output of the network, and the desired output, is used to update the parameters in the network.

4.2.2 Validation Data

Some networks can attain an almost arbitrarily small MSE on its training data. In most cases this means the network has been over-trained. An over-trained network will not generalize well, i.e. it will not accurately approximate data it has not been trained with (Svozil,

Kvasnicka, & Pospichal, 1997). Validation data is used to stop a network before over-training occurs. This is called the early-stopping method.

At certain intervals during training the current parameters of the network are frozen, and the network receives data it has not been trained with form the validation set. The MSE between the desired and actual output of validation data is calculated. If the MSE of the validation data set has reached its minimum value training is stopped. The point of best generalization has then been reached.

4.2.3 Testing Data

Although the validation error is used to determine when a network has reached the point when it is best able to generalize, validation error is normally not a good estimate of a network's ability to generalize (Svozil, Kvasnicka, & Pospichal, 1997). This is why a separate testing set is necessary. The network's ability to generalize is measured by how well it performs with the testing data set.

So in summary the validation data is used to decide when to stop training, and the testing set is used to test how well a network performs with unseen data (generalization).

5. Literature Review

In this chapter a literature review was conducted to understand the basics of ANNs, and to investigate the mechanisms behind selected neural networks. A basic understanding of neural networks was gained by first researching Multi-layer-perceptrons. More advanced networks followed, and the literature review concludes with an overview of neural network training.

5.1 Introduction

As stated in chapter one, ANNs refer to interconnected neurons, and these interconnected neurons has the ability to learn via an iterative training method.

This training method can be either supervised or unsupervised (Lippmann, 1987). In supervised training the network has access to the correct output, and the neural network then uses these values to calculate its error. The weights are updated so as to minimize this error (Rafiq, Bugmann, & Easterbrook, 2001). Although this data set includes the correct output values, and supervised training can then be applied, some of the network types combine unsupervised and supervised training methods. ANN's are used in 4 ways: pattern classification, associative memories, feature extraction and as dynamic networks (Neurodimension, 2010). For our purposes the networks are used to find a good input-output map, i.e. pattern classification. Neural networks are powerful approximators, and have found many applications in pattern classification.

It has been shown that a single hidden layer feed-forward ANN has approximation capabilities at least as powerful as those of the Fourier series (White, 1992). This finding led (Hornik & Stinchcombe, 1989) to state that their findings establish that multilayer feed-forward network can approximate, with any degree of accuracy, any function. Hornik & Stinchcombe showed that ANN's where universal approximators, but (Stinchcombe.M, 1993) later showed that other non-linear transformations in a feed-forward neural network also accomplished universal approximation.

5.2 Multilayer Perceptron

The multilayer perceptron (MLP) is the most basic neural network, and a thorough understanding of the MLP would aid in the understanding of the other network types. Thus the MLP was researched in detail.

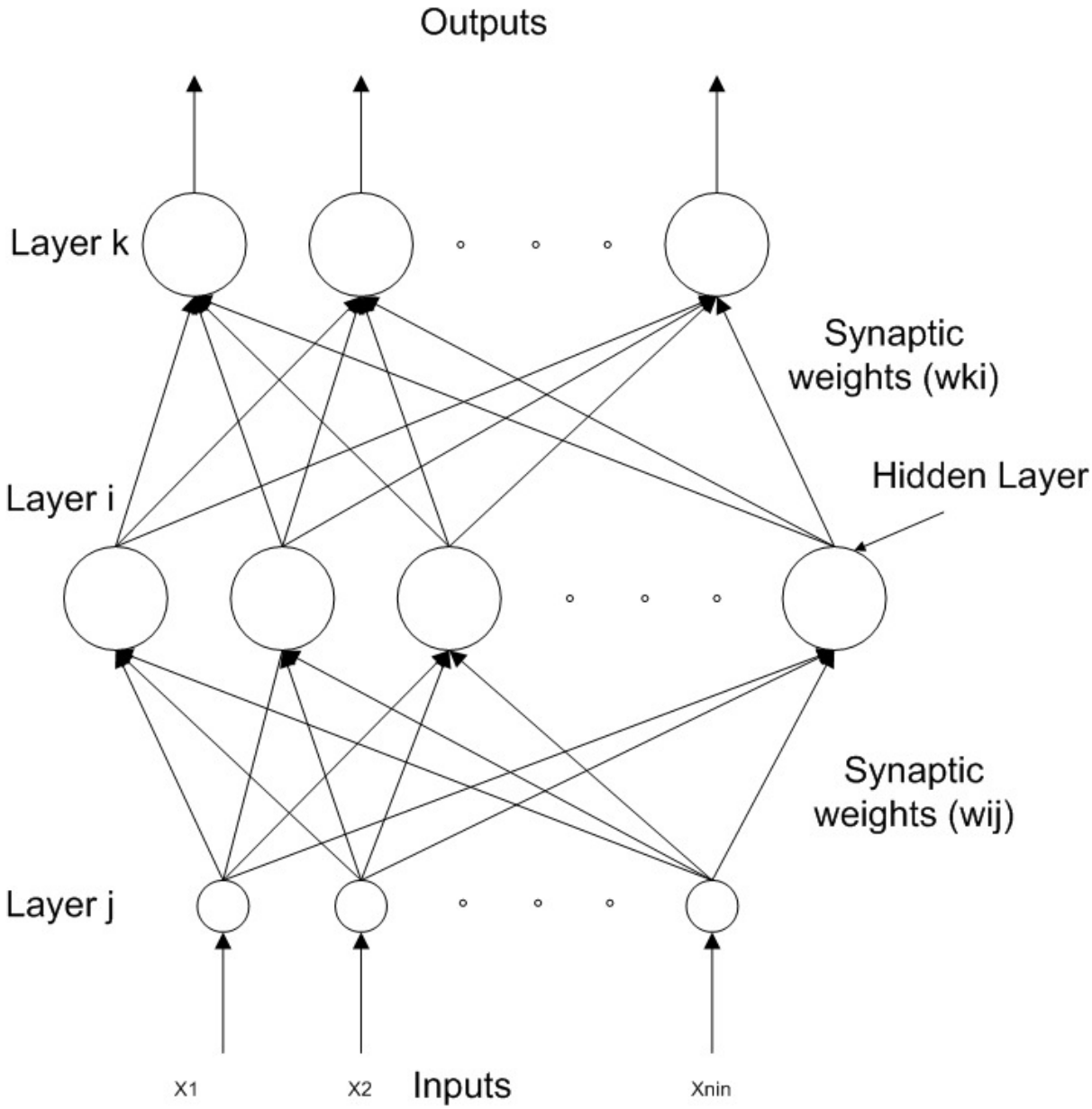

Figure 3: Generic Multi-layer Perceptron

As far as the application of ANN's go, the Multi-layer perceptron with the back-propagation training algorithm is the most popular method (Svozil, Kvasnicka, & Pospichal, 1997).In a MLP the neurons are grouped in layers. All neural networks have input layers and output layers, and any layers between these two would be called a hidden layer (Svozil, Kvasnicka, & Pospichal, 1997).At the input layer, each neuron's input is the input variable of the system, and the output layer's neurons correspond to the system outputs. The input of a neuron i in a hidden or output layer is given by

$$N_i = wt_i + \sum_{j=0}^{Nin} w_{ij} x_j \ . \qquad (2)$$

Where $wt_i$is known as the bias weight, x_j is the output from a neuron j in the previous layer, and w_{ij} is the synaptic weight associated with the current neuron i and neuron j in the previous layer. Note that each neuron in one layer is connected via synaptic weights to every neuron in the next level. N_{in} is the number of neurons in the previous layer. Equation (2) refers to the linear summation that is present in all PE's. N_i becomes the input of a transfer function so that

$x_i = \xi(N_i)$ (3)

x_i is the output of neuron I, and it is propagated through the network. Transfer functions can be linear or non-linear. Some common transfer functions include:

- The sigmoid function : $\frac{1}{1+e^{-N_i}}$ (4)
- The hyperbolic tan function : $\tanh(N_i)=\frac{e^{N_i}-e^{-N_i}}{e^{N_i}+e^{-N_i}}$ (5)
- Linear sigmoid(N_i)= $\begin{cases} 0 & N_i<0 \\ 1 & N_i>1 \\ N_i & \text{else} \end{cases}$ (4b)
- Linear tanh(N_i) = $\begin{cases} -1 & N_i<-1 \\ 1 & N_i>1 \\ N_i & \text{else} \end{cases}$ (5b)

5.3 Generalized Feed-forward models

These networks are exactly the same as MLP's, except that each processing element is connected to not only the next layer of the network, but to all successive layers in the network. The synapses can jump layers. These network types have been known to provide comparable result to MLP's with less training epochs (Neurodimension, 2010).

