Generalised Correlation Higher Order Neural Networks, Neural Network operation and Levenberg-Marquardt training on Field Programmable Gate Arrays

Janti Shawash

Department of Electronic and Electrical Engineering

University College London

A thesis submitted for the degree of

Doctor of Philosophy at University College London

January 12, 2012

Declaration Of Authorship

I, Janti Shawash, declare that the thesis entitled "Generalised Correlation Higher Order Neural Networks, Neural Network operation and Levenberg-Marquardt training on Field Programmable Gate Arrays" and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly in candidature for a research degree at University College London;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

Signed: .....

Date

To my father.

Acknowledgements

I would like to thank the Graduate School ORS for funding my research. I want to thank UCL and The Department of Electronic and Electrical Engineering for giving me the opportunity and a great environment to pursue my research ambitions. I would also like to thank my supervisor Dr David R. Selviah for funding my last year of research through a joint research project with the Technology Strategy Board.

During the course of my research I was motivated, advised and challenged by the individuals; mainly my supervisor Dr. David R. Selviah, Dr. F. Anibal Fernandez, my colleagues Kai Wang, Hadi Baghsiahi and Ze Chen. I would like to thank Imad Jaimoukha - Imperial College London- and Prof. Izzat Darwazeh for the talks and recommendations regarding various aspects of my research.

Most of all my thanks go to my family for motivating me to get this research degree, their enthusiasm and support made it all possible. I would like to thank Julia for her support and understanding and for making my life in London better than I would have ever expected.

Finally I would like to thank my friends Nicolas Vidal, Ioannes, Tsipouris and Miriam.

Abstract

Higher Order Neural Networks (HONNs) were introduced in the late 80's as a solution to the increasing complexity within Neural Networks (NNs). Similar to NNs HONNs excel at performing pattern recognition, classification, optimisation particularly for non-linear systems in varied applications such as communication channel equalisation, real time intelligent control, and intrusion detection.

This research introduced new HONNs called the Generalised Correlation Higher Order Neural Networks which as an extension to the ordinary first order NNs and HONNs, based on interlinked arrays of correlators with known relationships, they provide the NN with a more extensive view by introducing interactions between the data as an input to the NN model. All studies included two data sets to generalise the applicability of the findings.

The research investigated the performance of HONNs in the estimation of short term returns of two financial data sets, the FTSE 100 and NASDAQ. The new models were compared against several financial models and ordinary NNs. Two new HONNs, the Correlation HONN (C-HONN) and the Horizontal HONN (Horiz-HONN) outperformed all other models tested in terms of the Akaike Information Criterion (AIC).

The new work also investigated HONNs for camera calibration and image mapping. HONNs were compared against NNs and standard analytical methods in terms of mapping performance for three cases; 3D-to-2D mapping, a hybrid model combining HONNs with an analytical model, and 2D-to-3D inverse mapping. This study considered 2 types of data, planar data and co-planar (cube) data. To our knowledge this is the first study comparing HONNs against NNs and analytical models for camera calibration. HONNs were able to transform the reference grid onto the correct camera coordinate and vice versa, an aspect that the standard analytical model fails to perform with the type of data used. HONN 3D-to-2D mapping had calibration error lower than the parametric model by up to 24% for plane data and 43% for cube data. The hybrid model also had lower calibration error than the parametric model by 12% for plane data and 34% for cube data. However, the hybrid model did not outperform the fully non-parametric models. Using HONNs for inverse

mapping from 2D-to-3D outperformed NNs by up to 47% in the case of cube data mapping.

This thesis is also concerned with the operation and training of NNs in limited precision specifically on Field Programmable Gate Arrays (FPGAs). Our findings demonstrate the feasibility of on-line, real-time, low-latency training on limited precision electronic hardware such as Digital Signal Processors (DSPs) and FPGAs.

This thesis also investigated the effects of limited precision on the Back Propagation (BP) and Levenberg-Marquardt (LM) optimisation algorithms. Two new HONNs are compared against NNs for estimating the discrete XOR function and an optical waveguide sidewall roughness dataset in order to find the Minimum Precision for Lowest Error (MPLE) at which the training and operation are still possible. The new findings show that compared to NNs, HONNs require more precision to reach a similar performance level, and that the 2nd order LM algorithm requires at least 24 bits of precision.

The final investigation implemented and demonstrated the LM algorithm on Field Programmable Gate Arrays (FPGAs) for the first time in our knowledge. It was used to train a Neural Network, and the estimation of camera calibration parameters. The LM algorithm approximated NN to model the XOR function in only 13 iterations from zero initial conditions with a speed-up in excess of 3×10^6 compared to an implementation in software. Camera calibration was also demonstrated on FPGAs; compared to the software implementation, the FPGA implementation led to an increase in the mean squared error and standard deviation of only 17.94% and 8.04% respectively, but the FPGA increased the calibration speed by a factor of 1.41×10^6.

Contents

List of Figures

Acronyms, Abbreviations and Symbols

∂	partial derivative of a function
0.6_{10}	decimal based number representation
1.1001_2	binary, fixed-point based number representation
δw_{ij}	difference in the weight value with index ij
Δ	difference, change
$\frac{df}{dx}$	derivative of f with respect to x
λ	damping factor in the Levenberg-Marquardt algorithm
∇J	Gradient of the Jacobian
θ	vector of all parameters (weights)
$ADALINE$	Adaptive Linear Neuron Element
ANN	Artificial Neural Networks
b	bias in neural networks
d	unit root of order d
$Dim_{variable}$	Dimension of a variable Dim_{hid}
E	error vector
F	Function
H	Hessian matrix

J	Jacobian matrix
l	layer index
log	natural logarithm
$MaxIteration$	maximum iterations allowed when running the optimisation function
$MinMax$	Minimum and Maximum
MLP	Multi-Layer-Perceptrons
n	sample index
Net_{Hidden}	hidden layer output vector
Net_{input}	Network input vector
$Perf$	performance
r_t	returns at time t
SSE	sum of squared error
t	sample index at time t
W	weight matrix
X_i	input vector at index i
AccelDSP	MATLAB language-based design tool for implementing high performance Digital Signal Processing systems
ASIC	Application Specific Integrated Circuit
bit	binary digit
C++	C Plus Plus is a general-purpose programming language
CAD	Computer Aided Design
COT	Continually Online Training
CPU	Central Processing Unit
DSP	Digital Signal Processor

EDA	Electronic Design Automation
FFNN	Feed Forward Neural Network
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GTP	Power-efficient transceiver for Virtex-5 FPGAs
HONN	Higher Order Neural Network
HR	Hit Rate
ISE	world-class FPGA, DSP and Embedded Processing system design tools provided by Xilinx
MeanStdv	Mean and Standard Deviation
MSE	Mean Squared Error
NMAE	Normalised Mean Absolute Error
NMSE	Normalised Mean Squared Error
NRE	Non Recurring Engineering cost
PC	Personal Computer
PCA	Principle Component Analysis
PCI	Peripheral Component Interconnect - an industry standard bus for attaching peripherals to computers
R2	Correlation
RMSE	Root Mean Squared Error
SIC	Schwarz Information Criterion
SIMD	Single Instruction, Multiple Data
VHDL	Very-High-Speed Integrated Circuits Hardware Description Language
VLSI	Very-Large-Scale Integration
ZISC	Zero Instruction Set Chip

Chapter 1

Introduction

1.1 Motivation

Artificial intelligence enables us to solve highly complex problems. Neural Networks are a classic case in artificial intelligence where a machine is tuned to learn complex processes in an effort to mimic the operation of the human brain. Neural Networks (NNs) have a vital role in complex problems relating to artificial intelligence, pattern recognition, classification and decision making for several decades. NNs are used in applications such as; channel equalisation, intrusion detection and active filtering systems in communications, real time intelligent control and power systems. They are also used in machine vision applications such as; image processing, segmentation, registration, mapping.

1.2 Aim

This PhD thesis aims to showcase new research in the field of Neural Networks. During the course of my research I have co-authored three chapters on Neural Networks with my supervisor. The first chapter introduced and simulated a new type of Higher Order Neural Network called the Generalised Correlation Higher Order Neural Network. The research included several studies based on these new Higher Order Neural Networks (HONNs) in finance, camera calibration and image mapping.

My research interests led me to use the new HONNs to demonstrate the operation and learning of the networks in limited precision using two different learning algorithms, the error back-propagation and the Levenberg-Marquardt algorithm. Further research implemented and demonstrated the Levenberg-Marquardt algorithm on a Field Programmable Gate Array for solving the Exclusive Or (XOR) logic function approximated by a Neural Network and also parametric camera calibration.

1.3 Main contributions

The main contributions of my research are the following:

1.3.1 List of book chapters

- David R. Selviah and Janti Shawash. *Generalized Correlation Higher Order Neural Networks for Financial Time Series Prediction*, chapter 10, pages 212249. Artificial Higher Order Neural Networks for Artificial Higher Order Neural Networks for Economics and Business. IGI Global, Hershey, PA, 2008.

- Janti Shawash and David R. Selviah. *Artificial Higher Order Neural Network Training on Limited Precision Processors*, chapter 14, page 378. Information Science Publishing, Hershey, PA, 2010. ISBN 1615207112.

- David R. Selviah and Janti Shawash. *Fifty Years of Electronic Hardware Implementations of First and Higher Order Neural Networks*, chapter 12, page 269. Information Science Publishing, Hershey, PA, 2010. ISBN 1615207112.

1.3.2 List of papers submitted for peer-review

- Janti Shawash and David R. Selviah. *Higher Order Neural Networks for the estimation of Returns and Volatility of Financial Time Series.* Submitted to Neurocomputing. November 2011.

- Janti Shawash and David R. Selviah. *Generalized Correlation Higher Order Neural Networks for Camera Calibration.* Submitted to Image and Vision Computing. November 2011.

- Janti Shawash and David R. Selviah. *Real-time non-linear parameter estimation using the Levenberg-Marquardt algorithm on Field Programmable Gate Arrays.* Submitted to IEEE Transactions on Industrial Electronics. Accepted January 2012.

1.3.3 Talks and posters

- FTSE 100 Returns & Volatility Estimation; Algorithmic Trading Conference, University College London Conference Talk and Poster.

1.3.4 Papers to be submitted based upon the PhD research

Future work based on research findings to be used as material for conference and journal papers:

- The minimum lowest error precision for Levenberg-Marquardt algorithm on FPGAs.
- Run-time reconfigurable Levenberg-Marquardt algorithm on FPGAs
- Recursive Levenberg-Marquardt algorithm on FPGAs
- Signed-Regressor based Levenberg-Marquardt algorithm
- Higher Order Neural Networks for fibre optic channel electronic predistorion compensation
- Fibre optic channel electronic predistorion compensation using 2nd order learning algorithms on FPGAs
- Camera calibration operation and real-time optimisation on FPGAs
- Higher Order Neural Networks for well flow detection and characterisation
- Recurrent Higher Order Neural Network for return and volatility estimation of financial time series

1.4 Organisation of the thesis

This thesis is divided into two parts. Part I provides a review of the current state of research in two chapters. Chapter 2 provides a literature for the types of networks we investigate and use in new research. Chapter 3 provides a review of neural network operation and training on hardware field programmable gate arrays.

In Part II we showcase our new research. Chapter 4 investigates new types of Higher Order Neural Networks for predicting returns and volatility of financial time series. Chapter 5 compares the aforementioned Higher Order Neural Networks against parametric models for camera calibration and calibration performed using ordinary neural networks. Chapter 6 investigates the operation of two learning algorithms in an emulated limited precision environment as a precursor for the actual hardware implementation. Chapter 7 showcases the Levenberg-Marquardt algorithm on Field Programmable Gate Arrays used to estimate neural network and camera calibration parameters. Chapter 8 summarises all of the conclusions from the new research. Lastly, Chapter **??** provides an overview of further research opportunities based on the findings in our research.

Part I

Literature Review

Chapter 2

Neural Network Review

2.1 Development of Neural Networks

Artificial Neural Networks were first introduced by McCulloch and Pitts (1943) as a system derived to resemble neurophysiology models with a goal to emulate the biological functions of the human brain namely learning and identifying patterns. Brain functionality was modelled by combining a large number of interconnected neurons that aim to model the brain and its learning process. At first neurons were simple, they had linear functions that were combined to give us linear perceptrons with interconnections that were manually coded to represent the intended functionality.

More complex models such as the Adaptive Linear Neuron Element were introduced by Widrow and Hoff (1960). As more research was conducted, multiple layers were added to the neural network that provide a solution to problems with higher degrees of complexity, but the methodology to obtain the correct interconnection weights algorithmically was not available until Rumelhart et al. (1986) proposed the back propagation algorithm in 1986 and the Multi-Layer-Perceptrons were introduced. Neural Networks provided the ability to recognise poorly defined patterns, Hertz et al. (1989), where input data can come from a non-Gaussian distribution and noise, Lippmann (1987). NNs had the ability to reduce the influence of impulsive noise, Gandhi and Ramamurti (1997), they can tolerate heavy tailed chaotic noise, providing robust means for general problems with minimal assumptions about the errors, Masters (1993).

Neural Networks are used in wide array of disciplines extending from engineering and control problems, neurological function simulation, image processing, time series prediction and varied applications in pattern recognition; advertisements and search engines functionality and some computer software applications which take artificial intelligence into account are just a few examples. NNs also gained popularity due to the interest of

financial organisations which have been the second largest sponsors of research relating to neural network applications, Trippi et al. (1993).

2.2 Higher Order Neural Networks

One of the main features of NNs is that they learn the functionality of a system without a specific set of rules which relate network neurons to specific assignments for the rules that can be based on actual properties of the system. This feature was coupled with more demanding problems leading to an increase in complexity giving advantages as well as disadvantages. The advantages were that more complex problems could be solved. However, most researchers view that the "black-box" nature of NN training as a primary disadvantage due to the lack of understanding of the reasons that allow NNs to reach their decisions regarding the functions they are trained to model. Sometimes the data has higher order correlations requiring more complex NNs, Psaltis et al. (1988). The increased complexity in the already complex NN design process led researchers to explore new types of NN.

A neural network architecture capable of approximating higher-order functions such as polynomial equations was first proposed by Ivakhnenko (1971). In order to obtain a similar complex decision regions, ordinary NNs need to incorporate increasing number of neurons and hidden layers. There is a motivation to keep the models an as "open-box" models, where each neuron maps variables to a function through weights/coefficients without the use of hidden layers. A simple Higher Order Neural Network (HONN) could be thought of as describing elliptical curved regions as Higher Order functions (HO) can include squared terms, cubic terms, and higher orders. Giles and Maxwell (1987) were the first to publish a paper on Higher Order Neural Networks (HONNs) in 1987 and the first book on HONN was by Bengtsson (1990). Higher Order Neural Networks contain processing units that are capable of performing functions such as polynomial, multiplicative, smoothing or trigonometric functions Giles and Maxwell (1987); Selviah et al. (1991) which generate more complex decision regions which are multiply connected.

HONNs are used in pattern recognition, nonlinear simulation, classification, and prediction in computer science and engineering. Examples of using higher order correlation in the data are shown in engineering applications, where cumulants (higher order statistics) are better than simple correlation terms and are used to eliminate narrow/wide band interferences, proving to be robust and insensitive to the resolution of the signals under consideration, providing generalised improvements applicable in other domains, Ibrahim et al. (1999); Shin and Nikias (1993). It has been demonstrated that HONNs are always

faster, more accurate, and easier to explain, Bengtsson (1990). The exclusion of hidden layers allows for easier training methods to be used such as the Hebbian and Perceptron learning rules. HONNs lead to faster convergence, reduced network size and more accurate curve fitting, compared to other types of more complex NNs ,Zhang et al. (2002). In our research we attempt to continue the work already conducted by our group as presented in the following publications: Mao et al. (1992); Selviah (1994); Selviah et al. (1989, 1990).

2.3 Neural Network Structure

The HONN we consider in this research is based on first order Feed Forward Neural Networks (FFNNs) trained by supervised back propagation. This type of NN is the most common multi-layer-network in use as they are used in 80% of applications related to neural networks,Caudill (1992). It has been shown that a 3-layer NN with non-linear hidden layers and linear output can approximate any continuous function, Hecht-Nielsen (1989); White (1990). These properties and recommendations are used later in the thesis.

Figure 2.1 shows the diagram of typical neural network. The structure of the NN is described using the following notation, (Dim_{in} - Dim_{Hidden} - Dim_{out}), for example (3-4-1) expresses a NN with 3 input neurons 4 hidden neurons and one output neuron.

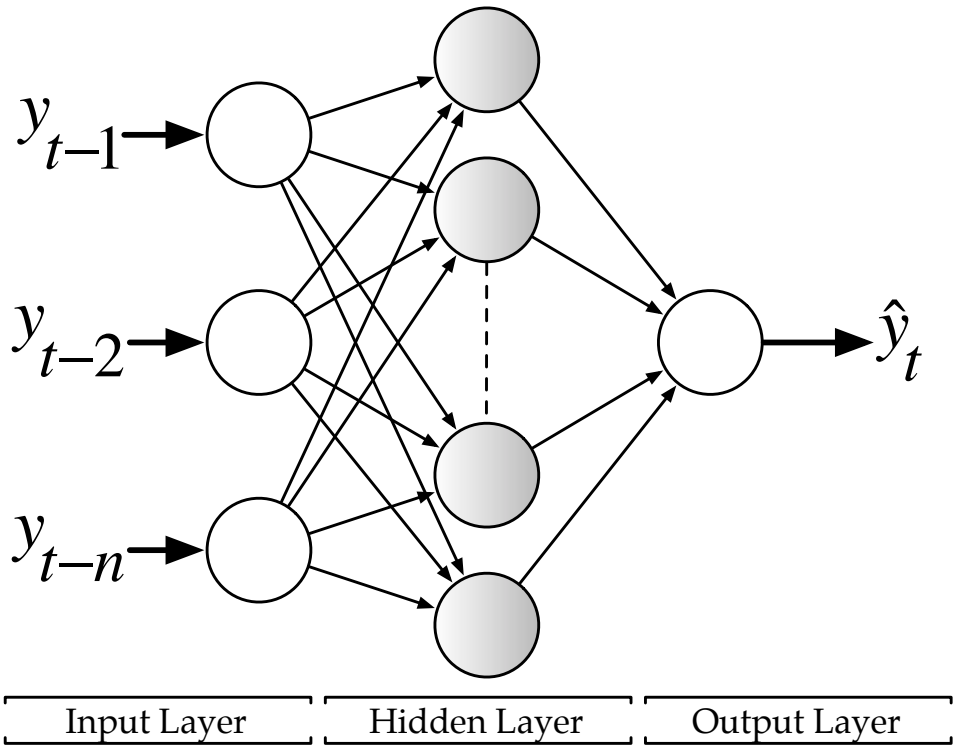

Figure 2.1: Neural Network with one hidden layer (3-4-1)

A NN is basically a system with inputs and outputs; the output dimension is determined by the dimension of the model we want to approximate. The input data length varies from one discipline to another, however; the input is usually decided by criteria suggested in literature, Fu (1994); Tahai et al. (1998); Walczak and Cerpa (1999); Zhang and Hu (1998). Successful design of NNs begins with an understanding of the problem solved, Nelson and Illingworth (1991).

The operation of the diagram in Figure 2.1 can be described in mathematical form as in (2.1), where the input of the NN comes from a sliding window of inputs taken from data samples y_t at times ranging from $t = i + 1, \ldots, n$, producing an output $\hat{y}_t$ as the latest sample by the interaction of the input data with network parameters (weights and biases) represented by $[W_{1,i}, W_{2,ii}, b_1, b_2]$.

$$\hat{y}_t = \sum_{ii=1}^{m} W_{2,ii} \times f\left(b_1 + \sum_{i=1}^{n} W_{1,i} \times y_{t-i}\right) + b_2 \tag{2.1}$$

NNs are able to take account of complex non-linearities of systems as the network's inherent properties include non-linear threshold functions in the hidden layers represented in (2.1) by f which may use the logistic or a hyperbolic tangent function as in equations (2.2), (2.3) and Figure 2.2. There are other types of non-linear functions, such as threshold and spiking functions. However, they are not relevant to the research in this thesis.

$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

If the network is to learn the average behaviour a logistic transfer function should be used while if learning involves deviations from the average, the hyperbolic tangent function works best, Klimasauskas et al. (1992). Non-linearity is incorporated by using non-linear activation functions in the hidden layer, and they must be differentiable to be able to perform higher order back-propagation optimisation; some of the most frequently used activation functions are the sigmoid, sometimes referred to as *logsig*, and hyperbolic tangent, *tansig*. Figure 2.2 shows both activation function.

The advantage of having no pre-specification models can give us the option of using training methods that use weight elimination to remove/reduce complexity in the NN as in Desai and Bharati (1998). By testing all possible combination and benchmarking their performance against information criteria taking into account the performance and the number of parameters used for estimation, finally we need to choose the internal structure of the NN. The more elements used to construct the network the more information

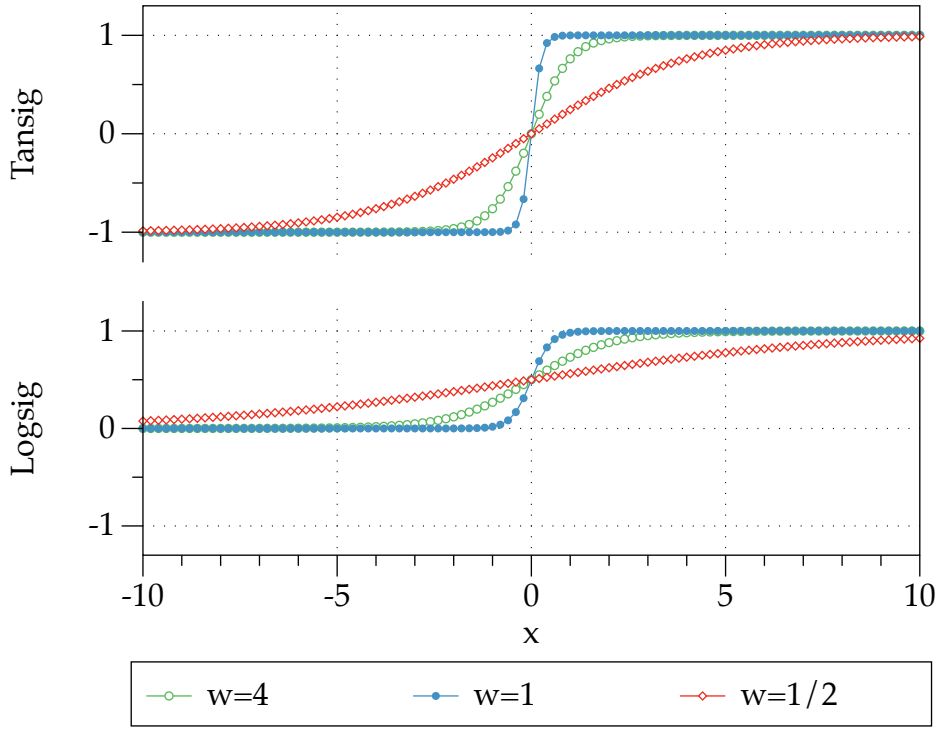

Figure 2.2: Hyperbolic Tangent and Logistic Function with varying weights

it can store about the data used to train it, this can be analogous to having a memory effect, over-fitting, that makes the network give better result for in-sample (training samples) estimations, but worse results for out-of-sample (data used for testing), this problem is minimised by ensuring we follow an information criteria that penalises increments in the number of parameters used to make a prediction. Swanson and White (1995) recommended the use information criteria increase the generalisation ability of the NN. The number of optimal hidden neurons can be found using Schwarz Information Criterion (SIC), Schwartz (1978), as suggested by Moody (1992); Moody et al. (1994). In most cases, simple parsimonious models generalise better Haykin (1999); Ioannides (2003).

The determination of the best size of the hidden layer is complex, Nabhan and Zomaya (1994). Studies showed that the a smaller size of the hidden layer leads to faster training but gives us fewer feature detectors, Dayhoff (1990). Increasing the number of hidden neurons presents a trade-off between the smoothness of the function and closeness of fit, Barnard and Wessels (1992), one major problem with the freedom we have with the hidden-layer is that it induces Over-fitting, Walczak and Cerpa (1999); where the NN stores the data already trained on in the weights linking the neurons together, degrading the generalisation ability of the network. Methods to avoid over fitting will be mentioned

in the next section.

The main principle is that the NN is required to be as simple as possible, Haykin (1999); Ioannides (2003) to provide better generalisation. As for the size of the hidden layer. Masters (1993) states the increasing the number of outputs of a NN degrade its performance and recommends that the number of hidden neurons, Dim_{hid}, - Dim for dimension- should be relative to the dimensions of the input and output of the network Dim_{in}, Dim_{out} as in (2.4).

$$Dim_{hid} = round(\sqrt{Dim_{in} \times Dim_{out}}) \tag{2.4}$$

Increasing the number of hidden nodes forms a trade-off between smoothness and closeness-of-fit, Barnard and Wessels (1992). In our studies we will examine NN with only one hidden layer as research already showed that one hidden layer NN consistently outperform a two hidden NN in most applications, Walczak (2001). Sometimes NN are stacked together in clusters to improve the results and obtain better performance similar the method presented by Pavlidis et al. (2006). Another way is to use Principle Component Analysis (PCA) or weighted network output selection to select the better performing networks from within that stack, Lai et al. (2006). Even though NNs were successfully used in financial forecasting, Zhang et al. (1998), they are hindered by the critical issue of selection an appropriate network structure, the advantage of having a non-parametric model sometimes leads to uncertainties in understanding the functions of the prediction of the networks, Qi and Zhang (2001).

All functions that compose and model NN should be verified statistically to check their feasibility, Amari et al. (1994) provides a statistical commentary on Neural Networks, were the functioning of the NN is explained and compared to similar techniques used in statistical problem modelling.

2.4 Neural Network Training

The training of neural networks aims to find a set of weights that give us a global minimum in the error function, meaning that it is the optimal performance that neural network can provide. The error surface of NNs is generally described to be complex, convex and contains concave regions, Fu (1994), it is more likely that we settle down for a local minimum than a global one. There are two methods to optimise a function, deterministic and probabilistic approaches, Lee (2007). In this study we will only use deterministic supervised learning methods as they tend to give better approximation, Lee et al. (2004),

such as back-propagation using Levenberg-Marquardt optimisation, Marquardt (1963); Press et al. (1992).

Say the signal we want to predict at time t is described by the variable y_t and the predicted signal is $\hat{y}_t$ and we try to find the set of weights that minimise the square of the error (distance) between those two values, with the error expressed by $E_t = y_t - \hat{y}_t$. Usually an energy function which is described by a single variable such as the mean square error (MSE) is used as in (2.5). Other examples of more robust error functions include the absolute error function which is less sensitive to outlier error, Lv and Yi (2005), but minimising MSE is the most widely used criterion in literature.

$$\min_{w} \frac{1}{N} \sum_{t=1}^{N} (E_t)^2 \tag{2.5}$$

In order to train and evaluate a network the data set is divided into training and test sets. Researchers presented some heuristics on the number of training samples, Klimasauskas et al. (1992) recommend having at least five training examples for each weight, while Wilson and Sharda (1994) suggests training samples is four times the number of parameters, with the data representing the population-at-large, for example the latest 10 months, Walczak and Cerpa (1999), as there is a general consensus that more weight to recent observation outperform older ones, Slim (2004).

In order to reduce network over-fitting and improve generalisation we should test randomly selected data, making the danger of a testing set characterised by one type of effect on data largely avoided, Kaastra and Boyd (1996). Another common way to reduce over-fitting is by dividing the data set into three sets, training, testing and validation data sets; we use the error from the evaluation of networks using the validation set as stopping parameter for training algorithms to determine if training should be stopped when the validation error becomes larger than the training error, this approach is called early stopping and used in most literature, Finlay et al. (2003); Haykin (1999).

Another way to avoid local minima is by using randomly selected starting points for the weights being optimised, Masters (1993), we use Nguyen-Widrow initialisation, Nguyen and Widrow (1990). Randomly selected training, validation and test sets ameliorate the danger of training on data characterised by one set of local type of market data, thus gaining a better generalisation ability to our network, Kaastra and Boyd (1996).

2.4.1 Error Back Propagation

The most famous and widely used learning algorithm is the back propagation algorithm, Rumelhart et al. (1986). Back-propagation (BP) trained NNs can approximate any con-

tinuous function in a satisfactory manner if a sufficient number of hidden neurons are used, Hornik et al. (1989). The BP algorithm is based on finding the parameter update values $w_{i,j}$ as in (2.6); the weight location in the NN is conveyed by subscripts. In (2.6) the new parameter is evaluated by using the amount of error, ∂E, that can be attributed to said parameter, ∂w_{ji}. The amount of change the new parameter exerts on the learning system is controlled by a damping factor, sometimes refereed to as learning rate, η. The subscript $_h$ is used to indicate that the learning factor can be either fixed or adaptable according to the specification of the BP algorithm used.

$$\Delta w_{ji} = -\eta_h \frac{\partial E}{\partial w_{ji}} \tag{2.6}$$

The back propagation algorithm was modified and advanced with operations that make it converge to the correct set of weights at a faster rate as in the Newton method for example. Even more advanced second-order methods converge even faster at the cost of more computational time and complexity such as the Levenberg-Marquardt (LM) algorithm, Marquardt (1963).

2.4.2 Levenberg-Marquardt Algorithm

Figure 2.3 shows a comparison of the closeness of fit performance of a sine function approximated using back-propagation versus the performance of the same function approximated using Levenberg-Marquardt algorithm.

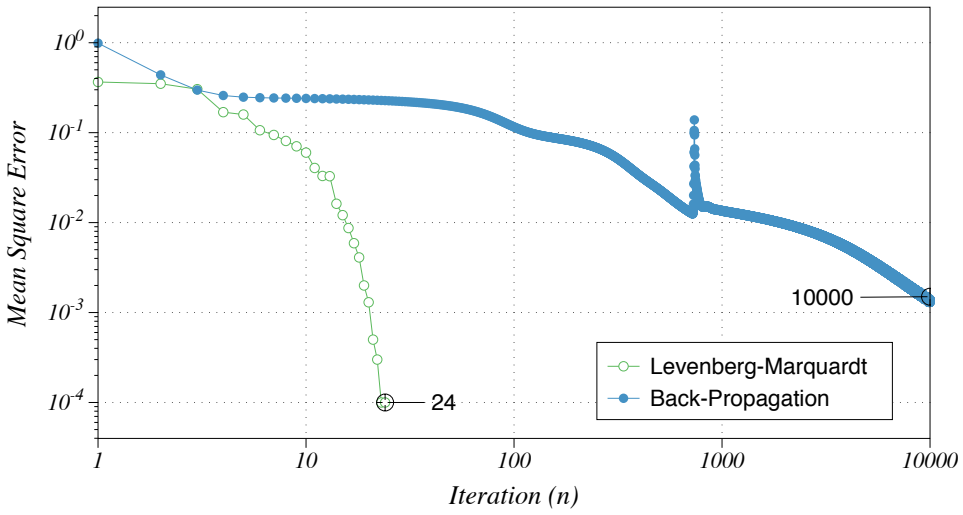

Figure 2.3: Back-Propagation versus Levenberg-Marquardt learning algorithm performance convergence

Levenberg-Marquardt reaches the optimal solution in just 24 iterations, while back-propagation continues for more than 10,000 iterations while still giving poorer results, hence we select the Levenberg-Marquardt algorithm as a more complex algorithm with which neural networks with an average number of parameters are approximated quickly and accurately. It should be noted that there are other learning techniques are not considered as they constitute a whole field of research on their own.

The Levenberg-Marquardt supervised learning algorithm is a process which finds the set of weights, W, that give us the best approximation as in (2.7). Where, J, is the gradient of error vector (Jacobian matrix), and ,$J^T J$, is the Hessian matrix of the error function, and λ is the trust region selected by the algorithm.

$$W_{new} = W_{old} - \left[J^T J + diag(J^T J) \times \lambda\right]^{-1} J \times E \tag{2.7}$$

NNs can be thought of as a non-linear least squares regression, which can be viewed as an alternative statistical approach to solving the least squares problem, White et al. (1992). Unsupervised training methods are available to train networks by partitioning input space, alleviating non-stationary processes, Pavlidis et al. (2006), but most unsupervised are less computational complex and have less capabilities in its generalisation accuracy compared to networks trained with a supervised method, Fu (1994). Back-propagation trained neural networks are superior to other networks as presented by various studies, Barnard and Wessels (1992); Benjamin et al. (1995); Walczak (1998). However, modelling problems that only have linear relationships and properties produces mixed results if modelled with NNs, Denton (1995); Zhang (2003), due to the reasons mentioned before, the added complexity and over-fitting. Nonetheless many studies have shown that the predictive accuracy is improved by using NNs, Desai and Bharati (1998); Hiemstra (1996); Kaastra and Boyd (1996); Lee et al. (1992); Qi and Maddala (1999); White (1988). Both algorithms are derived mathematical and algebraic form in A.1 and A.2.

2.5 Performance Evaluation Criteria

In order to evaluate NN performance it should be compared to other models, we must choose a criteria to compare their performance. The performance is evaluated by comparing the prediction that the NN provides as it is operated against the actual (target) value that it is expected to evaluate, similar to comparing network output and test, or train data sets. The most popular evaluation criteria include the mean square error (MSE), the normalised mean square error (NMSE), Theil's coefficient as used by Weigend et al. (1994) in the Santa Fe Time Series Competition. Other criteria include the root mean

square error (RMSE) , normalised mean absolute error (NMAE), R2 correlation coefficient, White (1988), and the directional symmetry known also as Hit Rate (HR). In camera calibration applications for example, the performance is evaluated by the sum of squared error, SSE, and the standard deviation of the model, σ, both in pixels.

2.6 Data Conditioning

After selecting the appropriate type of raw data to model with NNs, we need to process the data to eliminate some characteristics that make it difficult if not impossible to deal with. The raw data can be conditioned in a non-destructive manner without changing or disregarding vital information the data contains. Non-destructive conditioning means that we can revert to the original raw data from the transformed data.

Two popular methods for data conditioning are used in time series prediction. The first method is called minimum and maximum (MinMax) scaling where y_t is transformed to a range of $[-1, 1]$, linear scaling is still susceptible to outliers because it does not change uniformity of distribution, Kaastra and Boyd (1996). The other common type of scaling is called the mean and standard deviation scaling (MeanStdv) where y_t is changed to have a zero mean and a standard deviation equal to 1. In our studies we use the MinMax scaling to insure that the data is within the input bounds required by NNs.

Global models are well suited to problems with stationary dynamics. In the analysis of real-world systems, however, two of the key problems are non-stationarity (often in the form of switching between regimes) and over-fitting (which is particularly serious for noisy processes), Weigend et al. (1995). Non-stationarity implies that the statistical properties of the data generator vary through time. This leads to gradual changes in the dependency between the input and output variables. Noise, on the other hand, refers to the unavailability of complete information from the past behaviour of the time series to fully capture the dependency between the future and the past. Noise can be the source of over-fitting, which implies that the performance of the forecasting model will be poor when applied to new data, Cao (2003); Milidiu et al. (1999).

For example, in finance, prices are represented by p_t where, p is the price value at time $t \subset [1,\ 2,\ 3,\ ...\ ,\ n]$, , $t(1)$ is the first sample data, $t(n)$ is the latest sample, r_t is a stable representation of returns that will be used as input data as shown in (2.8).

$$r_t = 100 \times [log(y_t) - log(y_{t-1})] \tag{2.8}$$

Transforming the data logarithmically converts the multiplicative/ratio relationships in the data to add/subtract operations that simplify and improve network training, Masters (1993) , this transform makes changes more comparable, for example it makes a

change from $10 - to - 11$ similar to a change from $100 - to - 110$. The following transform operation is first differencing; that removes linear trends from the data, Kaastra and Boyd (1996), and Smith (1993) indicated that correlated variables degrade performance, which can be examined using the Pearson correlation matrix. Another way to detect integrated auto correlation in the data, is by conducting unit root tests. Say we have roots of order d , differencing d times yields a stationary series. For examples the Dicky-Fuller and Augmented-Dicky-Fuller tests that are used to examine for stationarity, Hæke and Helmenstein (1996). There are other tests that are applied when selecting input data, such as the Granger causality test for bidirectional effects between two sets of data that are believed to affect each other, some studies indicate that the effects of volatility to volume are stronger than the effects of volume on volatility, Brooks (1998). Cao et al. (2005) compared NNs uni-variate data and models with multi-variate inputs and found that we get better performance when working with a single source of data, providing further evidence to back our choice of input data selection.

2.7 Conclusions

We summarise this chapter as follows:

- NNs can approximate any type of linear and non-linear function or system.
- HONN extend the abilities of NN be moving the complexity from within the NN to an outside pre-processing function.
- The NN structure is highly dependent on the type of system being modelled.
- The number of neurons in NNs depend on the complexity of the problem on the and information criteria.
- NNs and HONNs used in a supervised learning environment can be trained using error back propagation.
- Faster and more accurate learning can be achieved by using more complex learning algorithms, such as the Levenberg-Marquardt algorithm.
- The NNs performance can be quantified by using various performance indicators which vary from field to field.
- Using NNs for modelling data requires intelligent thinking about the construction of the network and the type of data conditioning.

Due to the various decisions required to be made during the use of Higher Order Neural Networks and Neural Networks we will provide a brief review of the problem under investigation in its respective chapter.

Chapter 3

Neural Networks on Digital Hardware Review

This chapter provides a review of Neural Networks (NNs) in applications designed and implemented mainly on hardware digital circuits, presenting the rationale behind the shift from software to hardware, the design changes this shift entails, and a discussion of the benefits and constraints of moving to hardware.

3.1 Introduction

Neural Networks have a wide array of applications in hardware, ranging from telecommunication problems such as channel equalisation, intrusion detection and active filtering systems, Anguita et al. (2003); Pico et al. (2005), real time intelligent control systems that need to compensate for unknown non-linear uncertainties, Jung and Kim (2007), machine vision applications like image processing, segmentation and recognition of video streams that get data from a dynamic environment requiring operations that involve extensive low-level time consuming operations for the processing of large amounts of data in real-time; Dias et al. (2007); Gadea-Girones et al. (2003); Irick et al. (2006); Sahin et al. (2006); Soares et al. (2006); Wu et al. (2007); Yang and Paindavoine (2003). Another example is particle physics experimentation for pattern recognition and event classification providing triggers for other hardware modules using dedicated Neuromorphic NN chips that include a large-scale implementation of complex networks Won (2007), high speed decision and classification Krips et al. (2002); Miteran et al. (2003) and real-time power electronics Zhang et al. (2005b) are just a few examples of the implementations on hardware with Neural Networks that have non-linear and piecewise linear threshold functions. A further example is the use of hardware NNs in consumer electronics products has a wide

recognition in Japan, also hardware implementation is used where its operation is mission critical like in military and aerospace applications Xilinx (2008d) where the variability in software components is not tolerated, Chtourou et al. (2006).

3.2 Software versus hardware

The modern computer evolved in the past decades by the advances in digital electronics circuit designs and integration that give us powerful general purpose computational processors units (CPU). For example, Irick et al. (2006); Ortigosa et al. (2003) used NNs to discern patterns in substantially noisy data sets using hardware operating in fixed-point which achieves real-time operation with only 1% accuracy loss when compared to software implementing in floating-point. Numbers can be represented in two common ways fixed-point and floating-point, these representations will be expanded in later sections. Lopez-Garcia et al. (2005) demonstrated a 9 fold improvement with real-time operation on a compact, low power design. Maguire et al. (2007) achieved an improvement factor of 107.25 over a Matlab operation on a 2 GHz Pentium4 PC. However, the increase in performance compared to software depends on many factors. In practice, hardware designed for a specific task outperforms software implementations. Generally, software provides flexibility for experimentation without taking parallelism into account Sahin et al. (2006). Software has the disadvantage of size and portability when comparing the environment that they operate in; computer clusters or personal computers lack the power and space reduction features that a hardware design provides, Soares et al. (2006); see Table 3.1.

Table 3.1: Comparison of Computational Platforms

Platform	FPGA	ASIC	DSP	CPU	GPU
Precision	Fixed-point	Fixed-point	Fixed/Floating point	Floating point	Floating point
Area	More than ASIC	Least area	More than ASIC	Less than GPU	Larger than CPU
Embedded	Yes	Yes	Yes	Varies	No
Throughput	****	*****	***	*	**
Processing Type	Parallel	Parallel	Serial	Serial	SIMD
Power requirements	**	*	**	****	*****
Reprogrammability	Yes	No	Limited	Software	Software
Flexibility	Yes	No	No	Yes	Yes
NRE costs	Less than ASIC	Most	More than CPU	Minimal	More than CPU
Technology	New	Old	Old	Old	New
Trend	Increasing	Decreasing	Decreasing	Decreasing	Increasing

The information in this table was compiled from the references found in this chapter.

Traditionally Neural Networks have been implemented in software with computation processed on general purpose microprocessors that are based on the Von Newmann architecture which processes instructions sequentially. However, one of the NNs properties is

its inherent parallelism; which can offer significant performance increments if the designer takes this parallelism into account by designing it in hardware. Parallelism in hardware can process the forward-propagation of the NN, while simultaneously performing the back-propagation step in parallel providing a continuous on-line learning ability Girones et al. (2005).

The CPU is an example of a Very Large Scale Integration (VLSI) circuit. However, now it is possible to design VLSI circuits using Computer Aided Design (CAD) tools, especially Electronic Design Automation (EDA) tools from different vendors in the electronics industry. The tools give full control of the structure of the hardware allowing designers to create Application Specific Integrated Circuits (ASICs), making it possible to design circuits that satisfy application. However, this process is very time consuming and expensive, making it impractical for small companies, universities or individuals to design and test their circuits using these tools.

Although software has low processing throughput, it is preferred for implementing the learning procedure due to its flexibility and high degree of accuracy. However, advances in hardware technology are catching up with software implementations by including more semi-conductors, specialised Digital Signal Processing (DSP) capabilities and high precision fine grained operations, so the gap between hardware and software will be less of an issue for newer, larger, more resourceful FPGAs.

3.3 FPGA advantages and limitations

There are three main hardware platforms that are relevant to our work and a few related derivatives based on similar concepts. We begin our discussion with the most optimised and computationally power efficient design; the Application Specific Integrated Circuit (ASIC). ASICs provide full control of the design achieving optimal designs with smallest area with the most power efficient Very Large Scale Integrated circuits (VLSI) chips suitable for mass production. However, when the chip is designed it cannot be changed, any addition or alteration made on the design incurs increased design time and non-recurring engineering (NRE) costs making it an undesirable in situations where the funds and duration are limited, Zhang et al. (2005a). However, software implementations can be accelerated using other processing units; mainly the graphics processing unit; which is basically a combination of a large number of powerful Single Input Multiple Data (SIMD) processors that operate on data at a much higher rate than the ordinary CPU, also GPUs have a development rate trend that is twice as fast as the one for CPUs, Cope et al.

(2005); GPGPU (2008). But both those processing platforms do not play a major role in applications requiring high performance embedded, low power and high throughput.

The second platform to consider is the Digital Signal Processing (DSP) board in which the primary circuit has a powerful processing engine that is able to do simple mathematical arithmetic such as addition, subtraction, multiplication and division. These operations are arranged in a manner that can implement complex algorithms serially. Although DSPs are powerful enough to process data at high speed, the serial processing of data makes it a less desirable alternative compared to Field Programmable Gate Arrays (FPGAs) Soares et al. (2006); Yang and Paindavoine (2003). Hence, we propose the FPGA platform to implement our algorithms. Although FPGAs do not achieve the power, frequency and density of ASICs, they allow for easy reprogrammability, fast development times and reduced NRE, while being much faster than software implementations, Anguita et al. (2003); Gadea-Girones et al. (2003); Garrigos et al. (2007). The low NRE costs make this reconfigurable hardware the most cost effective platform for embedded systems where they are widely used. The competitive market environment will provide further reductions in price and increases in performance, Mustafah et al. (2007).

Field Programmable Gate Arrays (FPGAs), are semiconductor devices based on programmable logic components and interconnects. They are made up of many programmable blocks that perform basic functions such as logical AND and XOR operations or more complex functions such as mathematical functions. FPGAs are an attractive platform for complex processes as they contain pre-compiled cores such as multipliers, memory blocks and embedded processors. Hardware designed in FPGAs does not achieve the power, clock rate or gate density of ASICs; however, they make up for it in faster development time and reduced design effort. FPGA design comes with an extreme reduction in Non-Recurring Engineering (NRE) costs of ASICs, by reducing the engineering labour in the design of circuits. FPGA based applications can be designed, debugged, and corrected without having to go through the circuit design process. For examples, ASICs designs sometimes lead to losses amounting to millions of pounds, due to failure in the identification of design problems during manufacture and testing leading to designs that are thermally unstable which cause a meltdown in the circuit or its packaging, DigiTimes.com (2008); Tomshardware.co.uk (2008).

There are other hardware platforms available for complex signal processing, such as the wide spread CPU in personal computers and we have an active area in research in using Graphical Processing Units (GPUs) in doing scientific calculations with orders of magnitude in performance increase. But those solutions are not viable when we need an embedded processing platform with physical constraints in space and power and mission

critical processing. ASIC have greater performance compared to FPGAs, there are Digital Signal Processing (DSP) boards available used for real-time scientific computing but they do not provide the rich features that the FPGAs have to offer; most of the DSP functionality can be reproduced using FPGAs. Table 3.1 shows a comparison between the different signal processing platforms.

