

Wave Overtopping Prediction Using Global-Local Artificial Neural Networks

*A Thesis Submitted in Partial Fulfilment of the
Requirements for the Degree of Doctor of Philosophy.*

March 2006

By
David Christopher Wedge
Department of Computing and Mathematics
Manchester Metropolitan University

Contents

List of Tables	vii
List of Figures	ix
Abstract	xii
Declaration	xiii
Acknowledgements	xiv
Notation	xv
Abbreviations used	xviii
1 Introduction	1
1.1 Historical Overview	1
1.2 Hazard Levels and Wave Overtopping Rates	3
1.3 Empirical Curve-fitting	6
1.4 Numerical Modelling	7
1.5 Artificial Neural Networks	9
1.5.1 Introduction	9
1.5.2 Applications of ANNs	13
1.5.3 Early Research in ANNs	15
1.5.4 Multi-layer Perceptrons	18
1.5.5 Alternative training methods	25
1.5.6 Global training methods	27
1.5.7 Radial Basis Function networks	30
1.5.8 Hybrid Neural Networks	33
1.6 Artificial Neural Networks in Hydroinformatics	34

1.6.1	Freshwater Applications of ANNs	34
1.6.2	Coastal Applications of ANNs	38
1.6.3	Design Issues	40
1.7	CLASH and the development of a hybrid neural network	42
1.8	Overview	44
2	Mathematical Techniques for Neural Networks	46
2.1	Transfer Functions	46
2.1.1	Pseudo-linear transfer functions	47
2.1.2	Radial basis transfer functions	48
2.2	Gradient Descent	50
2.2.1	Adaptive linear elements	50
2.2.2	Multi-layer perceptrons	53
2.2.3	Modifications to back-propagation	54
2.3	Levenberg-Marquardt method	56
2.4	RBF centre selection	57
2.4.1	Fully interpolated networks	57
2.4.2	Least squares solution	58
2.4.3	Forward Selection	59
2.4.4	Orthogonal Least Squares	59
2.5	Regularisation	60
2.6	Summary	61
3	The CLASH Dataset	62
3.1	Data collection	62
3.2	Data selection and pre-processing	65
3.3	The nature of the CLASH dataset	75
3.3.1	Marginal distributions	75
3.3.2	Data clustering	76
3.3.3	Exponential relationships	77
3.3.4	Linearity of the CLASH dataset	79
3.4	Summary	80
4	CLASH prediction using MLP Networks	82
4.1	Introduction	82
4.2	Gradient Descent Pilot Studies	82

4.2.1	Pilot study 1: Learning rates and weight update mode	83
4.2.2	Pilot study 2: Stopping criterion	87
4.2.3	Pilot study 3: weight initialisation range	96
4.2.4	Pilot study 4: Output neuron transfer function	97
4.2.5	Pilot Study 5: Momentum coefficient	98
4.2.6	Pilot Study 6: Levenberg-Marquardt method	99
4.3	Model selection method	102
4.4	Method	105
4.5	Results and Discussion	105
4.5.1	Back-propagation	106
4.5.2	Levenberg-Marquardt training	112
4.5.3	Results summary	114
4.5.4	Speed and memory comparisons	117
5	CLASH prediction using RBF networks	119
5.1	Width Pilot Study	119
5.2	Method	122
5.3	Results and Discussion	124
5.3.1	Results without regularisation	124
5.3.2	Results with regularisation	126
5.3.3	Results with gradient descent optimisation	131
5.3.4	Speed and memory comparisons	132
5.4	Summary and a comparison with MLP networks	132
6	GL-ANN theory and algorithm	136
6.1	Background	136
6.1.1	Mathematics	136
6.1.2	Cognitive psychology	137
6.1.3	Computer Science	137
6.2	The ideas behind GL-ANNs	138
6.3	GL-ANN Algorithm	140
6.4	Summary	142
7	CLASH prediction using GL-ANN algorithm	144
7.1	Method	144
7.2	Results	146

7.2.1	Two-step algorithm	146
7.2.2	Three-step algorithm	149
7.2.3	Hybrid networks trained with regularisation	150
7.2.4	Speed and memory comparisons	153
7.3	Summary	153
8	GL-ANN Evaluation using Benchmark Datasets	159
8.1	Description of the benchmark datasets	159
8.1.1	Synthetic Datasets	160
8.1.2	Measured Datasets	164
8.2	Method	170
8.3	Results	174
8.3.1	Test errors	174
8.3.2	Optimum architectures	176
8.3.3	RBF spreads	177
8.3.4	RBF output weights	178
8.4	Summary	179
9	Conclusions and further work	181
9.1	Summary and conclusions	181
9.2	Original contributions	183
9.3	Further work	184
	Bibliography	187
	Appendices	208
A	Empirical Curve-Fitting	208
A.1	Besley	208
A.2	Van der Meer and Janssen	209
A.3	Hedges and Reis	210
B	The Conjugate Gradient Method	212
C	Comparisons with alternative methods	214
C.1	Synthetic datasets	214
C.2	Measured datasets	215

List of Tables

3.1	Variables in the CLASH database	64
3.2	Correlation coefficients for the CLASH data parameters	72
3.3	Parameters used in ANN training	73
4.1	Percentage of training errors that have converged within 5000 epochs .	89
4.2	Number of epochs required for convergence in training error, averaged over 10 runs	89
4.3	Percentage of test errors that have converged within 5000 epochs . . .	90
4.4	Number of epochs required for convergence in test error, averaged over 10 runs	90
4.5	Training parameters for slow, medium and fast convergence	91
4.6	Test MSEs for different hidden layer sizes and training-test splits . . .	104
4.7	Average errors for MLP with linear output neuron and BP training . .	107
4.8	Optimum errors for MLP with linear output neuron and BP training .	109
4.9	Average errors for MLP with sigmoid output neuron and BP training .	110
4.10	Optimum errors for MLP with sigmoid output neuron and BP training	111
4.11	Average errors for MLP with linear output neuron and L-M training .	112
4.12	Optimum errors for MLP with linear output neuron and L-M training .	113
4.13	Average errors for MLP with sigmoid output neuron and L-M training	114
4.14	Optimum errors for MLP with sigmoid output neuron and L-M training	116
4.15	Summary statistics for MLP training	117
5.1	Verification errors for networks containing RBFs of various widths, averaged for networks of the same size	122
5.2	Best errors achievable with RBF networks	126
5.3	Optimum errors for RBF networks with spread=0.4 trained with regu- larisation	129