One of the reasons why these networks generally train faster than a traditional MLP, is because these networks present a larger number of connections between processing elements (Asensio-Cuesta, Diego-Mas, & Alcaide-Marzal, 2010).

5.4 Radial basis function Networks

Radial basis function networks have a layer of additional PE's situated over the input space. Each of these PE's contains a radial basis function, and this layer can provide a smoothness constraint when the data is noisy (Acosta, 1995). This layer is trained using and

unsupervised learning rule. The RBF layer can then be summated and used directly as the network output. Hidden layers of linear or non-linear PE's may also be added between the RBF layer and the output layer.

In addition to MLP's good approximation properties RBF's have also shown good properties with respect the approximation of non-linear functions (Zhang & Morris, 1998). A Gaussian radial basis function that selects only a portion of the input space is normally used to produce the output of an RBF PE as follows (Rafiq, Bugmann, & Easterbrook, 2001):

$$G_i(x_i)=e^{\left(-\frac{\sum_j(x_j-w_{ij})^2}{2\sigma^2}\right)} \quad (6)$$

In (6) x_j is the input from the input layer, w_{ij} is known as the centre of the receptive field and σ is the half-width of the receptive field. Each RBF receives a discrete part of the input space, called a cluster. According to (Rafiq, Bugmann, & Easterbrook, 2001) the output of the Gaussian function is then normalized by the output activity of all the hidden neurons. The Gaussian functions differ significantly from the MLP transfer functions as they only respond to data located near the peak of the data field, and are referred to as a local function approximators (Neurodimension, 2010).

The last layer or output layer of a RBF is usually summated:

$$Output_i=\sum_{j=1}^{m} w_{ij}\, x_j \quad (7)$$

Here it was formulated such that the output layer is layer i and the previous layer of hidden neurons are level j. The optional hidden layers, as well as the RBF output synapses, are updated with a conventional supervised training method, as in MLPs (Neurodimension, 2010).

Neurosolutions uses a transfer function that differs from formula (6).

$$G_i(N_i)=e^{-Bi(Ni+wti)^2}$$

Here N_i is the weighted summation of inputs from the input layer, wt_i is the weight or bias value and B_i is the function width. Wt_i is linked to the linear summation and determines the

center of the receptive field.

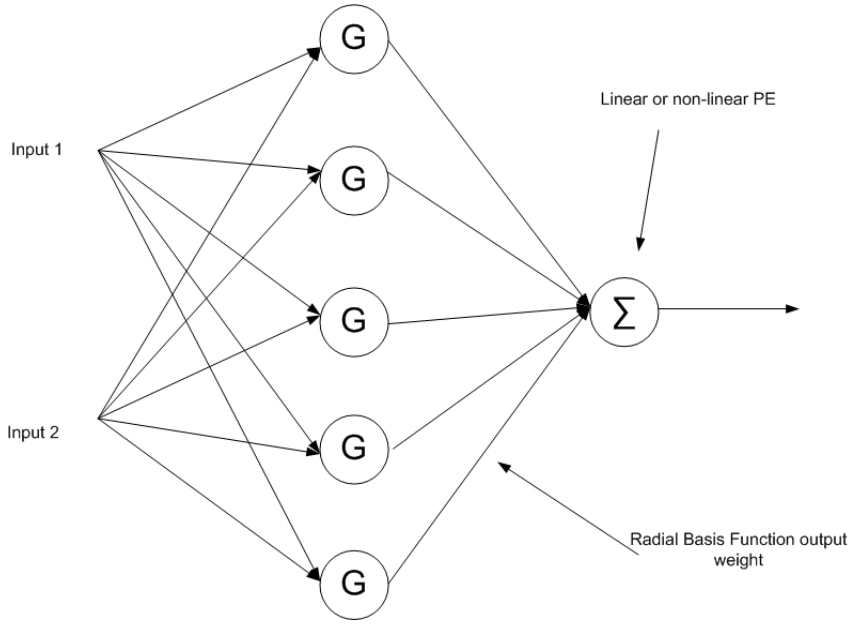

Figure 4: Generic Radial Basis Function Network

The hidden layer or layers of PE's would be situated between the output PE and the layer of RBF's as shown in figure 4 above.

5.5 Linear-Regression Models

Linear regression uses a linear function in its processing elements. These elements are also grouped in layers, and can contain hidden layers, similar to those found in MLPs. These networks can only express linear input-output maps (Baldi & Homik, 1995). This feature enables fast training, but may also yield a low network accuracy.

5.6 Modular neural networks

In these networks, separate networks are connected together in order to create faster networks, and avoid local minima (Tseng & Almogahed, 2009). Any network type can be used in a modular topology, but this literature review is restricted to MLPs that are connected in a modular network.

The different sub parts of the network partitions the sample space into subspaces (Zhao, 2009). As not all the PE's in a modular neural network are connected. This means that a smaller number of weights are required the same size MLP, this in turn can speed up training times and decrease network run times.

Neurosolutions only offers a limited choice in modular topology. All the topologies contain four separate single layer MLP's between the input and output layer. The sample space is divided into two subspaces. Each of these subspaces pass through two of the four single-layer MLPs. The input layer may or may not be directly connected to the output layer, and the first MLPs may or may not be directly connected to the output layer. Thus four different topologies are available.

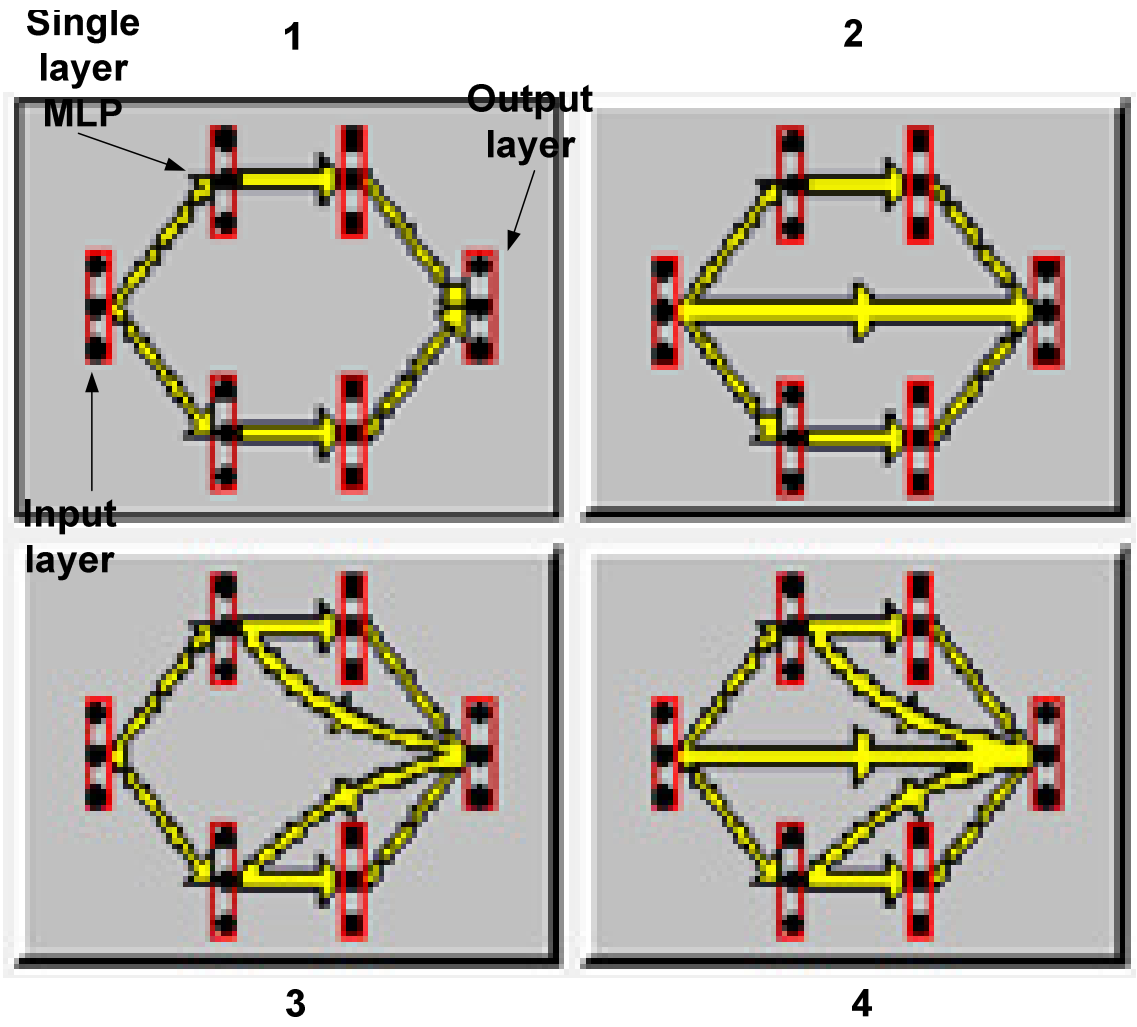

Figure 5: Modular topologies

5.7 Jordan/Elman Network

Jordan/Elman networks are networks that have extra PE's that are referred to as context units. These units store the past activity of other layers (Pham & Karaboga, 1999). These units are also trained using an unsupervised learning rule. One such network was proposed by (Elman, 1990) whose network copied the output of the first hidden layer to the context

units. The network proposed by (Jordan, 1986) copied the output of the output layer to the context units.