There are novel hardware derivatives which include a dedicated Neural Network implementation on a Zero Instruction Set Chip (ZISC) supplied by Recognetics.Inc (2008). This chip implements NNs by calculating the interaction of the system by multiplying the solution (weights) and the corresponding network structure using a multitude of highly tuned multiply-add circuits - the number of multipliers varies with chip models - but Yang and Paindavoine (2003) shows that the results it produces are not as accurate as those of the FPGAs and DSPs. Intel also produced an Electronically Trainable Artificial Neural Network (80170NB), Holler (1989), which had an input-output delay of 3 μs with a calculation rate of two billion weight multiplications per second, however, this performance was achieved at the cost of allowing errors by using reduced precision by operating at 7-bit accurate multiplication.

In the next section, we will show the architectural compromises that facilitate the implementation of Neural Networks on FPGA and how advances and development in FPGAs are closing the gap between the software and hardware accuracy.

3.4 Learning in Limited Precision

Most researchers use software for training and store the resultant weights and biases in memory blocks in the FPGA in fixed-point format Gadea et al. (2000); Soares et al. (2006); Taright and Hubin (1998); Won (2007). Empirical studies showed sudden failure in learning when precision is reduced below some critical level Holt and Hwang (1991). In general, most training done in hardware is ordinary first order back-propagation using differences in output error to update the weights incrementally through diminishing weight updates. When defining the original weights with a fixed word length as weight updates get smaller and smaller they are neglected due to having a value that is less than the defined precision leading to rounding errors and unnecessary weight updates. Babri et al. (1998) proposes a new learning method that alleviates this problem by skipping weight updates. However, this algorithm is still not as efficient as learning that is done in software with full double floating point precision, as limited precision induces small noise which can produce large fluctuations in the output.

For simple networks, it is possible to build the learning circuit alongside the feed forward NN enabling them to work simultaneously, this is called Continually On-line Training (COT) Burton and Harley (1998); Gadea-Girones et al. (2003); Petrowski et al. (1993). Other studies of more complex networks used the run-time reconfiguration ability of FPGAs to implement both feed-forward and back-propagation on the same chip, Ruan et al. (2005).

It is known that learning in low precision is not optimal, Zhu and Sutton (2003b) reports that a 16-bits fixed-point is the minimum allowable precision without diminishing a NNs capability to learn problems through ordinary back-propagation, while operation is possible in lower precision, Sahin et al. (2006). Activation functions were found to be used from a word lengths of 7-bits to 16-bits Gorgon and Wrzesinski (2006); Won (2007).

Zhu and Sutton (2003b) survey mentions that several training approaches have been implemented and that the development of an FPGA-friendly learning algorithm is still an open subject for research. So in conclusion, we train NNs using software and convert them to fixed point representations that are stored on the FPGA.

3.5 Signal Processing in Fixed-Point

Data processing initially was done on limited precision machines using binary representation. As the computer evolved, we gained more capability in representing the individual numbers in greater precision - floating point precision. The ability to deal with high precision data comes at the cost of more complex hardware design and lower processing throughput. In order to achieve the fastest possible processing we can find an adequate compromise between data representation and the processing capabilities of our hardware. Fixed-point signal is a binary representation of data with a finite number of bits (binary digits) as in Figure 3.1.

S	2^n	...	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-m}
Sign bit	Range/Magnitude							.	Fraction/resolution					

Figure 3.1: Diagram showing Fixed-point data representation

For example, we can represent the number "six" 6_{10} -subscript indicates that it is decimal based- is represented as 0110_2 where the subscript $_2$ stands for fixed-point binary format, we can add as many zeros to the left side of the number without affecting its value. Fractional representation is similar to decimal with the a radix point dividing the integer and fractional bits, where every bit represents multiples of 2^n where n is the location of

the number (bit). We can represent 2.65_{10} in fixed-point with bit width of 8 ($n = 8$) as 0010.1100_2, we notice that the number can be represented in only 4-bits as 10.11_2 forming the exact value.

Having more bit width allows for a higher range of numbers to be represented (magnitude) and/or smaller fractions (precision), depending on the position of the radix point, we have the ability to decide how to represent our signal in terms of range and precision depending on our processing needs, allowing us to design circuits to fit our exact needs giving absolute control over the data stream and processing flow. It should be noted that we should take into account the range and resolution of every signal we process, as incorrect representation leads to unexpected behaviour and functioning in our hardware. The data will adapts according to the data path structure, meaning that it will change depending on the design of our circuits, we can truncate, wrap or round the supplied number to match our design.

A decimal number 0.6_{10} is represented in 16-bit fixed-point as 0.100110011001101_2, converting the fixed-point back to floating results in the following value: 0.599969482_{10}, which is very close but not exact, we can keep increasing the number of digits to the right of the decimal points to get closer to the real value at the cost of more complex circuits.

Signed numbers are represented by assigning the left most bit as a sign indicator, 0 for positive number and 1 for negatives, we use twos complement to negate values, for example can be represented -4_{10} in 8 bits fixed point as 11111110_2, this is done by negating the value and adding 1_2 to the result of the negation. Floating point numbers are represented as in figures 3.2 and 3.3, for single and double floating point presentation.

S	exp(+127) 8 bits	.	Mantissa 23 bits
Sign bit	Exponent	.	Fraction

Figure 3.2: Single precision floating-point representation

S	exp(+1023) 11 bits	.	Mantissa 52 bits
Sign bit	Exponent	.	Fraction

Figure 3.3: Double precision floating-point representation

We benefit from fixed-point as it gives us better hardware implementation through simpler circuits that cover smaller areas with lower power consumption and costs, but it is more difficult to program application in fixed-point hardware compared to ordinary computer programs that usually take a fraction of time to develop. Fixed-point is more

suitable when we need high volume of devices with lower costs. Ordinary computers are better suited for low volume data processing where time and costs are not an issue.

3.6 Hardware Modelling and Emulation

Traditionally hardware designers and algorithm developers do not work simultaneously on a given problem; usually algorithm developers provide the hardware designers with algorithmic implementations without taking into account the difficulties in processing the data flow in finite precision which leads to discrepancies between the golden reference design (floating point) and the hardware model (fixed-point). Resolving these differences takes a significant amount of time for both developers and designers.

Field Programmable Gate Arrays contain many logic blocks and programmable interconnects that can be modified in a way to suit the application that they will be used for. One of the languages that defines the FPGA structure and configuration is called the Very-High-Speed Integrated Circuit Hardware Description Language (VHDL). In order to have a better understanding of the hardware design process and work-flow, I have attended an advanced VHDL course provided by Dulous Doulos (2008). All basic to advanced methods of logic and digital design on FPGAs were discussed, explored and tested in order to provide an understanding on how to model more complex algorithm in later stages. Attending the Advance Reconfigurable Computer System 07 Conference provided a clearer perspective on current trends in FPGA designs from research groups around the world, with a theme being about reconfigurable computing advances, manufacturers of FPGA demonstrated that there is less need to reconfigure the hardware during run-time, used to conserve and reuse circuit area at the expense of time lost due to reconfiguration. Advances in semi-conductors used to manufacture the FPGA are following Moores law Moore (1965) increasing the density and count of logic gates and interconnects by means of reduction in the hardware manufacturing process, alleviating the need to reconfigure the design at run-time.

3.7 FPGA Programming and Development Environment

Algorithm design and prototyping of networks is usually done in software using high level programming languages such as C++ , Java or Matlab. The hardware designer uses different languages and a different sets of tools to implement hardware designs. Traditionally hardware designers write VHDL programs that contain entities and architectures which

represent the building blocks of the algorithm. For small designs it is usually manageable to program all components and test them at the gate level in VHDL, but it becomes a tedious process in bigger projects; the implementation of static array multiplication can taking up to several pages of VHDL code.

With the advances in FPGAs and the ability to program them to do sophisticated algorithms, new high level languages have emerged such as Handel-C, Catapult-C and others, where we write the programs in a manner close to the C++ language. This method proved to be a real time saver by cutting down design time by at least 10 times, Maguire et al. (2007). The conversion from serial NN operation to parallel in high level language is done in a relatively short time; the same process would take a large amount of time to be done in VHDL Ortigosa et al. (2003).

Matlab is an environment that provides programs that are robust, accurate and quick to develop. It is the environment which we found the most suitable to integrate established algorithms to tools giving optimal results in the least amount of time. Xilinx (2008a,b) provides tools that enable the transfer of Matlab algorithms to hardware as bit-true and cycle-true accurate models. Ou and Prasanna (2005) used Matlab as the floating/fixed point design language and we use it to provide a testing environment for our algorithms allowing us to significantly reduce the development time and achieve rapid prototyping, by giving us the ability to examine the functionality of the algorithm as a whole instead of running time consuming simulations at the gate-level.

Matlab/Simulink designs can be automatically translated into an FPGA implementation making the design process more robust and less prone to errors. However, the design of an equivalent algorithm in VHDL might produce a more efficient design, but this comes at the cost of extensive increase in development time which sometimes makes the whole project infeasible to implement on hardware. The increased productivity achieved by switching to programming in Matlab and using Xilinx tools to obtain the Hardware models led to the development of other tools that are relevant to our project, such as the HANNA tool, Garrigos et al. (2007), that is a script providing modular templates for Neural Networks with varying sizes of layer and neurons. Ou and Prasanna (2004) designed a tool that measures the power efficiency of FPGA models by assigning power dissipation figures to the hardware resources from which the design is built, such as; the number of logic gates, memory and multipliers. However, we design our NN using generic component templates which comprise of matrix multiplication operations only.

3.8 Design Workflow

In this section we explain what steps are taken in order to make sure our software algorithm is implemented in hardware in a way that insures we do not lose the intended functionality of our designed algorithm; as explained in the previous section signals represented in hardware implementations are reduced from floating point operation to a fixed-point, where it is not possible to change the word length (bit width, bus width) of the information traversing through the FPGA during run-time; unless we include the ability to re-program the FPGA during run-time which we will discuss at a later stage. After examining the methods of implementing hardware design of algorithms in literature [VHDL, C++, Handel-C, Matlab], we concluded that we need to have the fastest and most cost effective way to transfer our algorithms into the hardware domain using tools that yield accurate results and integrated with our current algorithm development environment Matlab. Xilinx (2008c) provides the tools needed for hardware implementation and design; the tools include Xilinx ISE 10.1 design studio and Xilinx DSP tools such as SystemGenerator and AccelDSP that can be integrated to the Matlab and Simulink workflow.

Table 3.2 describes the workflow used to convert our golden reference algorithm that we have in floating point to its hardware represented counterpart that runs on the FPGA hardware. In this table, Q is the number of bits for representing the fixed-point number. Fixed-point number representation is comprised of three parts, a sign, Range bits R, and fractional bits F.

We start off with our floating point design, validate that its operational behaviour is as we intend it to be. Frequently functions we take for granted in floating point are extremely difficult to implement in hardware, as they require a very large area and design complexity leading to impractical or inefficient use of our hardware. For example, the square root and the sigmoid functions where we can replace the square root function by an absolute function value function as simplistic solution, while we can replace the sigmoid function with a look-up table of a specific resolution. We convert our code to a fixed-point and run a simulation to check that the behaviour is in line with our floating-point requirements. We explore how the trade-offs affect our algorithm by simulation and monitoring the behaviour of the changed algorithm and validate against our initial requirements to have the behaviour we require. VHDL code is obtain form AccelDSP or SystemGenerator depending on where we programmed our blocks, as they give us a bit true cycle true implementation of our the fixed point algorithm they are supplied with. At the final stage we transfer the VHDL code onto the hardware and test the feasibility of our design on real hardware, we might need to have a smaller area or some

Table 3.2: Finding Quantifiers that allow for the conversion from floating to fixed-point

1	*Parameter Range Estimation* Recording the minimum and maximum value a parameter takes during the operation and learning phases in floating-point
2	*Compute the maximum range the parameter takes* $Range = ceil(log2(Parameter) + 1)^*$
3	*Compute Fraction bits* Since $Q = R + F + 1$ Fraction length $= Q - R - 1$
4	*Construct quantifiers* Quantifiers take the form of signed fixed-point numbers with Range and Fractions as defined in the previous two steps
5	*Quantisation of the data operation* Use the quantifiers the limit to data operations to the fixed-point data type

* Ceil is function that maps a number to the an integer larger or equal to the number.

speed or latency constraints that we the automatic code did not take account of, we can go through the work-flow once more to address an issues preventing the algorithm from being implemented on hardware.

3.9 Xilinx ML506 XtremeDSP Development Board

There is a wide selection of FPGA chips available from different vendors that are suitable for different application depending on the hardware specification of the FPGA chip; for example the specification include logic cell count, operating frequency, power consumption, on-board memory, embedded microprocessors, DSP multipliers and adders. In neural networks, the main operation of neurons and interconnections performed is matrix multiplication with the weights matrix and the addition of biases followed by accumulation of signals, then performing a non-linear activation function using a look-up table, this process repeats for subsequent layers in the network. Therefore, we need to implement multiplication and addition operations; by using normal logic gate design, or by using specialised circuit blocks called DSP48E which provide design simplification and major speed up. The Xilinx ML506 XtremeDSP shown in the following figure is optimised for high-speed serial data processing with powerful DSP capabilities and system integration.

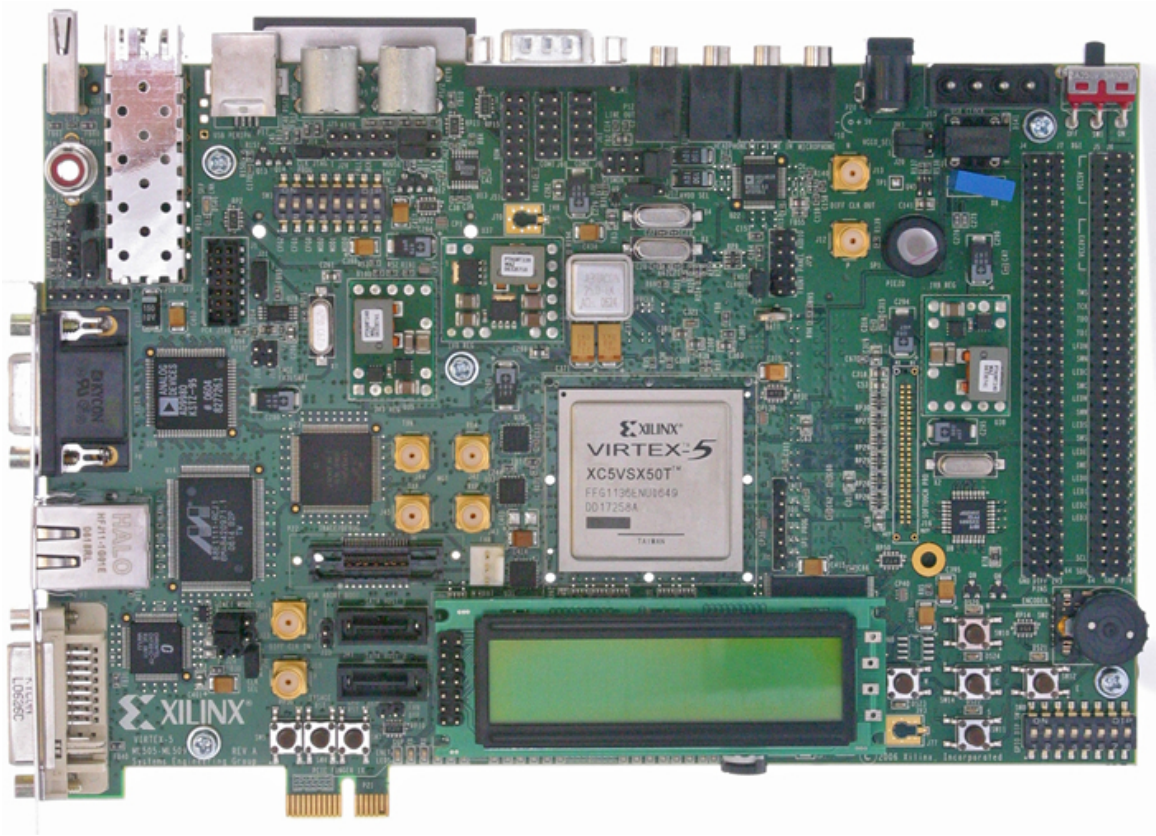

Figure 3.4: Xilinx Virtex-5 ML506 Development board

This FPGA development board is manufactured from express fabric measuring 65nm with an operating speed of 550 MHz, 8,160 logic slices, 288 (25x18 bit width) DSP48E slices, 4,752 kb Block RAM, 1 PCI Express endpoint, and 12 GTP transceivers running at 3.75 Gb/s and a total of 480 I/O connections.

A diagram of DSP48E is seen in figure 3.5, which is a special DSP MAC circuit designed by Xilinx. Multiplication operations designed using normal registers and LUTs, needs more clock cycles to do the required operation compared to the time it takes it on the DSP48E, this is achieved by using special fixed interconnection s built-in the FPGA chip during the manufacturing process, providing optimal structure at optimal power and maximum frequency operation.

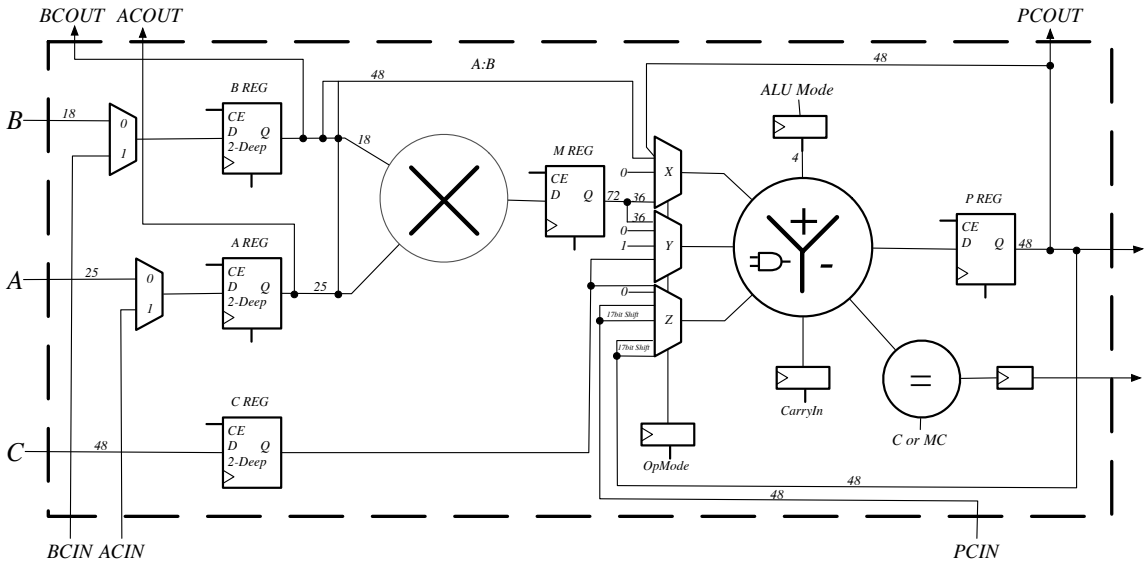

Figure 3.5: DSP48E fabric from Virtex-5 FPGA

3.10 Design Challenges

3.10.1 Design Challenges in Fixed-point

Since we have the ability to represent values in finite word lengths, we face quantisation errors of different types, depending on the way format our fixed point word length and fractional length. Table 3.3 shows the different errors due to quantisation and the way to avoid them. When programming hardware, we need to consider data ranges and take into account the quantisation artefacts and the degradation the algorithm suffers by the limited representation. We need to track all signal values to assign correct precision to

Quantisation Error			
Type	Cause	Effect	Solution
Overflow	Data larger than Operational range	Distortion	Increase word length, if not possible use directives to either saturate or wrap the number.
Underflow	Fraction smaller than least significant bit available	Granular noise	Increase fractional length, if not possible then truncate or round.

Table 3.3: Summary of Quantisation effects of data

limit the extent of error propagation throughout the operation of the design, and assure that it complies with the specifications of the prototype algorithm in full precision.

3.10.2 FPGA Design Challenges

In figures B.1, B.2 and B.3 we presented the multiplier cost when implementing FFNNs and their training algorithm showing the multiplier count against the size of the network parameters at different levels in the number of samples used for the optimisation process. We have drawn four horizontal lines indicating the number of DSP48E available in the FPGAs under consideration, currently we have the SXT50T Virtex-5 FPGA with 288 DSP48E units, we have the option to increase the number of DSP units by choosing a bigger FPGA such as the SXT240T that has 1056 DSP48E units and also it is possible to use multiple FPGAs to run our network and learning processes. Examining the figures B.1, B.2 and B.3 we notice that we reach the operational capacity of the FPGA quite quickly, as the number of parameters increases exponentially, when adding more FPGAs the increases in the DSP units is linear. So for big problems its not possible to run the optimisation processes with the limited number of DSP48Es that we have. Adding more FPGAs is not the optimal solution for this problem. We notice that the problem is compounded by the scarcity and the limited resources available to do arithmetic operations and the limited precision inherent in fixed-point.

Part II

New Research

Chapter 4

Higher Order Neural Networks for the estimation of Returns and Volatility of Financial Time Series

4.1 Introduction

Estimating the underlying processes that make up the data observed in the financial markets is decisive in making informed decisions. The Efficient Market Hypothesis proposed by Fama (1970) describes the financial data as following a random walk-type behaviour. It is well known that the logarithmic prices of high frequency financials can be estimated, Campbell (1987). The use of Neural Networks to estimate the logarithmic prices, usually referred to as returns, has been studied extensively, Deboeck (1994); Qi et al. (1996); Zhang and Hu (1998). This work aims to demonstrate the improvement in the estimation of the returns series with Higher Order Neural Networks, specifically the recently introduced Correlation Higher Order Neural Networks, Selviah and Shawash (2008). High frequency financial estimation has a residual that can be further deconstructed for volatility trends, Engle (1982). The most popular method is the Generalised AutoRegressive Conditional Heteroskedasticity (GARCH), Bollerslev (1986), a method we will use in combination with the new HONNs and compare it to the linear GARCH and non-linear EGARCH models, Brooks (1998).

The chapter is organised as follows; Section 4.2 provides the background for the research. Section 4.3 describes the procedure in which the simulations were carried out. Section 4.4 presents the simulation results and their analysis. Lastly, conclusions are presented in Section 4.5.

4.2 Returns Estimation

This work compares the ability of linear Auto Regression (AR), first order Neural Network (NN), and Higher Order Neural Networks (HONNs) in capturing the returns series dynamics and, hence, to make accurate forecasts. These models do not assume changes in volatility, the Generalised Autoregressive Conditional Heteroscedasticity (GARCH) and Exponential GARCH (EGARCH) models are used to capture the volatility present in the returns series residuals. As a general baseline validation, all of the more advanced models are compared with the Random Walk (RW) model. Each model is described in more detail below.

4.2.1 Random Walk (RW) Model

The random walk (RW) is a one step ahead forecasting method that assumes that the forecast value for tomorrow's price, $\widehat{y}_{t+1}$, is the same as today's value, y_t. The RW model is used as a benchmark to check if it is beneficial to forecast financial data using more complex models.

4.2.2 Linear Regression Model

Linear regression estimates future samples of a time series based on a weighted function of a number of previous values of the data. This method assumes that the forecast of the next point of a time series, $\widehat{y}_{t+1}$, is linearly dependent on previous data observations $y_{t-i}, i = 1, \ldots, n$, (4.1).

$$\widehat{y}_{t+1} = \beta_0 + \sum_{i=1}^{R} \beta_i \times y_{t-i} + \epsilon_t \tag{4.1}$$

In (4.1), R is the number of delayed or lagged terms with coefficients (weights), β_i, that are used in the regression of, y_{t-i}, to forecast, $\widehat{y}_t$, with error, ϵ_t. Theoretically, when the correct solution is found, after capturing all of the data dynamics, the residual error remaining in (4.1), ϵ_t, is a random variable, N, which is normally distributed with zero mean and a standard deviation of σ^2, $\epsilon_t \approx N(0, \sigma^2)$. The first parameter to consider is the input time window duration or lag length, R. A small lag length may not correctly represent the data, resulting in a residual error having non-zero autocorrelation between different data values, indicating that all of the data dynamics have been not captured, Hafer and Sheehan (1989). The optimal solution is reached when the set of weights (β_i) give a minimum difference between the forecast value and the actual value, $\epsilon_t = y_t - \widehat{y}_t$, usually represented as a single value as the Mean Square Error (MSE), (4.2), where N is

the number of elements in the residual, $\min_{\beta}(F(x))$ helps attain weights, β, which give the minimum value for the fitting function $F(x)$.

$$MSE = \min_{\beta}(\frac{1}{N}\sum_{t=1}^{N}\epsilon_t^2) \tag{4.2}$$

The RW and linear regression models are linear functions of the input data that do not take into account the nonlinear dynamics of financial time series data. Polynomial regression can be used to extrapolate the nonlinear relationships in the returns, however; we do not include a separate comparison with polynomial regression models because they are a subset of Higher Order Neural Networks that are able to capture polynomial nonlinear dynamics.

4.2.3 First Order Neural Networks Models

Neural Networks (NNs) were conceived in 1943, McCulloch and Pitts (1943), as systems derived from neurophysiological models aiming to emulate the biological operations of the brain; mainly for learning and identifying patterns. The range of NN applications increased after the back propagation training method was introduced in 1986, Rumelhart et al. (1986). NNs with hidden layers act as universal approximators, for example, a 3-layer NN, given a sufficient number of hidden neurons, can approximate any continuous function in a satisfactory manner to any degree of accuracy, Hornik et al. (1989). NNs can take a variety of different forms, depending on the number of inputs, the number of outputs, the underlying structure and the number of hidden layers. This study uses 3-layer feed forward neural networks as they represent 80% of all studies on multilayer NNs, Barnard and Wessels (1992). It has been shown that 3-layer networks consistently outperform 4-layer networks in most applications, Dayhoff and Deleo (2001); Walczak (2001). Equation (4.3) shows the operation of the NN, where $\widehat{y}_t$ is the estimated output that depends on the previous inputs $y_{t-i}, i = 1, \ldots, n$. $W_{2,l}$, $W_{1,i}$, b_1, and $b2$, are the various NN weights and biases that define how the network operates.

$$\widehat{y}_t = \sum_{l=1}^{m} W_{2,l} \times tanh(\sum_{i=1}^{m} W_{1,i} \times y_{t-i} + b_1) + b_2 \tag{4.3}$$

The hyperbolic tangent, $tanh$, (4.4), is one of the most used non-linear functions in NNs where it is incorporated into each of the hidden layers neurons, Klimasauskas et al. (1992); generally non-linear functions are preferably chosen to be differentiable.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.4}$$

As with the Linear models, Neural network training aims to reduce the discrepancy, ϵ_t, between the network output, $\widehat{y}_t$ and the desired value; details on training will be discussed later.

4.2.4 High Order Neural Network Models

As NN development progressed their structure increased in complexity. This complexity arose as a result of combining a large number of hidden layers and a large number of neurons in those layers, making the length of their training time and the explanation of their behaviour impracticable. Higher Order Neural Networks alleviate this problem by providing simpler NNs with all of the possible higher order multiplicative or functional interactions between the elements of the input vectors being provided explicitly. For example, HONNs can easily model the exclusive OR function (XOR) as they transform the input data from a linearly inseparable space to a space where the data can be classified more easily by simpler linear operators, Giles and Maxwell (1987); Lee et al. (1986). Later, recursive HONNs with feedback, known as HOFNETs, were introduced, Selviah et al. (1991). HONNs were also successfully applied to financial forecasting with a twofold improvement over ordinary NNs in some cases, Fulcher et al. (2006).

The transformations used in HONNs often dramatically speed up training as they help reduce the NN dependence on the hidden layers; at times eliminating them by using outer product or tensor models, Pao (1989). In this chapter we investigate HONNs that make use of one hidden layer along with input transformation. Figure 4.1 shows a higher order functional link transformation, together with a feed through of the untransformed input data, passing signals to the output through a single hidden layer. All of the single line links between the neurons in Figure 4.1 transmit a single value multiplied by a single weight. The thick lines in Figure 4.1 represent the outputs of the Higher Order Function, $H(X_i)$, where X_i is a vector of previous input values $y_{t-i}, n = 1, \ldots, n$. $H(X_i)$ contains a vector of output values; such as $(y_1 \times y_1, y_1 \times y_2, \ldots, y_1 \times y_n)$ for example.

$$\widehat{y}_t = \sum_{l=1}^{m} W_{2,l} \times tanh(\sum_{i=1}^{m} W_{1,i} \times \begin{bmatrix} X_i \\ H(X_i) \end{bmatrix} + b_1) + b_2 \tag{4.5}$$

The Higher Order Neural Network, HONN, (4.5), operates in a similar manner to first order NNs as described in (4.3), both equations are similar; however, the input to (4.5) includes a high order function, $H(X_i)$, that determines the type of Higher Order Neural Network being used. The full cross product function, $H(X_i)$, (4.6), is a higher order function that generates a matrix with second order multiplicative interactions, products

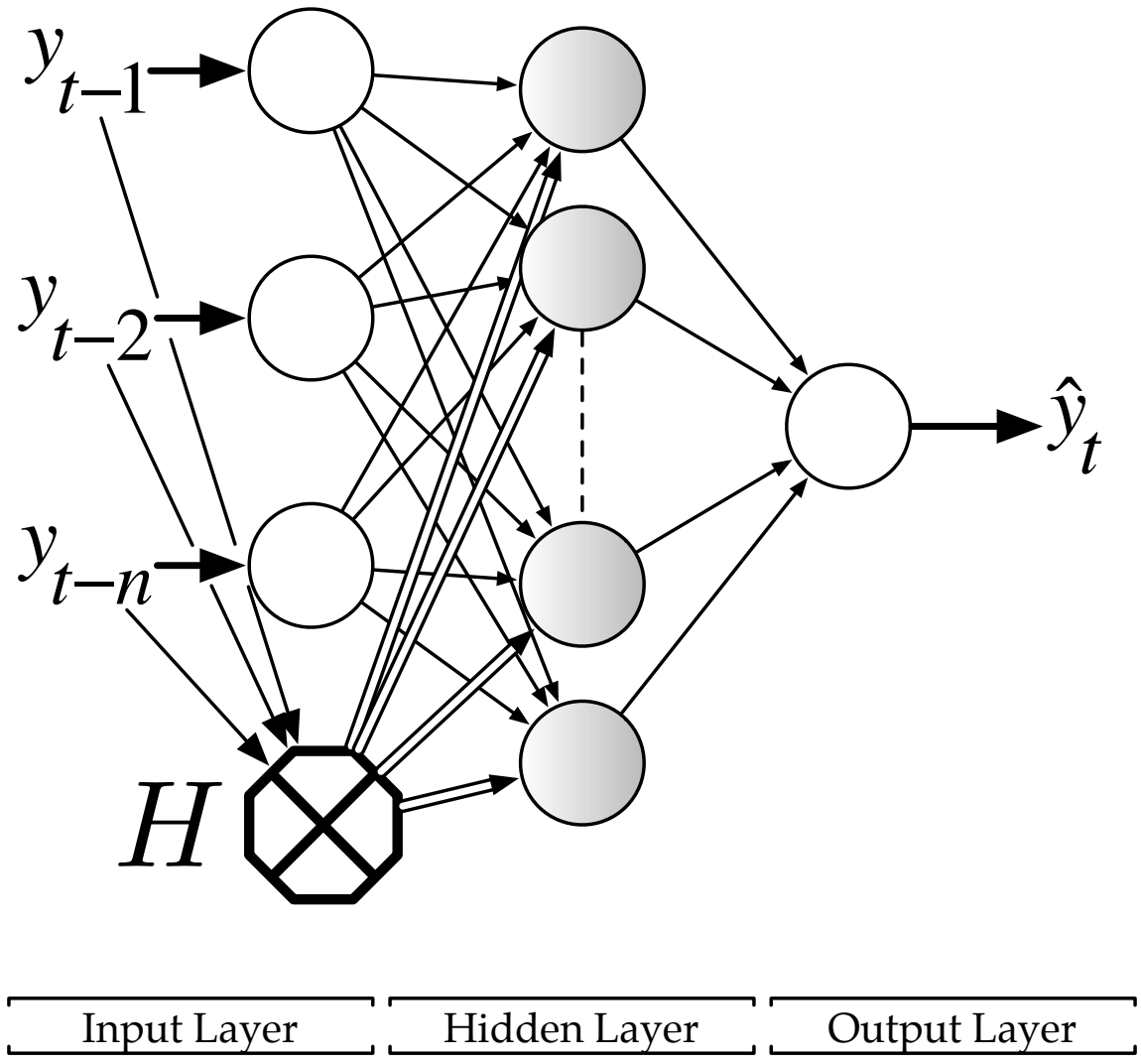

Figure 4.1: Schematic diagram of a Higher Order Neural Network structure

of pairs, of all of the network inputs. In this chapter, we examine HONNs of the types described in (4.6)-(4.9).

$$FXP - HONN = H(X_i) = X_i \otimes X_i^T \tag{4.6}$$

$$FXP - HONN = \begin{bmatrix} y_1 \times y_1 & y_1 \times y_2 & \cdots & y_1 \times y_m \\ y_2 \times y_1 & y_2 \times y_2 & \cdots & y_2 \times y_m \\ \vdots & \vdots & \ddots & \vdots \\ y_n \times y_1 & y_n \times y_2 & \cdots & y_n \times y_m \end{bmatrix}$$

$$CP - HONN = H_{i,j},\ i : 1 \to n,\ j : i \to n \tag{4.7}$$

$$C - HONN = \begin{pmatrix} \sum_{i=1,j=i}^{n} H_{i,j} & \sum_{i=2,j=i}^{n-1} H_{i,j+1} \\ \cdots & \sum_{i=n,j=i}^{1} H_{i,j+n-1} \end{pmatrix} \tag{4.8}$$

$$Horiz - HONN = \begin{pmatrix} \sum_{i=1,j=i}^{n} H_{i,j} & \sum_{i=2,j=i}^{n} H_{i,j} \\ \cdots & \sum_{i=n,j=i}^{n} H_{i,j} \end{pmatrix} \tag{4.9}$$

The first network, (4.6), contains all second order interactions between the input data values as captured in the Kronecker product matrix or covariance matrix $H(X_i)$, this network is referred to in this paper as the Full Cross Product Higher Order Neural Network (FXP-HONN). In equations (4.7)-(4.9) n represents the length of the input window, or lag length, of pervious values used to make the forecast. (4.7) shows the Cross Product Higher Order Neural Network (CP-HONN) where due to the symmetry of the Kronecker matrix the elements of the upper right triangular section are selected leading to an input vector of length $n + n^2/2$, where n is the number of data values of the input window. The recently introduced the Generalised Correlation HONN, Selviah and Shawash (2008), performs an additional localised autocorrelation operation of the input data vector providing an enhancement in performance. The recently introduced Correlation Higher Order Neural Network (C-HONN), (4.8), has a compound function which is the sum of the diagonal elements of the covariance matrix, giving the inner product terms of the autocorrelation, and the sums of the adjacent off diagonal elements of the covariance matrix, giving the outer product terms of the autocorrelation. The advantage of this compound function is that it increases the input dimension by only n elements leading to a model with fewer parameters. A further network of the class of Generalised Correlation HONNs is also examined which is referred to as the Horiz-HONN, (4.9), having a compound function of the horizontal sums of the covariance matrix values (as opposed to the diagonal sums of the C-HONN) leading to a similar size increase in the number of parameters as the C-HONN. The significance of the reduction in the number of parameters when compared to

the FXP-HONN is shown in Figure 4.2. In this figure the horizontal axis can be divided into 5 different regions (shades) with respect to each of the 5 networks. Each of the networks uses a number of input lags (top) and hidden neurons number (middle) and how these effect the number of parameters (bottom). An exponential increase in the number of model parameters is observed as the number of elements in the input vector increases for the FXP-HONN. Compared to the FXP-HONN, at a given number of hidden neurons, the CP-HONN reduces the number of higher order input parameters by half, while the number of parameters in the C-HONN and the Horiz-HONN increase linearly with respect to the number of inputs.

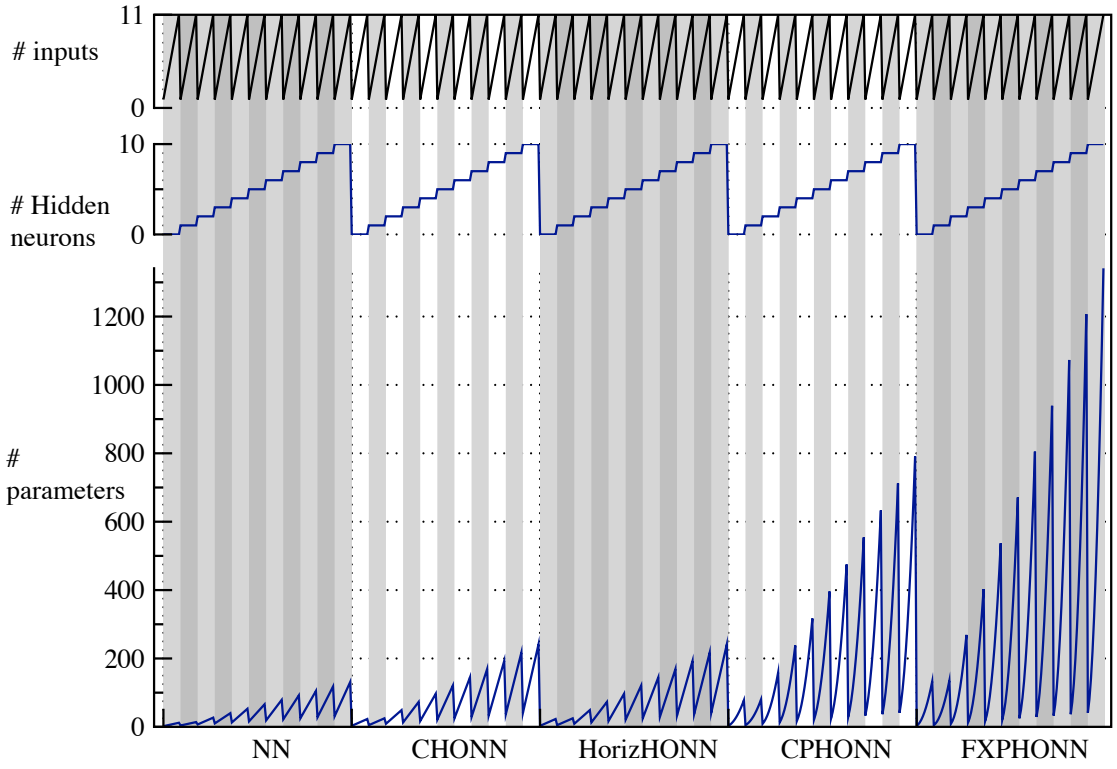

Figure 4.2: Number of model parameters as a function of the input dimension [1 to 11], the number of hidden neurons [0 to 10] and the type of Higher Order Neural Network.

Figure 4.2 shows the number of model parameters for 5 types of neural networks; a first order Neural Network (NN), a Correlation HONN (C-HONN), a Horizontal HONN (Horiz-HONN), Cross Product HONN (CP-HONN), and a Full Cross Product HONN (FXP-HONN). The number of model parameters depends on the type of NN and the number of inputs and the number of neurons in the hidden layer. The number of inputs or the input data length shown at the top of Figure 4.2 varies from 1 to 11. The number of hidden neurons varies from 0 to 10 neurons illustrated by the vertically striped pattern across

the figure and shown in the middle graph. The resulting number of model parameters for the networks increases slowly in the C-HONN and Horiz-HONN, and more significantly in the CP-HONN and FXP-HONN as a function of the number of inputs and the number of hidden layer neurons both independently and when combined. The models mentioned so far forecast returns ignoring changes in the volatility in the underlying data generating process. Models that incorporate neural networks and volatility capturing models have been studied before, Donaldson (1997); Li et al. (2005); Meissner and Kawano (2001). However, for the first time to our knowledge, this type of estimate of returns using HONN with the Generalised Autoregressive Conditional Heteroscedasticity model (GARCH) and the Exponential GARCH model (EGARCH) is performed to take into account changes in volatility with time.

4.2.5 Volatility Estimation

Linear regression, NN and HONN models are only able to model the linear or curvilinear relationships within data containing heteroskedastic error, meaning that they can only capture information with respect to the returns data without reduction of changing volatility, Fu (1994). However, the Autoregressive conditional heteroskedasticity (ARCH) model, Engle (1982) expresses the variance of the residual error as a function of the variances of previous errors, (4.10). Here, $\widehat{\sigma}_t^2$, is the variance provided by the model, that depends on q coefficients, α_i of the error, ϵ_t, and the mean volatility, α_0.

$$\widehat{\sigma}_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 \tag{4.10}$$

It is assumed that the variance, $\widehat{\sigma}_t^2$, is not constant with time and is linearly related to the random probability distribution of the signal, z_t, (4.11). This linear relationship is captured by, α_i, where, i, has a length of q. The volatility of the residual error ϵ_t, is related to the standardised residual, z_t, which comes from independent and identically distributed (i.i.d.) values with zero mean and a constant standard deviation, σ, $z_t \sim N(0, \sigma^2)$. The volatility can be extracted from the residual, ϵ_t by extracting the estimated volatility, σ_t driving the residual into a form close to z_t, and this, in effect, gives a standardised residual.

$$\epsilon_t = \sigma_t z_t \tag{4.11}$$

The Generalised AutoRegressive Conditional Heteroskedasticity (GARCH) model proposed by Bollerslev (1986) has become the standard used in economics and finance, (4.12), Figure 4.3. This model builds on the ARCH model by using a feedback loop containing

the previous volatility forecasts, $\widehat{\sigma}^2_{t-j}$, effectively performing a moving weighted average operation which is captured by the coefficients, β_j, that have p terms.

$$\widehat{\sigma}^2_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon^2_{t-i} + \sum_{i=1}^{p} \beta_j \widehat{\sigma}^2_{t-j} \tag{4.12}$$

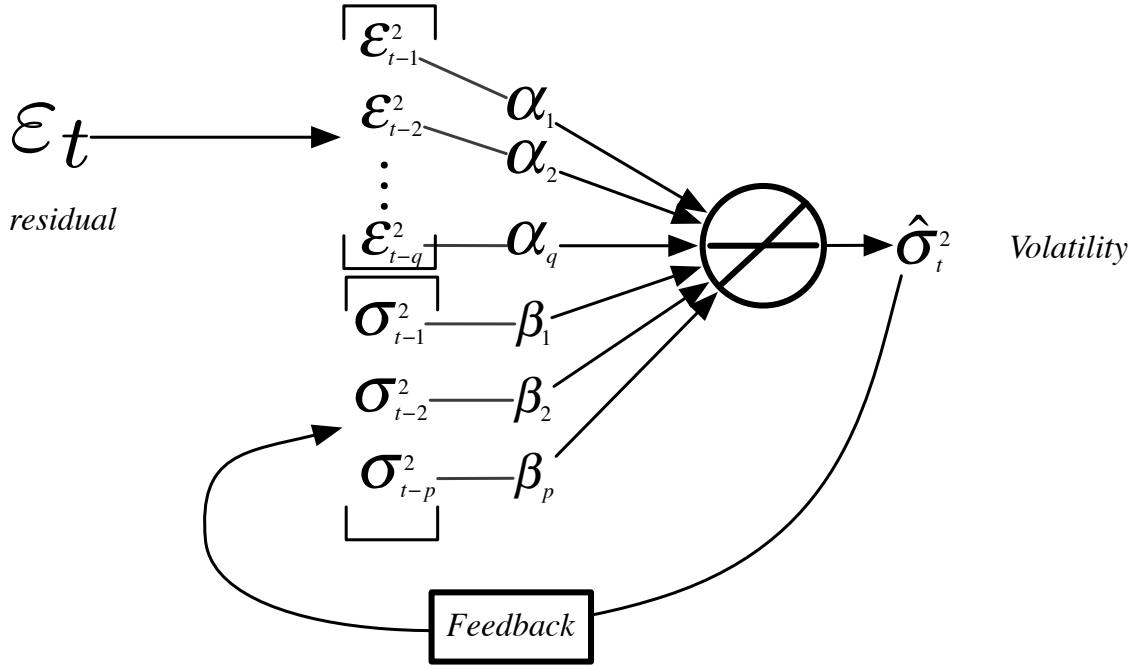

Figure 4.3: Schematic flow diagram of a GARCH model

The GARCH model incorporates the error, ϵ_t, and the error variance, σ^2_{t-j}, of previous samples to forecast the variance of error, $\widehat{\sigma}^2_t$, at the next time step. Figure 4.3 shows a schematic flow diagram of GARCH with a summation linear neuron at the output node similar to the output node of NNs. A more comprehensive explanation of the ARCH and GARCH models is found in Poon and Granger (2003). The linear GARCH(1,1) model, where p and q in (4.12) are set to 1, is accepted as being a better model than linear models alone, however; the inclusion of an additional nonlinear conditional mean specification, a nonlinear function of σ^2_{t-j}, provides further improvement, Gencay and Stengos (1998). Most literature examining nonlinear effects using GARCH adopt the Glosten Jagannathan Runkle GARCH (GJR-GARCH) and Exponential GARCH (EGARCH) models because they both have the ability to capture the asymmetry of the probability distribution function of the returns and volatility, as positive and negative returns have different effects on the volatility. Brooks (1998) demonstrated that EGARCH performs better than GJR-GARCH so this is included in the models under investigation in this

study. EGARCH, (4.13), has the following parameters; α_0 is the mean volatility, α_i are the coefficients of the previous squared errors, β_i are the coefficients of the previous forecasted volatilities. In (4.14), S^-_{t-i}, captures the asymmetry of the error with coefficients, L_i, of previous error terms.

$$log\widehat{\sigma}_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^{q} L_i S^-_{t-i} \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_j \widehat{\sigma}_{t-j}^2 \tag{4.13}$$

$$S^-_{t-i} = \begin{cases} 1 & \epsilon_{t-i} \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

4.3 Experimental methodology

There are two general types of data that can be used in models that estimate financial data; heterogeneous (technical) and homogeneous (fundamental) data, Tomek and Querin (1984). Homogenous data comes from one source of information, for example, the price series and its previous values. Heterogeneous data includes variables that are believed to have a relationship with the homogenous data such as; dividend, interest rate, money growth rate, volume and inflation rate; Deboeck (1994), this study only considers homogenous data.