5.4	Optimum errors for RBF networks with spread=0.6 trained with regularisation	130
5.5	Summary of the results of training RBF networks with the CLASH dataset	135
7.1	Best errors achievable with two-step GL-ANN	149
7.2	Optimum errors for hybrid networks with spread=0.4 containing 6 sigmoid neurons trained with regularisation	151
7.3	Performance indicators for MLP, RBF and GL-ANN networks, averaged across 30 networks	154
7.4	Performance indicators for the best performing MLP, RBF and GL-ANN networks	155
8.1	Summary of the synthetic benchmark datasets	166
8.2	Summary of the measured benchmark datasets	172
8.3	Mean square errors for the synthetic benchmark datasets	174
8.4	Mean square errors for the measured benchmark datasets	175
8.5	Number of hidden layer neurons for the synthetic benchmark datasets	176
8.6	Number of hidden layer neurons for the measured benchmark datasets	176
8.7	Synthetic datasets: optimum RBF spreads for pure RBF and hybrid networks	178
8.8	Measured datasets: optimum RBF spreads for pure RBF and hybrid networks	178
8.9	Output weights of RBF neurons in pure RBF and GL-ANN networks	178
C.1	MSEs for the synthetic datasets obtained using PRBFN, RT-RBF and GL-ANN algorithms	215
C.2	Relative error, as a percentage, for the measured benchmark datasets, trained using Quinlan's methods and GL-ANNs	215

List of Figures

1.1	Safe overtopping limits	4
1.2	Diagram of an artificial neuron	10
1.3	Diagram of an artificial neuron network	10
1.4	Diagram showing the neural network training process	12
1.5	Hopfield network architecture	18
1.6	Graph of a bipolar sigmoid function	20
1.7	Overfitting caused by oversized networks	22
1.8	Overfitting caused by overtraining	23
1.9	Cascade correlation architecture	26
1.10	Graph of a Gaussian radial basis function	31
1.11	The three-step training process used by GL-ANNs	43
1.12	The interaction between data and models	44
2.1	Linear and pseudo-linear transfer functions	49
2.2	Radial basis transfer functions	51
3.1	Cross-sectional view showing sea-wall structural parameters	64
3.2	Saturation in a sigmoid transfer function	66
3.3	Raw and transformed Normal probability plots for T_0	68
3.4	Raw and transformed Normal probability plots for h_{t0}	69
3.5	Raw and transformed Normal probability plots for B_{t0}	70
3.6	Raw and transformed Normal probability plots for q_0	71
3.7	Schematisation of a structure with a non-horizontal berm	74
3.8	k-nearest neighbour density estimates for the CLASH dataset	78
3.9	Besley predictions compared to measured overtopping rates	79
3.10	Plot of studentised residuals vs. estimated q_0 after linear regression on the CLASH dataset	80

4.1	Average test MSEs for networks trained with stochastic weight updates and 4 different learning rates	84
4.2	Average training MSEs for networks trained with stochastic weight updates and 4 different learning rates	85
4.3	Progression in test errors with batch weight updates	86
4.4	Test errors achieved with batch weight updates and $\eta = 0.0002$	87
4.5	Number of epochs required for convergence, assuming optimum learning rate and stochastic weight updates	91
4.6	Training progression within a slow training regime	93
4.7	Training progression within a medium training regime	94
4.8	Training progression within a fast training regime	95
4.9	Average test MSEs for different weight initialisation ranges	96
4.10	Number of epochs required to achieve convergence for different weight initialisation ranges	97
4.11	Average test MSEs for different output neuron transfer functions	98
4.12	Number of epochs required to achieve convergence with and without momentum	99
4.13	Average test MSEs with and without momentum	100
4.14	Average training and test MSEs with Levenberg-Marquardt algorithm	101
4.15	Progression in test errors during Levenberg-Marquardt training of a MLP network containing 8 neurons	101
4.16	Training, verification and test errors after BP training	108
4.17	Training, verification and test errors after L-M training	115
5.1	Variation in verification error with hidden layer size for RBF networks with various spread values	121
5.2	Dependence of verification errors on spread parameter	123
5.3	Dependence of hidden layer size on spread parameter	123
5.4	Training, verification and test errors for RBF networks with 2 different spreads	125
5.5	Verification and test errors of RBF networks as a function of regularisation parameter	127
5.6	Error progression for RBF network with spread 0.4 and $\lambda = 10^{-4}$	128
5.7	Error progression for RBF network with spread 0.6 and $\lambda = 10^{-4}$	128
5.8	Verification errors for RBF networks trained with an optimisation step	131

6.1	Diagrammatic representation of the GL-ANN training process	140
6.2	Flow chart summarising the GL-ANN algorithm	143
7.1	Progression in test error during the optimisation of hybrid networks containing 5 sigmoid neurons and 85 RBF neurons	145
7.2	Errors for hybrid networks with spread 0.4 averaged across fixed ar- chitectures	147
7.3	Errors for hybrid networks with spread 0.4 averaged across variable architectures	147
7.4	Errors for hybrid networks with spread 0.6 averaged across fixed ar- chitectures	148
7.5	Errors for hybrid networks with spread 0.6 averaged across variable architectures	148
7.6	Errors for three-step GL-ANNs with near optimum architectures	149
7.7	Verification errors for hybrid networks trained with regularisation . . .	152
7.8	$q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing MLP network .	156
7.9	$q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing RBF network .	157
7.10	$q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing GL-ANN network	158
8.1	Data densities for the sine 1D dataset	162
8.2	Data densities for the sine 2D dataset	162
8.3	Data densities for the impedance dataset	163
8.4	Data densities for the Hermite dataset	163
8.5	Plot of studentised residuals vs. estimated q_0 for the sine 1D dataset .	164
8.6	Plot of studentised residuals vs. estimated q_0 for the sine 2D dataset .	165
8.7	Plot of studentised residuals vs. estimated q_0 for the impedance dataset	165
8.8	Plot of studentised residuals vs. estimated q_0 for the Hermite dataset .	166
8.9	Data densities for the housing dataset	168
8.10	Data densities for the servo dataset	168
8.11	Data densities for the cpu dataset	169
8.12	Data densities for the auto-mpg dataset	169
8.13	Plot of studentised residuals vs. estimated q_0 for the housing dataset .	170
8.14	Plot of studentised residuals vs. estimated q_0 for the servo dataset . .	171
8.15	Plot of studentised residuals vs. estimated q_0 for the cpu dataset . . .	171
8.16	Plot of studentised residuals vs. estimated q_0 for the auto-mpg dataset	172
8.17	Graph showing a Gaussian function (blue) and a shifted sine curve (red)	175

Abstract

The construction of sea walls requires accurate predictions of hazard levels. These are commonly expressed in terms of wave overtopping rates. A large amount of data related to wave overtopping has recently become available. Use of this data has allowed the development of artificial neural networks, which have the aim of accurately predicting wave overtopping rates. The available data cover a wide range of structural configurations and sea conditions. The neural networks created therefore constitute a unified, generic approach to the problem of wave overtopping prediction.