Neural solutions offer these two, and two other configurations as shown below:

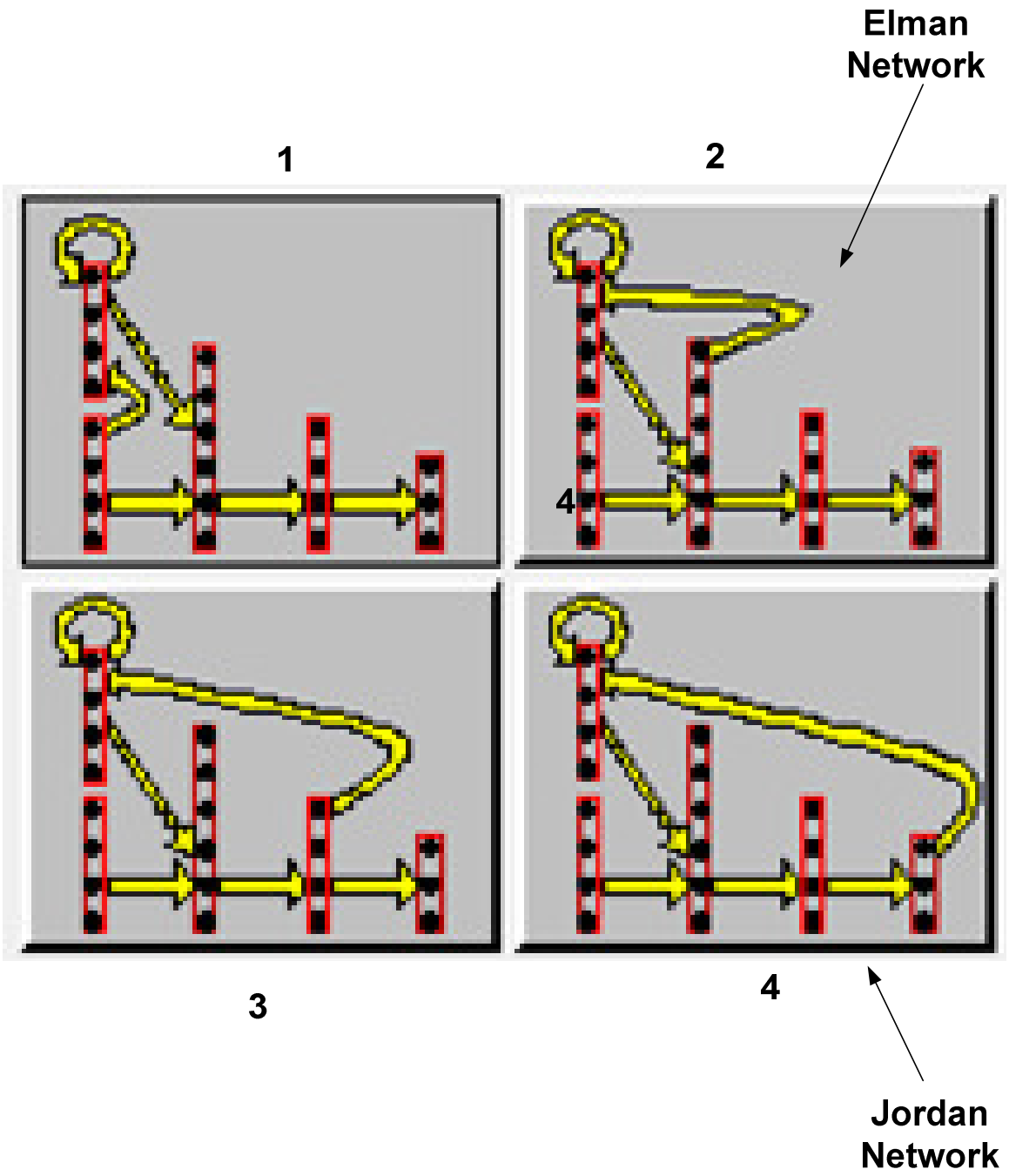

Figure 6: Jordan/Elman Topologies

5.8 Principal component analysis

Principal component analysis is a standard tool in pattern recognition, mainly for data-preprocessing and feature extraction (Karhunen, 1998). Principal component analysis is a technique used to transform the input space into a new space, where information about the input space is retained, but the dimensionality of the data is reduced. This makes classification easier, and thus can increase the effectiveness of the neural network.

One selects the number of principal components, and the input space is transformed, after which another network (in this case a MLP) is used to perform the classification of the data. These principle components are trained using an unsupervised learning rule.

5.9 Supervised Learning

Application of the mathematical chain rule is the most common method used to determine the effect hidden layers weights have on the outputs of an ANN (Murray, 1995). The methods name is derived from the fact that the method calculates the errors of the output layer before each preceding hidden layer's errors are calculated.

The new weights are attained by minimizing the MSE. The bias weights and synaptic weights are adjusted using the gradient descent method:

$$w_{ij}^{n+1}=w_{ij}^{n}-\lambda\left(\frac{\partial \varepsilon}{\partial w_{ij}}\right)^{n} \quad \text{(for synaptic weights)} \tag{8}$$

$$wt_{i}^{n+1}=wt_{i}^{n}-\lambda\left(\frac{\partial \varepsilon}{\partial wt_{i}}\right)^{n} \quad \text{(for bias weights)} \tag{9}$$

In the above equations λ is referred to as the learning rate and n is the iteration that has just been completed. The learning rate is an important parameter, if the value is too low, the learning can get stuck on a local optimum, and if the value is too high the weight might not converge (Neurodimension, 2010).

The MSE (denoted as ε in the derivatives) is calculated first; then by using the chain rule (Svozil, Kvasnicka, & Pospichal, 1997) the derivatives of the errors with respect to the weights are calculated:

$$\frac{\partial \varepsilon}{\partial w_{ij}}=\frac{\partial \varepsilon}{\partial x_i}\times\frac{\partial x_i}{\partial w_{ij}}=\frac{\partial \varepsilon}{\partial x_i}\xi'(N_i)x_j \tag{10}$$

$$\frac{\partial \varepsilon}{\partial wt_{i}}=\frac{\partial \varepsilon}{\partial x_i}\times\frac{\partial x_i}{\partial wt_{i}}=\frac{\partial \varepsilon}{\partial x_i}\xi'(N_i) \tag{11}$$

From (10) and (11) equation (12) can be easily found:

$$\frac{\partial \varepsilon}{\partial w_{ij}} = \frac{\partial \varepsilon}{\partial wt_i} x_j \quad (12)$$

When one has the derivatives of the error with respect to the bias- and synaptic weights in relation to each other as in (12) one still needs to compute $\frac{\partial \varepsilon}{\partial x_i}$. The formula for the derivative of the error with respect to the neuron output depends on whether one is working with an output or a hidden layer.

For output layer k:

$$\frac{\partial \varepsilon}{\partial x_k} = x_k - \acute{\alpha}_k \quad (13)$$

For hidden layer i:

$$\frac{\partial \varepsilon}{\partial x_i} = \sum_k \frac{\partial \varepsilon}{\partial wt_k} w_{ki} \quad (14)$$

After calculating (13) the transfer function derivatives are calculated and used to obtain the 2 error derivatives of the outer layer. These answers are then used to calculate (14) after which formulas (10) and (11) are again used to obtain the error derivatives with respect to the weights.

6. Methods and Mechanisms

After the literature review, the necessary knowledge has been attained concerning ANN's. Now the way in which the best ANN can be found is examined. The chapter opens with an introduction into the software that was used, Neurosolutions. The chapter concludes with a testing methodology. The methodology was deemed necessary due to the large number of parameters ANN's contain. The performance indicators are given in bold in section 6.2. The chapter closes with a graphical representation of the testing methodology.