To obtain results which are more generally applicable than to a single price series, two financial data sets were used: the Financial Times Stock Exchange 100 index (FTSE 100) was chosen because it represents 80% of the market capitalisation of the London Stock Exchange and the National Association of Securities Dealers Automated Quotations (NASDAQ) which represents the market capitalisation in America. Both data sets are sampled at daily intervals starting from 04/01/2000 and ending at 26/08/2009. This data is freely available on the internet at Yahoo! Finance (2009). Some studies, Huang et al. (2006b), recommend the use of mixed sample rates, for example, using a weekly average to forecast the following day price value. This study chose to use daily information because it contains a high level of information, fluctuation, and noise which the weekly and monthly data suppressed with the averaging process. We observed that the volatility of the two data sets was significantly reduced when a weekly sampling rate was considered. Weekly and monthly averaging dampens and reduces the effects of the daily volatility clustering information that shape daily market trading decisions.

Daily price series are non-stationary; this implies that the statistical property of the data generating process is time variant. This variation leads to gradual changes in the dependency between the input and output variables. Non-stationary system dynamics are not captured by first order NNs, Haykin (1999). A global solution is only applicable for

stationary data, Pavlidis et al. (2006). Most price series can be converted to a stationary returns series by rescaling the prices logarithmically and finding the difference, McNelis (2005), and this technique was used in this study. Equation (4.15) shows describes how the returns series is attained from the price series.

$$Returns_t = 100 \times (log(Price_t) - log(Price_{t-1})) \tag{4.15}$$

The logarithm transforms multiplicative or ratio relationships into additive relationships which simplify and improve network training, Kaastra and Boyd (1996). The difference removes linear trends as redundant correlations in the data reduce forecasting performance, Smith (1993). To avoid saturation at the nonlinear function in the hidden layer, the input data is rescaled to a range of [-1, 1], (4.16).

$$y_{MinMax} = \frac{(y_{max} - y_{min}) \times (x - x_{min})}{(x_{max} - x_{min})} + y_{min} \tag{4.16}$$

y_{max} and y_{min}; are the new maximum and minimum values for the rescaled data. x_{max}, x_{min}; are the maximum and minimum of the original data. y_{MinMax} is the rescaled data.

4.3.1 Neural Network Design

The estimation model structure depends on the input/output lengths. An output with dimension one alleviates the overall performance degradation observed in systems that have multiple outputs. Training focused on a single output outperforms training that has to deal with multiple output errors, Masters (1993); training focused on the major causes of output errors reduces the chances of lowering the less prominent causes of errors when compared to individual models dealing with single output error vectors. Models with input dimension of 1 (1 lag sample) are too simple and models with input lags larger than 9 samples are very rare, Huang et al. (2006a). It was shown, Hiemstra (1996), that NNs with nonlinear neuron hidden layers have better performance than linear neuron hidden layers. The number of neurons in the hidden layer is unknown initially and is complex to determine, Nabhan and Zomaya (1994). The optimum size and number of the hidden layers is highly dependent on the data set being modelled. Studies show that smaller sized hidden layers lead to faster training times as they have fewer feature detectors (weights), Dayhoff (1990); Selviah and Shawash (2008). One major problem that must be considered when designing the hidden layer is that large and multiple hidden layers cause over fitting, Walczak and Cerpa (1999), resulting in degraded generalisation ability and so higher errors. Optimal hidden layers size is selected using information criteria that will be discussed in a later section.

4.3.2 Neural Network Training

Iterative training aims to find the weights after convergence that give the global minimum of the models error surface. Error surfaces are generally complex, globally convex and contain local concave regions, Fu (1994). NNs are more likely to converge to a local minimum than a global one. There are two methods to optimise a function; deterministic and probabilistic, Lee (2007). In this study, deterministic supervised learning methods are used as they tend to achieve a better approximation of the data behaviour, Lee et al. (2004). Supervised learning is the process of finding the set of weights, a, that give the lowest mean squared error (MSE). The popular training method known as error back propagation is used together with the fast Levenberg-Marquardt optimisation technique, Marquardt (1963), (4.17); J, the Jacobian matrix containing the first derivatives of each layer with respect to the network parameters, W_{old}, error, ϵ, and Hessian matrix,$\nabla^2 J$.

$$J = \frac{\delta Layer}{\delta W_{Layer}} \tag{4.17}$$

$$W_{new} = W_{old} - [\nabla^2 J + \mu I]^{-1} J\epsilon$$

To avoid convergence to local minima a number of techniques are used. Firstly, randomly initialised weights are chosen using the Nguyen Widrow initialisation method which distributes the data equally within the neurons active region (avoiding saturation), Nguyen and Widrow (1990). Overfitting is avoided by early stopping, Haykin (1999); where the data is split into three parts, training, validation and test sets; the training set is used to optimise the weights in the network, validation is used to stop the training when the validation error does not improve or when the validation error becomes larger than the error obtained when using the training set. Repetition using different choices of random data for the training (in-sample), validation and test (out-of-sample) sets avoids the problem of training on specific data which only represents one type of local market condition resulting in better generalisation ability, Kaastra and Boyd (1996). An additional way to overcome over fitting is by removing complexity by reducing the number of elements in the NN which can be achieved by estimating and selecting the best information criterion will be discussed in Section 4.3.4.

4.3.3 Statistical analysis of the data sets

Figure 4.4 (a) shows the FTSE100 daily price series and histogram. Figure 4.4 (b) shows the FTSE100 daily returns series and its histogram. Both figures show higher order statistical information. Figure 4.4 (a) the histogram shows that the daily price distribution

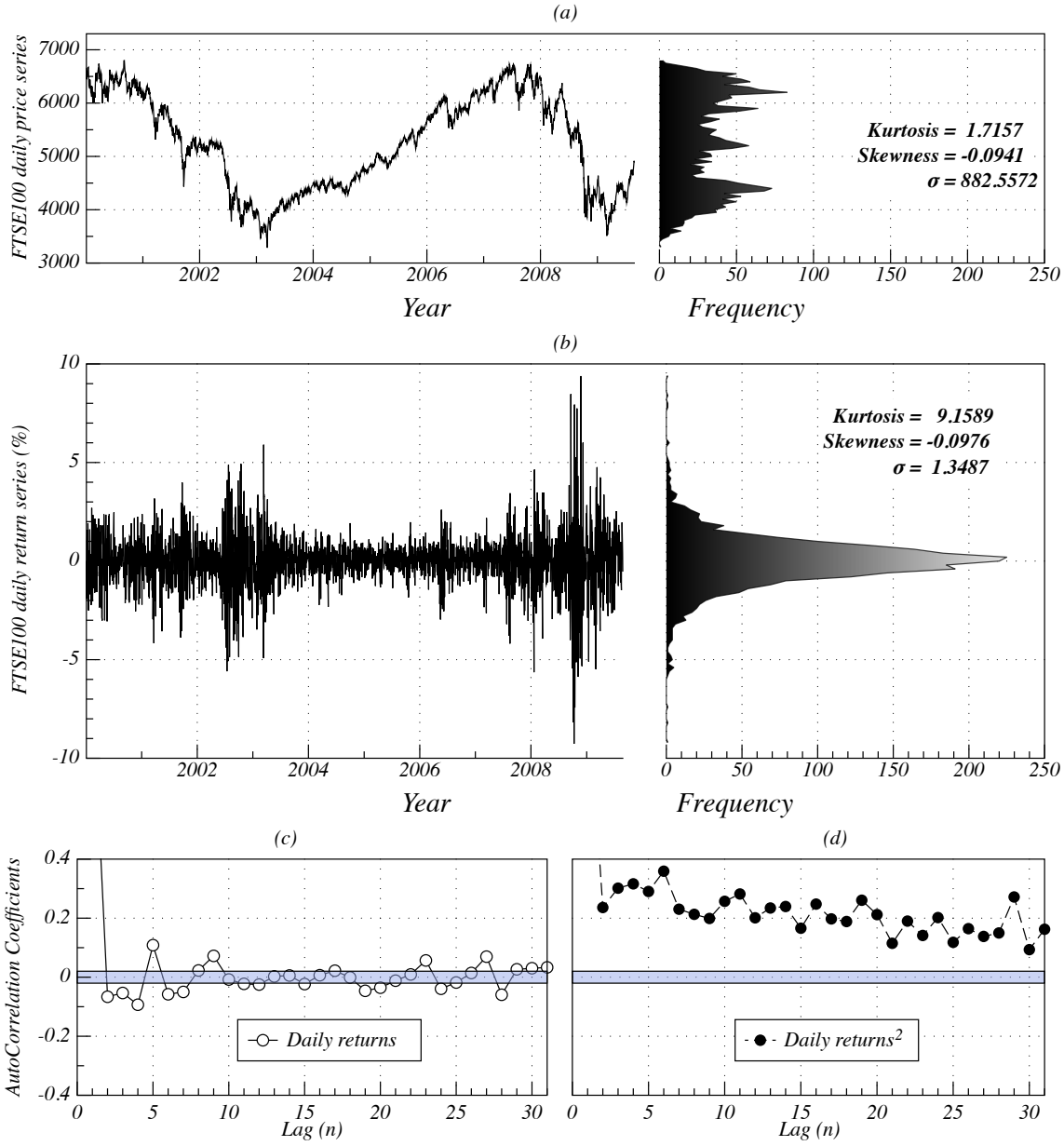

Figure 4.4: (a) FTSE 100 daily price series. (b) FTSE 100 daily returns series and daily returns histogram. Autocorrelation function of (c) daily returns and (d) daily squared returns and their 95% confidence interval.

does not resemble a Gaussian distribution with a similar mean and constant volatility, and Figure 4.4 (b) the histogram is distributed similarly to a Gaussian with a highly non normal kurtosis (Kurtosis > 3) as expected in literature, Slim (2004). The daily returns are slightly skewed to the negative side and they have a standard deviation of 1.3487. At a confidence level of 95% the autocorrelation of a random time series would lie within the shaded region along the sample axis. The position of this boundary is related to the number of samples, n, and is determined by $Bounds = \pm\frac{2}{\sqrt{n}}$, Box et al. (2008). Figure 4.4 (c) shows that the autocorrelation of the returns is almost random as it mainly lies within the 95% confidence interval. However, the squared returns in Figure 4.4 (d) have high correlation values for all values of lag, indicating a correlation in the volatility. Figure 4.5 (a) shows the daily price series and histogram of the NASDAQ time series. Figure 4.5 (b) shows the NASDAQ returns series and histogram; the statistical indicators show that the data is again highly non normal albeit with a positive skew and a standard deviation of 1.9516. Figure 4.5 (c) shows the autocorrelation of the returns. Figure 4.5 (d) shows the autocorrelation of the squared returns. There is a high correlation in the squared returns and almost random data properties in the returns series as for the FTSE100.

4.3.4 Estimation evaluation criteria

To compare the performance of the estimation models indicators must be chosen that evaluate their performance. A commonly used figure of merit in NN literature is the Root Mean Squared Error (RMSE), (4.18); the average distance between the target signal and the actual output of the model.

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - \widehat{y}_t)^2} \tag{4.18}$$

RMSE assumes that the error has a Gaussian distribution with zero mean and a constant standard deviation, σ. This is true for most data, but when dealing with complex data with changing volatility, $RMSE$ fails to take into account the changes in volatility of the data. To alleviate this problem the Log Likelihood Function (LLF) is used, (4.19); it incorporates varying volatility (standard deviation, σ_t) into the measurement of the performance, N is the length of the error vector, ϵ_t, and σ_t is the volatility at time t.

$$LLF = \frac{1}{2}\sum_{t=1}^{N}(log(2\pi\sigma_t) + \frac{\epsilon_t^2}{\sigma_t}) \tag{4.19}$$

However, model evaluation criteria should also include the number of system parameters as these affect the training time and memory capacity required and a small number give

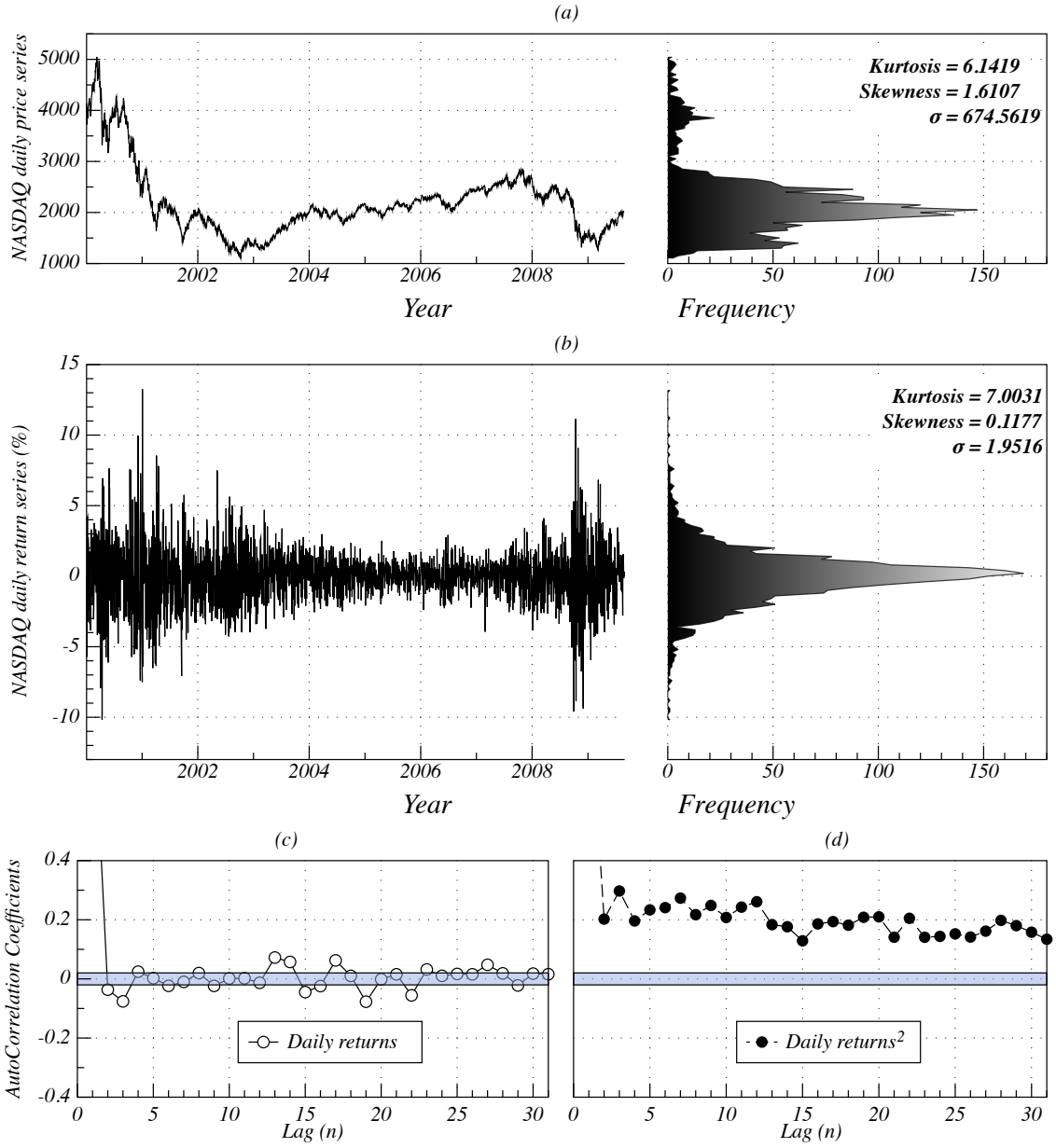

Figure 4.5: NASDAQ daily price series. (b) NASDAQ daily returns series and their histogram. Autocorrelation function of (c) daily returns and (d) daily squared returns and their 95% confidence interval.

better generalisation ability; one example that does this is the Akaike information criterion (AIC), (4.20), Akaike (1974). The AIC incurs a penalty when selecting a higher number of weights; preferring models with the least number of parameters, k, and the least error.

$$AIC(k) = \frac{LLF}{N} + \frac{k}{N} \tag{4.20}$$

The best input dimension and NN structure should be selected using the AIC criterion as simple models provide better generalisation ability, Haykin (1999); Ioannides (2003). The LLF need not be used as a separate evaluation criteria since it is a function of $RMSE$ and is incorporated in the AIC. So in this chapter the AIC is used as the main evaluation criterion but the $RMSE$ is also calculated and tabulated. The evaluation criteria mentioned so far are all based on the difference between the output and the real signal. In financial signals the direction of the signal forecast is also important and this is encompassed in the Hit Rate (HR).

$$\begin{aligned} HR &= \frac{1}{N}\sum_{t=1}^{N} D_t \text{where} \\ D_t &= \begin{cases} 1 & (\widehat{y}_t - y_{t-1}) \times (y_t - y_{t-1}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.21}$$

A good value for the correct direction detection is around 60%, Walczak (1998). However, even if 85% accuracy is achieved in direction forecast, it may still be unprofitable since the most profitable deviations are the largest, hardest and least probable to detect and most models are based on normal (Gaussian) distributions while the data is often non-Gaussian.

Even though the $RMSE$ value is used to reflect the performance of the NNs, during training the performance criteria to be minimised is usually represented by the Mean Square Error (MSE), (4.2). In this chapter we also examine the mean absolute error (MAE) training performance criterion, (4.22), as it has been described to be more robust indicator of performance when compared to the MSE, Bodyanskiy and Popov (2006).

$$MAE = \frac{1}{N}\sum_{t=1}^{N} |y_t - \widehat{y}_t| \tag{4.22}$$

To indicate how closely the residuals are to a normally distributed signal we use the χ^2 variance test parameter indicated by the parameter VAR.

4.3.5 Simulations

Five different neural networks were simulated using programs written in the Matlab 2009a language environment. Each neural network had input lengths of 1 to 11 lags. The hidden

layer varied from having no hidden layer to a hidden layer having 10 neurons. Each network was simulated 120 times with random starting points (weight distributions) and random data divisions. All of the forecast returns simulations took into account two error functions; the Mean Square Error (MSE), and the Mean Absolute Error (MAE). Two financial data sets were used so the total number of simulations was $11 \times 11 \times 120 \times 2 = 29,040$ simulations. After training, the best models, having minimum AIC for returns forecasting, were selected and their residual, ϵ_t, was used to estimate the volatility. When estimating volatility using GARCH and EGARCH, they had the following commonly used initial starting points; $\alpha_0 = 0$, $\alpha_1 = 0.9$, $\beta_1 = 0.01$, $L_1 = 0$, GARCH/EGARCH parameters were estimated using Maximum Likelihood Estimation (MLE) algorithms in Matlab.

4.4 Results and Analysis

4.4.1 Returns Simulation

In Figure 4.6 and Figure 4.7 the 9 graphs show the median of 120 simulations with MSE and MAE as error functions, random initial weights, and random data divisions. The median was used to indicate the average of the figures of merit as it provided a more robust average indicator of performance, since the median is less prone to outlier results than the mean. Each point on the diagram corresponds to a specific network with lags and hidden neurons similar to the ones in Figure 4.2. Figure 4.6 and Figure 4.7 show the AIC, $RMSE$ and Hit Rate for both in-sample and out-of-sample data labeled as $RMSE_i$, $RMSE_o$, HR_i and HR_o. The two lowest graphs show the number of training epochs and the training time for convergence to the optimal weights for the FTSE100 and NASDAQ respectively. The simulation results show the following trends; the best average AIC is achieved by the new networks C-HONNs and Horiz-HONNs. The $RMSE_i$ performance improves with increasing numbers of parameters, i.e. more lags and hidden neurons; of all of the HONNs the CP-HONN had the best average $RMSE_i$ performance for most network structures. However, a better performance in the in-sample did not lead to a better performance in the out-of-sample data. The out-of-sample $RMSE_o$ graph shows that the CP-HONN and FXHONN are the worst. The best out-of-sample $RMSE_o$ is obtained for low lags with a low number of neurons confirming that the best performance does not require a NN with many parameters. The in-sample Hit Rate had a trend similar to that in the $RMSE_i$, better performance with increasing parameters. Similarly, the out-of-sample Hit Rate had better performance in the NN and in the new

networks C-HONN, and Horiz-HONN; particularly when the hidden layer neuron count was 0, where the network reduces to a Linear Regression Network. The number of training epochs required did not correlate with the network structure or type, except when the NN reduces to a linear regression network when it converged to a solution in 2 epochs while for the rest of the networks the convergence occurred in 10 epochs. The time taken for training had a direct correlation with the number of weights. The only noticeable difference between MAE and MSE performance is in the Hit Rate and Epochs, where the MAE required a lower number of training epochs while giving better HR performance for small networks. Generally; the best model has the lowest AIC, $RMSE$, number of training epochs, training time, and the highest Hit Rate.

Figure 4.8 shows the results of the FTSE100 returns estimation using the new C-HONN; the C-HONN was chosen for this example because it had the best returns estimation AIC value. Figure 4.8 (a) shows the residual of the C-HONN estimation of the FTSE100 daily returns series. Figure 4.8 (b) shows the residual squared. Figure 4.8 (c) shows the autocorrelation coefficients of the residual. The squared residuals are shown in Figure 4.8 (d). Comparing Figure 4.8 (c-d) to Figure 4.4 (c-d) indicates that the returns estimation reduced the autocorrelation observed in the residuals, by moving the coefficients closer to zero or within the 95% confidence level. But the returns estimation had no effect in reducing the volatility clustering that is reflected in the non-negligible $Residuals^2$ autocorrelation coefficients. So the C-HONN captures the dynamics of the daily returns series resulting in reduced autocorrelation in the residuals; however, C-HONN had no effect on the squared returns autocorrelation coefficients; this can be observed in the bottom two graphs in Figure 4.8 and Figure 4.6.

Figure 4.8 (c) shows that the autocorrelations in the residuals are reduced and have autocorrelation values similar to a random variable and graph Figure 4.8 (d) shows the autocorrelation of the squared returns indicate a clustering in the volatility of the residuals which remained intact. The autocorrelation of the squared returns (volatility) is not captured by linear or NN models, hence, we use GARCH and EGARCH methods which estimate volatility that was unaccounted for when estimating the returns only.

Tables 4.1 and 4.2 show a summary of the FTSE100 and NASDAQ estimation models with the best results shown in bold font for each of the evaluation criteria. The tables are sorted according to the AIC criterion, with the best model at the top. Models estimating returns and volatility have significantly better AIC and $RMSE$ compared to the performance of returns only estimation models. The structure of the neural networks with the best AIC is provided in the column labeled (structure), where it indicates the

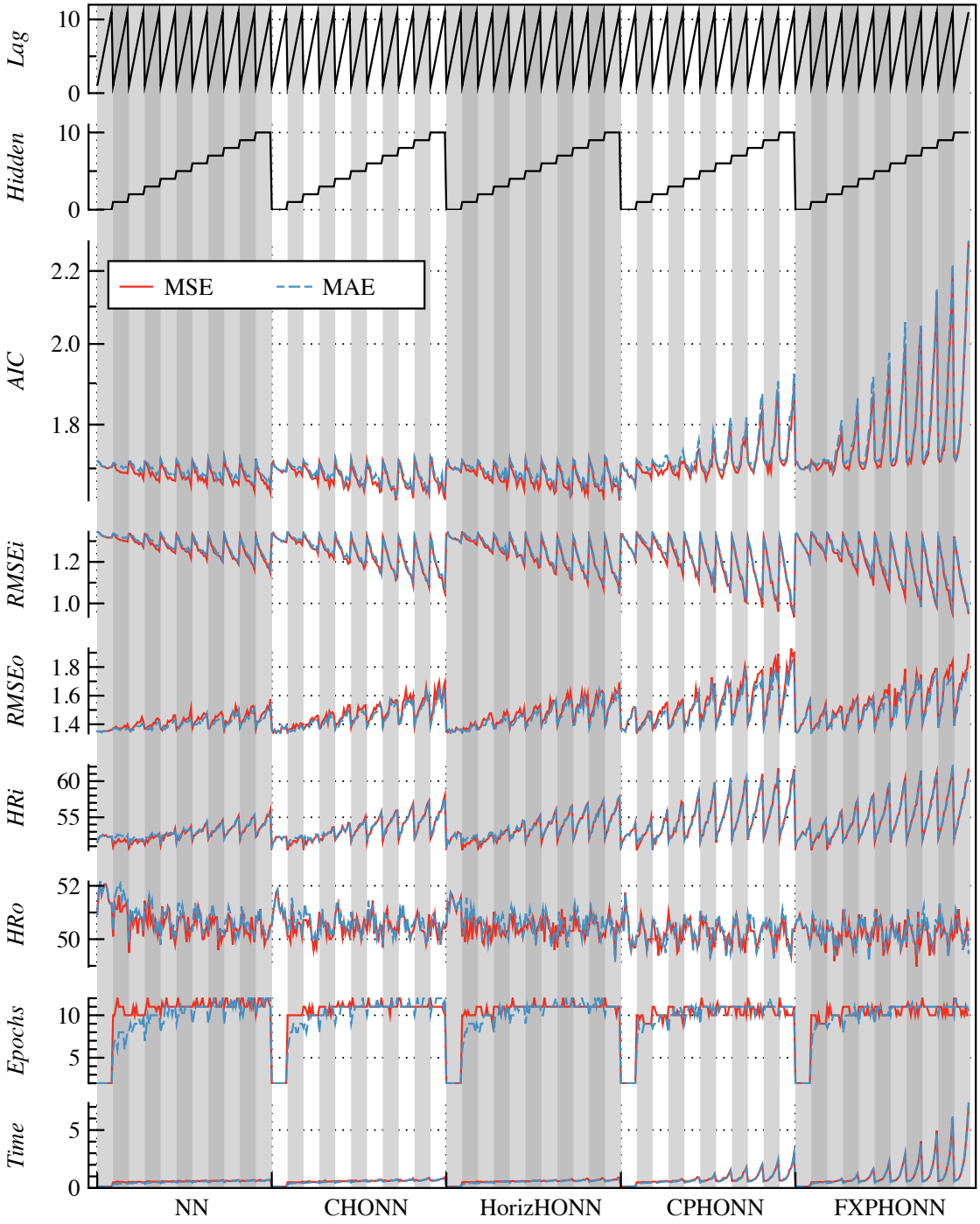

Figure 4.6: FTSE 100 Simulation results for a first order NN and 4 HONNs: AIC, in-sample and out-of-sample Root Mean Square Error, Hit Rate, and number of training epochs and training time in seconds (MSE in red, MAE in dashed blue).

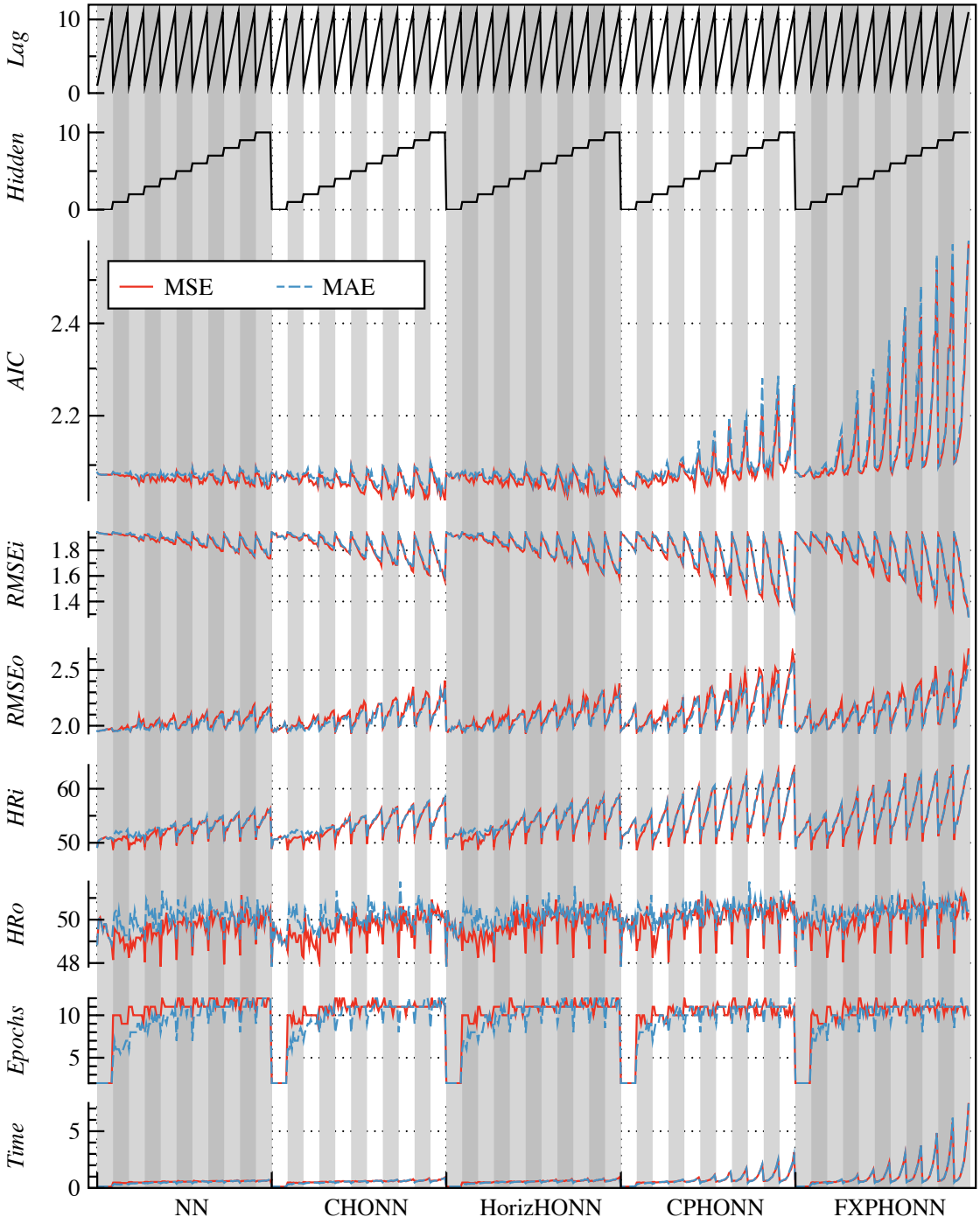

Figure 4.7: NASDAQ Simulation results for a first order NN and 4 HONNs: AIC, in-sample and out-of-sample Root Mean Square Error, Hit Rate, and number of training epochs and training time in seconds (MSE in red, MAE in dashed blue).

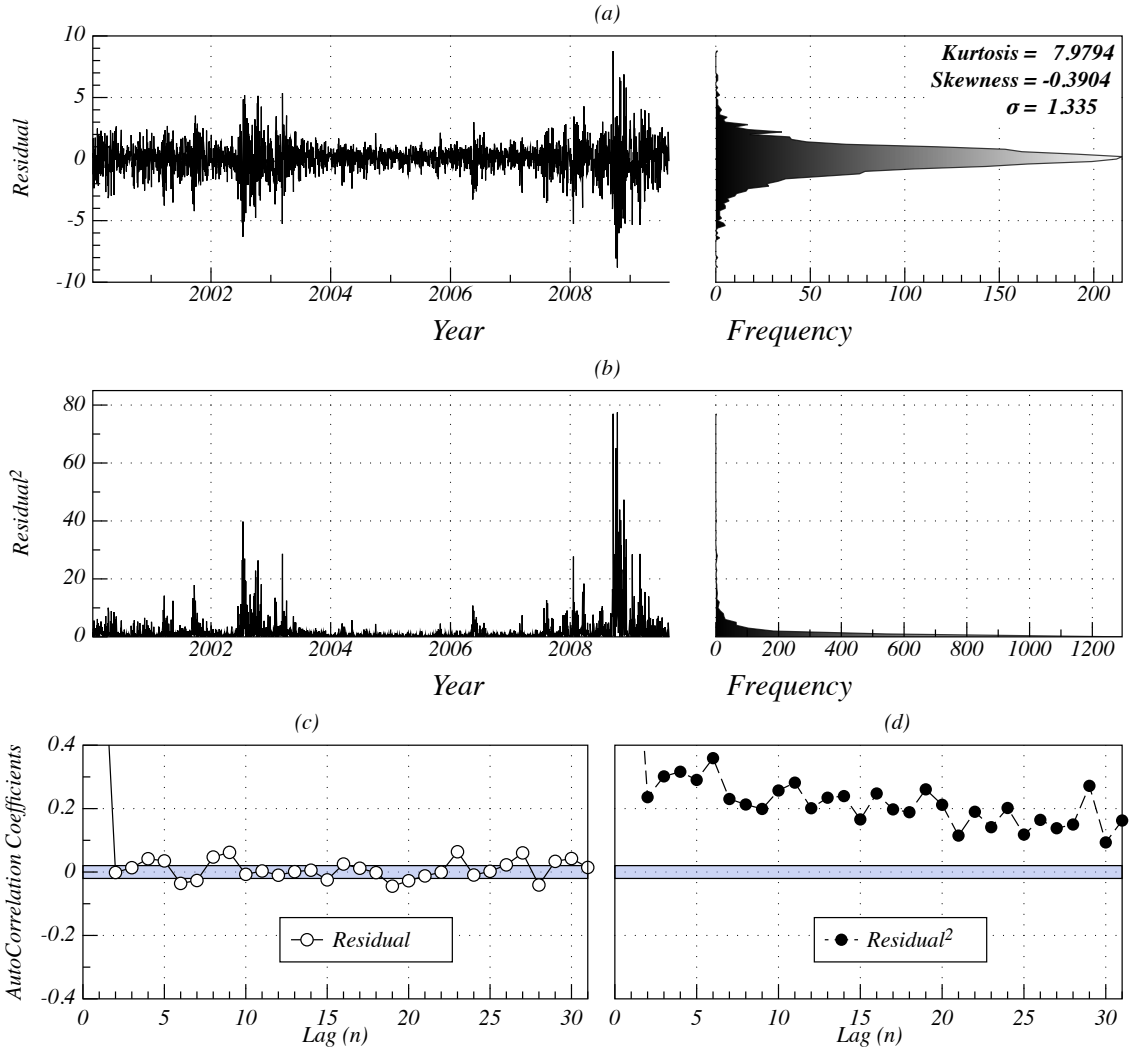

Figure 4.8: (a) Residual error of C-HONN network estimating FTSE100. (b) Squared residual errors. Autocorrelation function of (c) residual errors and (d) squared residual errors and their 95% confidence interval.

Table 4.1: FTSE100 returns estimation results (numbers in bold indicate best performance, lower AIC and $RMSE$ better, Higher $HRi/o/+$ better).

Model	$AIC(var)$	$RMSE_i$	$RMSE_o$	HR_i	HR_o	HR_o^+	Structure
C-HONN	**1.630**(0.0042)	1.047	1.342	57.83	51.75	59.14	11-10-1
Horiz-HONN	1.631(0.0004)	1.051	**1.336**	58.07	51.85	59.05	10-7-1
NN	1.648(0.0004)	1.149	1.352	55.94	51.65	**60.62**	11-10-1
CP-HONN	1.663(0.0005)	**0.935**	1.342	61.78	51.75	59.67	11-0-1
FXP-HONN	1.689(0.0008)	0.948	1.342	**61.92**	51.23	60.49	5-2-1
Linear	1.702(0.0004)	1.317	1.344	52.60	**52.16**	57.49	8-1
RW	2.097(0)	1.969	1.969	49.67	49.67	49.67	1-1

number of inputs, hidden neurons and outputs in the following manner (input-neurons-output). The structure column is for the cases with the best AIC performance. Lower AIC is an indicator of better performance in terms of errors and number of parameters. The table also indicates the variance of the AIC showing that the results closely match the indicated AIC values. The Random Walk model is, as expected, worst in both data sets indicating that we can extract patterns that are not i.i.d. The returns estimating models reduce the out-of-sample error when compared to linear regression by 4.2% for the FTSE100 and by up to 4.7% for the NASDAQ.The best HR in-sample performance is correlated with the number of parameters, so the FXP-HONN had the best performance in both data sets. However, the best out-of-sample Hit Rate (HR_o^+) is estimated by the NN in the FTSE100 data set to be 60.62% correct forecast and by the FXP-HONN in the NASDAQ data set to be 59.5% correct forecast. The best performance is an optimistic value to consider when having forecasting models with a high degree of variability, so a better more robust estimate is given by the average result which excludes outliers, both good and bad results, by using functions such as the median. So, more importantly the best average median HR_o performance was for the new C-HONN giving 51.85% for FTSE100 and the FXP-HONN gives 51.24% for NASDAQ. So in order to get the best HR estimation on average HONNs should be used. The tables also include a maximum HR_o referred to as HR_o^+. HONNs achieve the best performance possible for each criterion except HR_o^+ (in bold font).

4.4.2 Volatility Simulation

Figure 4.9 shows an example of one of the results of returns and volatility estimation. Figure 4.9 (a) shows the FTSE100 estimated volatility of the daily returns residuals after

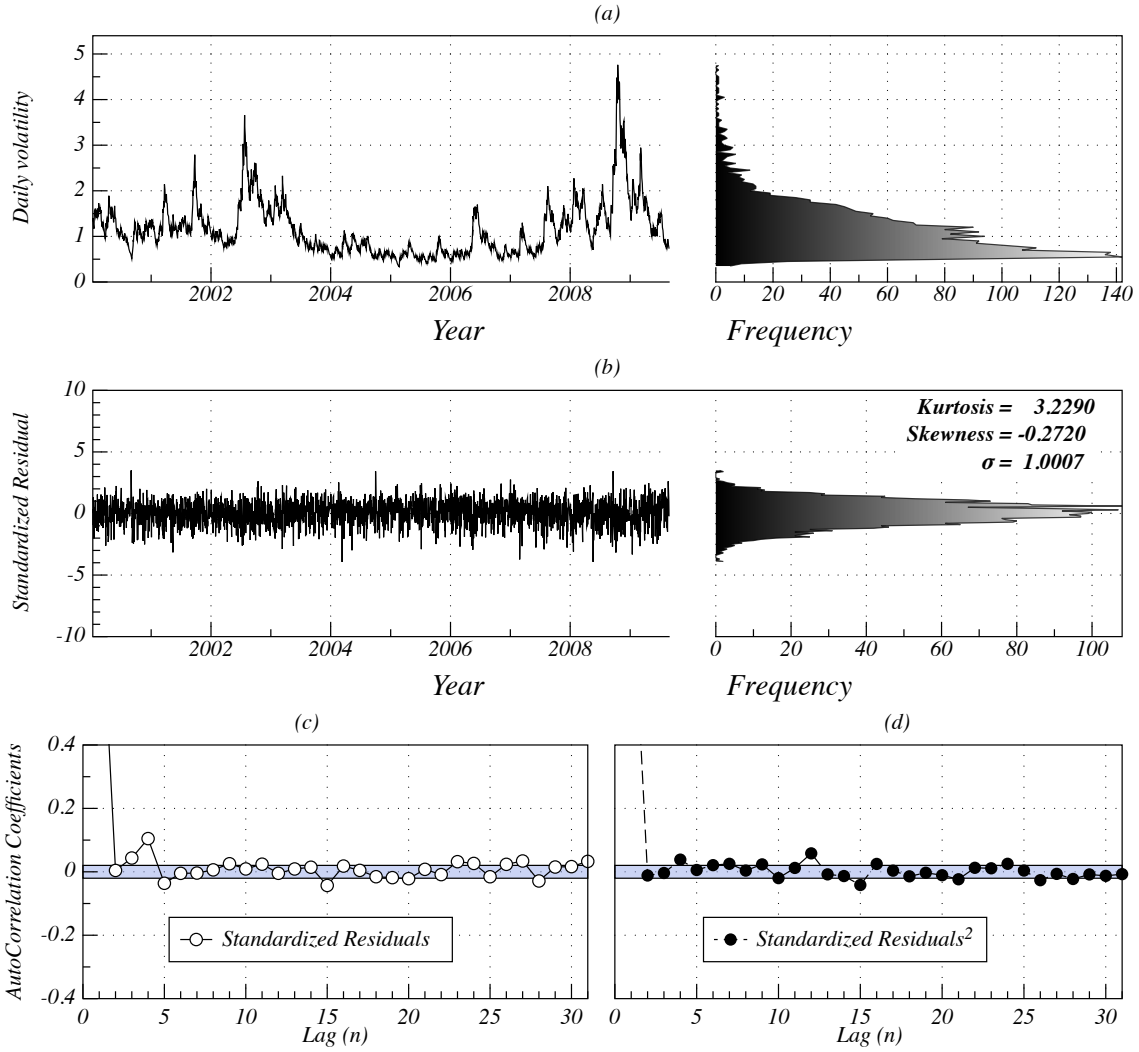

Figure 4.9: (a) Estimated FTSE100 daily returns volatility. (b) standardised Residuals. (c) Autocorrelation function of the standardised daily returns residual and the squared standardised daily returns residual when using C-HONN-EGARCH.

Table 4.2: NASDAQ returns estimation results (numbers in bold indicate best performance, lower AIC and $RMSE$ better, Higher $HRi/o/+$ better).

Model	$AIC(var)$	$RMSE_i$	$RMSE_o$	HR_i	HR_o	HR_o^+	Structure
C-HONN	**2.031**(0.0071)	1.562	**1.931**	58.21	50.93	58.97	11-10-1
Horiz-HONN	2.040(0.0002)	1.585	**1.931**	58.73	51.03	58.06	8-7-1
NN	2.054(0.0002)	1.737	1.95	56.25	51.13	58.56	9-10-1
CP-HONN	2.054(0.0014)	1.342	**1.931**	63.63	51.03	58.68	8-4-1
FXP-HONN	2.074(0.0002)	**1.279**	**1.931**	**64.39**	**51.24**	**59.5**	3-1-1
Linear	2.081(0.0002)	1.923	1.951	51.07	50.10	55.99	4-1
RW	2.452(0)	2.808	2.808	48.37	48.37	48.37	1-1

C-HONN estimation. Figure 4.9 (b) shows the standardised residuals obtained by dividing the returns residual by the EGARCH estimated volatility, $\epsilon_{standardised} = \epsilon_t/\sigma_t$, resulting in a residual that is similar to an independent and identically distributed signal, z_t, with volatility close to 1 and kurtosis close to 3. Figure 4.9 (c) displays the autocorrelation of the standardised returns and lastly the standardised squared returns are shown in Figure 4.9 (d). We notice that the hybrid C-HONN-EGARCH model captured all of the information in returns and volatility, by moving the residual autocorrelations, on average, closer to zero within the 95% confidence level of a random signal, this effect was also verified by a Ljung-Box-Pierce Q-test function provided by Matlab.

The AIC shows that volatility estimating models perform up to 6.2% better than returns only estimation for the FTSE100, and up to 2% better for the NASDAQ dataset which indicates that further information was captured from the residuals. The new networks C-HONN and Horiz-HONN had the best AIC performance when forecasting returns for both datasets with both networks having similar performance in the FTSE100, however; the Horiz-HONN had slightly worse out-of-sample performance, in both data sets the C-HONN model had better average performance, however; it also had the largest variance compared to other models. In spite of this large variance of the C-HONN model it still performed better than the NN and linear models.

We selected the new C-HONN as it had the lowest AIC to combine with the linear and nonlinear volatility estimating models GARCH and EGARCH shown in Tables 4.3 and 4.4. It can be seen that C-HONN and EGARCH provide the best AIC, $RMSE$, and HR in these simulations. The $RMSE$ of the volatility estimating models is lower due to the extraction of the volatility information from the returns residual. The χ^2 variance test, VAR, indicates that the volatility estimating models were able to capture up to 99.98% of the volatility in the data. The volatility estimating models $RMSE$ is that of

Table 4.3: FTSE100 volatility estimation results (numbers in bold indicate best performance, lower AIC and $RMSE$, Higher VAR).

Model	AIC	VAR	$RMSE$
C-HONN-EGARCH	**1.5274**	**0.9998**	**0.9998**
C-HONN-GARCH	1.5372	0.9678	1.001
Linear-EGARCH	1.5605	0.828	1.0527
Linear-GARCH	1.5358	0.9801	1.001

Table 4.4: NASDAQ volatility estimation results (numbers in bold indicate best performance, lower AIC and $RMSE$, Higher VAR).