Neural network models are developed using two standard approaches: multi-layer perceptron (MLP) networks and radial basis function (RBF) networks. A novel hybrid approach is then developed. The hybrid networks combine the properties of MLP and RBF networks. This is achieved firstly through a hybrid architecture, which contains artificial neurons of the types used in both MLP and RBF networks. Secondly, the hybrid networks are trained using a hybrid algorithm which combines the gradient descent method usually associated with MLP networks with a more deterministic forward-selection-of-centres method commonly used by RBF networks. The hybrid networks are shown to have better generalisation properties with the overtopping dataset than have basic MLP or RBF networks. They have been named ‘global-local artificial neural networks’ (GL-ANNs) to reflect their ability to model both global and local variation in an input-output mapping.

The properties of GL-ANNs are explored further through the use of a number of benchmark datasets. It is shown that GL-ANNs often contain fewer neurons than the corresponding RBF networks and have less need of regularisation when setting inter-neuronal weights. Some criteria for determining whether the GL-ANN approach is likely to be beneficial for a particular dataset are also developed. Such datasets are seen to be those that have inter-parameter relationships that operate on both a local and global level. The overtopping dataset used within this study is seen to be typical of such datasets.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification at this or any other university or other institution of learning.

Apart from those parts of this thesis which contain citations to the work of others and apart from the assistance mentioned in the acknowledgements, this thesis is my own work.

David Christopher Wedge

Acknowledgements

I would like to thank my supervisors Dr. David Ingram, Dr. David McLean, Mr. Clive Mingham and Dr. Zuhair Bandar for their support and guidance throughout my studies. I would also like to thank the European CLASH project for compiling and making available the overtopping data, without which this research would not have been possible. I would like to express my gratitude to CERN for use of the linear algebra package ‘Colt’ within some of the procedures used in this study. I would like to thank Mr. Darren Dancey for numerous informal discussions that have proved invaluable over the last 3 years. Finally I would like to thank my wife Sian and my children Thomas, Matthew and Rebecca for their support throughout this project.

Notation

Vector and matrix quantities are indicated by **bold** type. Where possible, the use of the same symbol for more than one purpose has been avoided. However, in cases where the same symbol is widely used in more than one area of study this symbol has been retained. The symbols concerned are C , g , u , v , α , η , λ and σ . When these symbols are used the intended meaning should be clear from the context.

A	empirically determined coefficient in parametric regression
\mathbf{A}	design matrix of a partly interpolated neural network
A_c	armour crest freeboard of a structure
B	empirically determined coefficient in parametric regression
B_h	width of berm of a structure
B_t	width of toe of a structure
C	empirically determined coefficient in parametric regression, capacitance of an electrical circuit
C_d	discharge coefficient
d	Euclidean distance between input and weight vectors
E	squared error of an artificial neural network
f	general function
f_i	fan-in to a neuron
\mathbf{f}_j	a column in the full design matrix
$\tilde{\mathbf{f}}_j$	a column in the orthogonal component of the full design matrix
F	Froude number
\mathbf{F}	design matrix of a fully interpolated neural network
$\tilde{\mathbf{F}}$	orthogonal component of the full design matrix \mathbf{F}
g	acceleration due to gravity, general function
\mathbf{g}	error gradient
G_c	width of structure crest

h	water depth at the base of the toe of a structure
h_B	water depth over the berm of a structure
h_t	water depth over the toe of a structure
h_*	wave breaking parameter
H	Hessian matrix
$H_{m0,toe}$	significant wave height at the toe of a structure, from spectral analysis
$H_{1/3,toe}$	average height of the highest 1/3 of the waves in a random wave-train
i	individual local input to an artificial neuron
i	local input vector to an artificial neuron
I	identity matrix
l	characteristic length in dimensional analysis
L	inductance of an electrical circuit
m_0	variance of the water surface elevation
p	pressure
P	general smoothing function
P	projection matrix
q	overtopping discharge per unit length of wall per unit time
q_0	the dimensionless overtopping discharge $q / (gH_{m0,toe})^{0.5}$
Q_*	a general dimensionless overtopping discharge
r	previous search direction during gradient descent
R	resistance of an electrical circuit
R_0	the dimensionless crest freeboard $R_c / H_{m0,toe}$
R_c	crest freeboard
R_{max}	maximum wave run-up
R_*	a general dimensionless crest freeboard
R^2	linear regression coefficient of determination
s	spread, or width, of a radial basis function
s	current search direction during gradient descent
S	sum of squared errors
$s_{m-1,0}$	wave steepness
t	target output
t	vector target of a neuron given a number of different inputs
T	pseudo-temperature used in simulated annealing
$T_{m-1,0,deep}$	wave period in deep water, from spectral analysis
$T_{m-1,0,toe}$	wave period at the toe of a structure, from spectral analysis

$T_{p,deep}$	peak wave period in deep water
T_0	dimensionless mean wave period, $T_{m-1,0toe} (g/H_{m0,toe})^{0.5}$
u	depth-averaged component of velocity in the x-direction , net input to a hidden layer artificial neuron
v	depth-averaged component of the velocity in the y-direction, net input to an artificial neuron
w	synaptic weight within an artificial neural network
\mathbf{w}	weight vector of an artificial neuron
x	individual input to an artificial neural network
\mathbf{x}	input vector to an artificial neural network
y	output of an artificial neuron
\mathbf{y}	vector output of a neuron given a number of different inputs
Z	impedance of an electrical circuit
α	momentum coefficient, zero of a general function
α_d	slope below berm
α_u	slope above berm
β	angle of wave attack relative to the normal, in degrees
γ	minimum line search coefficient
γ_b	empirical berm reduction factor
γ_f	roughness/permeability factor of a structure
γ_h	empirical depth reduction factor
γ_β	empirical wave attack angle reduction factor
δ	delta, a common factor used in calculating weight updates
Δw	weight update
η	learning rate, water surface elevation
λ	regularisation or weight decay coefficient, Levenberg-Marquardt coefficient
ξ	breaker parameter
ρ	density
σ	steepness parameter in a radial basis function, correlation coefficient
τ	characteristic time in dimensional analysis
φ	geopotential
ω	angular frequency