6.1 Neurosolutions

Neurosolutions uses a graphical user interface such as the one seen in figure 4. After one has selected the network type and the various parameters Neurosolutions generates an interface that represents the physical network architecture, its learning mechanisms, the network data as well as the performance measures.

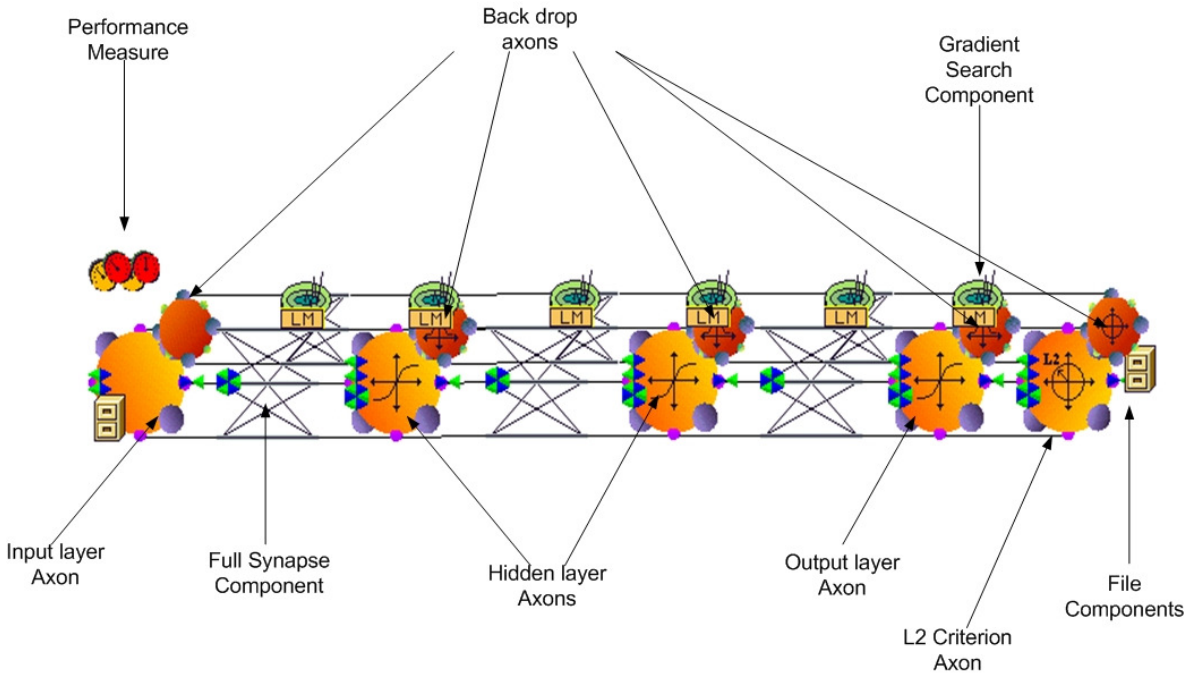

Figure 7: 2 layer MLP in Neurosolutions

The large yellow circles represent the activation family of objects. The Activation family consists of all the parts that form the actual neural network. The lines above indicate the full synapses, and the large yellow circles are called axons. Axons performs some identity map between its input and output activity (Neurodimension, 2010). The first and second last axons are the input and output layers respectively. The 2 axons between them represent the network's 2 hidden layers. The last axon calculates the cost function by using the Mean

Square Error method (see 4.1) and is part of the Error criteria sub-family. The second last axon represents the network output layer.

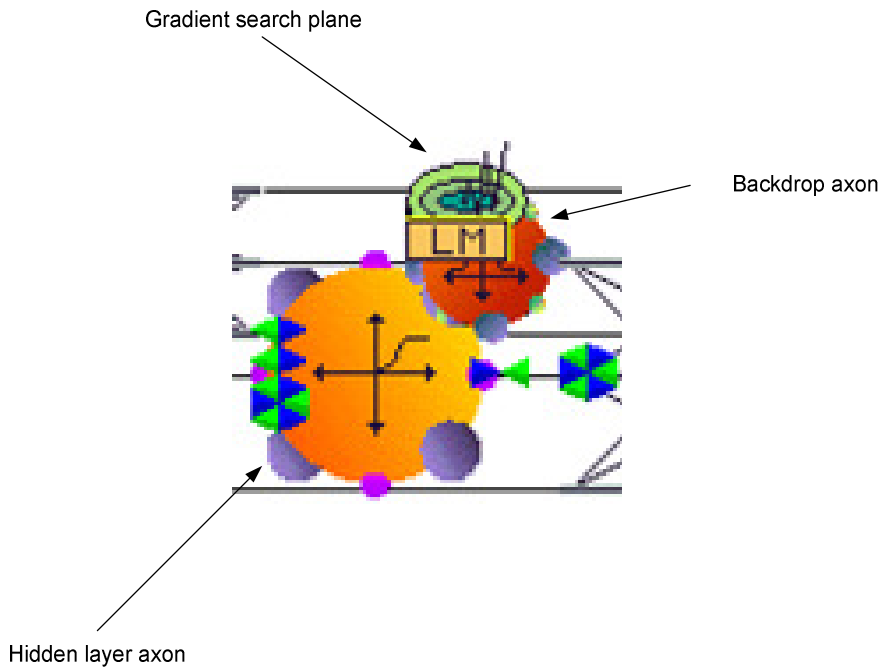

Figure 8: Axon, back-axon and gradient search component

The smaller red circles are called back axons, and form part of the Backdrop family. Each axon in the activation family has a corresponding back axon. These back axons perform two functions: 1) given their dual-axon's error, they must calculate the gradient information for all the bias and synapse weights (formulas 10, 11, 12 and 14) and 2) they must back-propagate the relative errors to their preceding back-axon components. Notice from figure 6 that the full synapse components do not have back axons. This is because the synapse weight errors are calculated in formulas 10, 11, 12 and 14.

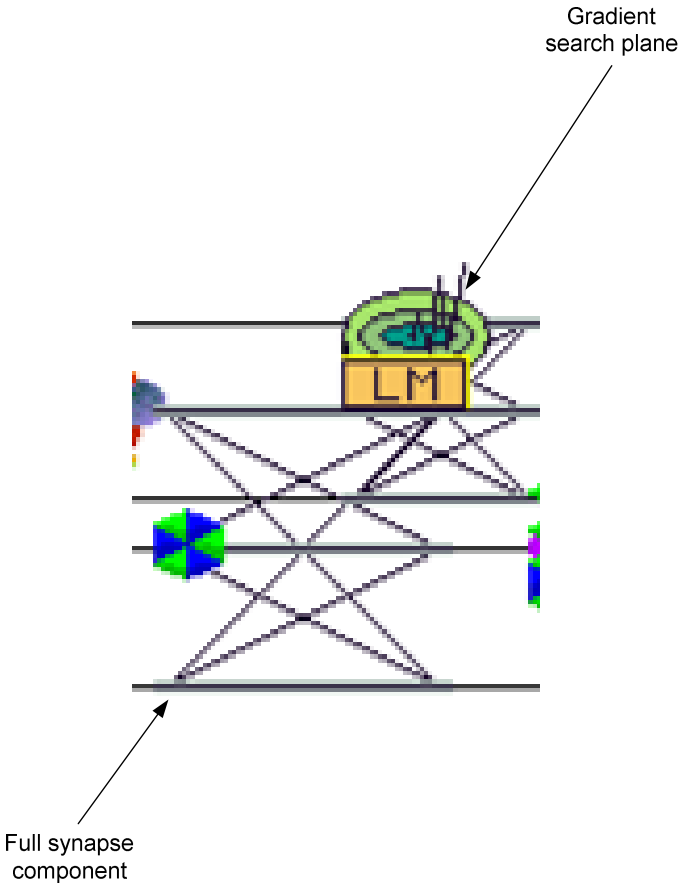

Figure 9: Full synapse and gradient search component

The smaller green components are in a class called the gradient search plane. These components use the relative gradient error information calculated by the backdrop family, to calculate the new bias and synaptic weights (formulas 8 and 9). Formulas 8 and 9 are examples of the most common method used to obtain the new weight values, the gradient descent method. Neurosolutions has different learning rules, all of which are more complex than the gradient descent method. A gradient search component is linked to each component in the activation family that has non-static weights, such as the hidden- and output layer axons and the full synapse components.

The small file icons indicate where actual data from the model is used, at the input axon and the L2 criterion error axon. The L2 criterion error axon uses information from the output of the model to determine the Mean-Square-Error.