Model	AIC	VAR	$RMSE$
C-HONN-EGARCH	**1.9957**	0.929	1.0021
C-HONN-GARCH	2.0072	**0.972**	1.0011
Linear-EGARCH	2.0512	0.638	**0.9864**
Linear-GARCH	2.0014	**0.972**	1.0013

the standardised residual errors. Models forecasting volatility had better out-of-sample $RMSE_o$; the volatility estimating models reduce the out-of-sample error when compared to linear regression by 10.25% for the FTSE100 and by up to 4.1% for the NASDAQ. In terms of AIC when combining the C-HONN with the established GARCH and EGARCH models they provide up to 2.1% and 2.7% improvement in AIC for FTSE100 and NASDAQ when compared to conventional models of Linear-EGARCH models. In both tables, Linear-E/GARCH indicate that the results were obtained from the best first order Neural Network which was reduced to a linear regression network followed by a volatility estimation model; this type of process is commonly used for returns and volatility forecasting. The results show the advantage of using a hybrid returns and volatility forecasting model using the new C-HONN-EGARCH combination.

4.5 Conclusions

This work presented an investigation into financial data forecasting using linear, First order Neural Network, and Higher Order Neural Network models. The performance of the HONN models was superior to other types of network or linear models. The new C-HONN provided the best performance for forecasting the daily returns series of FTSE100 and NASDAQ. The FXP-HONN had the best hit rate prediction for the NASDAQ time series at 59.5% and 60.62% was achieved for the FTSE100 using a first order NN. We conclude

that models that assume the data has a normal Gaussian distribution with constant volatility fail to capture all of the information available within the data as reflected in the AIC, $RMSE$ values and correlation tests. Even the best performing C-HONN did not capture the conditional volatility in the residual of the returns. This information is captured using models that take into account conditional volatility, such as GARCH and EGARCH. HONNs forecasting returns were combined with the GARCH and EGARCH models, for the first time, to give a hybrid model. It was observed that the hybrid model reduced the $RMSE$ error by 10.25% for the FTSE100 and by 4.1% for the NASDAQ datasets when compared to linear regression. The best performing model was the hybrid C-HONN-EGARCH combination model for the data sets considered.

Chapter 5

Higher Order Neural Networks for Camera Calibration

5.1 Introduction

Camera calibration is a process in which a real world image is mapped to match its projection as seen by the camera sensor. The image seen by the camera lies in a 2D Camera Coordinate System (CCS), and the reference scene lies in the 3D World Coordinate System (WCS). Most calibration models are based on geometrical relations which map the WCS to CCS based on the pinhole camera model.

Conventional camera calibration techniques divide the geometrical relationships analytically into linear transformations and non-linear distortions; referred to as extrinsic and intrinsic parameters, Bakstein (1999); Weng et al. (1992). The parametric calibration can only capture the information according to the analytical constructs they are designed to replicate. So, even a highly optimised parametric calibration process, in some cases, leaves a calibration residual error that has not been accounted for by the analytical model, Qiu and Song (1995).

This work proposes a new modification on a technique for camera calibration based on non-parametric mapping using Neural Networks. A Higher Order Neural Network is used to map the WCS to CCS using supervised learning. Where the HONN emulates the camera calibration model by accounting for statistical errors that were not considered in the parametric model. This chapter shows the improvements gained by non-parametric camera calibration by using Generalised Correlation Higher Order Neural Networks, Selviah and Shawash (2008) as a non-parametric solution for the camera calibration problem.

This chapter compares the parametric models to non-parametric models in two contexts. The first context is the comparison of the reduction of the calibration error when

mapping WCS-to-CCS using parametric, non-parametric, and a hybrid model combining the linear part of the parametric model with a HONN and a NN that rectifies the non-linear camera distortion and systemic errors. The second part of this study is concerned with CSS-to-WCS mapping using implicit models.

The chapter is organised as follows. Section 5.2 describes the existing and the new calibration techniques considered. Section 5.3 presents the experiments and simulations, followed by section 5.4 with the results. Section 5.5 ends with the conclusions.

5.2 Camera Calibration

Camera calibration can be ascribed as estimating a function which maps image coordinates from WCS to CCS. WCS occurs in 3D denoted by point $P = [x, y, z]^T$. CCS occurs in 2D, $p = [x, y]^T$.

In this section we provide a brief summary of the processes that comprise parametric, non-parametric and semi-parametric models.

5.2.1 Parametric Camera Calibration

Camera calibration is the process of finding explicit linear mapping in the projective space encoding the internal camera parameters described by the linear pinhole model. Furthermore, many real camera systems cannot be fully described by this linear pinhole model and require an estimation of the non-linear distortions transpiring in camera systems. There are many well established techniques that take into account both effects; Heikkila and Silven (1997); Tsai (1987); Zhang (2004).

Calibration procedures vary in their naming and the number of steps they involve. However, the equations describing these models revert to two types of functions that determine the mapping process. Linear functions are based on extrinsic parameters, while the non-linear functions are based on the intrinsic parameters. Extrinsic parameters include focal length, centring, scaling, translation and rotation parameters. Intrinsic parameters include distortions such as the radial and tangential lens distortions. This type of calibration is referred to as parametric calibration where all the aforementioned parameters form an equation that maps the WCS 3D data to the 2D image seen in CCS.

The Camera coordinate system (CCS) referred to as ($P = [x, y, z]^T$) usually measured in pixels which are coincident on the xy-plane, and optical axis along the z-axis. The centre of the image plane is at the origin, c, and the lens focal length of the lens f, Bacakoglu and Kamel (1997).

Equation (5.1) describes of the intrinsic linear mapping which projects points from WCS to CCS using a $[3 \times 3]$ rotation matrix R, and a $[3 \times 1]$ translation vector T, the $\hat{hat}$ denotes calculated parameters.

$$\begin{aligned} \hat{P} &= R \times P + T \\ \hat{p} &= f \times \hat{P} + c \end{aligned} \tag{5.1}$$

The non-linear radial and tangential distortion are calculated using (5.2). Radial distortion are caused by the shape of the lens, these distortions are highly dependent on the radius. Tangential distortion is due to manufacturing defects where the lens is not exactly parallel to the sensor causing the view of the image to occur at an angle perpendicular to the radius. The parameters which constitute the radial distortion depend on the radius, $r^2 = x^2 + y^2$. The tangential distortion is modelled on the following set of parameters; $a_1 = 2xy$, $a_2 = r^2 + 2x^2$, $a_3 = r^2 + 2y^2$. The equation in the first set of brackets indicates the radial distortion (with terms up to the 6th degree). The parameters in the brackets on the right account for the tangential distortion.

$$\begin{aligned} \hat{x}_{new} &= [x(1 + \rho_1 r^2 + \rho_2 r^4 + \rho_5 r^6)] + [\rho_3 a_1 + \rho_4 a_2] \\ \hat{y}_{new} &= [y(1 + \rho_1 r^2 + \rho_2 r^4 + \rho_5 r^6)] + [\rho_3 a_3 + \rho_4 a_1] \end{aligned} \tag{5.2}$$

The new projected points resulting from (5.1) and (5.2) are compared to the target coordinates in CCS to generate an error vector. The error vector, E, is used to estimate the error gradient with respect to the vector containing the parameters used to perform the projection, ϕ. The error gradient matrix, J, is used in the Levenberg-Marquardt non-linear learning algorithm. This algorithm finds the optimal parameters, ϕ, which reduce the error between the projected point and the target CCS values.

$$\phi_{new} = \phi_{old} + H^{-1} \times \nabla J \tag{5.3}$$

In (5.3), H is the Hessian matrix, $H = J^T J$, and $\nabla J = J \times E$ is the gradient of the error.

5.2.2 Non-Parametric Camera Calibration

This type of calibration is usually described as an implicit camera calibration technique which maps points from the WCS to CCS. Neural Networks are an example of this type of calibration where they are used for sensor and camera calibration, Maleki et al. (2004); Woo and Park (2009).

A feedforward neural network is used to learn the relationships, linear and non-linear, by finding the optimal neuron weights that link the reference grid to the distorted one. Neural networks are suited for learning complex nonlinear mappings where not all of the explicit analytical formulation is available, Wells (1996). This approach has several distinct features when compared to the parametric one. By using the non-parametric approach, no analytical form of the distortion surfaces is assumed. The distortion surfaces are derived directly from the training samples by compensating for the systematic mapping errors statistically, Qiu and Song (1995).

In this paper an advance type of NNs will be used. Ordinary Feed-forward neural networks are capable of handling linear and non-linear separations within the input space. Higher Order Neural Networks contain processing units that are capable of performing functions such as polynomial, multiplicative, smoothing or trigonometric functions, Giles and Maxwell (1987); Selviah et al. (1991), which generate more complex decision regions which are multiply connected. A simple HONN could be thought of as describing elliptical curve regions as HONN functions can include square terms, cubic terms, and higher orders. In order to obtain a similar complex decision regions ordinary NN need to incorporate increasing number of neurons and hidden layers.

HONNs will use implicit mapping instead of explicit mapping. Denoting that instead of computing the extrinsic and intrinsic parameters of the camera system as in (5.1),(5.2). A NN can describe the projection by using the formula in (5.4), where the Higher Order Function is represented by HO and the NN parameters are represented by $[W_1, W_2, b_1, b_2]$. N is the output from the first layer. $W1, b_1$ account for both the normal input and its higher order function. Network parameters initialization is described later. $tanh$ is hyperbolic tangent non-linear activation function in the hidden layer, see Figure 5.1.

$$\begin{aligned} N &= W_1 \times [P, HO]^T + b_1 \\ \hat{p} &= W_2 \times tanh(N) + b_2 \end{aligned} \tag{5.4}$$

One of the draw backs of HONNs is that they can consume large amounts of computing resources if set to encompass all the possible permutations of the input data. Each neuron in the input layer can be comprised of many multiplicative operations which when accumulated can begin to increase processing time, Selviah and Shawash (2008); Taylor (2009).

For example, if WCS P is considered, then the possible input patterns could be: x, y, z, xy, xz, yz, xx, yy, zz, xxy, xxz, etc. Which represent combination of all possible inter-multiplications of the input space. This type of function can be described as a Kronecker process, $\otimes$, in (5.5).

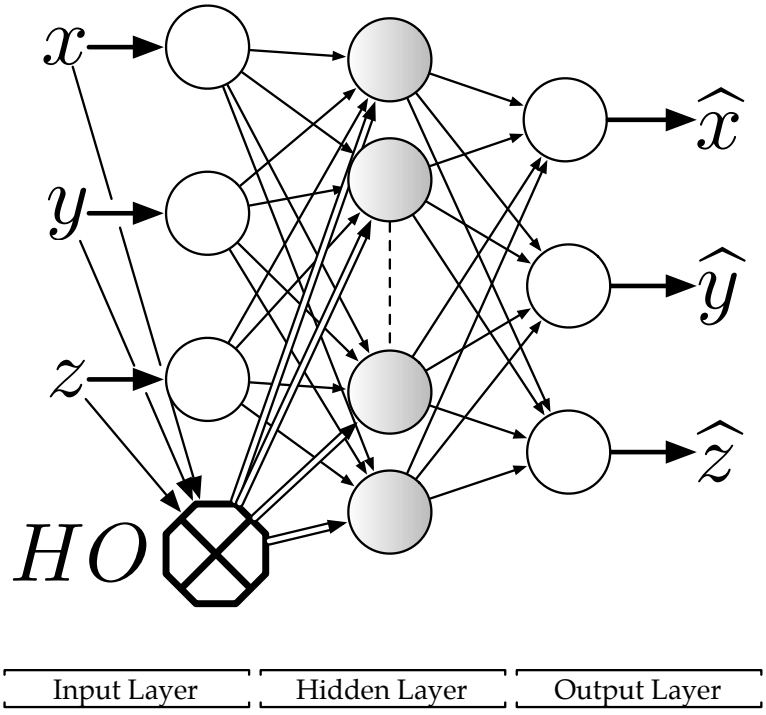

Figure 5.1: A Higher Order Neural Network with inputs, $P = (x, y, z)$ and a Higher Order Function represented by HO, N is the output from the first layer. The projection outputs are represented by $\hat{p} = (\hat{x}, \hat{y}, \hat{z})$.

$$\begin{aligned} H(P) &= P \otimes P \\ &= [x\ y\ z] \times [x\ y\ z]^T \end{aligned} \tag{5.5}$$

This type of operation constitutes a generic HONN which contains all the possible permutations, this network will be referred to as the Full Cross Product HONN (FXP-HONN) containing 9 parameters, (5.6).

$$FXP - HONN = \begin{pmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & yz & zz \end{pmatrix} \tag{5.6}$$

The output parameters from (5.6) contain redundant information, 6 parameters are non-unique. With the current data set under consideration, an elementary change performed by selecting only unique polynomial terms of FXP-HONN can reduce the number of parameters almost by one third, we refer to this network as Cross Product HONN (CP-HONN), (5.7).

$$CP - HONN = \begin{pmatrix} xx & xy & xz \\ & yy & yz \\ & & zz \end{pmatrix} \tag{5.7}$$

A further reduction on the number of parameters can be achieved by summing the diagonal and off diagonal traces of the CP-HONN to generate another two HONNs; the Correlation HONN (C-HONN), (5.8), and the Horizontal HONN (Horiz-HONN), (5.9). Both of these networks reduce the number of parameters by 33% when compared to the ordinary HONN.

$$C - HONN = \begin{pmatrix} xx + yy + zz \\ xy + yz \\ xz \end{pmatrix} \tag{5.8}$$

$$Horiz - HONN = \begin{pmatrix} xx + xy + xz \\ yy + yz \\ zz \end{pmatrix} \tag{5.9}$$

5.2.3 Semi-Parametric Camera Calibration

The semi-parametric camera calibration is a method that is similar to the parametric method insofar that it extracts the extrinsic camera parameters, however; it appends a non-linear approximation of the intrinsic parameters using a HONN or a NN instead of the

radial and tangential distortion parameter estimation. Semi-parametric calibration can be considered as combining steps that find the extrinsic parameters using the parametric model combined with an estimation of the intrinsic errors statistically using NNs and HONNs.

5.2.4 2D-to-3D mapping

In this chapter we also investigate inverse mapping by using non-parametric camera mapping from the CCS to WCS (2D-to-3D) as opposed to WCS to CCS (3D-to-2D). A number of studies solve this problem analytically, Anchini et al. (2006); Clarkson et al. (2001); Fanhuai et al. (2004); Phong et al. (1995); Shi et al. (2004); van de Kraats et al. (2005). Also, by using neural networks, Lynch (1999); Memon and Sohaib (2001). However, both those techniques require multiple views of the same plane. In this study we examine this type of mapping without the use of a secondary view. So the results from this results can only be applied on 2D data captured when the 3D coordinates occur on the same plane or distance from the camera.

5.3 Experiment

5.3.1 Test Data

In order to compare parametric calibration techniques with the new HONN non-parametric calibration we used two data sets that are publicly available on the Microsoft Research and the Caltech vision research websites, Bouguet (2008); Zhang (1998).

The data used in this experiment was obtained from the Caltech Camera Calibration Toolbox (CCT) repository. The first data set is the calibration planes used in, Zhang (1998) shown in Figure 5.2, this data will be referred to as "plane data". The plane data calibration points represent data extracted from multiple views of the same plane. The second calibration data set are the reference and distorted grids used in, Heikkila and Silven (1997), shown in Figure 5.3, this data will be referred to as "Cube data". The Cube data is extracted from a grid that is comprised of two sides of a cube which appear as two planes in WCS.

5.3.2 Simulation design

For parametric modelling we used the Caltech Camera Calibration toolbox for Matlab, also available in OpenCV. This toolbox provides a good benchmark since it encompasses

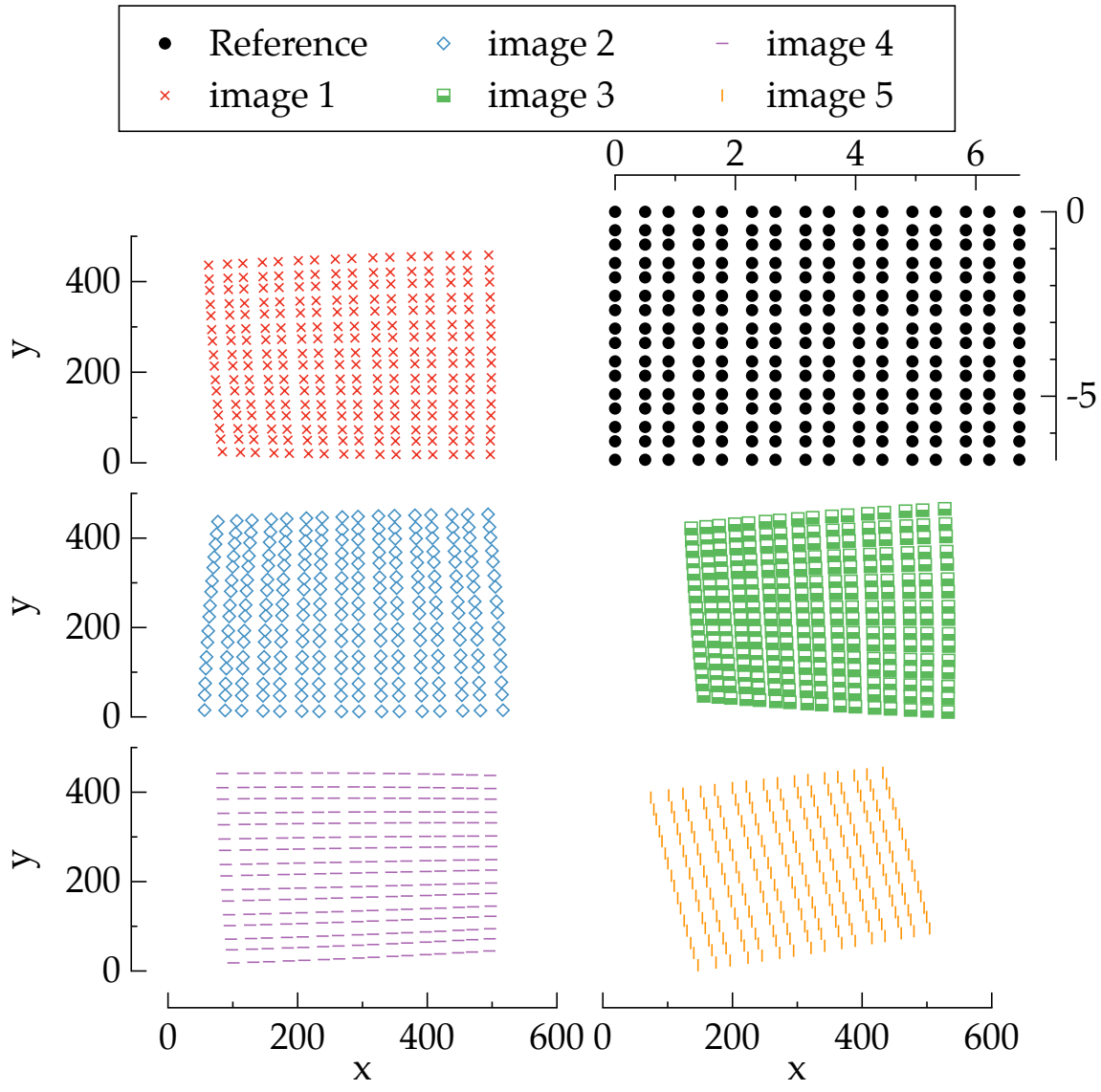

Figure 5.2: The 3D Reference grid and its plane distortion seen in 2D from 5 different views.

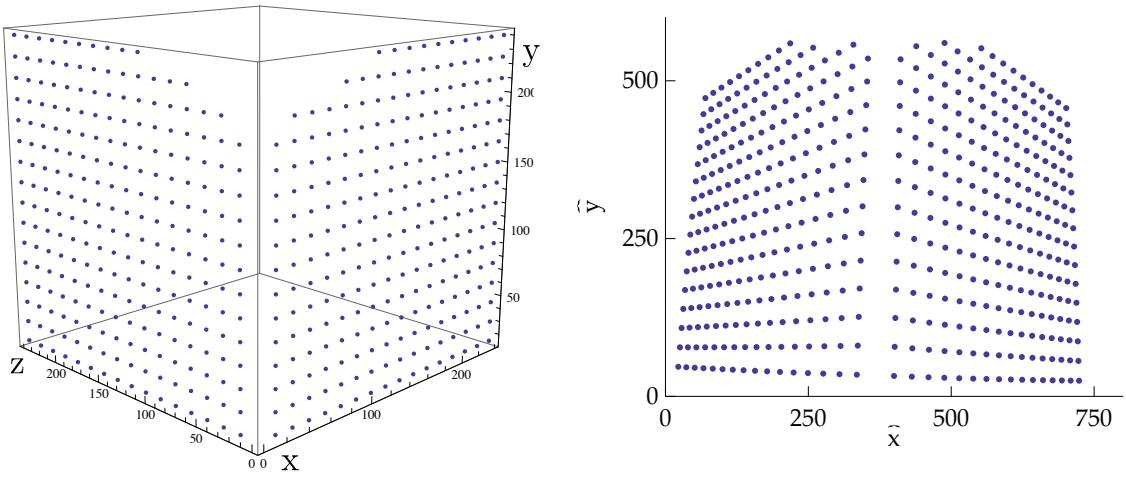

Figure 5.3: 3D Cube data (x, y, z) and its corresponding 2D plane $(\hat{x}, \hat{y})$.

a comparison of the other calibration techniques and camera calibration toolboxes such as, Heikkila and Silven (1997); Zhang (1998). For details on the techniques adopted by this toolbox please refer to the original author's documentation, Bouguet (2008). As for the non-parametric calibration we used Matlab 2009b, Mathworks (2009),which includes the Neural Networks toolbox, where the networks were designed, trained and tested.

The neural networks were designed with structures that included a single hidden layer with varying number of neurons. NNs were trained 10 times with different starting points that were initialised using the Nguyen-Widrow algorithm, Nguyen and Widrow (1990). The input and target data sets are randomly divided into three parts; training, validation and testing data having 70%, 15% and 15% points respectively. This division reduces over-fitting and makes sure the three data sets do not exclude regions with different properties in the area in the image. All input and output data points were scaled from their respective magnitudes to a range of [-1,1] which corresponds the desired input/output boundaries for optimal neural network training. NN training was performed using the Levenberg-Marquardt learning algorithm, (5.10), Marquardt (1963); this algorithm is also used by the CCT for parametric calibration.

$$\Delta\phi = -(H + I\mu)^{-1} \nabla J \tag{5.10}$$

After training is completed the output data from the NNs is scaled back to the domain of the test images and the error is extracted. Networks were trained on each of 5 planes and cube data separately. The neural networks reported in the results section were the ones whose structure gave consistently lower MSE on average. For ease of comparison with the CCT the performance will be reported as the standard deviation of the calibration residual.

Semi-parametric calibration was obtained by using the CCT to find the optimal intrinsic parameters. Then the error of the resultant projection was fed into a NN or HONN system that accounted for non-linear distortions. As for the CCS-to-WCS (2D-to-3D) calibration, the Neural Network toolbox was used to find NNs and HONNs that map CCS-to-WCS for every data set individually.

5.4 Results

This section is divided into two parts. The first part shows the results of simulations comparing WCS-to-CCS parametric, non-parametric and semi-parametric calibrations. The second part outlines the results of CCS-to-WCS mapping, i.e. the performance of obtaining 3D coordinates from 2D points.

5.4.1 3D-to-2D Mapping

Figure 5.4 shows the convergence of NN error with respect to increasing number of neurons in the hidden layer benchmarked against the performance of the CCT indicated by the thick line. From the two graphs we can see that all HONNs and NNs converged to a lower errors than the parametric method. Compared to NNs, HONNs error converges with a lower number of hidden neurons, 3 against 4 for the plane data, and 7 compared to 10 for the cube data.

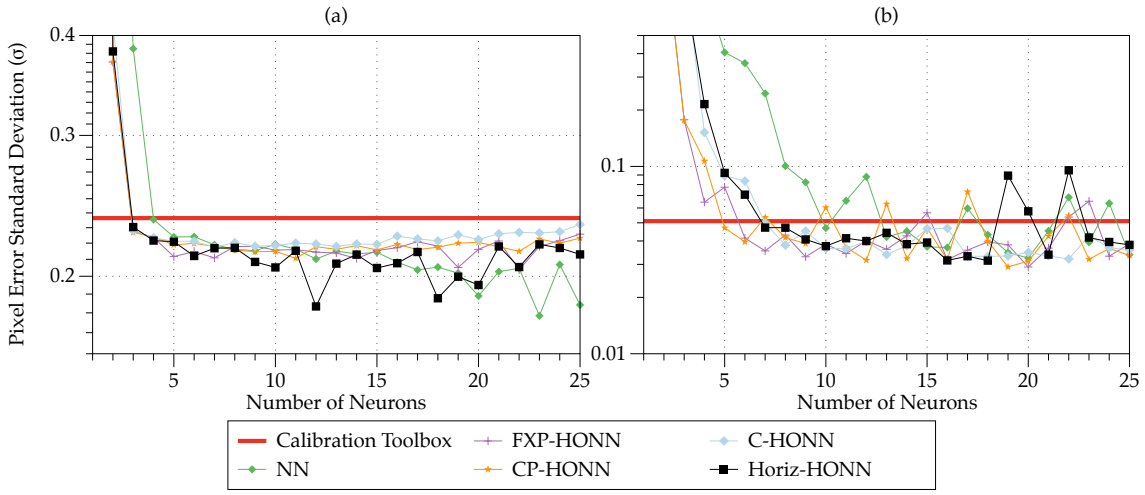

Figure 5.4: Calibration error convergence for 3D-to-2D parametric mapping compared to HONNs and NNs with varying hidden neurons for (a) Plane data. (b) Cube data.

Data	CCT	NN	FXP	CP	C	Horiz
image 1	0.2454	0.2103	0.2085	0.2088	0.2159	0.2047
image 2	0.1595	0.1086	0.1121	0.1214	0.1223	0.1220
image 3	0.3810	0.3558	0.3718	0.3678	0.3688	0.2756
image 4	0.1676	0.1319	0.1450	0.1475	0.1511	0.1467
image 5	0.1446	0.1211	0.1218	0.1256	0.1275	0.1173
All	0.2364	0.2071	0.2055	0.2058	0.2145	**0.1797**
Semi	0.7891	**0.1968**	0.2101	0.2028	0.2045	0.2156
Cube	0.0510	0.0324	**0.0291**	**0.0291**	0.0331	0.0314
Semi	0.6765	0.0343	**0.0334**	0.0343	0.0350	0.0359

Table 5.1: Calibration Error standard deviation (σ) for mapping with the CCT, NN and HONNs from WCS-to-CCS (3D-to-2D) and semi-parametric calibration for both the plane and the cube data.

Table 5.1 is divided into an upper and a lower segment. The upper segment shows the calibration error standard deviation resulting from 3D-to-2D mapping of the coordinates of the 5 planes indicated in the first column, with later columns indicating the calibration model. Best results are indicated in bold font. We notice that on average, all non-parametric models had 14% lower calibration error than the CCT. The Horiz-HONN had the lowest average error of $\sigma = 0.1797$ pixels which is 24% lower than CCT. The percentage enhancement of using non-parametric models is shown as graph (a) in Figure 5.5.

The final row in the upper segment shows the results of the semi-parametric calibration. The results are not conclusive with regards to any improvements when using the hybrid method. However, the average results of the semi-parametric method were lower by 12% compared to CCT.

The last two rows in Table 5.1 show the calibration error of the cube data; parametric vs non-parametric. And the last row shows the semi-parametric calibration results. The results show a similar pattern as in the previous data. The non-parametric reduced the mapping error by up to 43% in the cases of FXP and CP-HONNs, and by 39% on average when compared to CCT. The percentage enhancement of using non-parametric models is shown as graph (b) in Figure 5.5. As for semi-parametric cube calibration, the results indicate that the calibration error of the hybrid model are larger than the non-parametric model, however, they too are lower than the CCT by 32% on average with FXP-HONN having the largest reduction of error by 34%.

These results indicate that mapping errors are reduced when using non-parametric

models, with both the planar and cube mapping gaining significant calibration improvement. The hybrid model had better performance than the CCT, however, it did not give a lower calibration error than the fully non-parametric model. A benefit of the hybrid model is that it retains the intrinsic parameters from the linear analytical model.

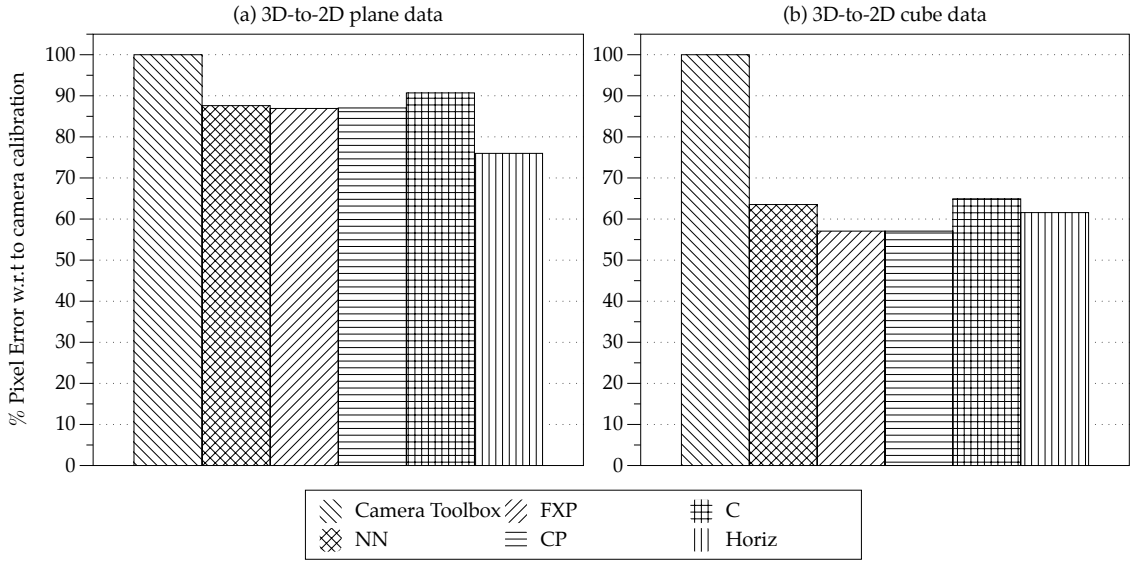

Figure 5.5: Calibration error σ for the camera calibration and the 5 Networks. (a) 3D-2D average performance of 5 plane images, (b) 3D-2D mapping of cube to grid.

5.4.2 2D-to-3D mapping

In this section we present the results of the inverse mapping problem of obtaining 3D coordinates from 2D coordinates. Figure 5.6, shows the convergence of HO/NNs when mapping (a) plane mapping, (b) cube data. Currently, we do not have an analytical model as a benchmark, so we set the lowest performing model as the benchmark in this case. In Figure 5.6 we notice a pattern similar to the one mapping WCS-to-CSS in terms of NN and HONN error convergence with respect to how many neurons are included in the hidden layer.

Table 5.2 shows the calibration error resulting from 2D-to-3D mapping of the plane data in the upper segment and the cube data in the last row. There are no results for the parametric model as it is not capable of performing the required transformation from CCS to WCS using this type of data, i.e data from a single view of the plane/cube. We notice the the NN and HONN CCS-to-WCS mapping of the plane data were almost similar, with FXP and CP HONN having a marginal performance improvement over the rest of

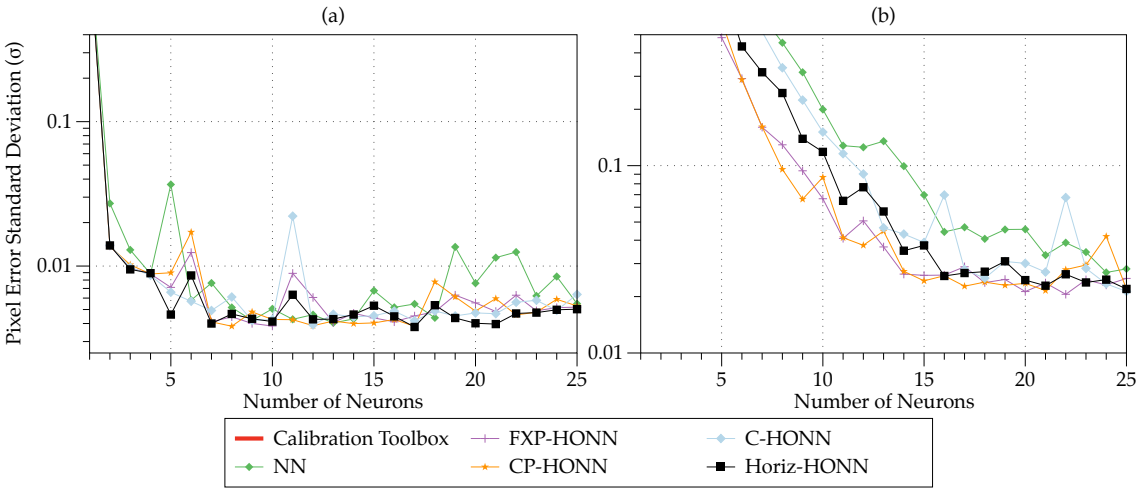

Figure 5.6: Calibration error convergence for CCS-to-WCS (2D-to-3D) mapping compared using HO/NNs for (a) Plane data, (b) Cube data.

Data	NN	FXP	CP	C	Horiz
image 1	0.0036	0.0037	0.0034	0.0037	0.0034
image 2	0.0019	0.0037	0.0018	0.0019	0.0029
image 3	0.0064	0.0061	0.0067	0.0076	0.0100
image 4	0.0025	0.0026	0.0027	0.0025	0.0034
image 5	0.0024	0.0036	0.0025	0.0027	0.0030
All	0.0037	**0.0036**	**0.0036**	0.0037	0.0037
Cube	0.0407	**0.0213**	0.0227	0.0251	0.0245

Table 5.2: Pixel Error in standard deviation (σ) for mapping with the CCT, NN and HONNs from CCS to WCS (2D-to-3D) for the 5 images and the cube data.

the models. Figure 5.7 (a) and (b) show calibration error of all HONN models compared to the NN model as a benchmark.

However, in the case of cube mapping we see a significant drop in the error when using HONNs compared to NNs, Figure 5.7 (b). The FXP and CP-HONN provide 47.6% and 44.2% lower mapping errors. This reduction in mapping error can be be attributed to the more complex nature required when mapping non-planar data by HONNs.

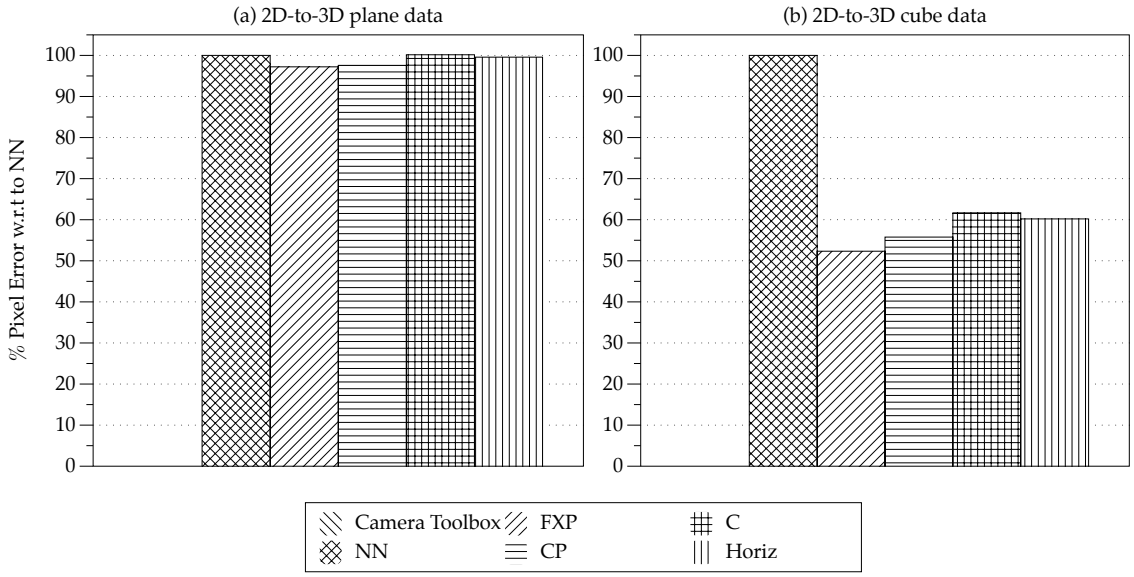

Figure 5.7: 2D-3D calibration error reduction in percentage compared against NNs for (a) Plane data (b) Cube data.

5.5 Conclusions

This work compared the mapping performance of a new type of Higher Order Neural Network, the Generalised Correlation Higher Order Neural Network, against standard analytical methods in three situations; 3D-to-2D mapping, a hybrid model combining HONNs with an analytical model, and the performance of HONNs compared to NNs when mapping 2D-to-3D. The study considered 2 types of data, multiple views of plane data, and a cube data comprising two planes on adjacent sides of a cube.

The results indicate that HONN calibration outperform both the standard analytical parametric model used by the Camera Calibration Toolbox (CCT) in all cases and the non-parametric model based on ordinary Neural Networks (NN) in some of the cases. HONN 3D-to-2D mapping reduced the error by more than 14% on average with the the

Horiz-HONN having 24% lower error than the parametric method for plane data. 3D-to-2D cube mapping had more drastic reduction in the error when using HONNs, with reduction of mapping error of up to 43% in the cases of FXP and CP-HONNs.

The new hybrid models combining analytical solution for linear parameters combined with a HONN based non-parametric model to account for non-linear distortions out performed the parametric modelling performed by the CCT. However, it did not provide a discernible mapping improvement in the case of plane data and it had a worse performance when mapping cube data. Inverse mapping from 2D-to-3D using HONNs did not provide discernible improvement when compared to the NN model. However, in the case of cube data the FXP and CP-HONNs provided 47.2% and 44.2% drop in error respectively. This can be attributed to the more complex nature of mapping non-planar data using HONNs, and the availability of non-degenerate information for the z-dimension. HONNs provide mapping improvements compared to the parametric model since it aims to eliminate the systematic error without being limited to a fixed analytical model. HONNs also outperform NNs as the Higher Order functions reformulate the input space into one that reflects the higher order interactions of the input data passing through a camera system. The simulations were limited to only two sets of data. Benchmarking HONN mapping capabilities against the standard model and the NN model presents a strong case for further research.

Chapter 6

Higher Order Neural Network Training on Limited Precision Processors

6.1 Introduction

Dedicated digital hardware, such as Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs) can achieve real-time, high speed, low latency operation, however, all those enhancements come with a penalty of reduced precision. There is a trade-off between the throughput of a design and the precision and resources required for satisfactory operation. There have been many studies on the operation of Artificial Neural Networks (ANNs) on real-time, low precision electronic hardware, Jung and Kim (2007); Sahin et al. (2006); Zhu and Sutton (2003b). Furthermore, efforts in improving NN operation and training performance has been accomplished by using floating-point libraries that make use of specific operation structure speed ups have been implemented in works such as Bilmes et al. (1997). However, these types of libraries are not applicable in the context of this study of NN training and operation in fixed-point. It was found that the ANN output error depends on the number of hidden layers and other factors, Piche (1995); Stevenson et al. (1990). Reducing the size of ANNs for simpler operation, learning can allow for accurate solutions. A number of researchers have found that they have to train ANNs offline on high precision floating point processors such as general purpose processors to preserve accuracy during training. The results of this training are then quantised to obtain a lower precision design which works in a limited precision environment.

In real time hardware, the ANN size poses a more serious problem than in software running on a floating point CPU due to the more limited circuit resources such as memory. The size of the ANN limits the learning offline in software as the time and memory requirements grow with ANN size. even though parallel hardware processors significantly increase the speed at which ANNs operate, they can only accommodate ANNs which do no exceed the available limited resource, Lopez-Garcia et al. (2005); Maguire et al. (2007). Another way to reduce the size of an ANN is to move the complexity from inside the hidden layers of the network to a pre-processing stage before it by using a Higher Order Neural Network (HONN). So we investigate the implementation of the recently introduced Correlation HONN (C-HONN), Selviah and Shawash (2008), and compare it with a first order ANN and a HONN in a limited precision environment. Dias et al. (2006) demonstrated that it is possible to implement an on-line Levenberg-Marquardt (LM) training algorithm in software; the use of online learning, as opposed to batch learning, reduces the memory requirements and operation time. The ability to operate LM algorithm online with reduced memory and operation complexity suggests that the LM algorithm may be suited for implementation on real time reduced precision hardware where it has not been used before. Therefore, we compare the LM online training with Back Propagation (BP) online training in a limited precision environment to find the lowest precision at which learning is feasible.

It was found that if training is performed in a limited precision environment the ANN converges correctly for high precision but below some threshold level of precision the training does not correctly converge. The most similar work in this area was done by Minghu et al. (2000), where a single type of HONN with BP training was investigated. To our knowledge no one has either trained nor even run or operated a HONN in a limited precision environment. We present the first demonstration of running and operating two HONNs in an emulated limited precision environment and show how to find the lowest precision which at which training and a convergence to a solution are still possible.

Section 6.2 describes HONNs and on-line learning algorithms. Section 6.3 details the experimental method. Sections 6.4, 6.5, and 6.6 present the simulations and the results. The discussion and conclusions are presented in sections 6.7.

6.2 Generalised Correlation Higher Order Neural Networks

ANN applications expanded to take into account the highly complex non-linear data available in various systems, such as communications systems and economic data. In general,

when the data complexity is increased, the ANN size needs to expand accordingly, with the possibility of reaching sizes that are impractical for ANN training even in software. One solution is to present the ANN with a function of the data, a function which exposes the interrelations that are difficult to capture using the hidden layer. These ANNs are sometimes referred to as functional-link networks, Masters (1993). HONNs provide a method to present the ANN with all of the possible high order interactions between the elements of the input vectors. HONNs can easily model the exclusive OR function (XOR) as they can transform the input from a linearly inseparable space to a space where data can be classified more easily by simpler linear operators. HONN also proved to be a better way to store information, Giles and Maxwell (1987); Lee et al. (1986); Personnaz et al. (1987). even though it increases the parameter count, the transformation helps reduce the ANN model dependence on the hidden layers; at times eliminating them by using outer products or tensor models, Pao (1989). Transforming input data for an ANN often dramatically speeds up training. HONNs were also successfully applied to financial prediction with an astonishing twofold improvement over ordinary ANNs in some cases, Fulcher et al. (2006).

Selviah and Shawash (2008) introduced the Generalised Correlation HONN that transforms input data by performing a localised correlation operation on the input data vector. Refer to Section 4.2.4 for details.

6.2.1 Artificial Neural Network Training Algorithm Review

The training of ANNs aims to find a set of weights that give a global minimum in the error function surface where the optimal performance can be achieved. There are two methods to optimise a function, the deterministic and the probabilistic methods, Lee (2007). In this study, we consider two popular deterministic supervised learning methods, Lee et al. (2004), the error Back-Propagation and the Levenberg-Marquardt algorithms, Marquardt (1963); Press et al. (1992). Since the error surface may be complex, convex and may have concave regions, Fu (1994), it is more likely that the network settles into a local minimum than a global one when using a deterministic method. This problem cannot be completely avoided in the deterministic training methods; however, it can be reduced in a number of ways. Firstly, early stopping, Haykin (1999) recommends that the data is split into three parts, training, validation and test sample sets; the training set is used to optimise the weights in the network, the validation set is used to stop the training when the validation error becomes less than a certain value with respect to the training error. Secondly, random selection of training, validation and test sets ameliorates the danger of training on data characterised by one set of a local type of data, thus giving a better generalisation

Table 6.1: Back-Propagation Algorithm

1	$while\, i < Max\ Iteration$	
2	$Net_{Hidden} = W_1 \times \begin{bmatrix} X_0 \\ 1 \end{bmatrix}$	Output of first layer
3	$X_{Hidden} = f(Net_{Hidden})$	Output after hidden layer
4	$Net_{Output} = W_2 \times \begin{bmatrix} X_{Hidden} \\ 1 \end{bmatrix}$	Network output
5	$E = Target - Net_{Output}$	
6	$\Delta E_{out} = f'_{linear}(X_{Hidden}) \cdot E$	Error in output layer
7	$\Delta W_2 = \Delta E_{out}$	Output layer weight change
8	$\Delta E_{Hidden} = W_2^T \times \Delta E_{out}$	
9	$\Delta W_1 = f'_{logistic}(X_{Hidden}) \cdot \Delta E_{Hidden}$	Hidden layer weight change
10	$W_{2_{new}} = \alpha \times W_{2_{old}} + (1-\alpha) \times \eta \times \Delta W_2 \cdot \begin{bmatrix} X_{Hidden} & 1 \end{bmatrix}^2$	New hidden-output layer weights
11	$W_{1_{new}} = \alpha \times W_{1_{old}} + (1-\alpha) \times \eta \times \Delta W_1 \cdot \begin{bmatrix} X_0 & 1 \end{bmatrix}^2$	New input-hidden layer weights

Check if training conditions are still true, if yes: repeat, otherwise exit training

ability to the network, Kaastra and Boyd (1996). Thirdly, local minima can be avoided by using randomly selected starting points for the weights being optimised Masters (1993), we use the Nguyen-Widrow initialisation method Nguyen and Widrow (1990). Lastly, Overfitting can also be reduced by removing ANN complexity by reducing the number of elements in the ANN by estimating and selecting the best AIC with respect to lag and layer size. From our study of ANN properties in software, it was recommended to use the smallest network size which provides valid operation. The same recommendations apply to hardware implementation to achieve the smallest circuit footprint and power efficiency, Marchiori and Warglien (2008). The two learning methods are summarised in Tables 6.1 and 6.2, where the stopping criteria (momentum, learning rates, and maximum training epoch number) are set to values used widely in literature. The derivation of both algorithms is available in A.