Abbreviations used

ADALINE	adaptive linear element
ANN	artificial neural network
ANOVA	analysis of variance
BP	back-propagation of error
CF	complexity factor
CLASH	Crest Level Assessment of coastal Structures by full scale monitoring, neural network prediction and Hazard analysis on permissible wave overtopping
FS	forward selection of centres
FS-OLS	forward selection with orthogonal least squares
GA	genetic algorithm
GL-ANN	global-local artificial neural network
K-NN	k-nearest neighbour
LLSSIM	linear least squares simplex
L-M	Levenberg-Marquardt
MCCV	Monte Carlo cross-validation
ME	mixture of experts
MLP	multi-layer perceptron
MSE	mean square error
OLS	orthogonal least squares
PRBFN	Perceptron Radial Basis Net
RT-RBF	regression tree radial basis function
RBF	radial basis function
RF	reliability factor
s.d.	standard deviation
SOM	self-organising map
SQUARE-MLP	square unit augmented radially extended multi-layer perceptron
SSE	sum of squared errors
SWEs	shallow water equations

Chapter 1

Introduction

1.1 Historical Overview

In England and Wales it is estimated that 1.8 million homes and 140,000 commercial properties are in areas at risk of flooding or coastal erosion [1]. The value of the assets at potential risk has been estimated at £237 billion [2]. Hazards range from damage to property and vehicles [3] to threats to human life - between 1999 and 2002 at least 12 lives were lost as a result of individuals being swept off coastal paths, breakwaters and seawalls [4]. In addition, flooding has ‘intangible’ effects on the people affected. A recent report from the Department for Environment Food and Rural Affairs (DEFRA) found that they experienced considerable health problems, particularly psychological effects [5].

Considerable time and money is devoted to the construction and maintenance of sea defences - the expected cost on infrastructure in England and Wales for the year 2005-6 is £320 million [2]. This investment is likely to rise as a result of the increase in mean sea levels and in the frequency of storm surges caused by global warming [1]. However, due to the cost and the environmental impact of sea-walls it is important not to over-engineer sea defences, so accurate methods for predicting the efficacy of a particular design are essential [6, 4].

Concern with the construction of sea defences is not new. For hundreds of years it has been considered necessary to protect human activities and property from the destructive power of the oceans.

In 1014 a ‘great sea flood’ hit a broad area along the South Coast of England. This storm is recorded in the ‘Anglo-Saxon Chronicle’ [7]. It caused major landslides at Portland and many towns were washed away. Water levels in London rose

to unprecedented levels [8].

In 1607, high water levels in the Bristol Channel caused flooding over an area of 520 km^2 in South-West England and South Wales, killing around 2000 people [8]. It is not known whether the water levels were caused by a storm surge or by a tsunami [9, 10, 11]. Shortly afterwards, Lord Coke declared that it was the responsibility of the state to defend the population against the sea.

by the Common Law ... the King of Right ought to save and defend his Realm, as well against the Sea, as against the Enemies, that the same be not drowned or wasted [12]

The ‘great storm’ of 1703 caused enormous damage to property and the ferocity of the storm inspired Daniel Defoe to write his first book, ‘The Storm’ [13], the following year [14]. Hundreds of ships were destroyed resulting in the deaths of at least 8000 seamen [14]. Wind-speeds are thought to have been in the region of 120 mph [15]. They caused enormous damage to buildings, destroying 400 windmills and blowing down thousands of chimney-stacks and millions of trees. Off the Plymouth coast, the recently completed Eddystone Lighthouse was destroyed, killing its builder Henry Winstanley [16]. Storm surges caused major flooding at Bristol and Brighton. The estimated costs of repairs in the United Kingdom was equivalent to £10 billion today [14].

On 31 January 1953 strong winds, low pressure and high tides led to storm surges along the East coast of England, reaching a height of nearly 3 metres at King’s Lynn. Flood defences were breached, affecting coastal towns in Lincolnshire, Norfolk, Suffolk, Essex and Kent. Over 300 people died and 24000 homes were flooded [17]. The clean-up operation took weeks and is estimated to have cost the equivalent of £5 billion today [18]. The affect on the Netherlands was even more devastating: 50 dykes burst and over 1800 people were killed. Following on from this flood, the British government put in place a storm warning system [19]. However, by 1993, a report found that 41% of these were in ‘moderate or significant’ need of repair [20]. In response to this report the Environment Agency was created in 1996, with responsibility for flood defences and flood warnings [18]. The British Government is currently developing a new strategy for flood and coastal changes within the context of sustainable development and climate change [21].

On an international level there have been two major coastal floods in the last year. On 26 December 2004 an earthquake occurred off the Indonesian coast. This triggered tsunami waves that affected thirteen countries including Indonesia, Thailand, Sri

Lanka, India and Somalia. Over 200,000 people were killed and 5 million made homeless by the tsunami. An Indian Ocean early warning system is now being designed at an estimated cost of \$20 million [22]. On 29 August 2005, a class 4 hurricane hit New Orleans. The water depth of Lake Pontchartrain rose dramatically as a result of heavy rainfall and a storm surge. This caused some of the city's levees to break, resulting in flooding to a depth of 6 metres in some parts of the city. The number of deaths is not yet accurately known but is expected to run into thousands, and the cost of repair is likely to be tens of billions of dollars [23].

1.2 Hazard Levels and Wave Overtopping Rates

Adequate defences require accurate predictions of the effectiveness of a particular design. One way to do this is to estimate the volume of water likely to 'overtop' a sea-wall, given information concerning the structure of the wall, the sea-state and meteorological information. This value is generally recorded as an average overtopping rate per metre of seawall, over the period of a storm. Safe overtopping rates have been estimated by Owen [24]. Different hazard levels have been identified for pedestrians, vehicles and buildings, as illustrated in figure 1.1. Franco [25] has further differentiated hazard levels according to the type of seawall.

There have been doubts expressed as to the accuracy of mean overtopping rates as a predictor of hazard level [4]. Maximum instantaneous overtopping rates or velocities are likely to be a better guide to hazard level. However, the prediction and measurement of peak instantaneous overtopping volumes is prone to considerable variability at the current time, so mean overtopping rates are still the most commonly used predictor of hazard levels. In recent years two paradigms have emerged that produce an estimate of this quantity: curve-fitting and numerical simulation.