6.2 Testing Methodology

Due to the large amount of parameters that neural networks generally have, it was decided that one parameter or feature would be changed, whilst the others would remain static. As mentioned earlier, 7 networks were identified as candidates for the data set. Each of these networks was tested using the following sequential steps. The best network in each step was then used in subsequent steps.

6.1 Transfer function test

In this step the four transfer functions (tanh, sigmoid, linear tanh and linear sigmoid) were tested for each network type. The network was trained for 100 epochs and the resulting test MSE was recorded and compared. The network with the lowest MSE was chosen for the learning rule test step.

6.2 Learning rule test

Neurosolutions uses 6 different learning rules to update the network weights. Each of these learning rules was tested, and the MSE's were compared. The network was also trained for 100 epochs.

6.3 Topology test

After the transfer function and the learning rule had been chosen for a network, the networks topology was determined. As this was the final parameter that would be changed for a particular ANN, the training **data error**, **non-training data error**, **# of runs before best generalization and the elapsed time**[3] was obtained in this phase. In this phase of testing the training data was divided as follows: 55%=training, 15% Cross validation and 30% Testing.

To obtain the training data error and # runs before best generalization the network was trained with the training data set, whilst using cross-validation to stop the network. Training was stopped when the cross-validation MSE reached 0.0001.

The network weights were then frozen and the testing data-set was used to obtain the **non-training data error.** All these results were obtained by averaging three runs.

After all the networks were tested in this manner and a comparison based on the performance indicators could be made. This comparison yielded the best network for this particular training set.

---

[3] As all the networks trained relatively quickly, the elapsed time was obtained by doing a training run of 1000 epochs and obtaining the elapsed time in seconds.

For some networks the topology test required an additional test step, whereby another parameter was tested. The best network here was then used to test the final topology parameters. These networks are the Radial basis function networks, the modular neural networks, the Jordan/Elman networks and Principal component analysis networks.

After all the network types had been tested, the best combination of parameters was obtained for each network type. These networks were then compared to each other and they were ranked for each performance indicator. Scores were allocated to each network according to each rank. The scores were as follows:

1^{st} = 7 points.
2^{nd} = 6 points.
3^{rd} = 5 points.
4^{th} = 4 points.
5^{th} = 3 points.
6^{th} = 2 points.
7^{th} = 1 point.

The score for each indicator rank was added, and the totals determined which network was best suited as the solution.

A graphical representation of the testing methodology is given in figure 7.

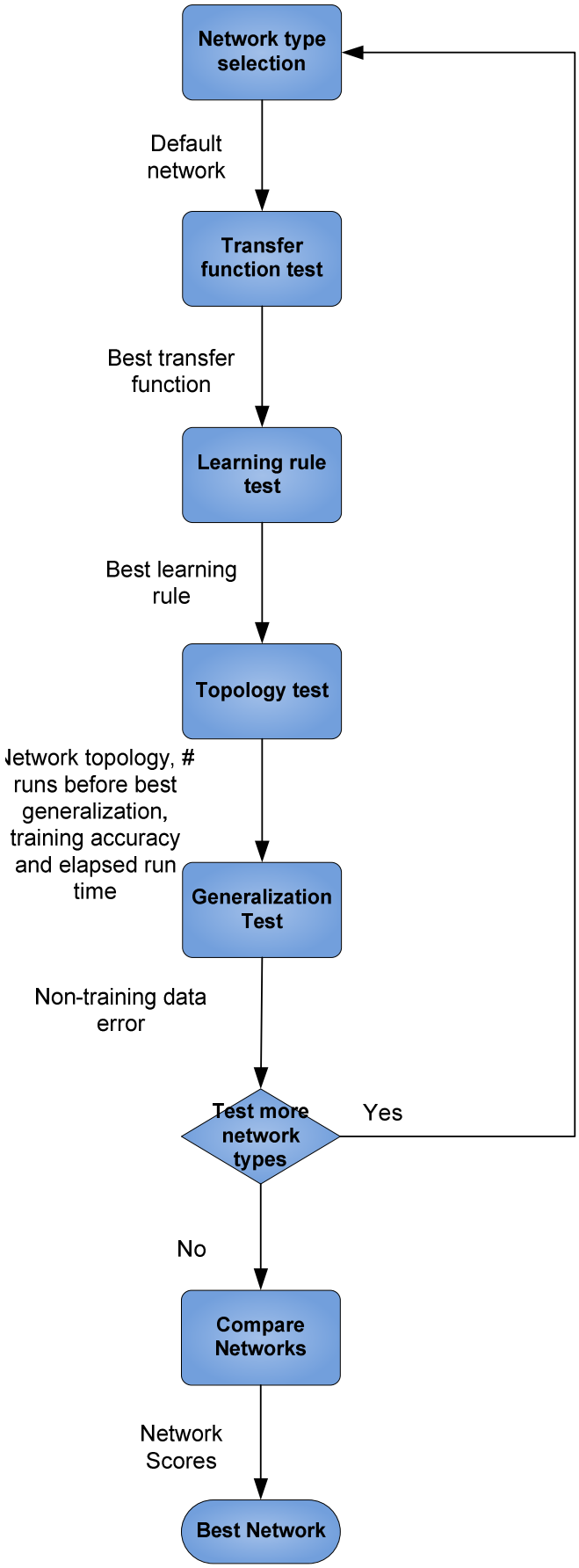

Figure 10: Testing methodology

7. Findings

In this chapter the methodology given in chapter 6 was used to test the various networks, the findings were recorded in each step of the methodology. Findings were tabulated and color was used to highlight which parameters were being tested. A different color also indicates which setting provided the best result. Each table gives the static parameters of the network being tested. The chapter concludes with the comparison of the networks, given in the final sub-heading: network summary.

7.1 Multilayer Perceptron

The default network consisted of a single layer network with 5 PE's.

7.1.1 Transfer Function

The transfer functions include the sigmoid, tanh, linear sigmoid and linear tanh function. The two linear functions are piecewise-linear approximation of their non-linear counterparts. Both of these functions are more computationally efficient than their counterparts as their map is easier to compute.

Multi-layer perceptron				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: I hidden layer, 5 PE's	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	2.18E-03	3.55E-05	2.17E-01	1.76E-02
% Error	3.83E-01	1.35E-01	6.17E+00	2.64E+00

The sigmoid transfer function provides the lowest error values, and will be used in the networks that follow.

7.1.2 Learning Rule

The learning rule refers to the way in which the weights are updated. In Section 5.6 a very basic method called the gradient descent method, is used to update the weights. Neurosolutions uses more advanced rules that are mostly based on the gradient descent method, and in the case of the MLP uses back-propagation.

Multi-layer perceptron						
Transfer function: Sigmoid		Learning rule: Variable		Topology: I hidden layer, 5 PE's		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	3.55E-05	5.40E-04	4.89E-02	5.06E-02	5.07E-02	5.09E-02
% Error	1.35E-01	6.39E-01	5.83E+00	5.93E+00	5.93E+00	5.94E+00

Only the conjugate gradient method delivers results that are comparable to that of the Levenberg Marquardt learning method. All the other methods perform similarly.

7.1.3 Topology

For MLP's a one hidden layer and two hidden layer topology was tested, with the # of PE's being the variable. For the 2 hidden layer network the same amount of PE's were used in both layers.

Multi-layer perceptron					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: I hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	2	3	3	3.5	4
# runs before best generalization	37.67	39.33	48.67	37.33	34.33
Training MSE	1.08E-04	8.79E-05	1.45E-04	4.21E-05	5.04E-05
Testing MSE	4.87E-05	5.42E-05	3.88E-05	2.39E-05	2.01E-05

Multi-layer perceptron					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layers, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	2	4	5	8	11
# runs before best generalization	34.33	39.33	41.00	49.67	43.67
Training MSE	8.52E-05	2.09E-04	4.78E-05	2.841E-04	3.05E-04
Testing MSE	2.01E-05	3.22E-05	2.57E-05	4.621E-05	1.53E-05

The MLP's provided very low run times, particularly the one layer varieties. Surprisingly most of the two layer network provided a lower training MSE than their one layer counterparts.

The 2 layer, 2 PE per layer MLP was chosen as it had the lowest elapsed time, # runs before generalization and provided good training and testing MSE. This network generalized well, was computationally very efficient, and trained quickly.

7.2 Generalized Feed-forward models

The default network here also consisted of one hidden layer with 5 PE's per layer.

7.2.1 Transfer Function

The same 4 transfer functions that were tested with MLP's are tested here.