Table 6.2: Levenberg-Marquardt Algorithm

1	$while\, i < Max\, Iteration$	
2	$Net_{Hidden} = W_1 \times \begin{bmatrix} X_0 \\ 1 \end{bmatrix}$	Output of first layer
3	$X_{Hidden} = f(Net_{Hidden})$	Output after hidden layer
4	$Net_{Output} = W_2 \times \begin{bmatrix} X_{Hidden} \\ 1 \end{bmatrix}$	Network output
5	$E = Target - Net_{Output}$	Error in output layer
6	$\theta = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}$	Weight vector
7	$J(\theta) = \begin{bmatrix} f'_{linear}(X_{Hidden}) \\ W_2 \times f'_{logistic}(X_{Hidden}) \end{bmatrix}$	Jacobian matrix
8	$\nabla J = J \times E$	Error gradient
9	$H = J^T \times J$	Hessian matrix
10	$H = H + \lambda \times diag(H)$	Updating Hessian matrix
11	$\Delta\theta = H^{-1} \times \nabla J$	Weight change
12	$\theta_{new} = \theta_{old} + \Delta\theta$	New weight vector
13	$W_{2_{new}} = W_{2_{old}} + \theta_{new}(W_2)$	New hidden-output layer weights
14	$W_{1_{new}} = W_{1_{old}} + \theta_{new}(W_1)$	New input-hidden layer weights
15	Updating λ	
16	$L = \Delta\theta^T \nabla J + \Delta\theta^T \Delta\theta \lambda_{old}$	Calculating update conditions
17	$\lambda = \begin{cases} \frac{\lambda}{2} & if \quad 2N(MSE - MSE_{new}) > 0.75L \\ 2\lambda & if \quad 2N(MSE - MSE_{new}) \leqq 0.25L \end{cases}$	New lambda
	Check if training conditions are still true,	
	if true: repeat or go to step 10 otherwise exit training	

Table 6.3: Floating-point to Fixed-point conversion workflow

1	*Golden Algorithm* Our Reference Design which we seek to implement in hardware
2	*Floating Point Design Verification* We need to validate our design before we continue with the conversion to hardware language
3	*Simulation of algorithm tradeoffs* Explore design tradeoffs of functions that are impractical to implement in hardware
4	*Conversion to fixed-point* Exploring a software implementation of fixed-point algorithmic behaviour
5	*Simulation of fixed-point implementation tradeoffs* Examine tradeoffs of word length variation, pipelining and loop unrolling if possible
6	*Generation of hardware code* Automatic hardware code generation
7	*Validation and verification of design deployed on hardware* Verification by analysing test-bench performance

6.3 Experimental Method

In order to investigate the effects of reduced precision on the learning process we followed algorithm in Table 6.3 to convert from floating point operation to fixed point. The floating point learning algorithm is referred to as the Golden Algorithm and is used as the benchmark for comparing the effect of reduced precision ranging from 4 to 28 bits fixed-point representation, Zarrinkoub (2006).

Data quantifiers are used that transform the data into the limited precision domain by rounding and truncation in a manner similar to that of digital circuits. Quantifiers were used before and after every operation presented in the network operation and learning modes, in tables 6.1 and 6.2. The precision of the quantifiers were varied depending on the section and data path in the network operation and learning modes. The best network structure was found in software using Matlab Neural Network Toolbox ver.2008a. The best structure was selected according to the AIC information criterion, where this parameter selects simplest and best performing ANN. The fixed-point toolbox in Matlab 2008a was chosen as the development environment as it allows for efficient prototyping by providing the ability to incorporate fixed-point functionality in signal processing algorithms with

further functionality that can produce device specific targeted codes which can be ported onto a hardware circuit at a later stage for direct hardware implementation, Bhatt and McCain (2005); Ou and Prasanna (2004, 2005); Xilinx (2008a).

6.4 Simulations

This section describes the way the simulations were structured in order to show the effect of reduced precision on network training algorithms. Matlab was used as our floating algorithm development environment and the fixed-point toolbox was used to convert the learning algorithms into fixed-point (reduced precision). The conversion from floating-point to fixed-point was applied to two learning algorithms BP, and LM. These two algorithms were implemented with precisions ranging from 4 to 28 bits and tested on two data sets: the XOR function and waveguide wall roughness measurements. These two functions were chosen as the data sets under consideration due to several factors; the XOR has discrete levels and the wall roughness is almost continuous, so they require different levels of quantisation in order to operate and learn and both data sets are used in electronic engineering applications.

6.4.1 Exclusive OR (XOR)

The XOR function was chosen as a demonstration as it is one of the functions that cannot be solved by linear models, ANNs without a hidden layer, or a single layer first order ANN. XOR input samples were generated by using a threshold function for 1000 values from a uniformly distributed pseudorandom number generator. The threshold function had a threshold level of 0.5.

$$Sample = \begin{cases} 1 & , \quad value > 0.5 \\ 0 & , \quad value \leq 0.5 \end{cases} \tag{6.1}$$

During the Fixed-point training and operation data was split into two parts, training and testing. The networks structures were set following the following pattern: (IN-Hidden-Output). In this pattern IN, the input dimension, depending on the estimation model used, the hidden layer has 4 neurons, and output dimension of 1.

Table 6.4 shows the maximum range which the data paths used during training with BP and LM displayed in different columns for clarity. The multiply and accumulate are given in algorithms 6.1 and 6.2 for these parameters. The table shows the minimum number of bits to take the maximum range into account, in order to prevent the algorithm from overflowing. Underflow is not considered as some numbers require an infinite precision to represent, however; underflow is reduced by increasing the overall precision.

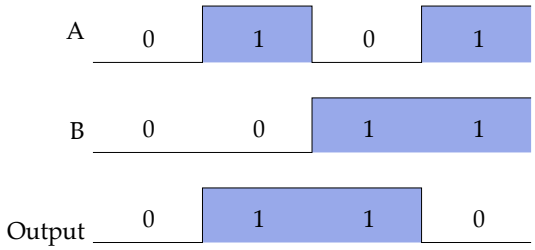

Figure 6.1: Exclusive OR function

Table 6.4: Minimum parameter range during training of the XOR function neural network models using two learning algorithms with three network types

BP					LM			
Range	ANN	C-HONN	CP-HONN		Range	ANN	C-HONN	CP-HONN
X_0	1	1	1		X_0	1	1	1
W_1	2	2	2		W_1	3	2	2
W_2	2	2	1		W_2	2	1	2
Net_{Hidden}	**3**	2	**3**		Net_{Hidden}	4	1	4
X_{Hidden}	1	1	1		X_{Hidden}	1	1	1
Net_{Output}	2	2	1		Net_{Output}	1	1	1
δW_1	1	1	1		E	2	1	1
δW_2	2	2	1		J	1	1	1
ΔW_1	1	1	1		H	**7**	6	7
ΔW_2	2	2	1		∇J	5	4	4
					$\Delta\theta$	2	1	1

6.4.2 Optical Waveguide sidewall roughness estimation

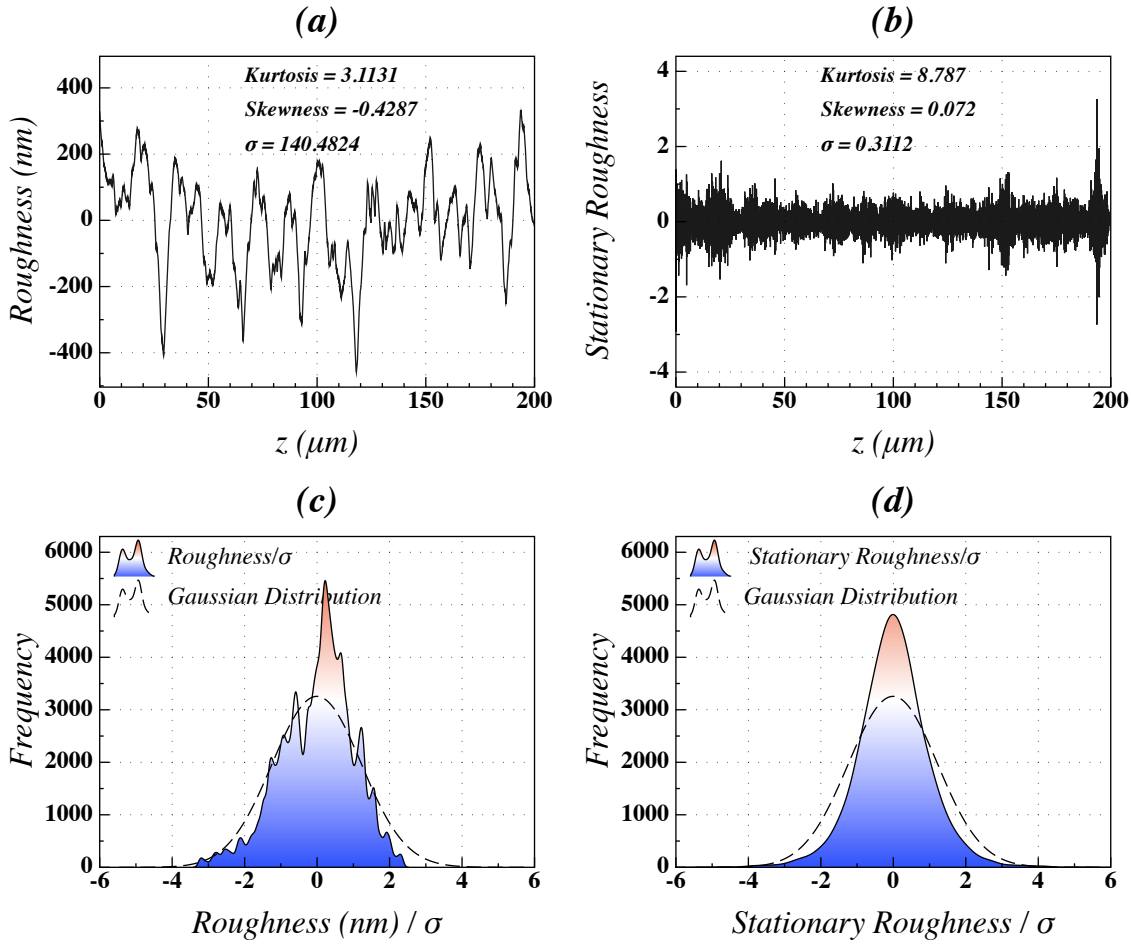

Figure 6.2: (a) Waveguide sidewall roughness measurements with an accuracy of 6 significant figures. (b) Stationary transformed waveguide sidewall roughness. (c) Probability distribution function (PDF) of waveguide sidewall roughness. (d) PDF of stationary waveguide wall roughness.

A good description or estimate of the sidewall nano-roughness of multimode rectangular core optical waveguides can facilitate for high bit rate short distance interconnects. As only small a region of the waveguide sidewall can be measured with an atomic force microscope (AFM); estimation enables a larger region to be synthesised for modelling. modelling allows us to investigate the coupling between bound modes and between bound and radiation modes which affect the equilibrium modal power distribution, cross-talk and loss, Papakonstantinou et al. (2008, 2009). Figure 6.2(a) shows the first experimentally measured data of polymer optical waveguide sidewall roughness consisting of

10,000 samples taken every 20 nm. The data is non-stationary making it difficult for networks to model. Figure 6.2(c) shows the histogram of the roughness measurements scaled by its standard deviation in order to compare it to the histogram in Figure 6.2(d). Figure 6.2(b) shows roughness measurements transformed into a stationary form by calculating the differences of the logarithms of the non-stationary signal for adjacent points, Chatfield (1989).

$$y_s = 100 \times [log(y_{ns}) - log(y_{ns-1})] \tag{6.2}$$

Where, y_s is the stationary transformation of the non-stationary data represented by, y_{ns}. This transformation operation is non-destructive, as no information is lost during this conversion, so the data can be converted back into the original non-stationary form when required. This transformation converts multiplicative (ratio) relationships in the data to simpler add (subtract) operations that simplify and improve network training, Masters (1993). Haykin (1999) mentions that ANNs alone are insufficient to capture the dynamics of non-stationary systems. However, non-destructive data transformations can transform the data to a stationary form, to alleviate this problem.

Figure 6.2(c) and (d) show two PDFs of the non-stationary and stationary data with a common Gaussian distribution fit to the non-stationary data in both the figures to emphasise the effect of the data transformation and the way it converts the distribution of the non-stationary data into a form more similar to a Gaussian distribution.We simulated various networks estimating wall roughness in software first to find the best for hardware implementation. All models were simulated with input dimension ranging from 1 to 9 input units, and hidden layers having 1 to 8 hidden neurones. To our knowledge there is no specific convention for choosing the correct ANN structure for use in limited precision, so we used the procedure given in Algorithm **??**, choosing the best network models in software and converting these models into a format suitable for limited precision hardware.

Comparing Table 6.4 with Table 6.5, the ranges of the various stages have increased due to the nature of the roughness data set taking values close to a continuous function with a Gaussian distribution, while the XOR data had discrete values of 0 and 1.

6.5 XOR Modelling Results

Table 6.6 shows the Linear and network evaluation criteria when BP training is preformed in floating point. The best performing models are in bold font. C-HONN has the best LLF and AIC criteria. All networks models converge to a solution; the linear model is unable to capture the solution for the non-linear XOR function as expected.

Table 6.5: Minimum parameter range during training for the estimation of waveguide sidewall roughness using two learning algorithms with three network types

BP				LM			
Range	ANN	C-HONN	CP-HONN	Range	ANN	C-HONN	CP-HONN
X_0	3	4	4	X_0	3	4	4
W_1	2	2	2	W_1	3	3	3
W_2	2	2	1	W_2	9	7	8
Net_{Hidden}	3	4	**5**	Net_{Hidden}	2	2	2
X_{Hidden}	1	1	1	X_{Hidden}	1	1	1
Net_{Output}	2	2	2	Net_{Output}	1	1	1
δW_1	2	2	2	E	1	2	1
δW_2	2	3	3	J	4	5	4
ΔW_1	2	4	4	H	21	**22**	19
ΔW_2	2	3	3	∇J	12	13	12
				$\Delta\theta$	3	2	2

Table 6.6: XOR BP floating point estimation results of log-likelihood function (LLF) and Akaike Information Criteria (AIC) and the Root Mean Squared Error ($RMSE$) lower values indicate better performance

	LLF	AIC	$RMSE$
C-HONN	**-8897.85423**	**-17.7657085**	**4.51E-09**
CP-HONN	-7915.16279	-15.8043256	3.23E-08
ANN	-6737.17429	-13.4563486	3.40E-07
Linear	73.9634	0.7696	0.7083

Figure 6.3 shows the RMSE versus the number of BP training epochs for several levels of precision, Q for 3 networks. The best performing networks have lowest RMSE and are fastest in reaching this lowest error. ANN-BP convergence in Figure 6.3 (a) is very slow compared to graphs C-HONN-BP convergence in (b) and CP-HONN-BP convergence in (c). The C-HONN-BP in graph (b) has the fastest initial convergence rate (epochs 3-30) compared to the other two networks with the same training method. Both C-HONN and CP-HONN convergence rates increase in later epochs. Based on these graphs in Figure 6.3, the new network C-HONN is best for modelling the XOR when BP training in limited precision.

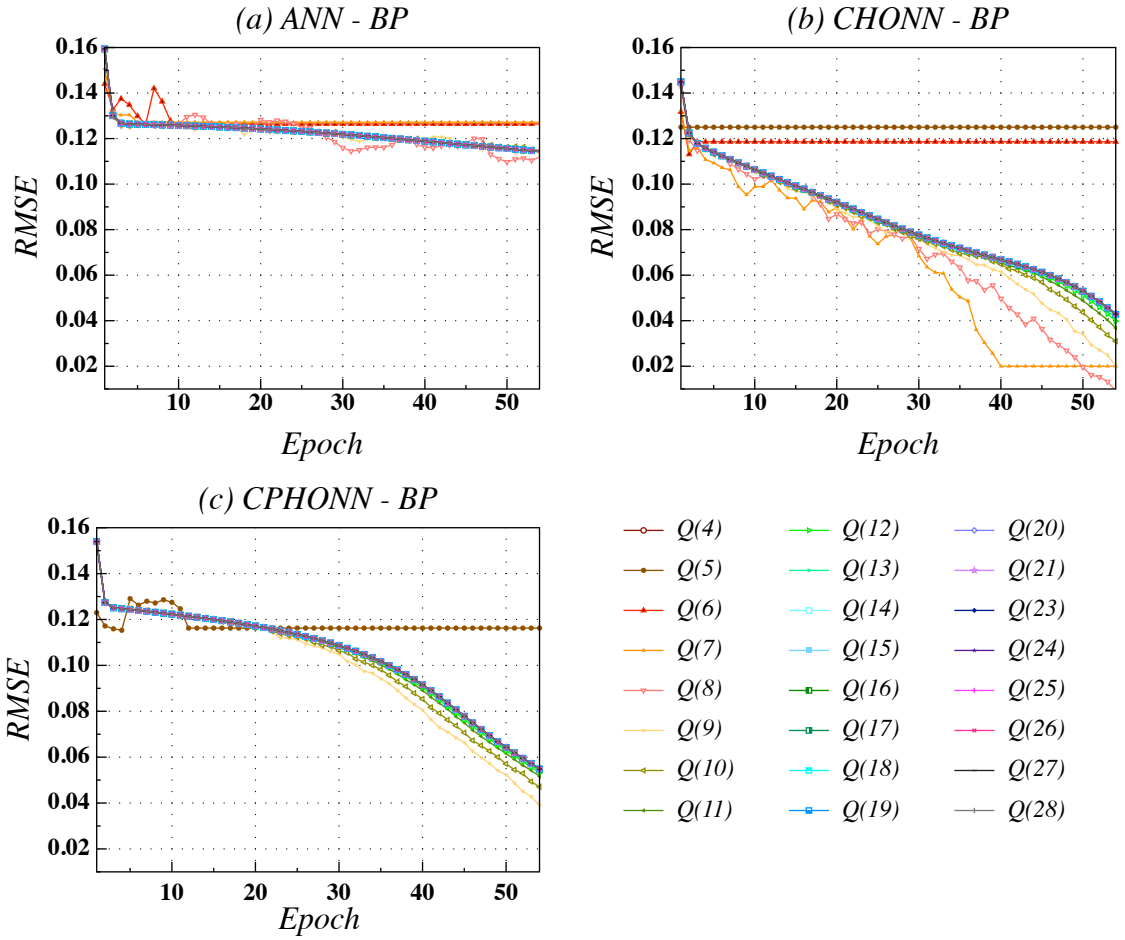

Figure 6.3: BP Training Error for several levels of precision, Q for XOR modelling

Figure 6.4 (a), (b), and (c) show that all networks reach lower RMSE values in a smaller number of epochs with the LM learning algorithm rather than the BP learning algorithm. Figures 6.4 (b) and (c) the HONNs converge after epoch 6 to high precision values (low

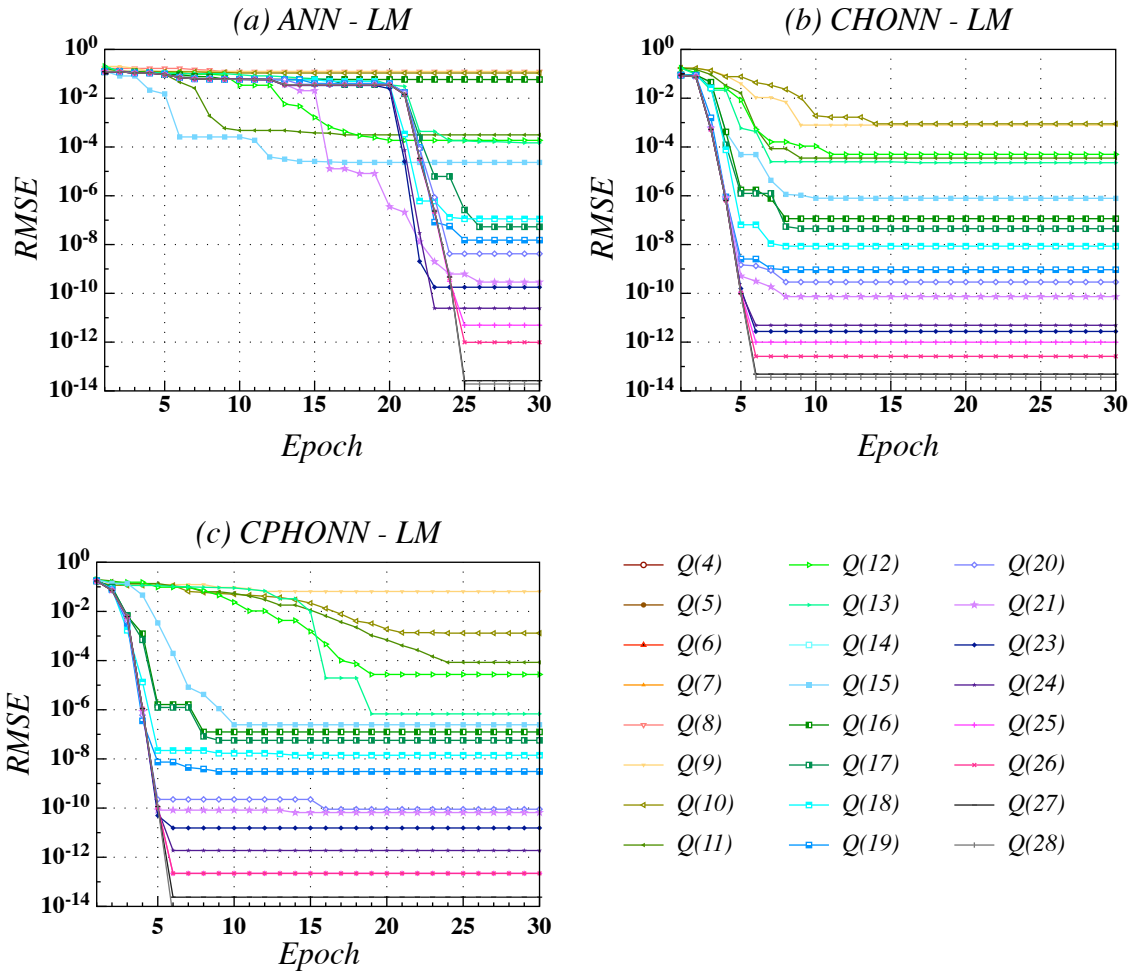

Figure 6.4: LM Training Error for several levels of precision, Q for XOR modelling

$RMSE$), while ANNs require at least 23 epochs to achieve the same precision, incurring an almost 4 fold increase in convergence time. Higher levels of precision give progressively lower errors as seen in figure 6.4. The $RMSE$ in the LM learning algorithm falls to very low values such as 10^{-7} due to the fixed-point nature of the number representation having two discrete levels at 0 or 1. As networks converge to the optimal solution, the values making up the error vector are rounded and truncated to 0, whereas the same parameters would leave small residuals in the error vector if operated in floating point. RMSE values lower than 10^{-6} can be considered as converged or optimal solutions with no need for further enhancement, as most learning algorithms use 10^{-6} as a training stopping criterion. The best choice for XOR modelling in reduced precision is either C-HONN or CP-HONN with LM training.

Figure 6.5 shows the $RMSE$ for all networks for both learning algorithms as a function of precision level, Q after 55 epochs. The two shaded regions to the left of the figure indicate the region in which the minimum ranges shown in bold font in Table 6.4 make some of the quantisers operate with a fraction with negative power, i.e. shifting the fraction point to the left, leading to more severe levels of underflow. The C-HONN gave the best error performance for BP training with the CP-HONN second and the ANN worst. The C-HONN and CP-HONN vie for the lowest $RMSE$ after LM training.In BP learning, beyond a certain level of precision, increases in precision no longer lead to increases in $RMSE$. We refer to the point at which $RMSE$ stops improving as the Minimum Precision for Lowest Error ($MPLE$). Setting the precision to the MPLE avoids increases in the design complexity and circuit size minimising the overall latency. The $MPLE$ for the XOR function under BP training is 12 bits. A further study needs to be done in order to determine the $MPLE$ for LM training for XOR modelling, as the $RMSE$ had not reached its floor before reaching the maximum precision of this study.

6.6 Optical Waveguide Sidewall Roughness Estimation Results

This section presents the results of operating and training networks with an input dimension of 9, 1 hidden layer with 1 hidden node, to estimate the next sample of the wall roughness. Floating point results are first presented followed by fixed-point simulations of the learning algorithms at different precisions. Table 6.7 shows that the C-HONN is a better estimator than the ANN and CP-HONN in floating point as indicated by bold font. Although CP-HONN has better RMSE levels the other networks, best models are selected using the AIC criterion as mentioned previously.

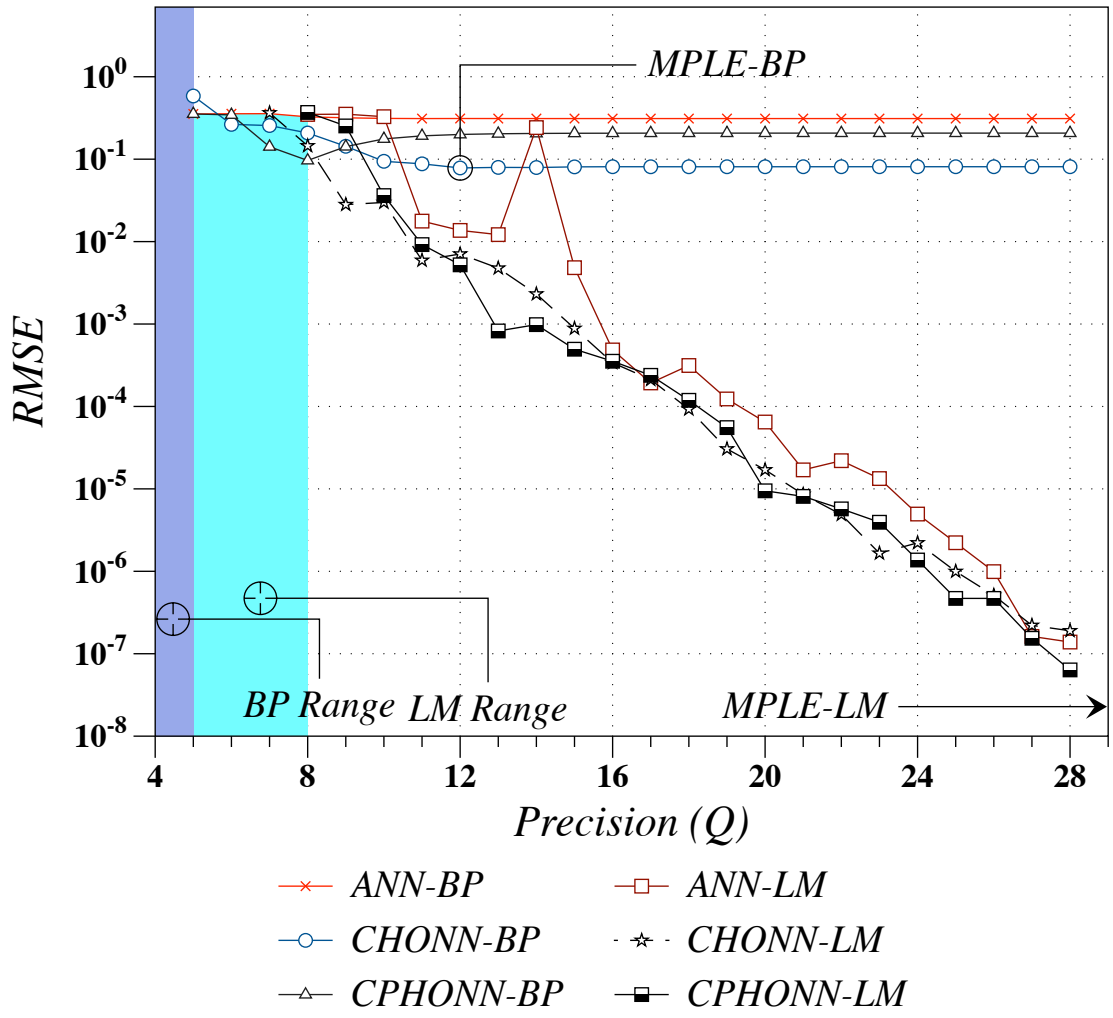

Figure 6.5: Networks output error after 55 epochs as a function of level of precision, Q for XOR modelling

Table 6.7: Waveguide roughness LM trained floating point estimation results Log-likelihood function (LLF), Akaike Information Criteria (AIC), Root Mean Square Error (RMSE), lower values are better, Hit Rate (HR) -higher values are better-

	LLF	*AIC*	*RMSE*	*HR*
C-HONN	**-1923.5642**	**-0.1905**	**0.019425**	77.796
ANN	-1906.1974	-0.1896	0.019559	**77.802**
CP-HONN	-1853.6991	-0.1799	0.018712	77.161

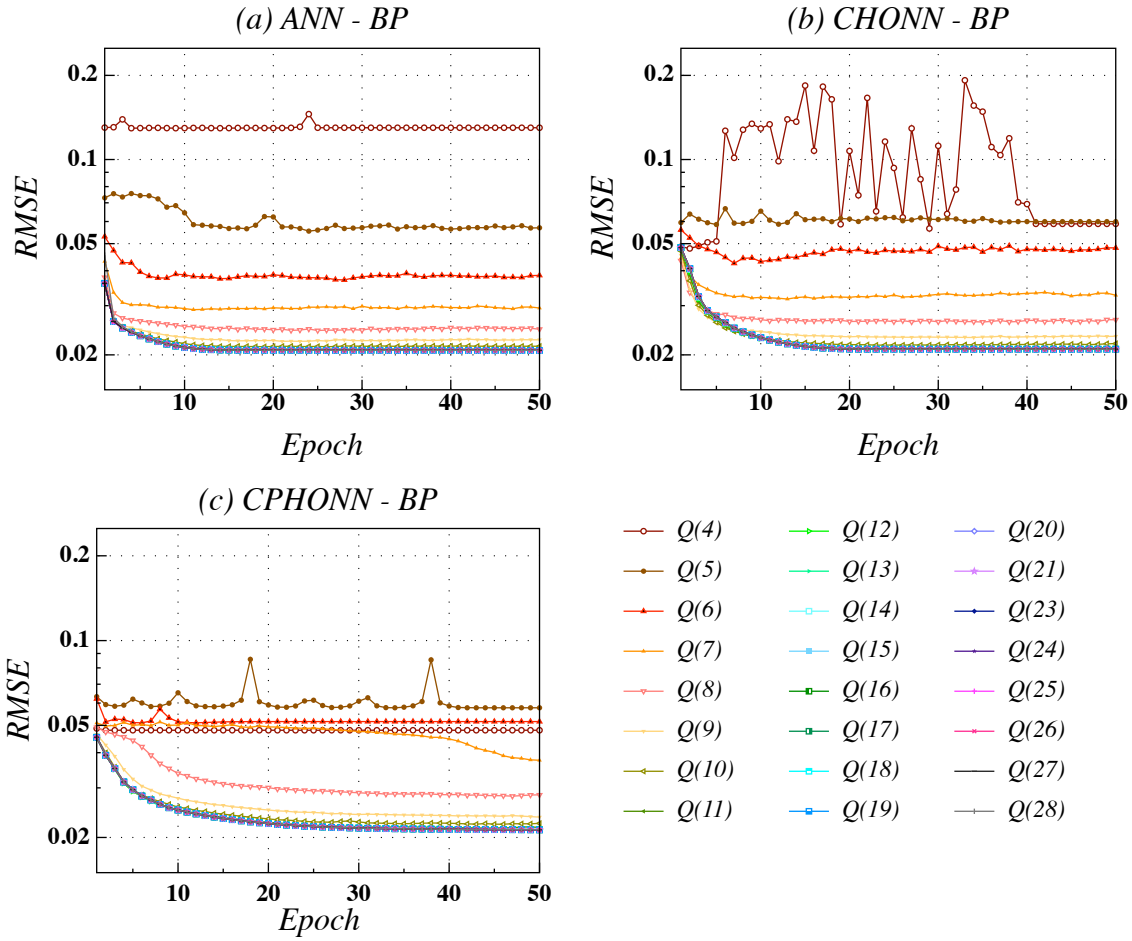

Figure 6.6: BP Training Error at several levels of precision, Q for estimating optical waveguide sidewall roughness

Figure 6.6 shows that all of the algorithms reach the optimum performance after 30 epochs. ANN-BP Figure 6.6 (a) convergences faster than the C-HONN and CP-HONN in Figure 6.6 (b) and (c) respectively. HONN operations require higher precision in order to achieve similar performance levels to the ANN, see Table 6.5. At epoch 10, the ANN is best for lowest $RMSE$ then C-HONN with the CP-HONN having the worst error.

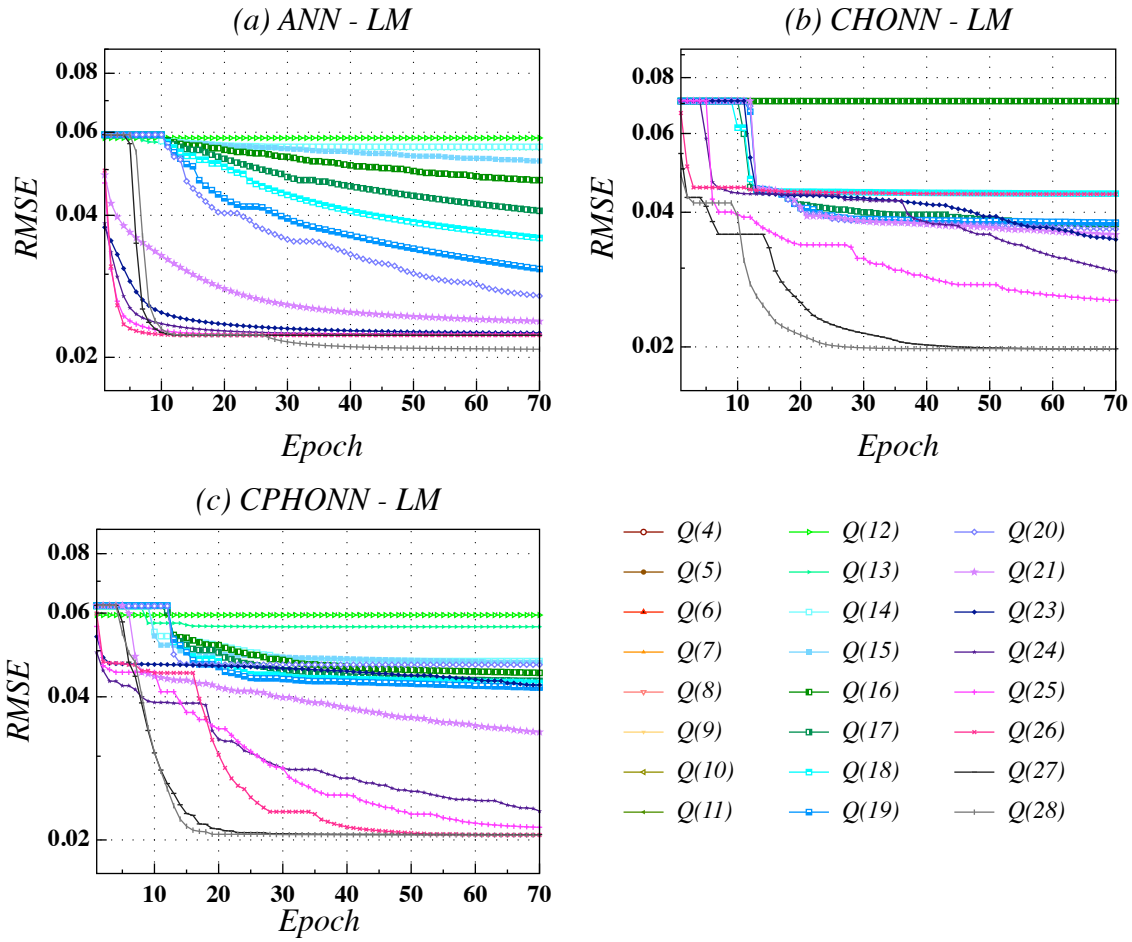

Figure 6.7: LM Training Error for several precisions, Q for estimating optical waveguide sidewall roughness

Figure 6.7 shows the LM training convergence curves for ANN in Figure 6.7 (a), C-HONN in Figure 6.7 (b), and CP-HONN in Figure 6.7 (c). After 10 epochs the ANN model had the fastest convergence compared to HONNs. The minimum range indicated in bold font in Table 3 leads to loss in fraction size due to higher range of high order interactions which in turn required higher precision to reach the optimum performance, these interactions led to information loss after Multiply and Accumulate (MAC) operations.

C-HONN and CP-HONN models showed better performance in floating point.Comparing graphs Figure 6.7 (b) and (c) the convergence of the C-HONN is slower for the lowest RMSE than that of the CP-HONN, due to the facts mentioned in the previous paragraph. As mentioned in section Fixed-point number representation the multiplication operation doubles the required precision and the range bits required an increase with every addition operation required by the C-HONN.

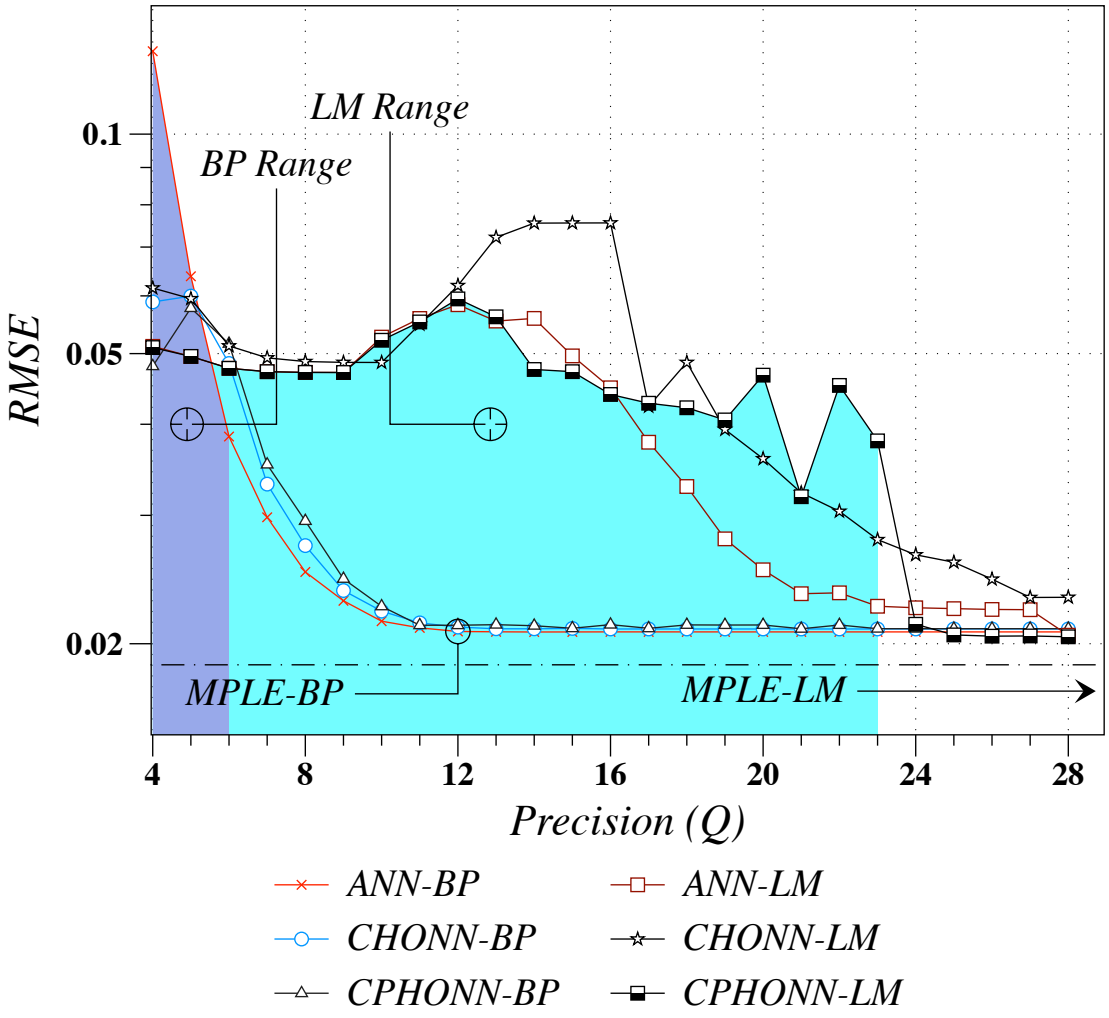

Figure 6.8: Output error after 70 epochs of BP and LM Training for several levels of precision for estimating optical waveguide sidewall roughness

Figure 6.8 shows the performance of all networks for both learning algorithms as a function of precision level for the waveguide sidewall roughness estimation. Figure 6.8 has two shaded regions that indicate the minimum required ranges as in Table 6.5. As in Figure 6.7, the effect of the HONN function leads to a deterioration of the estimating

models for both learning algorithms less than a certain precision. The deviation between the HONNs and ANN performance almost disappears for precisions higher than 11 bits for BP training whereas the discrepancies increased for LM training as the number of MAC operations required by the algorithm increased. In the case of BP learning for more than 12 bits of precision the RMSE does not change so, $MPLE - BP = 12bits$. The LM algorithm starts to converge to an optimal solution after 23 bit precision, having the best performance at 28 bits, so $MPLE - LM > 28bits$. The dashed-and-dotted line in Figure 6.8 shows the performance in floating point $RMSE = 0.0194$. The fixed-point RMSE reaches a minimum of 0.0204 when training the CP-HONN using LM. The BP algorithm converges to a value that is always 0.0015 higher than the floating-point $RMSE$.

6.7 Discussion and Conclusions

Simulations were conducted to find the effect of a limited precision environment on learning algorithms used to train ANNs and HONNs. It was found that the learning algorithms required less precision when training on discrete data due to the limited levels this data takes and they required higher precision when dealing with almost-continuous functions. The BP algorithm reaches the Minimum Precision for Lowest Error ($MPLE$) at 12 bits for both the discrete and continuous functions under consideration, being the XOR function and optical waveguide sidewall roughness. The LM algorithm provided a significant enhancement to the discrete function modelling without requiring many range bits due to the discrete nature of the MAC operation it requires; however, the LM training algorithm required a significant increase in the precision in order to accommodate for the expansion of MAC operations within the learning algorithm. The MPLE for the LM algorithm was not established as greater precision levels greater than 28 are needed for it to reach its error floor. The minimum precision for minimum error was required to be 24 bits so only the 4 highest precisions could be studied. The results of this study expand and support the findings of Piche (1995); Stevenson et al. (1990) where it was shown that discrete functions require less precision than continuous functions during network operation and indicate a precision level MPLE beyond which no performance gain is achieved. This measure allows the hardware designer to make an efficient design in the least area possible enabling the hardware to reach the highest operational frequency at lowest power. The study was made possible by using advanced high level languages such as Matlab ver.7.4 and Xilinx ISE ver.10 giving fast development time and model exploration with a high

level of abstraction in algorithmic blocks for circuit and signal track operations on digital hardware, Bhatt and McCain (2005); Xilinx (2008a); Zarrinkoub (2006).

Chapter 7

Levenberg-Marquardt algorithm implementation on Field Programmable Gate Arrays

7.1 Introduction

Artificial intelligence and machine learning are used to computationally solve complex problems. Some of these problems are solved statistically by using optimization algorithms that find the parameters which give the most accurate solution. A widely used algorithm is the Levenberg-Marquardt Algorithm (LM), Marquardt (1963). The LM-algorithm is used extensively for problems dealing with non-linear function and parameter approximation, Hamouda et al. (2011); Wilamowski et al. (2008), such as automated optical inspection systems Reed and Hutchinson (1996), which can use a single object observation from a single camera, Cho and Chow (2000), or systems dealing with projective geometry estimation Segvic and Ribaric (2001). The algorithm investigated can also be used for motion compensation systems Alam and Bal (2007); Lin et al. (2009), and robotic vision systems Ashrafiuon et al. (2008); Cho et al. (2009); Motai and Kosaka (2008); Ogawa et al. (2007); Xie et al. (2009).

To demonstrate non-linear parameter estimation we used Neural Network parameter estimation and Camera Calibration as practical examples. If we consider camera parameter estimation for an automated optical inspection system, the system will be able not only to tell that a fault occurred in a production line but also show how such a fault occurs and what caused the fault based on the value of the parameters estimated by the learning algorithm. It should be noted that this study is not limited to Neural Networks

or camera calibration it can integrated into a wider range of applications depending on non-linear parameter estimation.