Curve-fitting is an empirical approach. It takes results obtained from laboratory tests on scale models and uses them to set parameters within a parametric regression model. It has the advantage that, once the parameters have been set, the resulting curve may be used to predict results instantly for previously unknown scenarios. The process involved is essentially one of interpolation. However, empirical curve-fitting requires the generation of large amounts of accurate data from laboratory tests. It is therefore time-consuming and expensive. Further, each parametric model is only applicable to a limited range of structures, necessitating the generation of a series of alternative curves.

Numerical simulation yields results for a particular scenario more quickly than

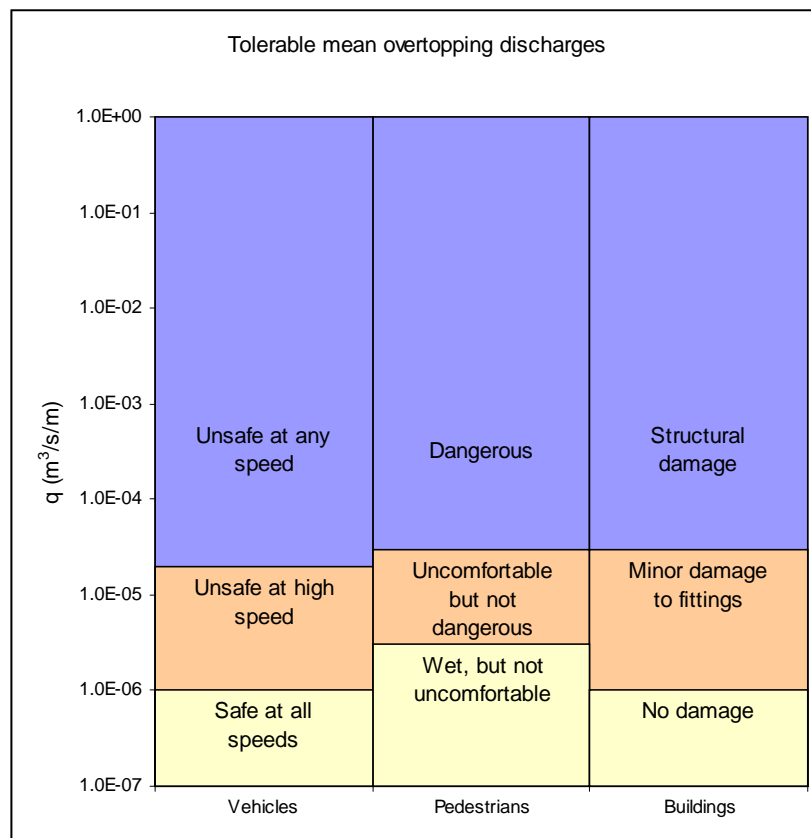


Figure 1.1: Safe overtopping limits

do laboratory models. Further, it gives a time-dependent picture of the progress of a storm. It is therefore able to provide information in addition to mean overtopping rates, such as instantaneous water pressure. However, this method also has its drawbacks. The results of the computational approach are not easily generalised. Whereas the empirical approach results in a curve that may be used for interpolation, numerical simulation must be repeated for each individual scenario.

When used to predict overtopping at ‘real’ seawalls, both approaches involve the use of certain approximations. The empirical approach is dependent on laboratory-scale data, and their validity therefore depends upon the scalability of results from freshwater scale-models to full-scale seawater sites. The approximations made during the scaling process are discussed in section 1.3. The numerical approach requires the parameterisation of very complex scenarios. In order to make the mathematical models tractable it is necessary to make assumptions and approximations, as described in section 1.4.

This thesis presents a new approach to wave overtopping prediction using artificial neural networks (ANNs). ANNs were originally envisaged as models of the mammalian brain. However, for the purposes of this study they may be seen as a method for achieving non-parametric (or semi-parametric) regression. They share the advantage of the curve-fitting approach: once their internal parameters have been set to appropriate values, they are able to interpolate (and in some cases extrapolate) to values that were not used in setting their parameters. However, unlike the curve-fitting approach, ANNs are not limited by the choice of any particular mathematical function. A single ANN may therefore be used as a generic prediction tool across a wide range of seawalls and sea-conditions. ANNs have the further advantage that they perform well in the presence of ‘noisy’ data. This means that an ANN may utilise data from full-scale sites measured under a variety of conditions.

The rest of this chapter reviews the existing state of research into wave overtopping prediction and relates it to the research presented within this thesis. Section 1.3 describes the empirical curve-fitting approach. Section 1.4 explains the use of numerical simulation techniques. Section 1.5 is a detailed history of ANNs. Section 1.6 reviews previous uses of neural networks in the area of hydroinformatics. Section 1.7 provides an outline of a new type of hybrid neural network to be presented in this thesis. Section 1.8 concludes this chapter and explains the structure of the rest of this thesis.

1.3 Empirical Curve-fitting

The most well-established method of predicting mean wave overtopping rates is that of empirical curve-fitting. This is a parametric approach in which the form of the relationship between the independent parameters and the overtopping rate is assumed. A small number of free parameters are then deduced by minimising a cost function, usually mean square error. This method is invariably linked to an experimental approach, in which results are obtained from scale models. These models generally contain simple idealised structures and flumes that give normal wave attack.

Besley [26] assumed an approximately exponential relationship between crest freeboard and mean overtopping discharge, following on from Owen [24]. He obtained empirical constants A and B for smooth, impermeable walls of various slopes to obtain the best fit for equation 1.1.

$$q_0 = AT_0 \exp\left(\frac{-BR_0}{T_0}\right) \quad (1.1)$$

In this equation $q_0 = q / (gH_{m0,toe}^3)^{0.5}$ is the dimensionless overtopping discharge, $R_0 = R_c / H_{m0,toe}$ is the dimensionless freeboard, $T_0 = T_{m-1,0toe} (g/H_{m0,toe})^{0.5}$ is the dimensionless mean wave period, q is the mean overtopping discharge rate in $m^3/s/m$, R_c is the crest freeboard, $H_{m0,toe}$ is the significant wave height at the toe of the wall, $T_{m-1,0toe}$ is the mean wave period at the toe of the wall and g is the acceleration due to gravity.