Generalized feed-forward network				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: I hidden layer, 5 PE's	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	7.66E-04	4.63E-06	4.86E-05	7.12E-04
% Error	2.37E-01	5.41E-02	8.24E-02	6.35E-01

Once again the sigmoid function offered the best approximation properties.

7.2.2 Learning Rule

Generalized feed-forward network						
Transfer function: Sigmoid		Learning rule: Variable		Topology: I hidden layer, 5 PE's		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	4.63E-06	5.62E-04	1.93E-02	5.11E-02	4.19E-02	4.23E-02
% Error	5.41E-02	6.13E-01	4.31E+00	5.96E+00	5.4E+00	5.43E+00

The Levenberg marquardt learning rule performed the best when used with the generalized feed forward network.

7.2.3 Topology

I hidden layer-, and 2 hidden layer networks were tested with the GFF type.

Generalized feed-forward network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: I hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	2	3	3	4	4
# runs before best generalization	38.67	32.67	44.33	36.00	32
Training MSE	9.59E-05	2.05E-04	1.39E-04	1.17E-04	6.36332E-05
Testing MSE	5.57E-05	4.21E-05	3.23E-05	2.35E-05	2.78008E-05

Generalized feed-forward network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layers, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	4	6	8	12	18
# runs before best generalization	36.66667	41	44.67	37.67	36.00
Training MSE	1.41E-04	1.06E-04	1.81E-04	8.907E-05	8.64E-05
Testing MSE	2.34E-05	3.65E-05	3.42E-05	3.283E-05	2.51E-05

The single layer networks generally provided quicker convergence as their # runs before generalization was lower. All the 1 layer networks also had a lower elapsed time, due to their less complex structure.

The 1 layer 6 PE network was chosen as it provided the lowest training MSE, the lowest # runs before generalization, a low elapsed time and a low testing MSE. This network is computationally efficient, yet it converges quickly and is very accurate.

7.3 Radial Basis Function Network

The radial basis function is the first network type that includes an unsupervised learning component. 7.3.1 and 7.3.2 refers to parameters that will be used with the layers following the RBF layer. The default network only contained a single neuron at the output layer, and no hiddenlayers.

7.3.1 Transfer Function

In this phase of testing the transfer function of the last neuron will be changed.

Radial basis function				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: 0 hidden layers	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	1.35E-03	1.66E-04	6.01E-04	1.77E-02
% Error	4.26E-01	2.98E-01	2.63E-01	2.67E+00

The sigmoid function provides the best approximation.

7.3.2 Learning Rule

The learning rule concerns the last sigmoid axon as well as the RBF output weights.

Radial basis function						
Transfer function: Sigmoid		Learning rule: Variable		Topology: 0 hidden layers,		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	1.66E-04	2.06E-04	2.90E-02	4.89E-02	5.39E-02	5.57E-02
% Error	2.98E-01	3.21E-01	4.57E+00	5.9E+00	3.91E+00	6.17E+00

The LM learning rule performed the best. The learning rule tests were performed at 200 epochs. The unsupervised- and supervised training was run for 100 epochs each.

7.3.3 TopologyType

The topology of the Radial basis function network is very important, and concerns the amount of RBF's situated in the RBF layer, as well as the PE's situated at the output. The network was tested with a single output PE at the end of the RBF's first, to determine the optimal number of RBF clusters.

For this step of the tests the unsupervised learning was set to stop when 100 epochs was reached.

Radial basis function network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: variable RBF clusters, 0 hidden layers	
	5	10	15	20	40
Elapsed time (in seconds)	2	3	3	4	7
# runs before best generalization	173	145	136	130	140
Training MSE	1.84E-03	5.41E-04	3.22E-04	3.23E-04	7.55E-04
Testing MSE	1.64E-03	6.76E-04	6.45E-04	5.23E-04	1.52E-03

20 radial basis functions was chosen as the optimal arrangement. This setting provided the best accuracy whilst still retaining a relatively low elapsed time. The main disadvantage of RBF networks though is the high # runs before generalization. This is because of the unsupervised training involved in RBF's. The MSE's were also significantly higher than the previous two network types.

7.3.4 Output Topology

After the number of RBF's covering the input space had been obtained, a single hidden layer between the RBF layer and the output layer was tested.In this step of testing the training was stopped in the normal way (when the cross-validation error became less than 0.0001).

Radial basis function network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 20 RBF's,1 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	6	13	18	34	42
# runs before best generalization	781	141	131	135	146
Training MSE	6.66E-05	3.29E-03	1.16E-04	1.92E-04	1.11E-04
Testing MSE	3.41E-04	1.11E-04	1.45E-04	1.11E-04	7.64E-05

The 4 PE network ranked 1st in # runs before best generalization, and 3rd in with the three other indicators. This made it the best all round network. It provides reasonable MSE's with a medium elapsed time, and quick convergence.

7.4 Linear-Regression Models

In the linear regression model the accuracy of the output of this data set does not depend on the number of processing elements but on the learning rule. The model converges to an answer map fairly quickly, possibly due to the fact that the learning rules are based on a gradient method. The transfer function was not varied, as the linear regression model is actually a multi-layer perceptron with a linear transfer function.

Linear regressor						
Transfer function: Sigmoid		Learning rule: Variable		Topology: I hidden layer, 5 PE's		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	1.61E-03	1.61E-03	1.67E-03	7.22E-03	3.56E-03	4.86E-03
% Error	5.59E-01	5.59E-01	5.65E-01	1.07E+00	7.66E-01	8.87E-01

The initial MSE's were the highest of all the networks tested thus far. After the learning rule the number of PE's was tested for 1 and 2 hidden layers. The stopping criteria for the linear network was also changed as the validation error never reached 0.001 but reached a constant value after a few epochs. The criteria set so that the training stopped when the validation error stopped changing.

Linear regressor					
Transfer function: Linear		Learning rule: Levenberg Marquardt		Topology: I hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	2	3	3	3	4
# runs before best generalization	31	26	26	26	26
Training MSE	1.43E-03	1.43E-03	1.43E-03	1.43E-03	1.43E-03
Testing MSE	1.40E-03	1.40E-03	1.40E-03	1.40E-03	1.40E-03

Linear regressor					
Transfer function: Linear		Learning rule: Levenberg Marquardt		Topology: 2 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	3	4	6	7	11
# runs before best generalization	26	26	26	26	27
Training MSE	1.43E-03	1.43E-03	1.43E-03	1.43E-03	1.43E-03
Testing MSE	1.40E-03	1.40E-03	1.40E-03	1.40E-03	1.40E-03

All the combinations provided the exact same accuracy, so the least complex network was chosen as it has the lowest elapsed time.

The error of all of these configurations converges to a fixed value at their # runs before best generalization.

7.5 Modular neural network

The optimal transfer function and learning rule was obtained in the normal way. After obtaining the learning rule however, the testing involved two steps for the topology tests. Firstly the four modular layouts provided by Neurosolutions were tested. Finally the number of PE's in each layer was tested.

7.5.1 Transfer Function

Modular neural network				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: 2 hidden layers, 4PE's, layout 1	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	5.09E-06	2.78E-06	2.09E-04	3.61E-02
% Error	2.27E-02	4.47E-02	1.88E-01	4.57E+00

7.5.2 Learning Rule

Modular neural network						
Transfer function: Sigmoid		Learning rule: Variable		Topology: 2 hidden layers, 4PE's, layout 1		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	2.399E-06	3.54E-02	5.58E-02	5.70E-02	5.63E-02	5.73E-02
% Error	3.57E-02	3.60E+00	6.59E+00	6.65E+00	6.62E+00	6.67E+00

7.5.3 Modular Topology

Modular neural network				
Transfer function: Sigmoid	Learning rule: Levenberg Marquardt		Topology: variable modular topology, 4 PE's	
	1	2	3	4
Elapsed time (in seconds)	15	16	18	18
# runs before best generalization	31	50	42	31
Training MSE	1.76E-04	1.96E-04	7.60E-05	6.75E-05
Testing MSE	1.78E-05	1.04E-04	2.44E-05	6.42E-05

Topology 4 was selected as the best network, it had a quick convergence and the lowest MSE's.

7.5.4 PE's per hidden layer

After the best performing modular layout was chosen, the PE's in each MLP was varied.

Modular neural network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	5	10	18	30	50
# runs before best generalization	46	50	31	43	40
Training MSE	1.23E-04	9.53E-05	6.75E-05	4.50E-05	3.15E-05
Testing MSE	3.38E-05	2.53E-05	6.42E-05	2.46E-05	2.99E-05

The network with 5 PE's per MLP was chosen as it provides the best combined training and non-training errors, and was in the middle of the group as far as quick convergence was concerned.