Replicating advanced learning algorithms on dedicated hardware is a challenging problem. In the field of Neural Networks, and non-linear parameter estimation, most studies are limited to the popular back-propagation algorithm and offline full precision training performed in a software environment Zhu and Sutton (2003a). This study builds on previous work which investigated the LM-algorithm in reduced precision, Shawash and Selviah (2010).

The chapter is organised as follows. Section 7.2 provides a brief background. Section 7.3 presents details of the two experiments conducted in this paper. Section 7.4 provides the results of these experiments. Ending with conclusions in section 7.5.

7.2 LM algorithm modelling

The Levenberg-Marquardt algorithm (LM) finds a solution of a system of non-linear equations, $y = \phi x$, by finding the parameters, ϕ, that link dependent variables, y, to independent variables, x, by minimizing an error of a function of said system by using error gradient information for every parameter considered in the system. The LM-algorithm in (7.1), estimates the parameters that make up a specific system function recursively until convergence by finding the appropriate change, $\Delta\phi$, leading to smaller errors. The LM-algorithm depends on error, E, the Hessian matrix H, the gradient of the error, ∇J, a scalar μ which controls the trust region, and I is the identity matrix.

$$\begin{aligned} H &= J \times J \\ \nabla J &= J \times E \\ \Delta\phi &= -\left(H + I\mu\right)^{-1} \nabla J \end{aligned} \tag{7.1}$$

Figure 7.1 shows a generic supervised learning procedure, in this case adapted to the LM-algorithm. The upper part in the diagram is the operation phase of a function, and the lower part is the phase which adjusts the parameters in the operational phase. The upper part of the diagram shows the operation of a function depending on certain weights and parameters. The function operation tends to start at specified initial starting weights which are adjusted by a learning algorithm to reach the desired state of operation with the lowest errors. The lower part in the diagram represents the LM-algorithm training algorithm. The arrows indicate the flow of signals and parameters to and from the operation and the training sections of the learning system. The operation phase tends to be less complex than the learning phase, so some studies managed to port simple learning

algorithms onto a Field Programmable Gate Array (FPGA). Porting learning algorithms onto an FPGA tends to be challenging, especially when it is related to on-line training Neural Networks (NN) on the FPGA, Zhu and Sutton (2003a). Usually, the processing is ported to hardware and the training remains in software or in some cases the error back propagation algorithm is performed on hardware Sahin et al. (2006). This study, for the first time to our knowledge, shows the implementation of the more complex LM-algorithm in hardware. In Figure 7.1 we propose to keep the high speed operation in software as its hardware counterpart has been comprehensively studied in literature. The hardware implementation of a complex 2nd order learning algorithm such as the LM-algorithm has not been extensively researched. Also, the amount of speed up gained from speeding up the learning phase on FPGA far exceeds the already fast implementation of the operation phase in software.

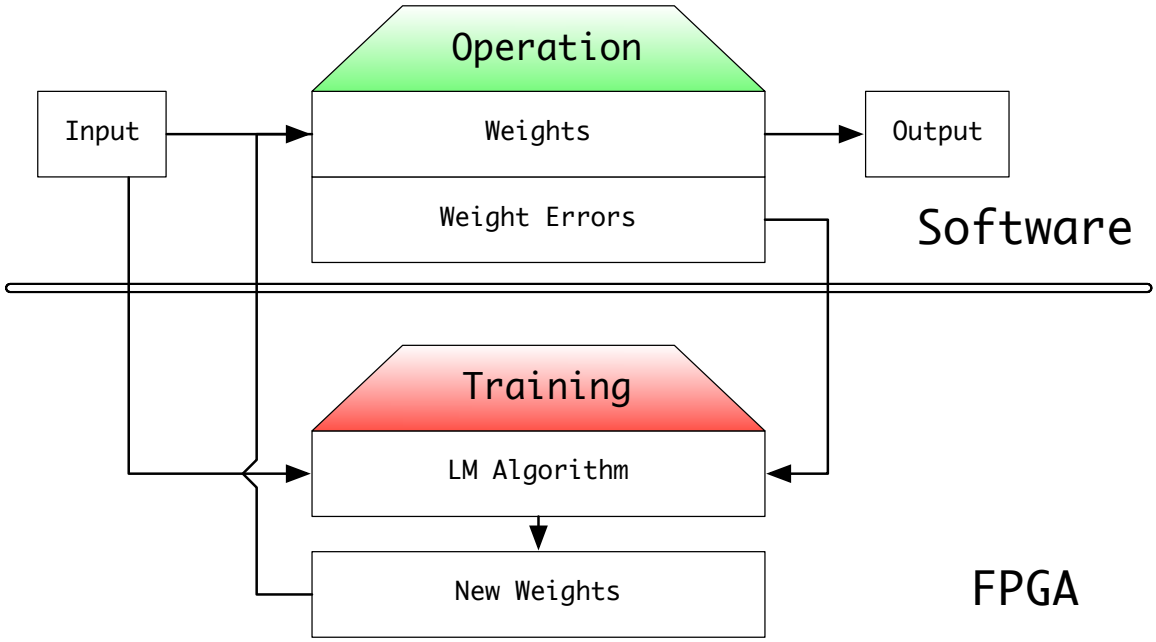

Figure 7.1: Diagram of proposed Levenberg-Marquardt-algorithm partitioning between Hardware (FPGA) and Software (CPU)

Equation (7.1) summarises the algebraic operation required to perform the LM-algorithm. To generate the hardware configuration to be implemented on the FPGA the algebraic operations were deconstructed to simpler parts. Even though the LM-algorithm solves non-linear functions its main operation resembles a solver of linear systems. In this study we use the Xilinx AccelDSP software package to construct and integrate three separate operations that comprise this solver, see Figure 7.2. In order to build this system for FPGA we use cores that make up the algebraic operations of the LM-algorithm.

Figure 7.2 provides a diagram composed of three cores that are implemented on FPGA. The three cores that comprise the LM-algorithm can be broken down into a QR factorisation, `QR`, a matrix multiplication, `mtimes`, and an upper triangular system solver, `triangSolver`. It was decided to split the LM-algorithm into two separate parts to allow for faster debug, core generation time reduction, and due to the limited resources on an FPGA. Both parts can be integrated into other projects requiring real-time non-linear parameter estimation. The first part contains the first two cores that factorize the Hessian matrix and multiply the Q matrix by the gradient of the Jacobian, ∇J. The second part solves the system using a back-substitution algorithm.

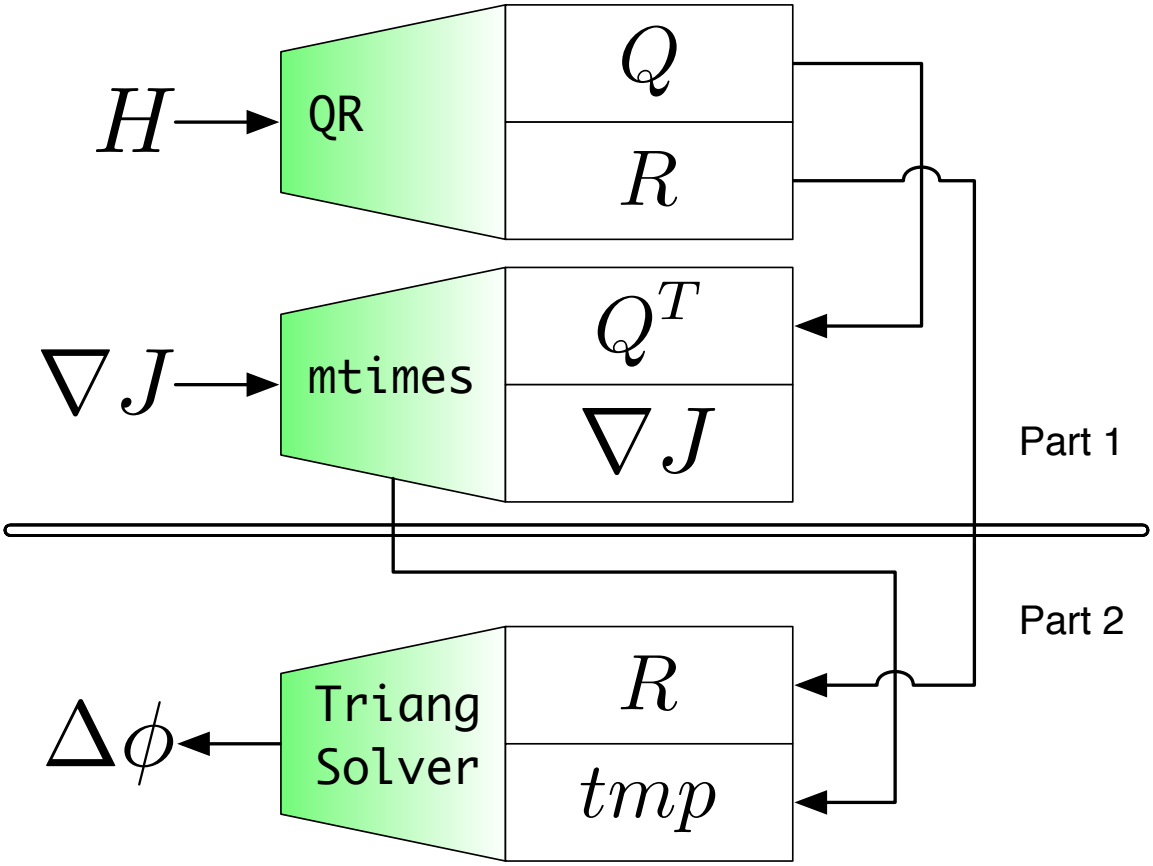

Figure 7.2: Levenberg-Marquardt-algorithm on the FPGA

We used the orthogonal, or QR, factorisation because it can deconstruct any rectangular matrix, A, to a product of a unitary matrix and an upper triangular matrix, $A = QR$, where Q is the orthogonal or unitary, and R is the upper triangular matrix. Unitary matrices are desirable for numerical computation because they preserve length, angles, and do not magnify errors, Gentle (2009). QR factorisation was chosen over LU factorisation because the QR factorisation does not require any pivoting or permutations

without resorting to division operations. The error gradient is then multiplied by the unitary matrix Q, ensuring that no increase in range occurs due to these multiplications since the maximum value Q can take is 1. After performing the factorisation and multiplication, the results of those two cores are routed to a triangular system solver which uses the back substitution method to find the solution $\Delta\phi$.

7.3 Experiment

Two experiments were conducted for a real-time LM-algorithm implementation on the FPGA. The first experiment used the FPGA-LM-algorithm to train a neural network to solve the XOR problem. The second experiment made use of the FPGA-LM-algorithm for explicit camera calibration.

In the following subsections we present how we implemented the LM-algorithm on the FPGA. To our knowledge, this is the first time this type of work has been done.

The design and simulation of the experiments required several tools and development environments. Matlab 2009b, Xilinx ISE Xilinx (2008a) with the AccelDSP development environment were used for software and hardware prototyping. An FPGA was used as a hardware-in-the-loop (HIL) system connected to the computer by ethernet; several studies used similar development environments Bhatt and McCain (2005); Ou and Prasanna (2005); Rosado-Munoz et al. (2009). The Neural Network toolbox in Matlab 2009b was used to simulate the XOR training in floating point as a benchmark. The real-time operation of the FPGA-LM-algorithm design was loaded onto the FPGA and was invoked by the Neural Network toolbox as a custom training function. The non-linear parameter estimation module in the Camera Calibration toolbox for Matlab, Bouguet (2008), was replaced with the real-time FPGA based LMA to perform the main optimisation operation in the camera calibration algorithm. A Xilinx Vertix-5 ML506 development board, an Intel Core2Duo 3.0 GHz processor, 4 GB RAM and Windows XP operating system was used to compile and connect the learning module on the FPGA.

It should be noted that AccelDSP limits the QR factorisation and the triangular system solvers specification to have a maximum word length of 24-bits and matrices dimensions to 32 units wide. Another limitation of the FPGA is the maximum speed of any design operating in a Hardware-in-the-loop configuration, all the designs were limited to a maximum of 100 MHz. This was found to be sufficient as the current system design is intended to be a proof of concept of real-time operating non-linear estimators to be used within stand-alone hardware systems requiring very high speed and low power solution

where the need of a real-time solution far outweighs the costs of having an integrated FPGA.

7.3.1 Exclusive OR (XOR)

The XOR estimation using a neural network is considered as a benchmark operation for various training techniques. The XOR is a function of two variables whose interaction forms a system that is linearly inseparable; meaning that it can not be solved using a linear system, see Figure 7.3.

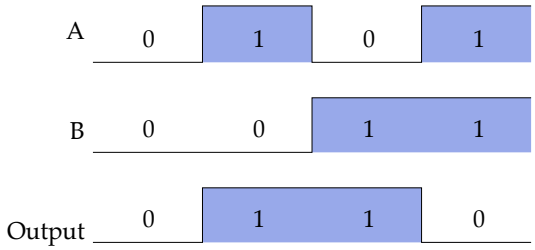

Figure 7.3: Exclusive OR function

To simulate NN training of the XOR function, two input signals with 32 binary samples were used to estimate the magnitude of the signals traversing through the NN. The data was divided into three segments; with 24 samples for training (75%), 4 samples (15%) for validation leaving 4 samples (15%) for testing. Neural Networks neuron parameter may have both weights and biases. The NN constructed had 2 layers, 4 weights linking the input-to-hidden layer with no biases and 2 weights linking the hidden-to-output layer with a 1 bias at the output neuron, see Fig. 7.4. The biases from the hidden layer were removed as the NN reached the same solution without them. This type of structure comprises a NN with 7 parameters. For repeatability the NNs were initialised to zero starting values.

In order to find the appropriate ranges for the variables used in the FPGA-LM-algorithm a floating point simulation of XOR-NN training was used to collate the variables and find the appropriate ranges, Zarrinkoub (2006). The input variables were quantised to fit the fixed-point format of Q(24,19); comprising a signed number with a range of 2^4 and 2^{-19} resolution. The output of the XOR-NN was set to Q(48,38), a precision generated using the AccelDSP workflow.

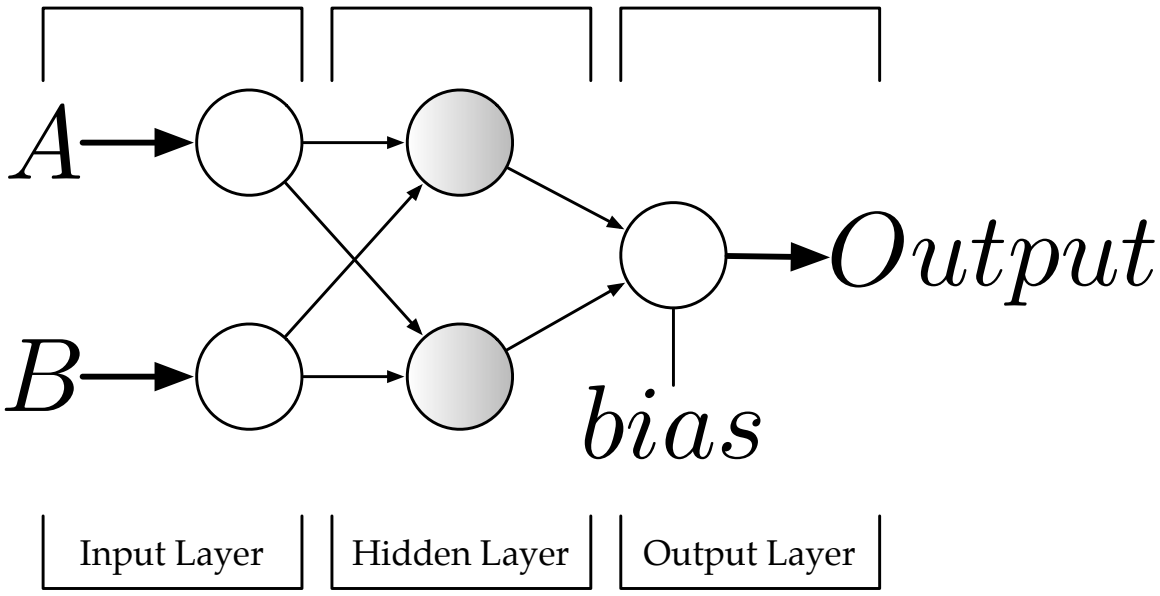

Figure 7.4: Neural Network for solving XOR

7.3.2 Camera Calibration

Camera Calibration is a process of finding the non-linear parameters which map World Coordinate System (WCS) in 3D to Camera Coordinate System (CCS) in 2D. The camera calibration operation and optimisation has a similar setting as neural networks when it comes to the optimisation to find the camera system parameters with studies performing the operation phase on hardware and the learning phase in software **??**. Camera calibration adopts an established analytical framework that uses the LM-algorithm for the estimation of both the linear and non-linear distortions of the camera system. We make use of the well known camera calibration toolbox used in both Matlab and OpenCV, Bouguet (2008). The calibration test data is found in Zhang (2000). Five planes and their reference grid were used for calibration, see Fig. 5.2. The FPGA-LM-algorithm was integrated into the toolbox by integrating it into the main optimisation routine.

The LM-algorithm is used to find the parameters mapping the reference 3D coordinates into the 2D camera coordinates. This type of mapping uses a combination of a linear and a non-linear system of equations. The linear system is used for the estimation of linear camera parameters which describe the rotation, translation and scaling vectors as in (7.2), where $P = [X, Y, Z]$ is a 3D reference point in world coordinate system (WCS), $\hat{p} = [x, y]$ is the 2D target point in the camera coordinate system (CCS). WCS points are mapped linearly to 2D points by using rotation matrix R, a translation matrix T, a focal point estimate f and centre point estimate c.

$$\begin{aligned} \hat{P} &= R \times P + T \\ \hat{p} &= f \times \hat{P} + c \end{aligned} \tag{7.2}$$

The non-linear effects of the camera system are estimated using equations dealing with radial and tangential distortions as in (7.3), where $\rho_{(1,2,5)}$ specify the extent of the radial distortion term $r^2 = x^2 + y^2$, and the tangential distortion is specified by $\rho_{(3,4)}$, where $a_1 = 2xy$, $a_2 = r^2 + 2x^2$, $a_3 = r^2 + 2y^2$.

$$\begin{aligned} \hat{x}_{new} &= [x(1 + \rho_1 r^2 + \rho_2 r^4 + \rho_5 r^6)] + [\rho_3 a_1 + \rho_4 a_2] \\ \hat{y}_{new} &= [y(1 + \rho_1 r^2 + \rho_2 r^4 + \rho_5 r^6)] + [\rho_3 a_3 + \rho_4 a_1] \end{aligned} \tag{7.3}$$

The number of unique parameters in both equations amount to a vector with a total of 12 parameters. It should be noted that the camera calibration toolbox for Matlab uses the data from all the images to construct a single vector containing all the parameters derived from all of the images. However, hardware development tools limit the size of the FPGA-LM-algorithm leading to image mapping of each image on an individual basis.

7.4 Results

7.4.1 XOR

Table 7.1 shows the specifications and the number of components used by the two parts of the FPGA-LM-algorithm, this summary of components excludes the logic required to time the internal functioning of the core, however, it is correlated with the number of cycles required per function call. It can be noted that only a few multipliers and adders were used, the clock cycles per function call were high. This is due to the process in which all cores were generated using the default settings in AccelDSP; mainly that all vector operations were rolled into loops and not performed in parallel. The loop-rolling process decreases the amount of multipliers and addition operations by using local memories which are triggered to supply intermediate values during algebraic operations. Loop unrolling reduces the clock cycles needed to perform the required operation, however, for the current study there was no need to produce cores with lower latency and higher resource usage as the FPGA used is large enough to accommodate for the XOR FPGA-LM-algorithm implementation. Both cores took up to 71% of the resources available in this FPGA.

Table 7.1: XOR FPGA-LM algorithm hardware utilisation summary

	Part 1	Part2
Parameter	Value	
Frequency (MHz)	100	100
Startup Clock Cycle	2011	452
Clock Cycles Per Function Call	2010	451
Multipliers	27	6
Adders	39	11
Subtractors	42	16
Number of Slice Registers	23375 out of 32640	
	71%	

After generating the cores, the FPGA-LM-algorithm was integrated into the Matlab Neural Network toolbox by designing a custom training function replacing the internal software LM-algorithm. The results of the software LM-algorithm and FPGA-LM-algorithm can be seen in Fig. 7.5. The upper graph in this figure shows that the software solution convergence stopped after only 9 iterations; if the back-propagation algorithm was used we can expect the optimization to require tens of thousands of iterations. The LM-algorithm on software reached the optimisation stopping criteria by reaching a level lower than the minimum gradient. The lower graph in Fig. 7.5 shows the convergence of the hardware trained NN. The solution converged in 13 iterations, incurring only 4 more training iterations than software. The FPGA-LM-algorithm optimisation ended when the maximum allowable μ value was reached. It can be assumed that the NN converged to the correct solution in both hardware and software as they both reached error levels lower than 10^{-6}.

Table 7.2 provides a summary of the software and hardware optimisation results. The summary indicates the number of epochs taken and the mean squared error (MSE) of the solution. We can note that the hardware solution had a slower rate of convergence after the 6th epoch attributed to the lower precision nature of the learning system on hardware.

In order to compare hardware and software performance, we compare the time it takes to perform the main LM-algorithm function in both cases. We assume that there are no signal routing delays or memory transfer delays to and from the FPGA chip. The only functions timed are the ones concerned with the LM-algorithm parameter update. The floating-point software implementations were timed to to be the average of a 100 runs, this

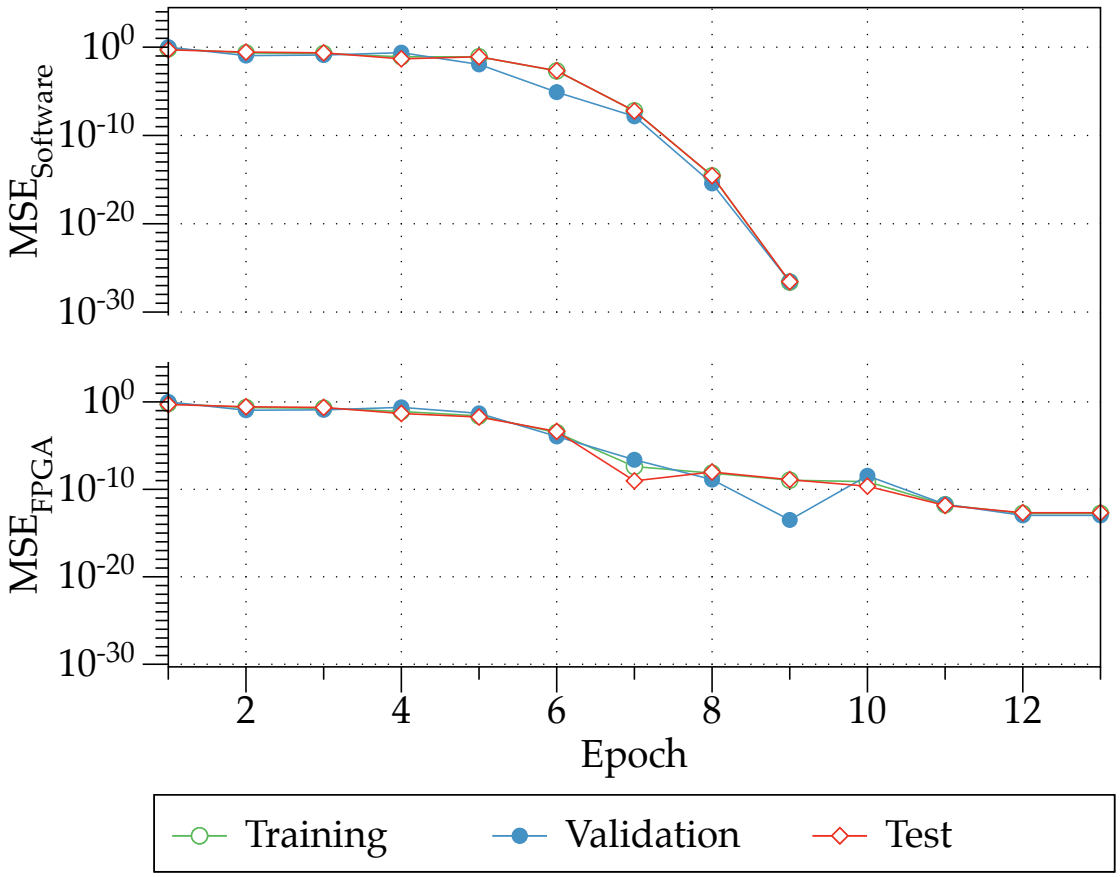

Figure 7.5: XOR LM algorithm training, validation and test performance trace in software and FPGA

Table 7.2: XOR LM training performance summary

Parameter	Epochs	MSE
Software	9	2.45×10^{-27}
FPGA	13	1.73×10^{-13}

way we accommodate for outlier events slowing down the algorithm calculation. Also, the floating-point operation in Matlab uses the BLAS library within the interpreted language of Matlab. It should also be noted that the LM implementation has been broken down to elementary multiply and matrix inversion operations. All function nesting and memory copying operation have been avoided. The 9 epochs in the software implementation of the XOR LM-algorithm took 0.3982 s. The FPGA LM-algorithm required 13 iterations at 100 MHz, taking up to 13 μs, accelerating the learning algorithm by a factor of 3.0633×10^6.

For reference, a recent study Shoushan et al. (2010) which implemented the training of a Neural Network using the popular back-propagation that solves the XOR problem in 760 iterations reaching an error levels 10 orders of magnitude higher than training performed in the LM-Algorithm.

7.4.2 Camera Calibration

Table 7.3 provides a summary of hardware utilisation required by the FPGA-LM-algorithm suitable to calculate the required 12 parameters for the WCS-to-CCS (3D-to-2D) mapping. It should be noted that part 1 of the LM-algorithm was generated and simulated in Fixed-Point on the computer due to the fact that the routing of the algorithm requires more resources than the one available on the Virtex-5 SX50T available for this study. The amount of logic required to time the design function and make it cycle true is correlated with the amount of clock cycles per function call, so the routing delay prohibited the generation of an FPGA netlist which fits on the FPGA under consideration. Several vector operations were unrolled to reduce the amount of timing, however, they did not reduce the amount of logic required in order to fit on the FPGA chip. A simulation of HDL code comprising the operation of Part1 shows that it introduced an error of a magnitude of only 10^{-5}. So for this experiment only the 12x12 back substitution system (part 2) was operating in real-time on the FPGA hardware taking only 7,273 slices from the total of 32,640 slices available on the Virtex5 SX50T. In order to fit the whole solution onto a single chip, we need to use either a newer Virtex-6 chip or an LX85 Virtex5 FPGA.

Figure 7.6 shows the parameter estimation convergence rate for the 12 parameters both in software, in the top graph, and through the FPGA, on the lower graph. The parameters initial and end values vary in their range extensively, this type of normalisation is used for the purpose of illustrating the convergence of the software and FPGA solutions of the LM algorithm. The graph displays the convergence rate after collating the parameters during the optimisation process and normalising them to a range of $[0, 1]$ and taking the absolute values. The software convergence is smoother than the FPGA and reaches a lower error level. From the FPGA system convergence figure we notice that the parameter estimation

Table 7.3: Camera Calibration FPGA-LM algorithm hardware utilisation summary

	Part 1	Part 2
Parameter	Value	
Frequency (MHz)	100	100
Startup Clock Cycle	6484	818
Clock Cycles Per Function Call	6483	817
Multipliers	19	6
Adders	24	11
Subtractors	24	16
Number of Slice Registers	62906 out of 32640	
for the complete solution	**192%**	
Number of Slice Registers	7273 out of 32640	
for the current solution	**22%**	

was not as smooth and that some of the parameters did not reach a lower level of error after the 16th iteration. The camera calibration optimisation system has a maximum of 30 iterations.

Figure 7.7 shows a comparison of the error scatter from the floating point optimisation on the left and the fixed-point FPGA optimisation on the right. The figure shows that the error resulting from the FPGA solution is almost similar to the software solution. Table 7.4 shows the summary of camera calibration in terms of the number of iterations and the Mean Squared Error (MSE) and standard deviation (σ) of the mapping error for the 5 images in their original format and also when the images and the reference grid are rescaled to [0,1]. When mapping the raw images, we notice that the floating LM-algorithm ends in 20 iterations because the criteria of lowest change in the norm of parameters reached a level below 10^{-9}. In all of the cases and data considered the FPGA-LM-algorithm terminated when the maximum number of iterations was reached. The FPGA-LM-algorithm had satisfactory performance in terms of error and standard deviation when compared to software. When the data was in its original scale the FPGA-LM-algorithm MSE performance increased by 173.5% on average, the standard deviation of the FPGA solution was also 60.10% larger than the software solution. The lower part of Table 7.4 shows the performance of the camera calibration mapping the rescaled data. All images and their reference grid were mapped to be in the range [0,1]. This type of rescaling process was first applied in this study to try to reduce the ranges of the resultant Hessian matrix, the error gradient matrices and also to reduce the condition

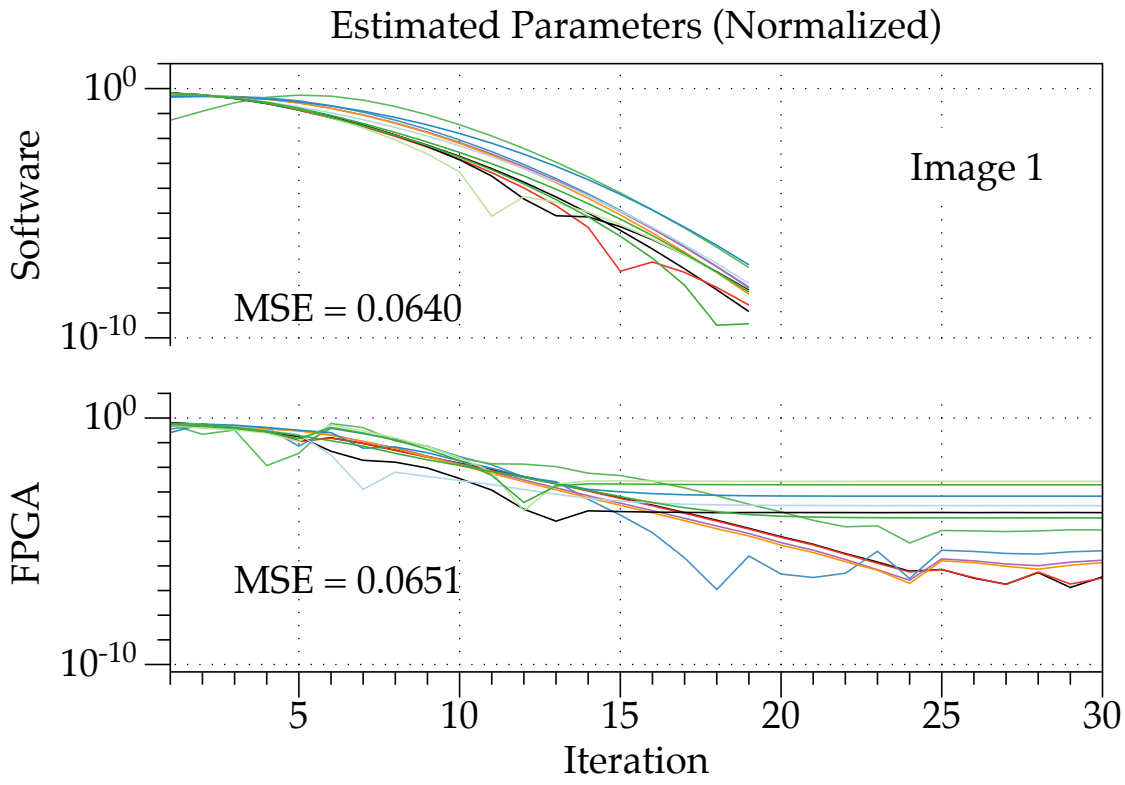

Figure 7.6: Camera LM algorithm parameter convergence for image 1 in software and FPGA

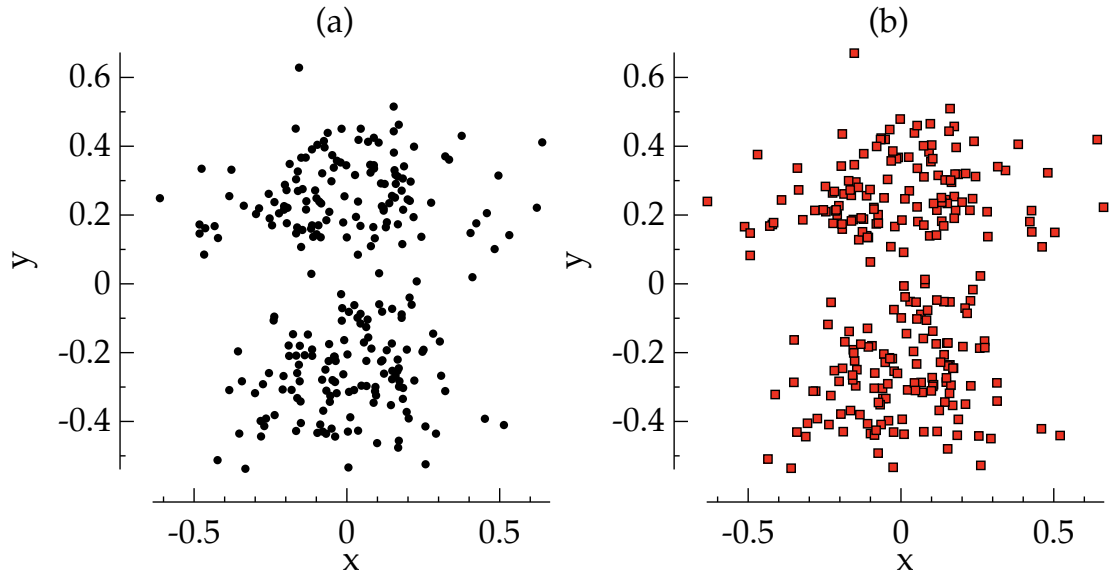

Figure 7.7: Calibration error for mapping reference grid to image 1 when both are rescaled to $[0, 1]$ in (a) Software. (b) FPGA.

Table 7.4: Camera Calibration LM algorithm training summary

		Iterations		$MSE(\sigma)$		
	image	Software	FPGA	Software	FPGA	% $MSE(\sigma)$ increase
Raw	1	19	30	0.0601 (0.2454)	0.0665 (0.2581)	10.65 (5.18)
Data	2	20	30	0.0254 (0.1595)	0.1165 (0.3417)	358.66 (114.23)
	3	20	30	0.1449 (0.3810)	0.2311 (0.4811)	59.49 (26.27)
	4	20	30	0.0280 (0.1676)	0.1109 (0.3334)	296.07 (98.93)
	5	20	30	0.0209 (0.1446)	0.0507 (0.2254)	142.58 (55.88)
					Average:	173.49 (60.10)
Data	1	19	30	0.0640 (0.2533)	0.0651 (0.2554)	1.72 (0.83)
scaled	2	30	30	0.1577 (0.3975)	0.1540 (0.3928)	-2.35 (-1.18)
to	3	—	—	—	—	—
$[0, 1]$	4	30	30	0.2438 (0.4943)	0.3899 (0.6251)	59.93 (26.46)
	5	21	30	0.0217 (0.1474)	0.0244* (0.1563)*	12.44 (6.04)
					Average:	17.94 (8.04)

The asterisk '*' indicates that the initial iteration of the optimisation algorithm was performed in software.

number of the Hessian matrix. The condition number of the matrix has a significant effect when dealing with quantised data which on occasion leads to singularities that do not occur when working in full precision. The rescaling reduced the initial condition number, however, the Hessian matrices that occur later in the algorithm are more difficult to solve even on a floating point precision, this can be seen in the worse performance of the software LM-algorithm. Image 3 had an ill-conditioned Hessian matrix that could not be solved on neither software nor hardware. Image 5 was solved on both platforms, however, the asterisk indicates that the first iteration of the FPGA-LM-algorithm was solved on floating point with subsequent iterations on hardware, a situation which would be preferable in real-world applications where the initial conditions are obtained using the highest precision available, and the adaptive system will only need to account for small changes that a system undergoes after calibration. Even though the rescaling rendered some of the mapping examples inoperable, the average increase in the hardware solution MSE was lowered to 17.94% more than the software solution while the standard deviation of the error remained almost the same only increasing by 8.04%.

The software implementation for the calibration of image 1 in software LM-algorithm took 0.4228 s. The FPGA LM-algorithm required 30 iterations at 100 MHz, taking up to 30 μs, resulting in an operation almost 1.41×10^6 faster than software.

7.5 Conclusions

The Levenberg-Marquardt Algorithm was programmed and used successfully in real-time on an FPGA, for the first time to our knowledge. It was found that the hardware implementation of the LM-algorithm was able to find Neural Network parameters to solve the XOR problem in just 13 iterations compared to existing studies of hardware learning using the back propagation algorithm. Comparing the optimised LM implementation in Matlab to the FPGA implementation the NN training speed was increased by a factor of 3×10^6. The same algorithm was also used to find the optimal mapping parameters in the camera calibration problem having a penalty of only 17.94% increase in the MSE and an 8.04% increase in the standard deviation when compared to a software solution. The FPGA-LM-algorithm for camera calibration speed was increased by a factor of 1.4×10^6. We were able to get these results by using high level software and hardware development environments. However, the limitations of the development environment restricted the variables to 24 bits of precision, and the FPGA hardware area restricted the size of the system that needs to be solved.

The FPGA hardware used in this study proved to be a restriction for portions of the algorithms used. By using a newer FPGA with more resources all the algorithms can be ported to hardware including the operation phase of the function estimation allowing for the elimination of some of the restrictions posed by the current deployed solution. A solution implemented fully in hardware would allow for a reduction in the latency of the solution and can remove the dependency on software.

This study provides a roadmap for the implementation of the LM-algorithm on fixed-point hardware which can be ported onto a dedicated ASIC for high speed low power applications. In most cases the requirement for a real-time low powered solution outweigh the costs of including an FPGA. To this extent the solution demonstrated here can be readily integrated into a robotics vision and control or in an optical inspection system. To operate viably, both of these systems ideally require a real-time parameter estimation module.

Chapter 8

Conclusions

8.1 Higher Order Neural Networks in Finance

The investigation of new Higher Order Neural Networks (HONNs) on financial data had the following results. The performance of the HONN models was superior to other types of network or linear models. The new C-HONN provided the best performance for forecasting the daily returns series of FTSE100 and NASDAQ. The FXP-HONN had the best hit rate prediction for the NASDAQ time series at 59.5% and 60.62% was achieved for the FTSE100 using a first order NN. We conclude that models that assume the data has a normal Gaussian distribution with constant volatility fail to capture all of the information available within the data as reflected in the AIC, $RMSE$ values and correlation tests. Even the best performing C-HONN did not capture the conditional volatility in the residual of the returns. This information is captured using models that take into account conditional volatility, such as GARCH and EGARCH. HONNs forecasting returns were combined with the GARCH and EGARCH models, for the first time, to give a hybrid model. It was observed that the hybrid model reduced the $RMSE$ error by 10.25% for the FTSE100 and by 4.1% for the NASDAQ datasets when compared to linear regression. The best performing model was the hybrid C-HONN-EGARCH combination model for the data sets considered, in summary:

1. A new type of HONN called the Correlation Higher Order Neural network was combined with a volatility model to estimate returns and volatility of time series.

2. This hybrid model reduced the $RMSE$ of the returns and volatility by 10.25% for the FTSE100 and by 4.1% for the NASDAQ datasets when compared to linear regression.

3. Ordinary Higher Order Neural Networks (FXP-HONN) provided the best Hit Rate estimates.

8.2 Higher Order Neural Networks for Camera Mapping

The new work comparing the mapping performance of a new type of Higher Order Neural Network, the Generalised Correlation Higher Order Neural Network, against standard analytical methods was tested in three situations; 3D-to-2D mapping, a hybrid model combining HO/NNs with an analytical model, and the performance of HONNs compared to NNs performing 2D-to-3D mapping. The study considered 2 types of data, multiple views of plane data, and a cube data comprising two planes on either side of a cube. The results indicate that HONN calibration outperform both the standard analytical parametric model used by the Camera Calibration Toolbox (CCT) in all cases and the non-parametric model based on ordinary Neural Networks (NN) in some of the cases. HONN 3D-to-2D mapping reduced the error by more than 14% on average with the the Horiz-HONN having 24% lower error than the parametric method for plane data. 3D-to-2D cube mapping had more drastic reduction in the error when using HONNs, with reduction of mapping error of up to 43% in the cases of FXP and CP-HONNs.

The new hybrid models combining analytical solution for linear parameters combined with a HO/NN based non-parametric model to account for non-linear distortions out performed the parametric modelling performed by the CCT. However, it did not provide a discernible mapping improvement in the case of plane data and it had a worse performance when mapping cube data. Inverse mapping from 2D-to-3D using HONNs did not provide discernible improvement when compared to the NN model. However, in the case of cube data the FXP and CP-HONNs provided 47.2% and 44.2% drop in error respectively. This can be attributed to the more complex nature of mapping non-planar data using HONNs, and the availability of non-degenerate information for the z-dimension. HONNs provide mapping improvements compared to the parametric model since it aims to eliminate the systematic error without being limited to a fixed analytical model. HONNs also outperform NNs as they Higher Order functions reformulate the input space into one that reflects the higher order interactions of the input data passing through a camera system. The simulations were limited to only two sets of data. Benchmarking HONN mapping capabilities against the standard model and the NN model presents a strong case for further research.

1. This is the first work comparing HONNs to NNs and parametric camera calibration models in terms of calibration and mapping error reduction.

2. HONNs were found to have a better performance for co-planar data than planar data when compared to NNs.

3. A new hybrid model combining intrinsic camera parameter with HONN modelling of camera data outperformed a fully parametric camera calibration model.

4. Hybrid camera calibration did not outperform a fully non-parametric models.

8.3 Learning in Limited Precision

The new work carried out on finding and simulating the effects of a limited precision environment on learning algorithms to train ANNs and HONNs found that the learning algorithms required less precision when training on discrete data due to the limited levels this data takes and they required higher precision when dealing with almost-continuous functions. The BP algorithm reaches the Minimum Precision for Lowest Error ($MPLE$) at 12 bits for both the discrete and continuous functions under consideration, being the XOR function and optical waveguide sidewall roughness. The LM algorithm provided a significant enhancement to the discrete function modelling without requiring many range bits due to the discrete nature of the MAC operation it requires; however, the LM training algorithm required a significant increase in the precision in order to accommodate for the expansion of MAC operations within the learning algorithm. The MPLE for the LM algorithm was not established as greater precision levels greater than 28 are needed for it to reach its error floor. The minimum precision for minimum error was required to be 24 bits so only the 4 highest precisions could be studied. To summarise:

1. This is the first research conducted to find the minimum precision for lowest error for the BP and LM algorithms on two types of data, discreet and continuous.

2. It was found that when dealing with the discreet systems have lower precision requirement than continuos systems when learning and optimising said systems.

3. MPLE for XOR was 12 bits. MPLE for LM was 24 bits minimum.

8.4 Levenberg-Marquardt algorithm on FPGAs

The new study implemented the Levenberg-Marquardt Algorithm, for the first time to our knowledge, on an FPGA. We found that the hardware implementation of the LM algorithm was able to find Neural Network parameters to solve the XOR problem in just 13 iteration compared to existing studies of hardware learning using the back propagation algorithm. The FPGA-LM algorithm NN training speed was increased by a factor of 3×10^6. The same algorithm was also used to find the optimal mapping parameters in the

camera calibration problem having a penalty of only 17.94% increase in the MSE and an 8.04% increase in the standard deviation when compared to a software solution. The FPGA-LM algorithm for camera calibration speed was increased by a factor of 1.4×10^6. We were able to get these results by using high level software and hardware development environments. However, the limitations of the development environment restricted the variables to 24 bits of precision, and the FPGA hardware area restricted the size of the system that needs to be solved. The FPGA hardware used in this study proved to be a restriction for portions of the algorithms used. By using a newer FPGA with more resources all the algorithms can be ported to hardware including the operation phase of the function estimation allowing for the elimination of some of the restrictions posed by the current deployed solution. A solution implemented fully in hardware would allow for a reduction in the latency of the solution and can remove the dependency on software. This study provides a roadmap for the implementation of the LM algorithm on fixed-point hardware which can be ported onto a dedicated ASIC for high speed low power applications. In summary:

1. This thesis provides the first implementation of a Levenberg-Marquardt algorithm to estimation NN parameters and calibrate a parametric camera model.