The method is only intended to be applied to smooth impermeable walls with slopes between 1:1 and 1:5, to waves of period less than 10 seconds approaching normal to the structure, and to values of R_0/T_0 between 0.05 and 3.0. Further, predictions are likely to be accurate only to within a factor of 10 [26]. Adaptations to the basic equation allow modifications for angled wave attack, bermed walls, rough slopes and wave return walls. However, the basic exponential form of the function is retained throughout. Details of the adaptations made to the basic equation, as well as alternative equations used by other researchers, are given in Appendix A.

The curve-fitting approach has the advantage that predictions are obtained very easily once the free parameters have been determined. Further, the input-output relationship is explicit and easy to understand. However, all curve-fitting approaches suffer from certain drawbacks.

- The parametric approach is inherently limited in its scope and requires knowledge of the relationship between the independent parameters and the overtopping rate on the part of the modeller.

- Predictions are limited to idealised structures, due to the small number of free parameters.
- The dependence on laboratory techniques means that the creation of appropriate data is time-consuming and expensive.
- The use of experimental data in predicting ‘real’ storms relies upon the validity of the scaling process. There are substantial approximations involved in the assumptions that surface tension and viscosity scale with size. A specific difficulty related to the use of freshwater in experimental tests has been described by Bullock *et al.* [27]. Saltwater has higher aeration levels than freshwater and therefore displays greater compressibility and lower impact pressures. This effect is particularly noticeable for violent situations, which lead to large amounts of trapped air.

1.4 Numerical Modelling

The mathematical modelling approach runs a numerical simulation of wave motion, within constraints including equations governing the underlying physics, given initial conditions such as water velocity and boundary conditions such as wall and bed slope geometry. The usual starting point is the Navier-Stokes equations, which are an expression of the fundamental laws of conservation of mass, momentum and energy. Solution of these equations for any but the simplest of scenarios is extremely computationally expensive [28]. However, in situations in which the depth of the water is small compared to the wavelength of the waves the non-linear Shallow Water Equations (SWEs), given in equation 1.2, are known to provide a good approximation [29].

$$\frac{\partial}{\partial t} \begin{pmatrix} \varphi \\ \varphi u \\ \varphi v \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} \varphi u \\ \varphi u^2 + \varphi^2/2 \\ \varphi uv \end{pmatrix} + \frac{\partial}{\partial y} \begin{pmatrix} \varphi v \\ \varphi uv \\ \varphi v^2 + \varphi^2/2 \end{pmatrix} = 0 \quad (1.2)$$

In this equation u and v are the velocity components in the horizontal plane, φ is the geopotential gh , g is the acceleration due to gravity and h is the water depth.

These may be derived from the Navier-Stokes equations by assuming that the vertical velocity is small compared to the horizontal velocity. This is equivalent to the assumption of hydrostatic pressure. When waves are impacting, this assumption is

incorrect. However it has been shown that the SWEs give reasonable accuracy even under certain breaking conditions [30, 31].

Numerical schemes have been designed for solving these equations. Typical is the approach in Hu *et al.* [28], which uses a finite-volume solver with a Godunov-type upwind scheme. Such a scheme may be used to model particular scenarios with considerable accuracy, terms being added to allow for factors such as bed stress or bed dryness [32]. The wall and bed geometry are included in the model using appropriate boundary conditions.

Mathematical modelling typically gives results within a factor of 2 of the measured overtopping discharges [28]. This is a considerable improvement on the curve-fitting regime. This is expected, since modelling is applied to a particular scenario rather than a family of scenarios.

Such schemes have so far been applied only to near-ideal walls in laboratory-controlled tests, mainly due to the great computational cost of running such simulations. It is to be expected that mathematical modelling of ‘real’ scenarios would require additional terms, and therefore computer time, in order to achieve similar accuracy. The number of uncontrolled variables in ‘real’ seas will also affect the accuracy of predictions made by numerical solvers.

The underlying mathematical model used within numerical modelling normally contains a number of assumptions, such as shallow water inviscid flow, in order to reduce the high computational cost of running the simulations. Modelling ‘real’ scenarios requires more accurate models and leads to greatly increased computation time. Shiach *et al.* [31] made comparisons between a numerical model (based on the Shallow Water Equations) and experimental observations. They found that for strongly impacting waves the model was too inaccurate to be of practical use and a more detailed model had to be employed. Under these circumstances Volume of Fluid (VoF) models [33] or free surface capturing models [34] had to be employed, resulting in a dramatic increase in computational cost.

A further disadvantage of the numerical simulation approach is that it is situation-specific. A detailed knowledge of both the sea wall geometry and the exact sea conditions is required, so a small design change necessitates a complete rerun of the simulation.

1.5 Artificial Neural Networks

1.5.1 Introduction

Artificial Neural Networks (ANNs) were originally devised as models of the human brain. It was hoped that ANNs could reveal useful information about the structure of the brain and the processes that occur within the brain. The use of ANNs as a tool for exploring brain function has become increasingly widespread within cognitive psychology and neurophysiology in recent years. However, this study is primarily interested in ANNs as a tool for solving mathematical problems. In particular, ANNs are used to identify unknown multivariate functions from samples of data. Aspects concerning the biological validity of an ANN architecture or of a training algorithm are only occasionally considered.

In a biological neuron, electrical signals are passed from neuron to neuron via synaptic connections. The strength of the incoming electrical signal is moderated by the excitatory or inhibitory nature of the synaptic connection. Several incoming signals may be combined within the main cell body. The overall output signal from a neuron then passes along a long axon. The signal strength is maintained along much of the axon's length and may activate neighbouring neurons. Each of these neurons therefore receives roughly the same signal.

In an ANN a neuron is represented by a simple processing unit that has three functions: it takes one or more inputs, performs a mathematical transformation on these inputs and outputs the resulting value. From a signal processing point of view it therefore has the essential features of a biological neuron [35]. The transformation performed by the neuron is known by several names. Throughout this study it is referred to as a 'transfer function'. Transfer functions may take many mathematical forms, and the formula chosen will often have a large effect on the computational algorithms used, the problems which an ANN can solve and the speed with which solutions may be obtained. This thesis is particularly concerned with the difference between local transfer functions that only have significant outputs across a small volume of input space and more diffuse transfer functions. Radial basis and sigmoidal functions are representative of these two types of function, and are described in detail in section 2.1. Their associated training algorithms are described in sections 2.2-2.5.