7.6 Jordan/Elman network

Neurosolutions has 4 types of Jordan/Elman networks. The difference is in the inputs and outputs of the context units. These four types were tested first in the topological tests. Finally the number of PE's was tested. All the Jordan/Elman networks consisted of 2 hidden layers.

7.6.1 Transfer Function

The four standard transfer functions were chosen and tested.

Jordan/Elman network				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: 1 hidden layer, 6 PE's, layout 1	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	3.99E-05	5.30E-06	1.15E-03	2.88E-02
% Error	6.69E-02	5.97E-02	3.81E-01	4.06E+00

7.6.2 Learning Rule

Jordan/Elman network						
Transfer function: Sigmoid		Learning rule: Variable		Topology: 1 hidden layer, 6 PE's, layout 1		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	1.46E-05	9.52E-04	5.59E-02	5.98E-02	5.60E-02	5.79E-02
% Error	8.39E-02	7.95E-01	6.59E+00	6.80E+00	6.59E+00	6.71E+00

7.6.3 Jordan/Elman topology

Jordan/Elman network				
Transfer function: Sigmoid	Learning rule: Levenberg Marquardt		Topology: variable Jordan/Elman topology, 4 PE's	
	1	2	3	4
Elapsed time (in seconds)	7	8	8	7
# runs before best generalization	35	46	35	38
Training MSE	1.14E-05	1.32E-04	1.22E-04	1.22E-04
Testing MSE	1.48E-05	3.09E-05	3.20E-05	2.99E-05

The first configuration had the lowest and best MSE's and the quickest convergence.

7.6.4 Topology

The two layers of the Jordan/Elman network's number of PE's were varied.

Jordan/Elman network					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	3	5	7	11	18
# runs before best generalization	45	40	35	40	36
Training MSE	7.24E-05	5.39E-05	4.76E-05	5.10E-05	5.22E-05
Testing MSE	7.69E-05	2.19E-05	2.99E-05	4.56E-05	1.72E-05

The Jordan/Elman networks all had relatively low elapsed times considering the 2-hidden-layer topology. Generally more PE's resulted in faster convergence, but not necessarily a higher accuracy. The four PE network converged the fastest, provided the lowest training MSE, and third lowest testing MSE and was chosen as the overall best network.

7.7 Principal component analysis

In the final network type the topological tests consisted, once again, of two separate steps. The first step was used to determine the number of principal components in front of the MLP. The final step involved varying the number of PE's and the number of hidden layers.

7.7.1 Transfer Function

Principal component analysis				
Transfer function: Variable		Learning rule: Levenberg Marquardt	Topology: I hidden layer, 2 principal components, 6PE's	
	Tanh	Sigmoid	Linear Tanh	Linear Sigmoid
MSE	4.49E-05	9.89E-07	1.19E-02	3.51E-02
% Error	8.30E-02	2.01E-02	6.11E-01	3.83E+00

7.7.2 Learning Rule

Principal component analysis						
Transfer function: Sigmoid		Learning rule: Variable		Topology: I hidden layer, 2 principal components, 6PE's		
	Levenberg Marquardt	Conjugate Gradient	Delta Bar Delta	Quick Prop	Momentum	Step
MSE	9.89E-07	3.50E-02	5.54E-02	5.76E-02	5.74E-02	5.90E-02
% Error	2.01E-02	3.77E+00	6.56E+00	6.69E+00	6.68E+00	6.77E+00

7.7.3 # of principal components

Principal component analysis					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layer, variable principle components	
	2	3	4	5	6
Elapsed time (in seconds)	3	3	4	4	5
# runs before best generalization	132	138	148	140	129.67
Training MSE	1.57E-04	9.25E-05	6.16E-05	5.25E-05	1.94E-05
Testing MSE	2.13E-05	2.01E-05	4.19E-05	3.61E-05	1.69E-05

The 6-principle-components network provided the best # runs before best generalization and the lowest MSE's while still retaining a good elapsed time. Further tests were done using this network.

7.7.3 MLP Topology

The networks that followed the principle components, was tested in a 1 and 2 hidden-layer variety.

Principal component analysis					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 1 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	2	3	3	4	4
# runs before best generalization	141	139	137	132	137
Training MSE	7.26E-05	1.98E-04	7.54E-05	6.91E-06	1.82E-04
Testing MSE	6.71E-05	1.48E-05	1.42E-05	1.67E-05	3.89E-05

Principal component analysis					
Transfer function: Sigmoid		Learning rule: Levenberg Marquardt		Topology: 2 hidden layer, variable PE's	
	2	3	4	5	6
Elapsed time (in seconds)	4	6	8	14	21
# runs before best generalization	140	149	137	141	148
Training MSE	1.77E-04	1.71E-04	2.61E-05	1.61E-04	1.23E-04
Testing MSE	1.74E-05	3.35E-05	2.50E-05	1.86E-05	1.88E-05

Once again the 1-hidden-layer networks generally achieved a lower MSE and converged faster than the 2-hidden-layer networks. The 1-hidden-layer, 5-PE network was chosen as it provided quick convergence (considering the 100 unsupervised training runs), a relatively low elapsed time and the best MSE's.

7.8 Network Summary

In sections 7.1-7.7 each network type was tested in various configurations and a network was chosen out of each network type. These networks were chosen based on the 4 performance indicators given in section 3. In this chapter each of these 7 networks are compared using these indicators. The networks are then scored according to their rank (see

section 6.2) . The matrix below contains the performance indicators on the horizontal axis, and the network type on the vertical axis.

Network Type	Elapsed Time	Rank	# runs for best generalization	Rank	Training MSE	Rank	Testing MSE	Rank	Score
Multi layer Perceptron	2	1(tied)	34.33	3	8.52E-05	5	2.01E-05	2	20
Generalized feed-forward	4	2(tied)	32	2	6.36E-05	4	2.78E-05	5	19
Radial basis function	18	4	131	6	1.16E-04	6	1.45E-04	6	10
Linear Regression	2	1(tied)	31	1	1.43E-03	7	1.40E-03	7	16
Modular Multi-layer perceptron	30	5	43	5	4.50E-05	2	2.46E-05	3	17
Jordan/Elman network	7	3	35	4	4.76E-05	3	2.99E-05	4	18
Principal component Analysis	4	2(tied)	132	7	6.91E-06	1	1.67E-05	1	21

Given all four the relevant performance indicators the principal component analysis provided the best network. Both the multilayer perceptron and the Generalized feed-forward network came a very close second and third.

The Linear regression network and the RBF network performed the worst as far as output accuracy was concerned. The network that contained an unsupervised learning part (PCA and RBF) converged the slowest.

8. Conclusion

With 7 network types, 130 network variations, 520 runs and countless epochs, a vast and variegated amount of networks were tested. And with the exception of the Radial basis function- and linear regression network, all the network types delivered high and comparable accuracy.

The principle component analysis network provides the best training- and non-training data errors. This network has a low run time that would enable it to adapt quickly to changes in the G2 Gensym model. This particular network is the best suited of those tested, in terms of Ite-Consultlink's requirements.

Even though many networks did meet the requirements, research and testing provided the best network type, and topology. There is no doubt that the use of a neural network in the G2 Gensym environment could offer a substantially reduced run-time compared to the current model.

If a neural network is implemented in the G2 environment, the requirements for low run time, low training time, and good accuracy will be met, ultimately providing a time, and cost-saving.

Bibliography

Acosta, F. (1995). Radial basis function and related models: An overview. *Signal Processing , 45*, 37-58.

Asensio-Cuesta, S., Diego-Mas, J., & Alcaide-Marzal, J. (2010). Applying generalised feedforward neural networks to classifying industrial jobs. *International Journal of Industrial Ergonomics , I* (7).

Baldi, P. F., & Homik, K. (1995). Learning in Linear Neural Networks: A Survey. *IEEE transaction on neural networks , 6* (4).

Cybenko, G. (1996). Neural networks in computational sciences and engineering. *IEEE computational science & engineering , 96*, 36-42.

Elman, J. (1990). Finding structure in time. *Cognitive Science , 14*, 179-211.

Gensym. (2009). *Mission Critical Solutions : Gensym Web Site*. Retrieved March 23, 2010, from Gensym Web site: http://www.gensym.com/index.php?option=com_content&view=article&id=68&Itemid=71

Hornik, K., & Stinchcombe, M. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks , 2* (5), 359-366.

Hsu, Y.-Y., & Yang, C.-C. (1995). Electrical load forecasting. In A. F. Murray, *Applications of neural networks* (pp. 157-189). Kluwer Academic Publishers.