2. Integrating a trust region factor on hardware is crucial to problem with a very large condition number.

3. Training small networks on FPGAs had a speed up in excess of 3×10^6 compared to software.

4. Calibrating camera systems on FPGA gained a speed up of 1.4×10^4 when compared to software.

5. The FPGA used in this research restricted the size of the optimisation problems that can be solved feasibly.

6. Rescaling data for Parametric Camera calibration system incurs a reduction in performance on the calibration system.

7. HONNs and NNs managed to find the image calibration with the data rescaled from it original range.

Appendix A

Back Propagation and Levenberg-Marquardt Algorithm derivation

A.1 Error Back-propagation Algorithm

The following section will present the training of a Neural Network using the Back-Propagation BP algorithm. The network weights are initialised, either randomly or by using other initialisation methods. The Back-propagation algorithm consists of three steps:

1. Network simulation
2. Back-propagating the error
3. Updating the weights

Neural Networks has a number N of input vectors X with length n,

$$X = x_0, x_1, x_2, ..., x_n$$

$$Net_{input} = X_0, X_1, X_2, ..., X_N$$

We implement the feed forward stage of the network by multiplying with the corresponding weights as follows

$$Net_{hidden_i} = W_1 \times X_i$$

Net_{hidden_I} is the output of the first hidden layer at sample i before applying the non-linear threshold function.

$$Net_{output_i} = W_2 \times f(Net_{hidden_i})$$

where the non-linear function in the hidden layer is the logistic function.

$$f_{hid}(x) = \frac{1}{1+e^{-x}}$$

In order to back-propagate the error, the transfer functions need to be differentiable.

$$f'_{hid} = \frac{df}{dx} = f(x) \times (1 - f(x))$$

while the output layer will have a linear function

$$f_{out}(x) = x$$

$$f'_{out} = \frac{\partial f_{out}}{\partial x} = 1$$

We attempt to minimise the following energy function over the training set with W representing all the networks weights

$$E(W) = \sum_{n=1}^{N} (target_n - Net_{output_n})^2$$

$$\delta = -\frac{\partial E}{\partial Net_{output}}$$

As the *target*

is given, we can calculate the weight update using the delta rule.

$$\Delta w_{ij} = -\frac{\partial E}{\partial Net_{output}} \frac{\partial Net_{output}}{\partial w_{ij}}$$

where i is the index of the layer the weight is located in, and j is its position within that layer.

Then the weight difference in the hidden layer is calculated,

$$\frac{\partial Net_{output}}{\partial w_i} = \frac{\partial}{\partial w_{ij}} \sum w_i Net_{Hidden}$$

Using the chain rule, we find the weight update in the hidden layers

$$\Delta w_{ij} = \delta_i Net_{Hidden_j}$$

$$\Delta w_{ji} = -\eta_h \frac{\partial E}{\partial w_{ji}}$$

As the error is calculated as previously mentioned, it can be written in a simple way for every sample presented to the NN as follows.

$$\delta = target - Net_{output}$$

We expand the difference (the error) using the delta chain rule,

$$\delta_{ij} = \frac{\partial E}{\partial Net_{output}} \times \frac{\partial Net_{output}}{\partial Net_{Hidden}} \times \frac{\partial Net_{Hidden}}{\partial w_{ji}}$$

The Back-Propagation algorithm in matrix form:

$$\hat{y} = \sum_{l=1}^{L} f_l(W_l \times X_l + b_l)$$

Where l , is the layer index, $l = 1, 2, \ldots, L$. X_l is the input to layer l , W_l and b_l are the weights and biases of their respective layers.

The error of the output layer is,

$$\delta = target - \hat{y}_L$$

.

The error back propagation is as follows,

$$\delta_l = \left(W_{l+1}^T \times \delta_{l+1}\right) \cdot f_l^{'}\left(X_l\right)$$

Weight updates,

$$\Delta W_l = \delta_l X_{l-1}^T$$

$$\Delta b_l = \delta_l$$

We repeat the forward simulation and back-propagation followed by weight updates for as many times as required by our conditions of operation.

A.2 Levenberg-Marquardt Algorithm

We want to fit function that has output, y, when it operates on input, x, having parameters, w,

$$y = F(x; w)$$

Where the energy function we want to minimise is the same as before

Table A.1: Back-Propagation Algorithm

1	$while\, i < Max\, Iteration$	
2	$Net_{Hidden} = W_1 \times \begin{bmatrix} X_0 \\ 1 \end{bmatrix}$	Output of first layer
3	$X_{Hidden} = f(Net_{Hidden})$	Output after hidden layer
4	$Net_{Output} = W_2 \times \begin{bmatrix} X_{Hidden} \\ 1 \end{bmatrix}$	Network output
5	$E = Target - Net_{Output}$	
6	$\Delta E_{out} = f'_{linear}(X_{Hidden}) \cdot E$	Error in output layer
7	$\Delta W_2 = \Delta E_{out}$	Output layer weight change
8	$\Delta E_{Hidden} = W_2^T \times \Delta E_{out}$	
9	$\Delta W_1 = f'_{logistic}(X_{Hidden}) \cdot \Delta E_{Hidden}$	Hidden layer weight change
10	$W_{2_{new}} = \alpha \times W_{2_{old}} + (1-\alpha) \times \eta \times \Delta W_2 \cdot \begin{bmatrix} X_{Hidden} & 1 \end{bmatrix}^2$	New hidden-output layer weights
11	$W_{1_{new}} = \alpha \times W_{1_{old}} + (1-\alpha) \times \eta \times \Delta W_1 \cdot \begin{bmatrix} X_0 & 1 \end{bmatrix}^2$	New input-hidden layer weights

Check if training conditions are still true, if yes: repeat, otherwise exit training

$$E(w) = \sum_{i=1}^{n} [y_i - y(x_i; w)]^2$$

The gradient of, E, with respect to the parameters, w, which will be zero at, E, minimum, K, is the layer number.

$$\frac{\partial E}{\partial w_k} = -2 \sum_{i=1}^{N} [y_i - y(x_i; w)] \times \frac{\partial y(x_i; w)}{\partial w_k} \qquad k = 1, 2, ..., M$$

Taking an additional partial derivative gives

$$\frac{\partial^2 E}{\partial w_k \partial w_l} = 2 \sum_{i=1}^{N} [\frac{\partial y(x_i; w)}{\partial w_k} \frac{\partial y(x_i; w)}{\partial w_l} - [y_i - y(x_i; w)] \times \frac{\partial^2 y(x_i; w)}{\partial w_k \partial w_l}]$$

Its common to remove the factors of 2 by defining

$$\beta_k = -\frac{1}{2} \frac{\partial E}{\partial w_k} \qquad \alpha_k = \frac{1}{2} \frac{\partial^2 E}{\partial w_k \partial w_l}$$

Setting, $\alpha = \frac{1}{2} D$, we can rewrite the equation as a set of linear equations, D, is a square Hessian matrix

$$\sum_{l=1}^{M} \alpha_{kl} \delta w_l = \beta_k$$

Solved for increments of, δw_l, in the context of least-squares, matrix α, being equal to one-half times the Hessian matrix.

$$\delta w_l = constant \times \beta_l$$

Matrix form of this algorithm:
As previously stated in the back-propagation algorithm, we first perform the network simulation and extract the resulting error,

We cast the weight parameters to form a single column vector as follows,

$$\theta = \begin{bmatrix} W_L \\ W_{L-1} \\ \vdots \\ W_1 \end{bmatrix}$$

Now we calculate the Jacobian matrix of partial differentials of the outputs with their respective weights and inputs,

$$J(\theta, N)$$

Elements in J relating to the weight vectors is calculated as follows,

$$J \begin{pmatrix} W_L \\ W_{L-1} \\ \vdots \\ W_1 \end{pmatrix} = \begin{bmatrix} f'_L(W_{L+1} \times X_L) \\ f'_{L-1}(W_L \times X_{L-1}) \\ \vdots \\ f'_1(W_2 \times X_1) \end{bmatrix}$$

where $l = 1, 2, \ldots, L$ is the layer number and X_l is the input to layer l. The Hessian matrix is calculated as follows,

$$H(\theta) = J^T J$$

The gradient of the error vector also calculated,

$$\nabla J(\theta) = J(\theta) \times E$$

And the weight update is,

$$\Delta\theta = \left(H + \lambda \times diag(H)\right)^{-1} \nabla J(\theta)$$

Table A.2: Levenberg-Marquardt Algorithm

1	$while\, i < Max\, Iteration$	
2	$Net_{Hidden} = W_1 \times \begin{bmatrix} X_0 \\ 1 \end{bmatrix}$	Output of first layer
3	$X_{Hidden} = f(Net_{Hidden})$	Output after hidden layer
4	$Net_{Output} = W_2 \times \begin{bmatrix} X_{Hidden} \\ 1 \end{bmatrix}$	Network output
5	$E = Target - Net_{Output}$	Error in output layer
6	$\theta = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}$	Weight vector
7	$J(\theta) = \begin{bmatrix} f'_{linear}(X_{Hidden}) \\ W_2 \times f'_{logistic}(X_{Hidden}) \end{bmatrix}$	Jacobian matrix
8	$\nabla J = J \times E$	Error gradient
9	$H = J^T \times J$	Hessian matrix
10	$H = H + \lambda \times diag(H)$	Updating Hessian matrix
11	$\Delta\theta = H^{-1} \times \nabla J$	Weight change
12	$\theta_{new} = \theta_{old} + \Delta\theta$	New weight vector
13	$W_{2_{new}} = W_{2_{old}} + \theta_{new}(W_2)$	New hidden-output layer weights
14	$W_{1_{new}} = W_{1_{old}} + \theta_{new}(W_1)$	New input-hidden layer weights
15	Updating λ	
16	$L = \Delta\theta^T \nabla J + \Delta\theta^T \Delta\theta \lambda_{old}$	Calculating update conditions
17	$\lambda = \begin{cases} \frac{\lambda}{2} & if \quad 2N(MSE - MSE_{new}) > 0.75L \\ 2\lambda & if \quad 2N(MSE - MSE_{new}) \leqq 0.25L \end{cases}$	New lambda
	Check if training conditions are still true, if true: repeat or go to step 10 otherwise exit training	

Appendix B

Learning algorithms Hardware Cost analysis

B.1 Back-Propagation Hardware cost analysis

Performance and requirements of the calculations constituting the learning algorithm are presented in the tables in this section. The location of Outputs, Operands and the type of operation implemented can be seen in prior NN diagrams:

1. Setting the parameters: momentum α, $\alpha_{min,max}$, learning rate η, $\min(E)$, $\min(Gradient)$, maximum iterations.

2. Initialise random weight matrices with the following dimensions:

   - Assume input signals is of dimension D_{in} and has N samples. With a corresponding output signal of dimension D_{out} and the same number of samples.
   - W_1 is a matrix with dimension [$Hidden$, $D_{in} + 1$].
   - W_2 is a matrix with dimension [D_{out}, $Hidden + 1$].

3. We calculate forward propagation and store intermediate stages for later use in back propagation, the operations are shown in Table B.1. In the following tables we present the number of multiplication and addition operations needed in order to perform the functions that the Neural Network is built upon.

   - h_1 : Output from Hidden layer before applying threshold.
   - y_1 : Output after applying threshold on h_1.
   - y_2 : Output of the neural network.
   - E : Error vector, difference between the target and the output of the network.

Table B.1: Cost of Forward Propagation

Output	Operation	Shape	Multiply	Add	LUT
h_1	$W_1 \times \boxed{IN, N_{Samples}}$	$(Hid., D_{in}+1) \times (D_{in}+1, N)$	$Hid. \times D_{in} + 1 \times N$	$(D_{in}+1) \times N$	No
y_1	$logsig \begin{pmatrix} h_1 \\ b \end{pmatrix}$	$(Hid.+1, N)$	—	—	Yes
y_2	$W_2 \times \begin{bmatrix} y_1 \\ b \end{bmatrix}, N$	$(D_{out}, Hid.+1 \times Hid.+1, N)$	$D_{out} \times Hid.+1 \times N$	$(Hid.+1) \times N$	No
E	$Y - y_2$	(D_{out}, N)	—	$D_{out} \times N$	No
SSE	$\sum E \times E^T$	$(D_{out}, N) \times (N, D_{out})$	$D_{out} \times N \times D_{out}$	$N \times D_{out}$	No
$Perf$	$SSE/2N$	$(1)/(1) \times (1)$	3	—	No

Table B.2: Cost of the Back-Propagation of Error

Output	Operation	Shape	Multiply	Add	LUT
δE_2	E	(D_{out}, N)	—	—	No
E_1	$W^T_{2(1:Hid.)} \times \delta E_2$	$(Hid., D_{out}) \times (D_{out}, N)$	$Hid. \times D_{out} \times N$	$D_{out} \times N$	No
$temp$	$y_{1(1:Hid.)} \times (1 - y_{1(1:Hid.)})$	$(Hid., N) \cdot (Hid., N)$	$Hid. \times N$	$D_{out} \times N$	Yes**
δE_1	$temp \cdot E_1$	$(Hid., N) \cdot (Hid., N)$	$Hid. \times N$	—	Yes**

**:: values in the *temp* vector can be stored in a LUT, we also can avoid multiplication by using LUTs.

- SSE :sum of squared errors. $Perf$: mean squared errors /2.

4. Back-Propagation of Error. In Table B.2 we present the cost of back-propagating the error through the Neural Network.

5. In this stage we present the costs associated with the calculation of Error gradients shown in Table B.3.

6. In tables B.4, B.5 and B.6, we present the costs associated with weight update in the presence and absence of the momentum parameter. Additional operations are recalled in order to determine the conditions of Neural Network training as in Table B.7, where we notice that there is a sorting operation of the Gradient parameters in order to determine the maximum gradient in both hidden and output weight matrices, typically a sorting operation is of complexity $O(nlogn)$, so we need to account for the delay the sorting operation takes.

7. We repeat the NN training while conditions $\neq$ true, we go back to step 3 and repeat until condition test results are true, meaning that the NN training operation is complete.

B.2 Levenberg-Marquardt Hardware cost analysis

1. Setting the parameters: momentum: α , learning rate: η , min(E), min$(Gradient)$, $\alpha_{min,max}$, damping factor: λ, maximum iterations.

Table B.3: Cost of the calculating weight changes

Output	Operation	Shape	Multiply	Add	LUT
G_2	$\delta E_2 \times \begin{bmatrix} y_1 \\ b \end{bmatrix}^T$	$(D_{out}, N) \times (N, Hid.+1)$	$D_{out} \times N \times (Hid.+1)$	$N \times (Hid.+1)$	No
G_1	$\delta E_1 \times \begin{bmatrix} IN \\ b \end{bmatrix}^T$	$(Hid., N) \times (N, D_{in}+1)$	$Hid. \times (D_{in}+1) \times N$	$N \times (D_{in}+1)$	No

Table B.4: Cost of the weight updates depending on the value of α

If $\alpha = 0$

Output	Operation	Shape	Multiply	Add
$W_{2_{new}}$	$W_2 + \eta \times G_2$	$(D_{out}, Hid.+1) + (S.) \times (D_{out}, Hid.+1)$	$D_{out} \times (Hid.+1)$	$D_{out} \times (Hid.+1)$
$W_{1_{new}}$	$W_1 + \eta \times G_1$	$(Hid., D_{in}+1) + (S.) \times (Hid., D_{in}+1)$	$Hid. \times (D_{in}+1)$	$Hi. \times (D_{in}+1)$

Table B.5: Cost of the weight updates depending on the value of α

If $\alpha \neq 0$

Output	Operation	Shape
ΔW_2	$\alpha \Delta W_{2_{old}} + (1-\alpha)\eta \times G_2$	$(Scaler) \times (D_{out}, Hid.+1) + (Scaler) \times (Scaler) \times (D_{out}, Hid.+1)$
$W_{2_{new}}$	$W_2 + \Delta W_2$	$(D_{out}, Hid.+1) + (D_{out}, Hid.+1)$
ΔW_1	$\alpha \Delta W_{1_{old}} + (1-\alpha)\eta \times G_1$	$(Scaler) \times (Hid., D_{in}+1) + (Scaler) \times (Scaler) \times (Hid., D_{in}+1)$
W_1	$W_1 + \Delta W_1$	$(Hid., D_{in}+1) + (Hid., D_{in}+1)$

Table B.6: Cost of the weight updates depending on the value of α, continued from Table B.5

If $\alpha \neq 0$

Output	Operation	Shape
ΔW_2	$2 \times Scaler \times D_{out} \times (Hid.+1)$	$D_{out} \times (Hid.+1)$
$W_{2_{new}}$	—	$D_{out} \times (Hid.+1)$
ΔW_1	$2 \times Scaler \times Hid. \times (D_{in}+1)$	$Hid. \times (D_{in}+1)$
W_1	—	$Hid. \times (D_{in}+1)$

Table B.7: Cost of the Checking conditions

Output	Operation	Shape	Multiply	Add	LUT
$Perf_{vector}$	$Perf$	$(1,1)$	—	—	Yes
$Param_{difference}$	$\eta Max(Max(\lvert G_1 \rvert), Max(\lvert G_2 \rvert))$	$(1,1)$	1	Sorting operation $\times 2$	Yes
$Grad_{max}$	$Param_{difference}/\eta/N$	$(1,1)$	2	—	Yes

Table B.8: Cost of computing the Jacobian matrix

Output	Operation	Shape
θ	$\left[reshape(W_2^{'}, parameters_2, 1), reshape(W_1^{'}, parameters_1, 1)\right]$	$(\theta, 1)$
$J(\theta, N)$	(# of parameters, N)	(θ, N)
$J(\theta_{W_2})$	y_1	$(Hid.+1, N)$
For j=1:Hid. $temp_{temp}$ $temp$ $J(\theta_{W_1} index(j) : index(j)+IN, :)$ end	 $(1-y_1(j)) \cdot y_1(j)$ $W_2(j) \times temp_{temp}$ $[1 : length(IN+1)]^T \times [temp] \cdot \begin{bmatrix} IN \\ b \end{bmatrix}$	 $(1, N) \cdot (1, N)$ $(1) \cdot (1, N)$ $(IN+1, 1) \times (1, N) \cdot (IN+1, N)$
∇J	$J \times E$	$(\theta, N) \times (N, 1)$

2. Initialise random weight matrices with the following dimensions:

   - Assume input signals is of dimension D_{in} and has N samples. With a corresponding output signal of dimension D_{out} and the same number of samples. W_1 is a matrix with dimension [$Hid.$, $D_{in}+1$]. W_2 is a matrix with dimension [D_{out} , $Hid.+1$].

3. We calculate forward propagation and store intermediate stages for later use in back propagation which is exactly the same as in the back-propagation algorithm feed-forward cost analysis in Table B.1.

4. Computing the Jacobian and Gradient of the network error needs the operations presented in Table B.8.

5. Computing the Hessian Matrix and updating the weights is shown in Table B.10, we set a flag dw indicating that the weights are calculated. Search direction h obtained by solving a system of linear equations, $X = A \backslash B \rightarrow AX = B$, by using Gaussian elimination. we finish the updating by converting θ back to its corresponding W_1 and W_2 weight matrices.

6. Forward propagation to calculate new damping factor: λ . We re-simulate the Network as previously demonstrated and store the corresponding E_{new} , SSE_{new} and $Perf_{new}$ scalars for use in the calculation of a reference value that will indicate the way we change the value of λ as in Table B.11.

7. Then check if conditions are still in effect, if training is to continue, go back to step 4, as we already have the obtained E_{new} from the beginning of step 6; allowing us calculate the Jacobian matrix and proceed with the training. Training is suspended if the training conditions are no longer hold.

Table B.9: Cost of computing Jacobian matrix, continued from B.8

Output	Multiply	Add	LUT
θ	—	—	Yes
$J(\theta, N)$	—	—	Yes
$J(\theta_{W_2})$	—	—	Yes
For j=1:Hid.			
$temp_{temp}$	$(1) \times N$	$(1) \times N$	Yes**
$temp$	$(1) \times N$	$(1) \times N$	No
$J(\theta_{W_1} index(j) : index(j) + IN, :)$	$((IN+1) \times 1 \times N) + (IN+1, N)$	$(IN+1) \times N$	Store
∇J	$\theta \times N$	θ	Store

Table B.10: Cost of computing the Hessian matrix and updating the weights

Output	Operators and Operands	Shape	Multiply	Add	LUT
H	$J \times J^T$	$(\theta, N) \times (N, \theta)$	$\theta \times N \times \theta$	$N \times \theta$	Yes
$H(diag)$	$H(diag) + (\lambda - \lambda_{old})$	$(1, \theta) + (1)$	—	θ	Yes
H^{-1}	—	$(\theta, \theta) \times (\theta, \theta)$	θ^3	θ^2	—
h	$H^{-1} \nabla J$	$(\theta, \theta) \times (\theta, 1)$	$\theta \times \theta \times 1$	θ	Store
θ_{new}	$\theta + h$	$(1, \theta) + (1, \theta)$	—	θ	Store

Table B.11: Cost of computing the λ update

Output	Operators and Operands	Shape	Multiply	Add
L	$(h' \times \nabla J) + (h' \times (h \cdot \lambda))$	$((1, \theta) \times (\theta, 1)) + ((1, \theta) \times (\theta, 1) \cdot (1))$	$2 \times \theta$	θ
λ_{old}	λ	(1)	—	—
$temp$	$2 \cdot N \cdot (PI - PI_{new})$	$(1) \cdot (1) \cdot (1)$	3	1
λ	$\lambda/2 \quad if \quad temp > 0.75 \cdot L \quad or$ $\lambda \times 2 \quad if \quad temp < 0.25 \cdot L$	$[(1) \cdot (1)]$ if $><$ $(value)$	2	—

B.3 DSP48E Component Summary

Analysing the algorithms implementing the FFNN, BP and LM algorithms we can summarise the information in the tables from the previous section to equations relating the size of the network and the length of the training vectors. Let N be the number of the training vectors, In is the length of the input vector, $Hid.$ is the number of hidden neurons, we assume the output dimension is equal to 1, the activation function is assumed to be linear or stored in LUT, further reducing the burden on the circuit design.

B.3.1 Area of Neural Networks

We will express the number of DSP48E components needed as a function of number of input samples, length of the size of the input vector and the structure of the hidden layer,

$$Multipliers = f(N, In, Hid.)$$

$$Multipliers = A + B$$

A, is the multiplication count in the first-to-hidden layer, while B is the number of multiplications in the hidden-to-output layer

$$A = (In + 1) \times Hid. \times N$$

$$B = (Hid. + 1) \times N$$

$$= N \times (((In + 1) \times Hid.) + (Hid. + 1))$$

Assuming the size of the input vector and the number of hidden neurons is large we can ignore the bias factor of 1.

$$Multipliers = N \times Hid. \times In$$

Figure B.1 shows the multiplier count at various sample sizes, N, and the network parameters, θ, as a function of, In and $Hid.$.

$$\theta(In, Hid.) = ((Hid. + 1) + (Hid. \times (In + 1)))$$

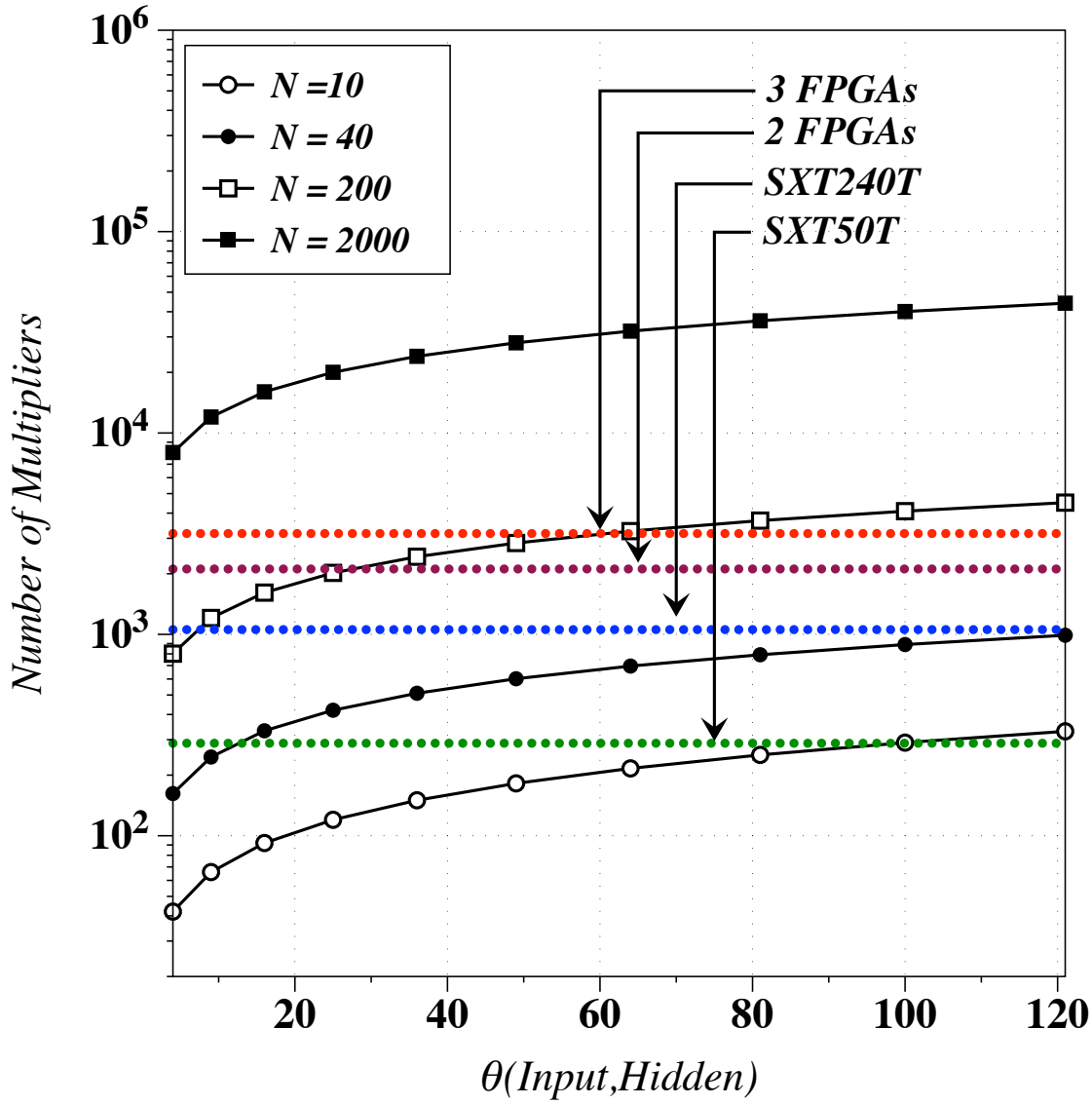

Figure B.1: Area of FeedForward Neural Network with respect to increasing number of parameters

B.3.2 Area of Back-Propagation

We assume that the momentum parameter $\alpha = 0$, to reduce complexity. We can segment the BP algorithm as in the following equations.

$$Multipliers = A + B + C + D + E + F + G$$

Where, A is the partial error in the output layer, B is the derivative of the activation function, C is the partial error in hidden layer, D is update calculation for weights linking the hidden layer to the output layer, E is the weights update of the input-to-hidden layer matrix, F and G are the updating costs when multiplying by the learning rate η.

$$A = Hid. \times N$$

$$B = Hid. \times N$$

$$C = Hid. \times N$$

$$D = (Hid. + 1) \times N$$

$$E = Hid. \times N$$

$$F = Hid. + 1$$

$$G = Hid. \times (In + 1)$$

$$= 4 \times (Hid. \times N) + (Hid. + 1) \times N + Hid. + 1 + Hid. \times (In + 1)$$

Removing small factors,

$$Multipliers = Hid.(4 \times N + In) + N$$

The relationship between θ and the length of the training vector N is shown in figure B.2.

B.3.3 Levenberg-Marquardt Multiplier Area

$$Multipliers = A + B + C + D + E + F$$

Where, A is the Jacobian matrix of the input-to-hidden layer matrix , B gradient function ∇J , C is the Hessian, D is the inverse of the Hessian, E is the weight update vector and F is the calculation of the λ updating parameter.

$$A = Hid. \times (N + N + N(In + 1))$$

$$B = \theta \times N$$

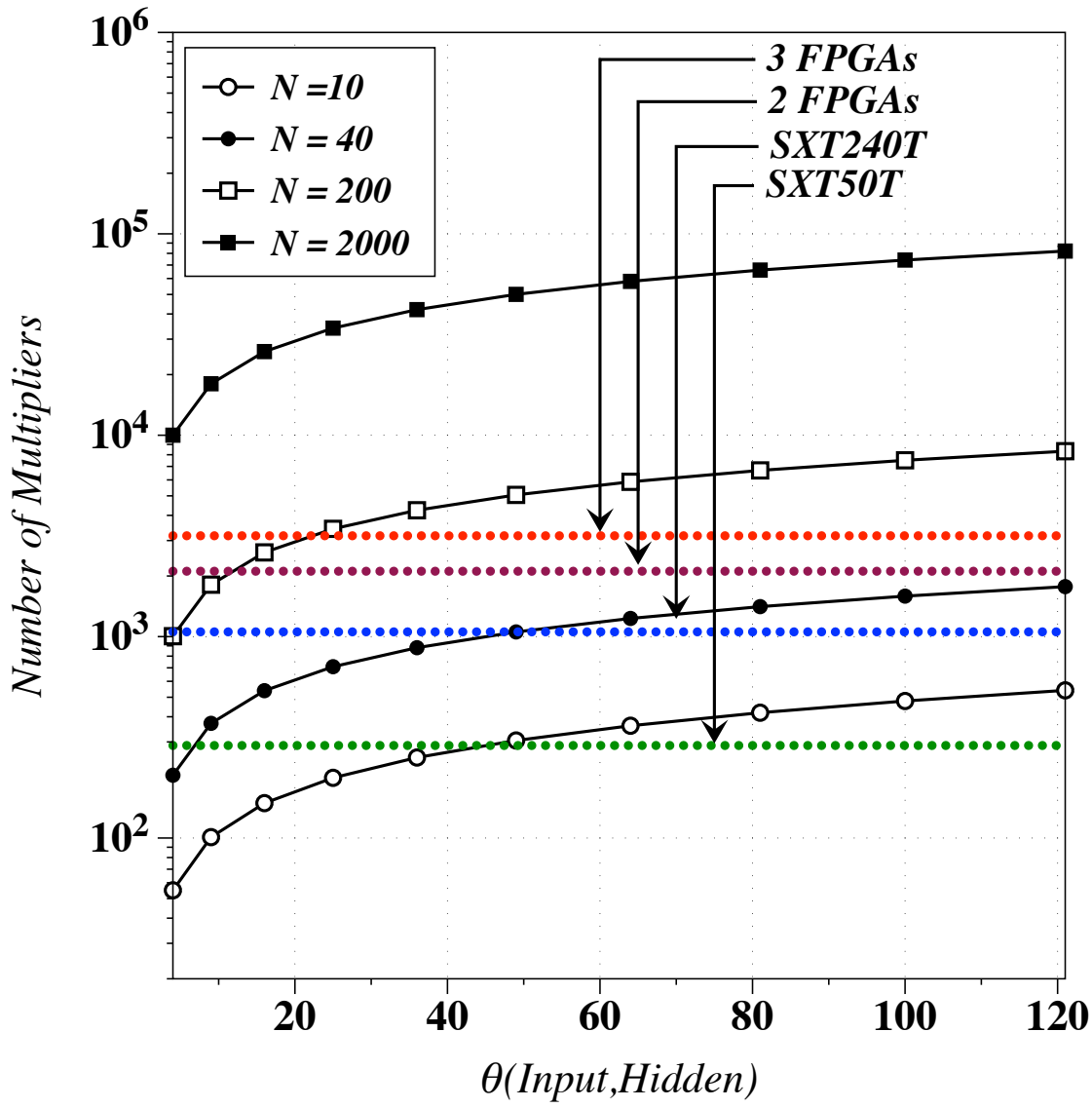

Figure B.2: BP algorithm multiplier cost

$$C = \theta \times N \times \theta$$

$$D = \theta^3$$

$$E = \theta^2$$

$$F = \theta + \theta$$

$$= N \times Hid. \times (3 + In) + \theta(N + \theta N + \theta^2 + \theta + 2)$$

Removing small factors,

$$Multipliers = N \times Hid. \times In + \theta(N + \theta N + \theta^2 + \theta)$$

The relationship between θ and the length of the training vector N is shown in figure B.3.

In figures B.1, B.2 and B.3 we presented the multiplier cost when implementing FFNNs and their training algorithm showing the multiplier count against the size of the network parameters at different levels in the number of samples used for the optimisation process. We have drawn four horizontal lines indicating the number of DSP48E available in the FPGAs under consideration, currently we have the SXT50T Virtex-5 FPGA with 288 DSP48E units, we have the option to increase the number of DSP units by choosing a bigger FPGA such as the SXT240T that has 1056 DSP48E units and also it is possible to use multiple FPGAs to run our network and learning processes. Examining the figures B.1, B.2 and B.3 we notice that we reach the operational capacity of the FPGA quite quickly, as the number of parameters increases exponentially, when adding more FPGAs the increases in the DSP units is linear. So for big problems its not possible to run the optimisation processes with the limited number of DSP48Es that we have. Adding more FPGAs is not the optimal solution for this problem. We notice that the problem is doubled by the scarcity and the limited resources available to do arithmetic operations and the limited precision inherent in fixed-point. In the next chapter we propose future work that deals with those issues, explaining the rational and suggest solutions for this research problem.

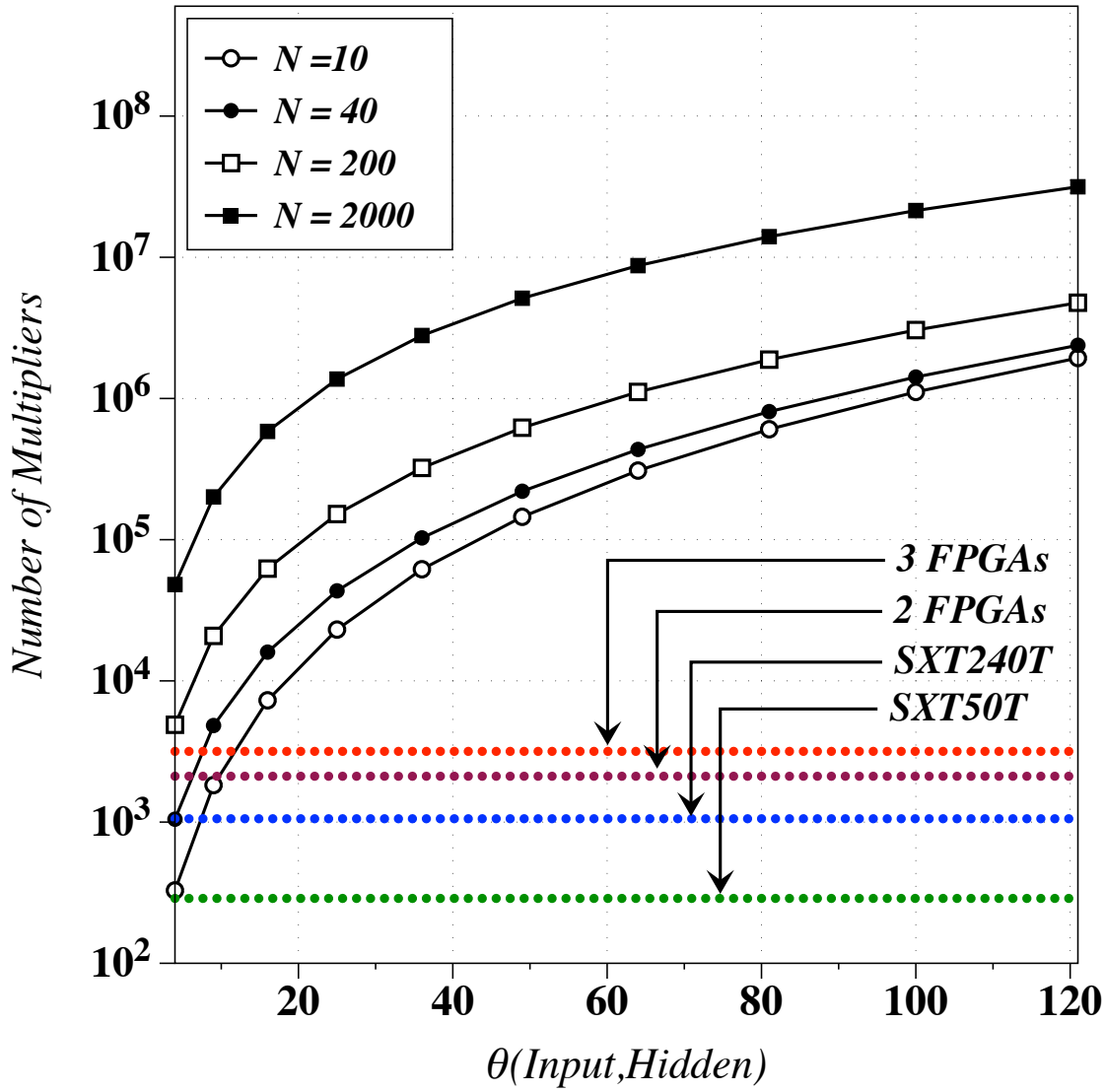

Figure B.3: LM algorithm multiplier cost

Appendix C

Example of NN smoothing function on a FPGA

Neural network operation has two modes, operation and training. In the operation mode, the signals are propagated thought the NN structure as in (C.1). The propagation provides an evaluation of the deviation of the results from the target values. These deviation are used in the second, training mode of operation. The deviation form the errors that are used in the back-propagation algorithm.

$$\begin{aligned} N &= W_1 \times X + b_1 \\ \hat{Y} &= W_2 \times sigmoid(N) + b_2 \end{aligned} \tag{C.1}$$

$$y = \frac{1}{1 + e^{-x}} \tag{C.2}$$

$$y = c_k + m_k \times x \tag{C.3}$$

Table C.1 shows the error in the estimation of the piecewise linear estimation of the non-linear sigmoid function. Where the k defines the number of piecewise segments that the interpolation look up table (LUT) will be divided to approximate; the number of segments is divided into 2^k segments. The LUT covers $2^k + 1$ segments in the active region of [0,8] as can be seen in Figure C.1. This region was chosen due to the symmetry of the sigmoid around the x-axis and its convergence to 1 when its input is larger than 8. The second column in Table C.1 shows the total number of coefficients based on Eq (C.3). The last three columns show the MSE error when comparing the LUT operation to the floating point operation using the Eq (C.2). $MSE_{\text{“}double\text{”}}$ shows the error of Eq (C.2) and Eq (C.3) when both are operating in floating point format. $MSE_{\text{“}Q\text{”}}$ compares Eq (C.2) in floating point to Eq (C.3) when all the variables are quantised in the precision indicated in Table C.2, however; all operation are carried out in floating point precision.

Table C.1: *sigmoid* approximation error of LUT operation in floating and fixed-point format

k	# coefficients	$MSE_{\text{“}double\text{”}}$	$MSE_{\text{“}Q\text{”}}$	$MSE_{\text{“}FPGA\text{”}}$
3	16	1.6812×10^{-5}	1.7258×10^{-5}	1.7259×10^{-5}
4	32	1.0751×10^{-6}	1.1699×10^{-6}	1.1707×10^{-6}
5	64	6.7583×10^{-8}	9.2152×10^{-8}	9.2968×10^{-8}

Table C.2: Quantizer resolutions in the LUT based *sigmoid* function

Data	in	c1	c2	out
Sign	1	1	1	1
Word length	16	16	16	16
Fraction length	11	14	14	12

Finally, $MSE_{\text{“}FPGA\text{”}}$ compares Eq (C.2) in floating point to Eq (C.3) with the actual values obtained from the FPGA implementation with signals and variables casted to the precision shown in Table C.2, however; this time all operation are carried out in fixed-point precision. The MSE results were obtained from implementing the sigmoid function using 2000 intermediate values in the region of $[0, 8]$.

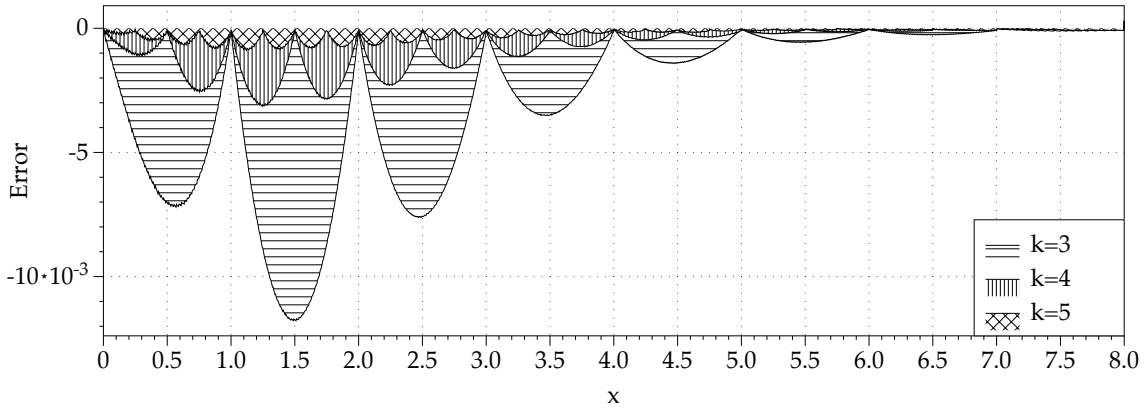

Figure C.1: *sigmoid* approximation error of quantised LUT operation at three k-values

Figure C.2 shows the error when approximating the *sigmoid* function using piecewise linear function in double and quantised precisions with k ranging from [1-14]. We can see that the error of the LUT double approximation reaches the limits of the computer precision. The quantised precision error converges after $k = 7$ with minute levels of improvement.

Initially the weights were calculated using the Neural Networks toolbox operating in full double precision mode. This network parameters were converted to fixed-point format and loaded onto a Neural Network running in the FPGA.

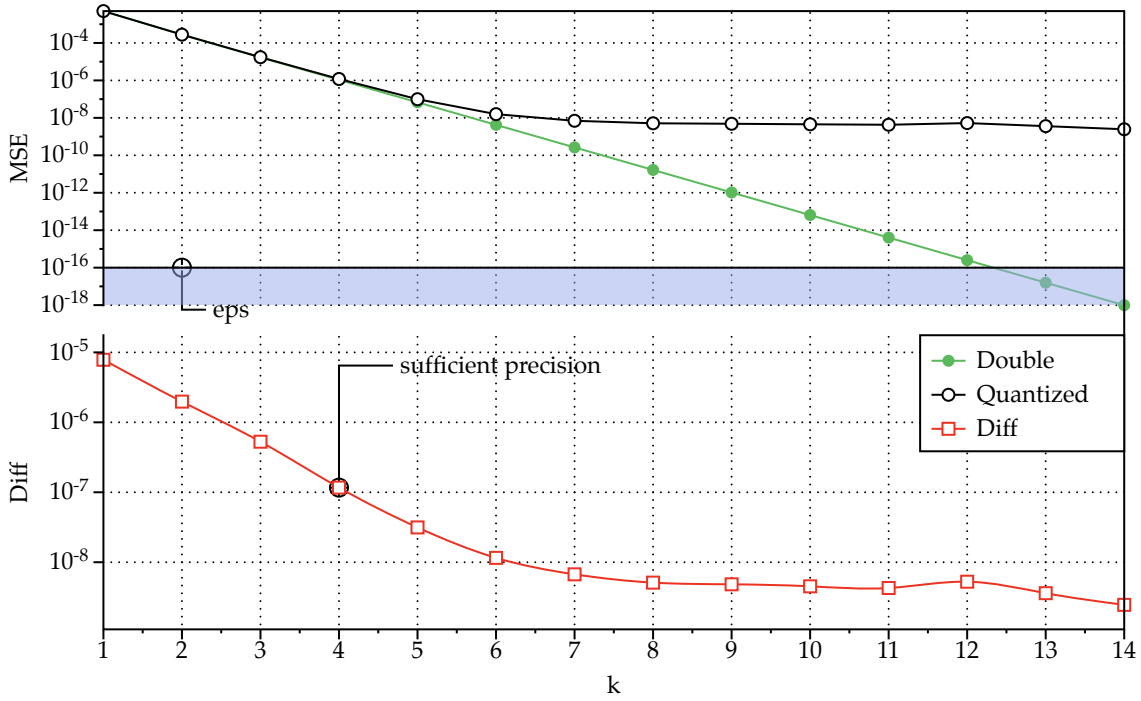

Figure C.2: Double and quantised Piecewise Linear Approximation error for k ranging from 1 to 14

Table C.3: XOR Neural Network (2-2-1) on FPGA utilisation summary

IO utilisation:	#	%
Number of bonded IOBs:	59 out of 480	12%
Number of LOCed IOBs:	59 out of 59	100%
IOB Flip Flops:	39	
Specific Feature utilisation:		
Number of BlockRAM/FIFO:	6 out of 132	4%
Number using BlockRAM only:	6	
Total primitives used:		
Number of 36k BlockRAM used:	6	
Total Memory used (KB):	216 out of 4,752	4%
Number of BUFG/BUFGCTRLs:	8 out of 32	25%
Number used as BUFGs:	3	
Number used as BUFGCTRLs:	5	
Number of IDELAYCTRLs:	1 out of 16	6%
Number of DCM_ADVs:	1 out of 12	8%
Number of DSP48Es:	1 out of 288	1%

Appendix D

Floating point LM algorithm using QR factorisation

Table D.1 shows the difference in the parameters of the trained neural network using the Matlab Neural Network LM algorithm and the modified version using the QR solver method, the fixed-point precision was set to 32-bits.

Table D.2 shows the difference in the parameters of the trained neural network using the Matlab Neural Network LM algorithm and the modified version using the QR solver method, the fixed-point precision was set to 52-bits.

Table D.3 shows the difference the number of epochs required to complete training with different quantisation constraints. The QR in the first column was performed in floating point, in the second column only the Hessian matrix was quantised. In the final column both the Hessian matrix and the Jacobian were quantised to 52 bits.