Like a human brain, ANNs contain a number of neurons that may be interconnected in various ways. When a connection is present, the inter-neuronal signal is moderated by a synaptic 'weight'. In figure 1.2 i_p are the inputs to the neuron, w_p are the input

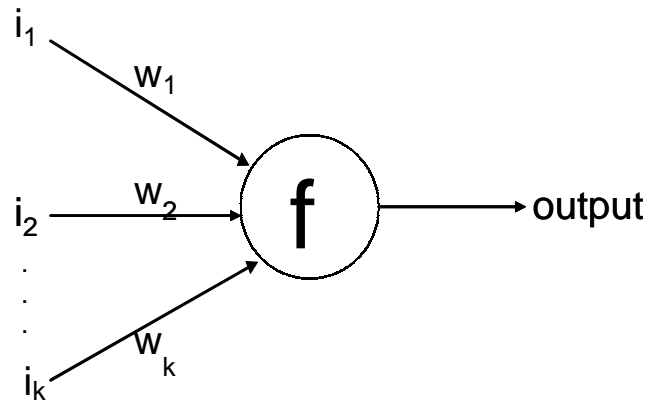


Figure 1.2: Diagram of an artificial neuron

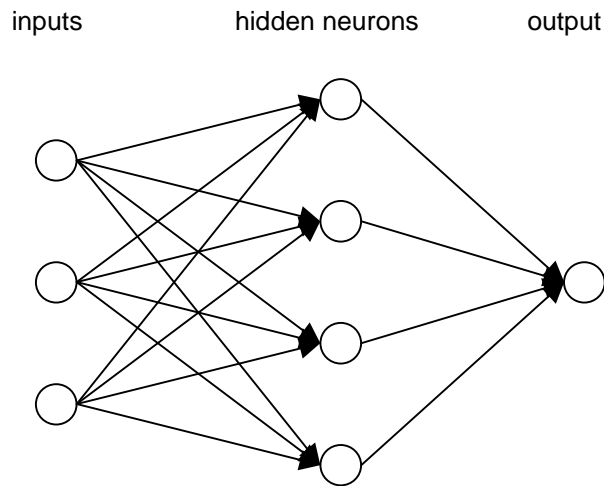


Figure 1.3: Diagram of an artificial neuron network

weights and f is the transfer function.

The neurons are generally arranged in layers, making the transmission of information through a network easier to track. The layers include an input layer, an output layer and may contain one or more intermediate layers (figure 1.3). The latter are usually referred to as ‘hidden’ layers, since they do not hold information that may be immediately interpreted in a symbolic way. However, the hidden layer neurons perform much of the processing that makes ANNs such powerful mathematical tools.

Human brains are known to develop through three processes: the growth of new neurons, the loss of older neurons and an alteration of the strength of synaptic connections. The first two processes have artificial equivalents in constructive and pruning algorithms for resizing ANNs. Some of these will be discussed in detail in future sections. They include the cascade-correlation algorithm and forward selection of centres

in RBF networks. However, the learning process on which neural network research has been primarily focused is the process of weight adaptation.

Humans learn from experience. Physiological knowledge concerning the neuronal structure of the brain indicates how this learning comes about. Particular patterns of neuronal activity correspond to particular psychological responses. When we find ourselves in a specific situation, the same neurons that were excited last time we were in a similar situation will ‘fire’ again. Our behaviour at any time is therefore governed to a large extent by our behaviour at previous times. However, the strength of synaptic connections is being adjusted all the time in response to external stimuli. For example, if a particular action has achieved the desired ends, the synaptic connections firing at that time are likely to be strengthened. If, on the other hand, an action is unsuccessful, an inhibitory effect will be induced. The state of our synaptic connection strengths at any one time may therefore be seen as the result of our responses to all of our previous experiences [36].

The strength of a synaptic weight is represented in an ANN by a connection weight. In order for an ANN to learn, these weights must be adjusted. ‘Learn’ is used here to mean ‘give an improved response’. Humans learn by adapting their responses to their environment. By introducing an assessment function we can ensure that ANNs learn by improving their score on this assessment function. We can now see that the ANN learning process is a series of weight adjustments, moderated by an assessment function. Due to the introduction of an assessment function, the process is also referred to as ‘training’. The training process is illustrated diagrammatically in figure 1.4.

When there are a large number of neurons, the ANN method results in the determination of a large number of free parameters (the inter-neuronal weights). It may be seen as a method for performing non-parametric regression analysis: the large number of free parameters means effectively that there is no assumption concerning the functional form of the input-output relationship. This may be contrasted with curve fitting approaches in which an overall functional form is assumed for the relationship between the variables. The small number of free parameters in such approaches imposes a considerable restriction on the possible approximating functions produced.

Unlike statistical methods such as linear regression, ANNs are almost invariably non-linear. Their non-linearity arises from the use of non-linear transfer functions within individual neurons. The parallel structure of a neural network means that the overall input-output relationship may be a highly complex, non-linear function although the individual transfer functions represent fairly simple non-linearities.

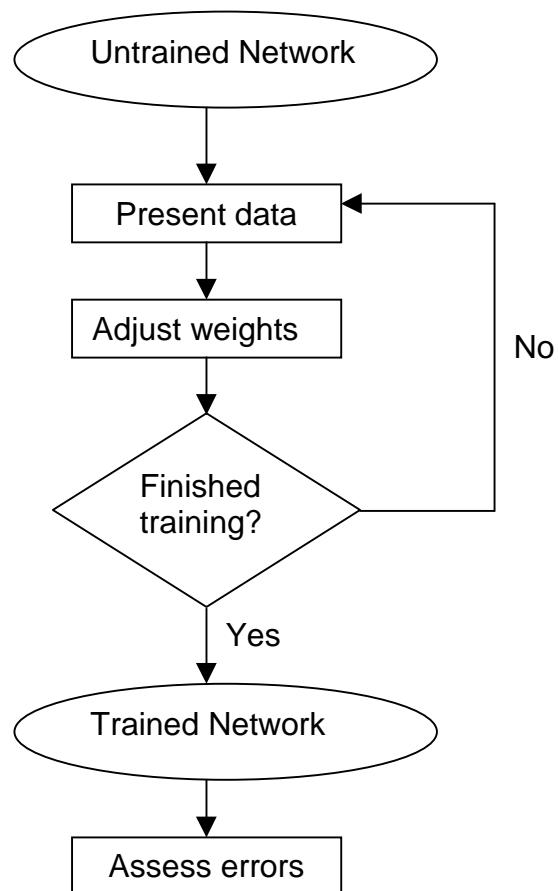


Figure 1.4: Diagram showing the neural network training process

The training of ANNs has proven to be a complex process. Methods of training are highly varied: some attempt to approximate the processes of biological neurons but many diverge greatly from them in an attempt to find more computationally efficient methods to achieve optimal or near-optimal weights. Apart from the method used to train them, ANNs may be differentiated in many ways. The following perspectives give alternative ways of classifying ANNs, although we shall see that the different perspectives are intertwined in complex ways-

- Choice of transfer function [37].
- Selection of assessment function [38].
- Choice of network architecture [38].