Ite Consulting. (2007). *Main: Ite Consulting*. Retrieved March 4, 2010, from Ite Consulting Web site: http://www.ite.co.za/main.html

Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist. *Proc. Eighth Annual Conf. of the Cognitive, Amherst, MA,* , 531-546.

Karayiannis, N. B., & Venetsanopoulos, A. N. (1993). *Artificial Neural Networks: Learning Algorithms,Performance Evaluation and Applications.* Kluwer Academic Publishers.

Karhunen, J. (1998). Principal Component Neural Networks - Theory and Applications- book review. *Pattern Analysis & Application* , 74-75.

Lippmann, R. (1987). An introduction to computing with neural net. *IEEE ASSP Magazine* , 318-362.

Mao, K., Tan, K., & Ser, W. (2000). Probabilistic Neural-Network Structure Determination for Pattern Classification. *IEEE TRANSACTIONS ON NEURAL NETWORKS , 11* (4), 1009-1016.

Martin, A., & Bartlett, P. L. (1999). *Neural network learning: Theoretical foundations.* Cambridge: Cambridge University Press.

Meade, A., & Zeldin, B. (1998). Establishing criteria to ensure successful feedforward artificial neural network modelling of mechanical systems. *Mathematical Computer Modelling , 27* (5), 61-74.

Murray, A. F. (1995). Neural Architectures and algorithms. In A. F. Murrey, *Applications of neural networks* (pp. 1-34). Dordrecht: Kluwer Academic Publishers.

Neurodimension. (2010). Neurosolutions 6 Help function.

Pham, D., & Karaboga, D. (1999). Training Elman and Jordan networks for system identification using. *Artificial Intelligence in Engineering* , 107-117.

Rafiq, M., Bugmann, G., & Easterbrook, D. (2001). Neural network design for engineering application. *Computers and structures , 79*, 1541-1552.

Sietsma, J., & Dow, R. (1988). Neural net pruning - why and how. *Proceedings of IEEE International Conference on Neural networks*, (pp. 325-333). San Diego.

Stinchcombe.M. (1993). Universal Approximation using feedforward networks with non-sigmoid hidden layer activation function. In H. White, *Artificial Neural Networks:Approximation and Learning Theory* (pp. 29-40). Oxford: Blackwell.

Svozil, D., Kvasnicka, V., & Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems* (39), 43-62.

Tseng, C. H., & Almogahed, B. (2009). Modular neural networks with applications to pattern profiling problems. *Neurocomputing , 72*, 2093-2100.

White, H. (1992). *Artificial Neural Networks:Approximation and Learning Theory.* Oxford: Bladkwell.

Willis, M., Di Massimo, C., Montague, G., Tham, M., & Morris, A. (1999). On neural networks in chemical process control. *Proc IEE,PtD , 138* (3), 256-266.

Willis, M., Montague, G., & Peel, C. (1995). On the application of artificial neural networks to process control. In A. F. Murray, *Applications of neural networks* (pp. 191-219). Kluwer Academic Publishers.

Zhang, J., & Morris, A. (1998). A sequantial learning approach for single hidden layer neural networks. *Neural Networks , 11* (1), 65-80.

Zhao, Z.-Q. (2009). A novel modular neural network for imbalanced classification problems. *Pattern Recognition Letters , 30*, 783-788.

Appendix A –Data set

Wc	Wm	R		Wc	Wm	R		Wc	Wm	R
0	0.667	971.038		0.067	0.333	953.434		0.4	0.5	854.09
0.133	0.267	933.035		0.133	0.8	926.266		0	0.933	967.36
0.5	0.1	832.759		0.467	0.2	840.095		0.333	0.533	871.523
0.067	0.667	949.005		0.8	0.2	763.422		0	0.467	973.816
0.1	0.4	941.82		0.533	0.2	823.553		0.867	0.067	750.839
0.2	0.6	908.615		0.267	0.333	892.403		0.2	0.133	914.308
0.467	0.333	838.715		0.1	0.2	944.432		0.7	0.1	785.819
0.267	0.667	888.522		0.1	0.8	936.638		0.3	0.6	879.94
0.267	0.733	887.749		0.2	0.7	907.404		0.2	0.733	907.001
0.267	0.067	895.533		0.2	0.3	912.267		0.733	0.267	777.025
0.2	0.4	911.046		0.133	0.467	930.485		0.467	0.533	836.654
0.267	0.467	890.846		0.067	0.867	946.367		0.6	0.133	808.288
0.1	0.5	940.519		0.467	0.467	837.34		0.5	0	833.778
0.1	0	947.059		0.067	0.733	948.124		0.133	0.133	934.743
0.333	0.667	870.038		0.3	0.5	881.079		0	0.7	970.577
0.133	0.533	929.638		0.133	0.4	931.334		0.133	0.2	933.888
0.267	0.2	893.965		0.067	0.933	945.491		0.8	0.133	763.993
0	0.8	969.196		0.2	0.8	906.197		0	1	966.445
0	0.333	975.676		0.6	0.1	808.608		0.333	0.467	872.267
0.667	0.267	791.734		0.333	0.333	873.759		0.333	0.4	873.013
0.733	0.2	777.616		0.4	0.1	858.396		0.4	0.333	855.879
0.2	0.2	913.491		0.4	0	859.48		0.6	0.067	808.928
0.9	0	744.702		0.4	0.6	853.02		0.8	0.067	764.565
0.067	0.8	947.245		0.4	0.467	854.447		0.733	0	779.396
0.533	0.067	824.883		1	0	725.329		0.933	0.067	737.598

0.467	0.4	838.027		0.4	0.067	858.757		0.133	0.6	928.793
0.6	0	809.57		0.2	0.467	910.234		0.267	0.6	889.295
0.7	0.2	784.913		0.2	0.267	912.674		0.467	0	842.173
0.2	0	915.948		0.3	0.2	884.512		0	0.3	976.142
0.4	0.133	858.036		0.533	0.133	824.217		0.2	0.5	909.829
0.7	0.3	784.009		0	0.5	973.352		0.1	0.9	935.352
0.733	0.067	778.802		0.267	0.4	891.624		0.3	0.4	882.22
0.4	0.267	856.597		0.2	0.1	914.718		0.3	0.1	885.662
0.467	0.267	839.405		0.067	0.267	954.325		0.8	0	765.138
0.3	0.7	878.805		0.933	0	738.131		0.267	0.133	894.748
0	0.067	979.418		0.267	0.267	893.183		0.267	0	896.318
0.5	0.4	829.715		0.333	0.067	876.759		0.8	0.1	764.279
0	0.867	968.277		0	0.9	967.818		0.9	0.1	743.889
0	0.733	970.116		0.067	0.133	956.111		0.067	0.6	949.888
0.2	0.067	915.127		0.5	0.5	828.705		0	0.2	977.544
0.1	0.3	943.124		0.133	0.867	925.427		0.4	0.2	857.316
0.533	0.333	822.227		0.267	0.533	890.07		0.667	0	794.196
0.4	0.533	853.733		0	0.133	978.48		0.4	0.3	856.238
0.133	0.067	935.599		0.333	0.267	874.507		0.7	0	786.727
0	0.4	974.745		0	0.267	976.609		0.867	0	751.392
0.1	0.6	939.222		0.667	0.067	793.579		0.667	0.333	791.121
0	0	980.358		0.2	0.533	909.424		0.467	0.067	841.479
0.4	0.4	855.162		0.333	0	877.513		0.067	0.067	957.007
0.6	0.4	805.738		0.133	0.667	927.949		0.533	0.467	820.905
0.3	0.3	883.365		0.533	0	825.549		0.067	0.4	952.545
0.067	0.467	951.658		0	0.533	972.888		0	0.6	971.962
0.867	0.133	750.288		0.6	0.2	807.649		0.133	0.733	927.107

0.067	0	957.905		0.2	0.333	911.86		0.133	0	936.457
0.5	0.2	831.742		0.2	0.667	907.808		0.067	0.2	955.217
0.667	0.2	792.348		0.333	0.133	876.007		0.6	0.3	806.692
0.333	0.6	870.78		0.5	0.3	830.727		0.1	0.7	937.928
0.3	0	886.816		0	0.1	978.949		0.533	0.267	822.889
0.533	0.4	821.565		0.6	0.333	806.374		0.6	0.267	807.011
0.733	0.133	778.209		0.667	0.133	792.963		0.333	0.2	875.257
0.067	0.533	950.772		0.467	0.133	840.786				
0.133	0.333	932.184		0.1	0.1	945.744				