Table D.1: Difference of the default LM approach vs the proposed QR @ 32 bits

Parameters	$MSE_{\text{“}floating\text{”}}$	$MSE_{\text{“}quantised(H)\text{”}}$	$MSE_{\text{“}quantised(H,J)\text{”}}$
W_1	1.4204×10^{-25}	6.5723×10^{-13}	2.8667×10^{-7}
W_2	9.7234×10^{-25}	9.5571×10^{-13}	1.0499×10^{-7}
B_2	3.2950×10^{-26}	1.7972×10^{-13}	2.4145×10^{-7}

Table D.2: Difference of the default LM approach vs the proposed QR @ 52 bits

Parameters	$MSE_{\text{“}floating\text{”}}$	$MSE_{\text{“}quantised(H)\text{”}}$	$MSE_{\text{“}quantised(H,J)\text{”}}$
W_1	1.4204×10^{-25}	2.1137×10^{-25}	1.3263×10^{-19}
W_2	9.7234×10^{-25}	1.6532×10^{-25}	4.9710×10^{-20}
B_2	3.2950×10^{-26}	1.4824×10^{-25}	1.1104×10^{-19}

Table D.3: $QR - MSE$ at different quantisation configurations

Parameter	$floating$	$quantised(H)$	$quantised(H, J)$
Epochs	8	8	13
Performance	2.45×10^{-27}	2.53×10^{-27}	3.57×10^{-18}

References

Hirotsugu Akaike. New look at statistical-model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. URL ISI:A1974U921700011. 47

Mohammad S. Alam and Abdullah Bal. Improved Multiple Target Tracking via Global Motion Compensation and Optoelectronic Correlation. *IEEE Transactions on Industrial Electronics*, 54(1):522–529, February 2007. ISSN 0278-0046. doi: 10.1109/TIE.2006.885513. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4084638. 93

S. Amari, A. R. Barron, E. Bienenstock, S. Geman, L. Breiman, J. L. McClelland, B. D. Ripley, R. Tibshirani, B. Cheng, and D. M. Titterington. Neural networks - a review from a statistical perspective - comments and rejoinders. *Statistical Science*, 9(1):31–54, 1994. URL ISI:A1994NG28500002. 10

R. Anchini, C. Liguori, V. Paciello, and a. Paolillo. A Comparison Between Stereo-Vision Techniques for the Reconstruction of 3-D Coordinates of Objects. *IEEE Transactions on Instrumentation and Measurement*, 55(5):1459–1466, October 2006. ISSN 0018-9456. doi: 10.1109/TIM.2006.881034. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1703886. 64

D. Anguita, A. Boni, and S. Ridella. A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation. *IEEE Transactions on Neural Networks*, 14(5):993–1009, 2003. ISSN 1045-9227. URL ISI:000186478900004. 17, 20

Hashem Ashrafiuon, Kenneth R. Muske, Lucas C. McNinch, and Reza a. Soltan. Sliding-Mode Tracking Control of Surface Vessels. *IEEE Transactions on Industrial Electronics*, 55(11):4004–4012, November 2008. ISSN 0278-0046. doi: 10.1109/TIE.2008.2005933. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4657369. 93

H.A. Babri, Y.Q. Chen, and T. Yin. Improving backpropagation learning under limited precision. *Pattern Recognition Letters*, 19(11):1007–1016, 1998. ISSN 0167-8655. URL ISI:000076968200003. 21

H. Bacakoglu and M.S. Kamel. A three-step camera calibration method. *IEEE Transactions on Instrumentation and Measurement*, 46(5):1165–1172, 1997. ISSN 00189456. doi: 10.1109/19.676732. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=676732. 59

Hynek Bakstein. *A Complete DLT-based Camera Calibration with a Virtual 3D Calibration Object*. Diploma, Charles University, Prague, 1999. URL http://terezka.ufa.cas.cz/hynek/toolbox.html. 58

E. Barnard and L. Wessels. Extrapolation and interpolation in neural network classifiers. *IEEE Control Systems*, 12(5):50–53, 1992. 9, 10, 13, 34

Mats Bengtsson. *Higher Order Artificial Neural Networks*. Diane Publishing Company, Darby PA, USA, June 1990. ISBN 0941375927. 6, 7

C.O. Benjamin, S.C. Chi, T. Gaber, and C.A. Riordan. Comparing BP and ART-II Neural-Network Classifiers for Facility Location. *Computers & Industrial Engineering*, 28(1):43–50, 1995. URL ISI:A1995QE25900003. 13

T. M. Bhatt and D. McCain. Matlab as a development environment for FPGA design. *42nd Design Automation Conference, Proceedings 2005*, pages 607–610, 2005. URL ISIP:000230430300122. 79, 92, 97

J. Bilmes, K. Asanovic, and J. Demmel. Using PHiPAC to speed error back-propagation learning. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 4153–4156. IEEE Comput. Soc. Press, 1997. ISBN 0-8186-7919-0. doi: 10.1109/ICASSP.1997.604861. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=604861. 73

Y. Bodyanskiy and S. Popov. Neural network approach to forecasting of quasiperiodic financial time series. *European Journal of Operational Research*, 175(3):1357–1366, December 2006. URL ISI:000241411500003. 47

Tim Peter Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. URL ISI:A1986D184300004. 32, 39

Jean-Yves Bouguet. Camera Calibration Toolbox for Matlab, 2008. URL http://www.vision.caltech.edu/bouguetj/calib_doc/index.html#system. 64, 66, 97, 99

George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control (Wiley Series in Probability and Statistics)*. Wiley, New Jersey, 2008. ISBN 0470272848. URL http://www.amazon.com/Time-Analysis-Forecasting-Probability-Statistics/dp/0470272848. 45

Chris Brooks. Predicting stock index volatility: Can market volume help? *Journal of Forecasting*, 17(1):59–80, 1998. URL ISI:000072240000004. 15, 32, 40

B. Burton and R. G. Harley. Reducing the computational demands of continually online-trained artificial neural networks for system identification and control of fast processes. *IEEE Transactions on Industry Applications*, 34(3):589–596, 1998. ISSN 0093-9994. 22

J. Y. Campbell. Stock returns and the term structure. *Journal of Financial Economics*, 18(2), 1987. URL WOS:A1987J008900007. 32

L. J. Cao. Support vector machines experts for time series forecasting. *Neurocomputing*, 51, 2003. URL ISI:000181912600020. 14

Q. Cao, K.B. Leggio, and M.J. Schniederjans. A comparison between Fama and French's model and artificial neural networks in predicting the Chinese stock market. *Computers & Operations Research*, 32(10):2499–2512, October 2005. URL ISI:000228207700002. 15

M. Caudill. The view from now. *AI Expert*, pages 24–31, June 1992. 7

C. Chatfield. *The Analysis of Time Series - An Introduction*. Chapman & Hall, London, 4th edition, 1989. ISBN 1584883170. URL http://www.amazon.com/Analysis-Time-Introduction-Statistical-Science/dp/1584883170. 82

Jung Uk Cho, Quy Ngoc Le, and Jae Wook Jeon. An FPGA-Based Multiple-Axis Motion Control Chip. *IEEE Transactions on Industrial Electronics*, 56(3):856–870, March 2009. ISSN 0278-0046. doi: 10.1109/TIE.2008.2004671. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4624557. 93

Siu-Yeung Cho and Tommy W.S. Chow. Shape and Surface Measurement Technology by an Improved Shape-From-Shading Neural Algorithm. *IEEE Transactions on Industrial Electronics*, 47(1):225–230, 2000. 93

S. Chtourou, M. Chtourou, and O. Hammami. Neural network based memory access prediction support for SoC dynamic reconfiguration. *IEEE International Joint Conference on Neural Network Proceedings*, 1-10:2823–2829, 2006. URL ISIP:000245125905020. 18

M.J. Clarkson, D. Rueckert, D.L.G. Hill, and D.J. Hawkes. Using photo-consistency to register 2D optical images of the human face to a 3D surface model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1266–1280, 2001. ISSN 01628828. doi: 10.1109/34.969117. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=969117. 64

B. Cope, P.Y.K. Cheung, W. Luk, and S. Witt. Have GPUs made FPGAs redundant in the field of video processing? In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, pages 111–118. IEEE, 2005. ISBN 0-7803-9407-0. doi: 10.1109/FPT.2005.1568533. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1568533. 19

Judith E. Dayhoff. *Neural Network Architectures: An Introduction*. Van Nostrand Reinhold, New York, 1990. 9, 42

Judith E. Dayhoff and Jim M. Deleo. Artificial neural networks - opening the black box. In *Conference on Prognostic Factors and Staging in Cancer Management*, volume 91, pages 1615–1635. Wiley, 2001. URL ISI:000168094800006. 34

Guido J. Deboeck. *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*. Wiley, New York, 1994. 32, 41

J.W. Denton. How Good Are Neural Networks for Causal Forecasting? *Journal of Business Forecasting*, 14(8):17–20, 1995. 13

V.S. Desai and R. Bharati. A comparison of linear regression and neural network methods for predicting excess returns on large stocks. *Annals of Operations Research*, 78:127–163, March 1998. URL ISI:000073378300008. 8, 13

F. Dias, F. Berry, J. Serot, and F. Marmoiton. Hardware, design and implementation issues on a FPGA-based smart camera. *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 17–23, 2007. URL ISIP:000254383500003. 17

F. M. Dias, A. Antunes, J. Vieira, and A. Mota. A sliding window solution for the on-line implementation of the Levenberg-Marquardt algorithm. *Engineering Applications of Artificial Intelligence*, 19(1):1–7, 2006. URL WOS:000234724400001. 74

DigiTimes.com. Channel vendors demand card makers recall faulty nvidia products. http://www.digitimes.com/news/a20080725PD206.html, July 2008. URL http://www.digitimes.com/news/a20080725PD206.html. 20

R Donaldson. An artificial neural network-GARCH model for international stock return volatility. *Journal of Empirical Finance*, 4(1):17–46, January 1997. ISSN 09275398. doi: 10.1016/S0927-5398(96)00011-4. URL http://linkinghub.elsevier.com/retrieve/pii/S0927539896000114. 39

Doulos. Advanced VHDL. http://www.doulos.com/content/training/ advanced_vhdl_training.php, Feb 2008. URL http://www.doulos.com/content/training/advanced_vhdl_training.php. 24

Robert F. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United-Kingdom Inflation. *Econometrica*, 50(4):987–1007, 1982. URL ISI:A1982NW70600008. 32, 39

Eugene F. Fama. Efficient capital markets - review of theory and empirical work. *Journal of Finance*, 25(2):383–423, 1970. URL ISI:A1970G412100008. 32

Shi Fanhuai, Zhang Xiaoyun, Liu Hongjian, and Liu Yuncai. Estimation of camera pose using 2D-3D occluded corner correspondence. In Yuan Baozong, Ruan Qiuqi, and Tang Xiaofang, editors, *7th International Conference on Signal Processing*, pages 1256–1259, Beijing, China, 2004. IEEE. ISBN 0-7803-8406-7. doi: 10.1109/ICOSP.2004.1441553. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1441553. 64

D.D. Finlay, D.D. Finlay, C.D. Nugent, P.J. McCullagh, N.D. Black, J.A. Lopez, and C.D. Nugent. Evaluation of a statistical prediction model used in the design of neural network based ECG classifiers: a multiple linear regression approach Evaluation of a statistical prediction model used in the design of neural network based ECG classifiers: a multiple linear regression approach. pages 258–260, 2003. 11

Limin Fu. *Neural Networks in Computer Intelligence*. McGraw-Hill, New York, 1994. ISBN 0079118178. 8, 10, 13, 39, 43, 75

John Fulcher, Ming Zhang, and Shuxiang Xu. *Application of Higher-Order Neural Networks to Financial Time-Series Prediction*, chapter 5, pages 80–108. Artificial Neural Networks in Finance and Manufacturing. Idea Group Publishing, Hershey, PA, 2006. ISBN 1591406714. URL http://eprints.utas.edu.au/638/. 35, 75

R. Gadea, J. Cerda, F. Ballester, and A. Mocholi. Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation. *13th International Symposium on System Synthesis, Proceedings*, pages 225–230, 2000. URL ISIP:000167102200034. 21

R. Gadea-Girones, A. Ramirez-Agundis, J. Cerda-Boluda, and R. Colom-Palero. FPGA implementation of adaptive non-linear predictors for video compression. *Field-Programmable Logic and Applications, Proceedings*, 2778:1016–1019, 2003. URL WOS:000186329700109. 17, 20, 22

P. P. Gandhi and V. Ramamurti. Neural networks for signal detection in non-gaussian noise. *IEEE Transactions on Signal Processing*, 45(11), 1997. URL ISI:A1997YE90500023. 5

J. Garrigos, J.J. Martinez, J. Toledo, and J.M. Ferrandez. HANNA: A tool for hardware prototyping and benchmarking of ANNs. volume 4528, pages 10–18, 2007. URL ISIP:000247804300002. 20, 25

Ramazan Gencay and Thanasis Stengos. Moving average rules, volume and the predictability of security returns with feed forward network. *Journal of Forecasting*, 17 (5-6):401–414, 1998. URL ISI:000077416900005. 40

James E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer New York, New York, 1 edition, 2009. ISBN 1441924248. URL http://www.amazon.com/Matrix-Algebra-Computations-Applications-Statistics/dp/1441924248. 96

Clyde Lee Giles and T. Maxwell. Learning, Invariance, and Generalization in High-Order Neural Networks. *Applied Optics*, 26(23):4972–4978, 1987. URL ISI:A1987L307700009. 6, 35, 61, 75

R.G. Girones, R.C. Palero, J.C. Boluda, and A.S. Cortes. FPGA implementation of a pipeflned on-line backpropagation. *Journal of VlSI Signal Processing Systems for Signal Image and Video Technology*, 40(2):189–213, 2005. URL WOS:000229477700003. 19

M. Gorgon and M. Wrzesinski. Neural network implementation in reprogrammable FPGA devices - An example for MLP. volume 4029, pages 19–28, 2006. URL WOS:000239600000003. 22

GPGPU. General-purpose computation using graphics hardware, May 2008. URL http://www.gpgpu.org. 20

Christian Hæke and Christian Helmenstein. Neural networks in the capital markets: An application to index forecasting. *Computational Economics*, 9(1):37–50, February 1996. URL http://www.springerlink.com/content/h5547h3807700068. 15

Rik W. Hafer and Richard G. Sheehan. The sensitivity of VAR forecasts to alternative lag structures. *International Journal of Forecasting*, 5(3):399–408, 1989. URL ISI:A1989CH65300011. 33

Mahmoud Hamouda, Handy Fortin Blanchette, Kamal Al-Haddad, and Farhat Fnaiech. An Efficient DSP-FPGA-Based Real-Time Implementation Method of SVM Algorithms for an Indirect Matrix Converter. *IEEE Transactions on Industrial Electronics*, 58(11):5024–5031, November 2011. ISSN 0278-0046. doi: 10.1109/TIE.2011.2159952. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5892884. 93

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New York, 2nd edition, June 1999. ISBN 8120323734. 9, 10, 11, 41, 43, 47, 75, 82

R. Hecht-Nielsen. *Neurocomputing*. Addison Wesley, Menlo Park, CA, 1989. 7

J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112. IEEE, 1997. ISBN 0-8186-7822-4. doi: 10.1109/CVPR.1997.609468. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=609468. 59, 64, 66

J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, Reading, 1989. 5

Ypke Hiemstra. Linear regression versus back propagation networks to predict quarterly stock market excess returns. *Computational Economics*, 9(1):67–76, February 1996. URL http://www.springerlink.com/content/hmp77v8t5212276t. 13, 42

M. Holler. An Electrically Trainable Artificial Neural Network (ETANN) with 10240 'Floating Gate' Synapses) with 10240 'Floating Gate' Synapses. In *International Joint Conference Neural Network*, volume 2, pages 191–196, Washington, DC, June 1989. 21

J. L. Holt and J. N. Hwang. Finite precision error analysis of neural network electronic hardware implementations. In *IEEE International Joint Conference on Neural Networks*, volume 1-2, pages A519–A525, New York, July 1991. 21

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL ISI:A1989AT49300003. 12, 34

W. Huang, S.Y. Wang, L. Yu, Y.K. Bao, and L. Wang. A new computational method of input selection for stock market forecasting with neural networks. *Computational Science - ICCS 2006, Pt 4, Proceedings*, 3994:308–315, 2006a. URL ISI:000238417500046. 42

Wei Huang, Lean Yu, Shouyang Wang, Yukun Bao, and Lin Wang. Comparisons of the different frequencies of input data for neural networks in foreign exchange rates forecasting. volume 3994, pages 517–524, 2006b. URL ISI:000238417500072. 41

H.M. Ibrahim, R.R. Gharieb, and M.M. Hassan. A higher order statistics-based adaptive algorithm for line enhancement. *IEEE Transactions on Signal Processing*, 47(2):527–532, February 1999. URL ISI:000078123700022. 6

Michalis Ioannides. A Comparison of Yield Curve Estimation Techniques Using UK Data. *Journal of Banking & Finance*, 27(1):1–26, 2003. URL ISI:000180076200002. 9, 10, 47

K. Irick, T. Theocharides, V. Narayanan, and M.J. Irwin. *A real time embedded face detector on FPGA*. 40th Asilomar Conference on Signals, Systems and Computers. 2006. ISBN 1058-6393. URL ISIP:000246925202011. 17, 18

A G Ivakhnenko. Polynomial Theory of Complex Systems Polynomial Theory of Complex Systems. *IEEE Trans, on Systems, Man, and Cybernetics*, 1(4):364–378, 1971. 6

S. Jung and S.S. Kim. Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems. *IEEE Transactions on Industrial Electronics*, 54(1):265–271, 2007. URL WOS:000244334100026. 17, 73

Iebeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, April 1996. URL ISI:A1996UH42800002. 11, 13, 14, 15, 42, 43, 76

Casimir C. Klimasauskas, R.R. Trippi, and E. Turban. *Applying Neural Networks*, pages 47–72. Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance. Probus, Chicago, 1992. 8, 11, 34

M. Krips, T. Lammert, and A. Kummert. FPGA implementation of a neural network for a real-time hand tracking system. *First IEEE Internation Workshop on Electronic Design, Test and Applications, Proceedings*, pages 313–317, 2002. URL ISIP:000174705900056. 17

Kin Keung Lai, Lean Yu, Wei Huang, and Shouyang Wang. *Multistage Neural Network Metalearning with Application to Foreign Exchange Rates Forecasting*, pages 338–347. 2006. URL http://www.springerlink.com/content/e0077462w35nn443. 10

Dae Won Lee, Hyung Jun Choi, and Jaewook Lee. A regularized line search tunneling for efficient neural network learning. *Advances in Neural Networks - ISNN 2004, Pt 1*, 3173:239–243, 2004. URL ISI:000223492600041. 10, 43, 75

Jaewook Lee. A novel three-phase trajectory informed search methodology for global optimization. *Journal of Global Optimization*, 38(1):61–77, 2007. URL ISI:000245887100004. 10, 43, 75

K.Y. Lee, Y.T. Cha, J.H. Park, M.S. Kurzyn, D.C. Park, and O.A. Mohammed. Short-Term Load Forecasting Using An Artificial Neural Network. *IEEE Transactions on Power Systems*, 7(1):124–132, 1992. URL ISI:A1992HQ19600016. 13

Yee Chun Lee, Gary Doolen, Hudong Hen Chen, Guo Zheng-Zhen Sun, T. Maxwell, H. Y. Lee, and Clyde Lee Giles. Machine Learning using a Higher-Order Correlation Network. *Physica D*, 22(1-3):276–306, 1986. URL ISI:A1986F340700020. 35, 75

Weimin Li, Jianwei Liu, and Jiajin Le. Using GARCH-GRNN Model to Forecast Financial Time Series. In P Yolum, editor, *International Symposium on Computer and Information Sciences*, pages 565 – 574. Springer-Verlag, 2005. 39

Faa-jeng Lin, Senior Member, Ying-chih Hung, and Syuan-yi Chen. FPGA-Based Computed Force Control System Using Elman Neural Network for Linear Ultrasonic Motor.

IEEE Transactions on Industrial Electronics, 56(4):1238–1253, April 2009. ISSN 0278-0046. doi: 10.1109/TIE.2008.2007040. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4663727. 93

R. Lippmann. An introduction to computing with neural nets. *IEEE Acoustics, Speech, and Signal Processing Magazine.*, 4(2):4–22, 1987. 5

J. C. Lopez-Garcia, M. A. Moreno-Armendariz, J. Riera-Babures, M. Balsi, and X. Vilasis-Cardona. Real time vision by FPGA implemented CNNs. In *European Conference on Circuit Theory and Design*, volume 1, pages 281–284, 2005. ISBN 0-7803-9066-0. URL ISIP:000234406900070. 18, 74

J.C. Lv and Z. Yi. An improved backpropagation algorithm using absolute error function. *Advances in Neural Networks - ISNN 2005, Pt 1, Proceedings*, 3496:585–590, 2005. URL ISI:000230166900093. 11

M Lynch. The use of feedforward neural networks for machine vision calibration. *International Journal of Production Economics*, 60-61(3):479–489, April 1999. ISSN 09255273. doi: 10.1016/S0925-5273(98)00199-6. URL http://linkinghub.elsevier.com/retrieve/pii/S0925527398001996. 64

L.P. Maguire, T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin. Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing*, 71(1-3):13–29, 2007. URL WOS:000251500600003. 18, 25, 74

N Maleki, A Safavi, and F Sedaghatpour. Single-step calibration, prediction and real samples data acquisition for artificial neural network using a CCD camera. *Talanta*, 64 (4):830–5, November 2004. ISSN 1873-3573. doi: 10.1016/j.talanta.2004.02.041. URL http://www.ncbi.nlm.nih.gov/pubmed/18969677. 60

Z. Q. Mao, D. R. Selviah, and J. E. Midwinter. Optical High-Order Feedback Neural Network Using an Optical Fiber Amplifier. *Artificial Neural Networks*, 1-2(2):1479–1482, 1992. 7

D. Marchiori and M. Warglien. Predicting human interactive learning by regret-driven neural networks. *Science*, 319(5866):1111–1113, 2008. URL WOS:000253311700049. 76

Donald Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Math*, 11(2):431–441, 1963. 11, 12, 43, 66, 75, 93

Timothy Masters. *Practical Neural Network Recipies in C++*. Morgan Kaufmann, New York, 1st edition, 04 1993. ISBN 0124790402. 5, 10, 11, 14, 42, 75, 76, 82

Mathworks. Matlab, 2009. URL http://www.mathworks.com/. 66

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity (reprinted from bulletin of mathematical biophysics, vol 5, pg 115-133, 1943). *Bulletin of Mathematical Biology*, 52(1-2):99–115, 1943. URL ISI:A1990DC78200005. 5, 34

Paul D. McNelis. *Neural Networks in Finance: Gaining Predictive Edge in the Market (Academic Press Advanced Finance)*. Academic Press, San Diego, CA, 2005. ISBN 0124859674. URL http://www.amazon.com/Neural-Networks-Finance-Predictive-Academic/dp/0124859674. 42

Gunter Meissner and Noriko Kawano. Capturing the volatility smile of options on high-tech stocksA combined GARCH-neural network approach. *Journal of Economics and Finance*, 25(3):276–292, September 2001. ISSN 1055-0925. doi: 10.1007/BF02745889. URL http://www.springerlink.com/index/10.1007/BF02745889. 39

Qurban Memon and Khan Sohaib. Camera calibration and three-dimensional world reconstruction of stereo-vision using neural networks. *International Journal of Systems Science*, 32(9):1155–1159, September 2001. ISSN 0020-7721. doi: 10.1080/00207720010024276. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/00207720010024276&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3. 64

R. L. Milidiu, R. J. Machado, and R. P. Renteria. Time-series forecasting through wavelets transformation and a mixture of expert models. *Neurocomputing*, 28, 1999. URL ISI:000082665400011. 14

Jiang Minghu, Zhu Xiaoyan, Lin Ying, Yuan Baozong, Tang Xiaofang, and Lin Biqin. Analysis of the effects of quantization in high-order function neural network. *WCC 2000 - ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000*, pages 1629–1632, 2000. doi: 10.1109/ICOSP.2000.893413. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=893413. 74

J. Miteran, S. Bouillant, and E. Bourennane. SVM approximation for real-time image segmentation by using an improved hyperrectangles-based method. *Real-Time Imaging*, 9(3):179–188, 2003. URL WOS:000186021100003. 17

J. Moody. *The Effective Number of Parameters, an Analysis of Generalization and Regularization in Nonlinear Learning Systems*. Morgan Kauffmann, San Mateo, CA, 1992. 9

J. Moody, J. Utans, and A.N. Refenes. *Architecture Selection Strategies for Neural Networks, Application to Corporate Bond Rating Prediction*. John Wiley & Sons, New York, 1994. 9

G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), Apr 1965. URL ftp://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf. 24

Yuichi Motai and Akio Kosaka. HandEye Calibration Applied to Viewpoint Selection for Robotic Vision. *IEEE Transactions on Industrial Electronics*, 55(10):3731–3741, October 2008. ISSN 0278-0046. doi: 10.1109/TIE.2008.921255. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4475519. 93

Y.M. Mustafah, A.W. Azman, A. Bigdeli, and B.C. Lovell. An automated face recognition system for intelligence surveillance: Smart camera recognizing faces in the crowd. In *First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 142–147, Vienna, September 2007. URL ISIP:000254383500020. 20

Tarek M. Nabhan and Albert Y. Zomaya. Toward generating neural-network structures for function approximation. *Neural Networks*, 7(1):89–99, 1994. URL ISI:A1994MY13700007. 9, 42

McCord M. Nelson and W. T. Illingworth. *A Practical to Guide to Neural Nets*. Reading, MA, 1991. 8

Derrick Nguyen and Bernard Widrow. Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights. *IJCNN International Joint Conference on Neural Networks*, 1-3:C21–C26, 1990. URL ISIP:A1990BS06N00284. 11, 43, 66, 76

Genya Ogawa, Katsuyuki Kise, Tsuyoshi Torii, and Tomoharu Nagao. Onboard Evolutionary Risk Recognition System for AutomobilesToward the Risk Map System. *IEEE*

Transactions on Industrial Electronics, 54(2):878–886, April 2007. ISSN 0278-0046. doi: 10.1109/TIE.2007.891654. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4126820. 93

E.M. Ortigosa, P.M. Ortigosa, A. Canas, E. Ros, R. Agis, and J. Ortega. Fpga implementation of multi-layer perceptrons for speech recognition. *Field-Programmable Logic and Applications, Proceedings*, 2778:1048–1052, 2003. URL WOS:000186329700117. 18, 25

J. Z. Ou and V. K. Prasanna. PyGen: A MATLAB/Simulink based tool for synthesizing parameterized and energy efficient designs using FPGAs. In *FCCM'04 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 47–56, 2004. URL ISIP:000225131700005. 25, 79

J. Z. Ou and V. K. Prasanna. MATLAB/Simulink Based Hardware/Software Co-Simulation for Designing Using FPGA Configured Soft Processors. In *Parallel and Distributed Processing Symposium*, pages 148b–148b, April 2005. URL http://ieeexplore.ieee.org/search/searchresult.jsp?history=yes&queryText=((matlab%2Fsimulink+based+hardware%2Fsoftware+co-simulation+for+designing+using+fpga+configured+soft+processors)%3Cin%3Emetadata). 25, 79, 97

Yoh Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison Wesley, Boston, MA, USA, June 1989. ISBN 0201125846. 35, 75

I. Papakonstantinou, D. R. Selviah, R. C. A. Pitwon, and D. Milward. Low-Cost, Precision, Self-Alignment Technique for Coupling Laser and Photodiode Arrays to Polymer Waveguide Arrays on Multilayer PCBs. *IEEE Transactions on Advanced Packaging*, 31(3):502–511, 2008. 81

I. Papakonstantinou, R. James, and D. Selviah. Radiation and bound mode propagation in rectangular multimode dielectric channel waveguides with sidewall roughness. *IEEE Journal of Lightwave Technology*, 27(18):4151–4163, 12 2009. 81

Nicos G. Pavlidis, Dimitris K. Tasoulis, Vassilis P. Plagianakos, and Michael N. Vrahatis. Computational intelligence methods for financial time series modeling. *International Journal of Bifurcation and Chaos*, 16(7):2053–2062, July 2006. URL ISI:000240860400013. 10, 13, 42

L. Personnaz, I. Guyon, and G. Dreyfus. High-Order Neural Networks: Information Storage without Errors. *Europhysics Letters*, 4(8):863–867, 1987. URL http://www.iop.org/EJ/abstract/0295-5075/4/8/001/. 75

A. Petrowski, G. Dreyfus, and C. Girault. Performance analysis of a pipelined backpropagation parallel algorithm. *IEEE Transactions on Neural Networks*, 4(6):970–981, 1993. ISSN 1045-9227. 22

Thal Quynh Phong, Radu Horaud, Adnan Yassine, and Pham Dinh Tao. Object pose from 2-D to 3-D point and line correspondences. *International Journal of Computer Vision*, 15(3):225–243, July 1995. ISSN 0920-5691. doi: 10.1007/BF01451742. URL http://www.springerlink.com/index/10.1007/BF01451742. 64

S.W. Piche. The selection of weight accuracies for madalines. *IEEE Transactions on Neural Networks*, 6(2):432–445, 1995. ISSN 1045-9227. URL ISI:A1995QJ92200013. 73, 91

F.I. Pico, A.G. Olivo, F.G. Crespi, and A. Camara. An electronic reconfigurable neural architecture for intrusion detection. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Pt 2, Proceedings*, 3562:376–384, 2005. ISSN 0302-9743. URL ISI:000230386700039. 17

Ser Huang Poon and Clive W. J. Granger. Forecasting volatility in financial markets: A review. *Journal of Economic Literature*, 41(2):478–539, 2003. URL ISI:000183622600003. 40

William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2 edition, October 1992. ISBN 0521431085. 11, 75

D. Psaltis, D. Brady, and K. Wagner. Adaptive optical networks using photorefractive crystals. *Applied Optics*, 27(9):1752–1759, May 1988. ISSN 0740-3224. 6

M. Qi and G.S. Maddala. Economic factors and the stock market: A new perspective. *Journal of Forecasting*, 18(3):151–166, May 1999. URL ISI:000080749800001. 13

M. Qi and G.P. Zhang. An investigation of model selection criteria for neural network time series forecasting. *European Journal of Operational Research*, 132(3):666–680, 2001. URL ISI:000169231600015. 10

Min Qi, G. S. Maddala, and C. R. Rao. *Financial applications of artificial neural networks*, pages 525–529. Elsevier Science, Amsterdam, 1996. 32

NaoLin Qiu and De Ma Song. The nonparametric approach for camera calibration. In *Proceedings of IEEE International Conference on Computer Vision*, pages 224–229, Cambridge, MA, 1995. IEEE Comput. Soc. Press. ISBN 0-8186-7042-8. doi: 10.1109/ICCV.1995.466782. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=466782. 58, 61

Recognetics.Inc. Bringing infinite intelligence to the application of pattern recognition. http://www.recognetics.com/, June 2008. URL http://www.recognetics.com/. 21

J.M. Reed and S. Hutchinson. Image fusion and subpixel parameter estimation for automated optical inspection of electronic components. *IEEE Transactions on Industrial Electronics*, 43(3):346–354, June 1996. ISSN 02780046. doi: 10.1109/41.499806. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=499806. 93

a. Rosado-Munoz, M. Bataller-Mompean, E. Soria-Olivas, C. Scarante, and J. Guerrero-Martinez. FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches. *IEEE Transactions on Industrial Electronics*, 2009. ISSN 0278-0046. doi: 10.1109/TIE.2009.2023641. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5067312. 97

Z. Ruan, J.G. Han, and Y.Z. Han. BP neural network implementation on real-time reconfigurable fpga system for a soft-sensing process. volume 1-3, pages 959–963, 2005. URL ISIP:000236575101073. 22

David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, D. E. Rumelhart, and J. L. McClelland. *Learning Internal Representations by Error Propagation*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986. 5, 11, 34

S. Sahin, Y. Becerikli, and S. Yazici. Neural network implementation in hardware using FPGAs. In *NIP, Neural Information Processing*, volume 4234 Part 3, pages 1105–1112, Berlin, 2006. Springer Verlag. ISBN 0302-9743. URL ISI:000241759000122. 17, 18, 22, 73, 95

G. Schwartz. Estimating dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978. URL ISI:A1978EQ63300014. 9

S. Segvic and S. Ribaric. Determining the absolute orientation in a corridor using projective geometry and active vision. *IEEE Transactions on Industrial Electronics*, 48(3):696–710, June 2001. ISSN 02780046. doi: 10.1109/41.925597. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=925597. 93

D. R. Selviah. Neural computer uses optical fingers. *Physics World*, 7(7):29–30, 1994. ISSN 0953-8585. 7

D. R. Selviah, J. E. Midwinter, A. W. Rivers, and K. W. Lung. Correlating matched-filter model for analysis and optimization of neural networks. *IEE Proceedings-F Radar and Signal Processing*, 136(3):143–148, 1989. ISSN 0956-375X. 7

D. R. Selviah, Z. Q. Mao, and J. E. Midwinter. Optoelectronic High-Order Feedback Neural Network. *Electronics Letters*, 26(23):1954–1955, 1990. ISSN 0013-5194. 7

D. R. Selviah, Z. Q. Mao, and J. E. Midwinter. *A high order feedback net (HOFNET) with variable non-linearity*, pages 59–63. IEEE, IEEE, London, UK, 11 1991. URL http://eprints.ucl.ac.uk/archive/00002684/. 6, 35, 61

David R. Selviah and Janti Shawash. *Generalized Correlation Higher Order Neural Networks for Financial Time Series Prediction*, chapter 10, pages 212–249. Artificial Higher Order Neural Networks for Artificial Higher Order Neural Networks for Economics and Business. IGI Global, Hershey, PA, 2008. 32, 37, 42, 58, 61, 74, 75

Janti Shawash and David R. Selviah. Artificial Higher Order Neural Network Training on Limited Precision Processors. In *Artificial Higher Order Neural Networks for Computer Science and Engineering: Trends for Emerging Applications*, chapter 14, page 378. Information Science Publishing, Hershey, PA, 2010. ISBN 1615207112. URL http://www.amazon.com/Artificial-Networks-Computer-Science-Engineering/dp/1615207112/ref=sr_1_1?ie=UTF8&s=books&qid=1267789032&sr=8-1. 94

F Shi, X Zhang, and Y Liu. A new method of camera pose estimation using 2D3D corner correspondence. *Pattern Recognition Letters*, 25(10):1155–1163, July 2004. ISSN 01678655. doi: 10.1016/j.patrec.2004.03.010. URL http://linkinghub.elsevier.com/retrieve/pii/S0167865504000790. 64

D.C. Shin and C.L. Nikias. *Adaptive Noise Canceler for Narrowband/Wideband Interferences Using Higher-Order Statistics*, volume 3. Minneapolis, MN, USA, April 1993. ISBN 0-7803-0946-4. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=319510&isnumber=7686. 6

Liu Shoushan, Chen Yan, Xu Wenshang, and Zhang Tongjun. A single layer architecture to FPGA implementation of BP artificial neural network. In *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010)*, pages 258–264. IEEE, March 2010. ISBN 978-1-4244-5192-0. doi: 10.1109/CAR.2010.5456553. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5456553. 103

C. Slim. Forecasting the volatility of stock index returns: A stochastic neural network approach. *Computational Science and Its Applications - ICCSA 2004, Pt 3*, 3045, 2004. URL ISI:000221852900098. 11, 45

Murray Smith. *Neural networks for statistical modeling*. Van Nostrand Reinhold Company, 1993. 15, 42

A.B. Soares, A.A. Susin, and L.V. Guimaraes. *Automatic Generation of Neural Networks for Image Processing*. IEEE International Symposium on Circuits and Systems. 2006. URL ISIP:000245413503146. 17, 18, 20, 21

M. Stevenson, R. Winter, and B. Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Transactions on Neural Networks*, 1(1):71–80, March 1990. URL MEDLINE:18282824. 73, 91

N.R. Swanson and H. White. A model-selection approach to assessing the information in the term structure using linear-models and artificial neural networks. *Journal of Business & Economic Statistics*, 13(3):265–275, 1995. URL ISI:A1995RF32000003. 9

A. Tahai, S. Walczak, J.T. Rigsby, P. Siegel, K. Omer, A. deKorvin, and A. Zebda. *Improving Artificial Neural Network Performance through Input Variable Selection*, pages 277–292. JAI Press, Stamford, CT, 1998. 8

Y. Taright and M. Hubin. FPGA implementation of a multilayer perceptron neural network using VHDL. volume 1-2, pages 1311–1314, 1998. URL ISIP:000081007500319. 21

Brian J. Taylor. *Methods and Procedures for the Verification and Validation of Artificial Neural Networks*. Springer US, 2009. ISBN 1441939350. URL http://www.amazon.com/Procedures-Verification-Validation-Artificial-Networks/dp/1441939350. 61

W. G. Tomek and S. F. Querin. Random-processes in prices and technical analysis. *Journal of Futures Markets*, 4(1):15–23, 1984. URL ISI:A1984SF23400002. 41

Tomshardware.co.uk. Nvidia Takes A Big Hit In The Second Quarter. http://www.tomshardware.co.uk/ Nvidia-Q22008,news-28930.html, August 2008. URL http://www.tomshardware.co.uk/Nvidia-Q22008,news-28930.html. 20

R. R. Trippi, E. Turban, R. R. Trippi, and E. Turban. *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real-World Performance*. Chicago, 1993. 6

R. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, 1987. ISSN 0882-4967. doi: 10.1109/JRA.1987.1087109. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1087109. 59

Everine B van de Kraats, Graeme P Penney, Dejan Tomazevic, Theo van Walsum, and Wiro J Niessen. Standardized evaluation methodology for 2-D-3-D registration. *IEEE transactions on medical imaging*, 24(9):1177–89, September 2005. ISSN 0278-0062. doi: 10.1109/TMI.2005.853240. URL http://www.ncbi.nlm.nih.gov/pubmed/16156355. 64

S. Walczak and N. Cerpa. Heuristic principles for the design of artificial neural networks. *Information and Software Technology*, 41(2):107–117, 1999. URL ISI:000079065800005. 8, 9, 11, 42

Steven Walczak. Neural network models for a resource allocation problem. *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics*, 28(2):276–284, 1998. URL ISI:000072641600016. 13, 47

Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*, 17(4):203–222, 2001. URL ISI:000167939400009. 10, 34

A. S. Weigend, M. Mangeas, and A. N. Srivastava. Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting. *International Journal of Neural Systems*, 6(4), 1995. URL ISI:A1995TZ33600001. 14

A.S. Weigend, N.A. Gershenfeld, A.S. Weigend, and N.A. Gershenfeld. *Time Series Prediction, Forecasting the Future and Understanding the past, Proceedings in the Sfi Studies of Complexity*. Addison-Wesley, Reading, MA, 1994. 13

G Wells. Vision-based robot positioning using neural networks. *Image and Vision Computing*, 14(10):715–732, December 1996. ISSN 02628856. doi: 10.1016/0262-8856(96)89022-6. URL http://dx.doi.org/10.1016/0262-8856(96)89022-6. 61

J Weng, P Cohen, and M Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):965–980, 1992. ISSN 01628828. doi: 10.1109/34.159901. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=159901. 58

H. White. Economic prediction using neural networks: The case of IBM daily stock returns. volume 2 of *Proceedings of the IEEE International Conference on Neural Networks*, pages 451–8, 1988. 13, 14

H. White. Connectionist Nonparametric Regression - Multilayer Feedforward Networks Can Learn Arbitrary Mappings. *Neural Networks*, 3(5):535–549, 1990. URL ISI:A1990EE89500004. 7

H. White, A.R. Gallant, K. Hornik, M. Stinchcombe, and J. Wooldridge. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Cambridge, MA, 1992. 13

Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electric Show and Convention*, volume Part 4, pages 96–104, 1960. 5

B.M. Wilamowski, N.J. Cotton, Okyay Kaynak, and G. Dundar. Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks. *IEEE Transactions on Industrial Electronics*, 55(10):3784–3790, October 2008. ISSN 0278-0046. doi: 10.1109/TIE.2008.2003319. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4602720. 93

R.L. Wilson and R. Sharda. Bankruptcy Prediction Using Neural Networks. *Decision Support Systems*, 11(5):545–557, 1994. URL ISI:A1994NR16100011. 11

E. Won. A hardware implementation of artificial neural networks using field programmable gate arrays. *Nuclear Instruments & Methods in Physics Research Section A-Accelerators Spectrometers Detectors and Associated Equipment*, 581(3):816–820, 2007. ISSN 0168-9002. URL ISI:000251148000026. 17, 21, 22

Dong-Min Woo and Dong-Chul Park. Implicit Camera Calibration Using MultiLayer Perceptron Type Neural Network. In *First Asian Conference on Intelligent Information*

and Database Systems, pages 313–317, Vietnam, 2009. URL http://www.computer.org/portal/web/csdl/doi/10.1109/ACIIDS.2009.11. 60

Y.L. Wu, J.Y. Zhou, and H.H. Chen. Real-time infrared imaging system based on FPGA. *14Th International Conference on Mechatronics and Machine Vision in Practice 2007, Proceedings*, pages 97–99, 2007. URL ISIP:000252829700021. 17

Wen-Fang Xie, Zheng Li, Xiao-Wei Tu, and Claude Perron. Switching Control of Image-Based Visual Servoing With Laser Pointer in Robotic Manufacturing Systems. *IEEE Transactions on Industrial Electronics*, 56(2):520–529, February 2009. ISSN 0278-0046. doi: 10.1109/TIE.2008.2003217. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4602717. 93

Xilinx. AccelDSP Synthesis Tool. http://www.xilinx.com/tools/dsp.htm, 06 2008a. 25, 79, 92, 97

Xilinx. System Generator for DSP. http://www.xilinx.com/ise/optional_prod/ system_generator.htm, June 2008b. URL http://www.xilinx.com/ise/optional_prod/system_generator.htm. 25

Xilinx. ISE Foundation. http://www.xilinx.com/ise/ logic_design_prod/foundation.htm, June 2008c. URL http://www.xilinx.com/ise/logic_design_prod/foundation.htm. 26

Xilinx. Aerospace and defense products, June 2008d. URL http://www.xilinx.com/products/silicon_solutions/aero_def/products.htm. 18

Yahoo! Finance. FTSE100, NASDAQ. http://finance.yahoo.com/, September 2009. URL http://finance.yahoo.com/. 41

F. Yang and M. Paindavoine. Implementation of an RBF neural network on embedded systems: Real-time face tracking and identity verification. *IEEE Transactions on Neural Networks*, 14(5):1162–1175, 2003. URL WOS:000186478900017. 17, 20, 21

Houman Zarrinkoub. Fixed-point Signal Processing with Matlab and Simulink, April 2006. URL http://www.mathworks.com/webex/recordings/fixedpt_042006/fixedpt_042006.html. 78, 92, 98

D. Zhang, H. Li, and S.Y. Foo. A simplified FPGA implementation of neural network algorithms integrated with stochastic theory for power electronics applications. *IECON 2005: Thirty-First Annual Conference of the IEEE Industrial Electronics Society, Vols 1-3*, pages 1018–1023, 2005a. URL ISIP:000236873601023. 19

G.P. Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, 2003. URL ISI:000180567700009. 13

Guoqiang Zhang and Michael Y. Hu. Neural network forecasting of the british pound us dollar exchange rate. *Omega-International Journal of Management Science*, 26(6): 786–786, 1998. URL ISI:000077631400011. 8, 32

Guoqiang P. Zhang, Eddy B. Patuwo, and Michael Y. Hu. Nonlinear time series forecasting with artificial neural networks. *Decision Sciences Institute*, 1-3:1023–1025, 1998. URL ISIP:000083105000450. 10

M. Zhang, S.X. Xu, and J. Fulcher. Neuron-adaptive higher order neural-network models for automated financial data modeling. *IEEE Transactions on Neural Networks*, 13(1): 188–204, 2002. URL WOS:000173440100016. 7

Y.M. Zhang, L. Guo, L.Y. Wu, and C.B. Feng. On stochastic neutral neural networks. *Advances in Neural Networks - ISNN 2005, Pt 1, Proceedings*, 3496:69–74, 2005b. URL ISI:000230166900010. 17

Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. ISSN 01628828. doi: 10.1109/34.888718. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=888718. 99

Zhengyou Zhang. A Flexible New Technique for Camera Calibration. Technical report, Microsoft Research, Redmond, WA, 1998. URL http://research.microsoft.com/en-us/um/people/zhang/Calib/. 64, 66

Zhengyou Zhang. Camera calibration with one-dimensional objects. *IEEE transactions on pattern analysis and machine intelligence*, 26(7):892–9, July 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.21. URL http://www.ncbi.nlm.nih.gov/pubmed/18579947. 59

J. H. Zhu and P. Sutton. *FPGA implementations of neural networks - A survey of a decade of progress*, volume 2778, pages 1062–1066. Springer, Berlin/Heidelberg, 09 2003a. ISBN ISSN 0302-9743 (Print) 1611-3349 (Online) ISBN 978-3-540-40822-2. URL WOS:000186329700120. 94, 95

J.H. Zhu and P. Sutton. FPGA implementations of neural networks - A survey of a decade of progress. In *Field-Programmable Logic and Applications Proceedings*, volume 2778, pages 1062–1066, 2003b. URL WOS:000186329700120. 22, 73