The next sub-section (1.5.2) describes some of the applications of ANNs. The rest of this section details the historical development of neural networks. This development may be seen as constituted from a number of strands. 1.5.3 describes early research into ANNs (pre 1985). 1.5.4 describes the development of the most widely used ANN, the ‘multi-layer perceptron’ (MLP). 1.5.5 details some improvements to the basic MLP method, while section 1.5.6 describes some global methods for training these types of networks. 1.5.7 describes the development of an alternative to the MLP, known as a ‘radial basis function’ (RBF) network. Finally, 1.5.8 describes some hybrid networks that combine the MLP and RBF approaches.

1.5.2 Applications of ANNs

This section aims to give a review of the practical applications of ANNs. These applications may be split into three main classes: pattern association, pattern recognition and function approximation. These areas will be treated in turn.

Pattern Association

A neural network may be trained to act as an ‘associative memory’. The process of training stores a set of patterns (vectors), which may be retrieved from the network. The aim may be to retrieve a clean pattern when a noisy version of the pattern is presented to the network (‘auto-association’). Alternatively, the aim may be to retrieve a pattern that is different from the input pattern, but has been paired with that particular pattern (‘hetero-association’). Association is a pertinent model for memory within the

human brain [36, 39]. It has been applied within the areas of market basket analysis [40], information retrieval [41, 42], syntax analysis [43] and image recognition [44].

Pattern Recognition

During pattern recognition, an ANN is required to identify the class to which an input pattern belongs. During this task, neural networks divide up ‘decision space’ into regions, each one corresponding to a single class. Again, this model is analogous to processes within the human brain, such as the process by which we identify familiar objects despite variation in viewing angle, lighting conditions and other distortions to our visual inputs [45]. There are two ways in which the classification may be performed. During unsupervised learning the network discovers clusters in the data itself. Alternatively, a supervised learning approach may be applied. In this case a network is trained to reproduce known outputs (categories), from which it may then generalise to unseen inputs. Applications of pattern recognition techniques include fingerprint identification [46], optical character recognition (OCR) [47] and number plate identification [48, 49]. The NETTalk program is a well-known program that converts written language into phonemes, allowing a computer to learn to ‘speak’ [50]. Other applications of pattern recognition include medical diagnosis [51] and financial risk assessment [52].

Function Approximation

Within function approximation, tasks may be divided into modelling and forecasting. In the latter, time series data is available and the aim is to predict future data from past data. Examples include exchange rate prediction [53], house price indexing [54] and solar activity prediction [55]. Modelling tasks have covered a wide range of topics including domestic energy consumption [56] and various aspects of control engineering including vehicle manoeuvre, electric power, chemical engineering and blood pressure management [57, 58].

The main application investigated within this thesis, wave overtopping prediction, comes into the category of modelling. Systems that have been developed for predicting overtopping levels and other hydraulic parameters are discussed in section 1.6.

1.5.3 Early Research in ANNs

McCulloch and Pitts are widely credited with founding the modern ANN tradition with their 1943 paper, 'A Logical Calculus of Ideas Immanent in Nervous Activity' [59]. They used neurons with net inputs equal to a simple weighted sum of their inputs. They used a stepped threshold transfer function: if a net input was positive the output was 1, otherwise it was 0.

They showed that, given sufficient neurons and correctly set weights, a network made up of such neurons could compute any computable function.

Hebb [60] noted that the strength of a synaptic connection is increased if the neurons on either side of the connection are activated synchronously. He used this observation as the basis of a learning rule that could adjust synaptic weights. Later authors [61, 62] added the converse rule that the connection strength is decreased if the neurons on either side of the synapse fire asynchronously. For a pair of connected neurons, their connection weight is therefore increased if both activations deviate from their mean values in the same direction and decreased if the activations deviate in opposite directions.

In 1956 von Neumann [63] introduced the idea of redundancy. Neural networks contain a large number of neurons that collectively represent an individual concept. The overall system is robust in the sense that one or more of the neurons may be faulty, giving an 'incorrect' output, and yet the system as a whole can still give a 'correct' response.

Rosenblatt invented the perceptron in 1958 [64]. His perceptron is an ensemble of neurons arranged in a single layer. Each neuron has a fixed bias in addition to the applied inputs. The outputs from the neuron are again stepped, but have values +1 and -1. Rosenblatt demonstrated that a single neuron could separate inputs into two separate classes given any linearly separable function, resulting in an output of +1 for one class and -1 for the other. By using more than one neuron, the inputs may be divided into more than two classes. Effectively a single-neuron perceptron creates a hyperplane in input space, which separates the inputs into two categories. When training the network, it is therefore necessary that the investigator inputs the target category into the system alongside the input vector.

The perceptron training rule is the first example of a supervised training rule. In supervised training, target outputs are presented to the ANN and the ANN attempts to reduce the error between the actual and the target outputs. It can effectively position the decision surface, starting from a position with all weights set to zero or from a random

position, in a finite number of training steps [64]. The weight changes at each step are proportional to the output of the neuron in question and to the difference between the target and the actual output. The weights will therefore only be adjusted if the target and actual weights are different, i.e. the ANN is misclassifying the input pattern. Each of the input vectors is presented in turn. If the inputs are not correctly classified at this point, the process is repeated until classification is correct. Each presentation of the set of all input patterns is known as an ‘epoch’.

Widrow and Hoff [65] introduced a new training rule, often described as the ‘least squares rule’, and used it to train an ANN they called an adaptive linear element, or ‘ADALINE’. This is related to the perceptron training rule, but the error is calculated as the difference between the net input and the target output, rather than the difference between the stepped output and the target output. This means that learning will occur even when the classification of an input pattern is correct and learning is therefore quicker. The name ‘least squares rule’ has arisen because its use leads to convergence to the least mean square solution. A full mathematical treatment is given in section 2.2.1.

In 1969 Minsky and Papert’s book ‘Perceptrons’ cast serious doubt on the potential development of neural networks [66]. They pointed out many of the limitations of Rosenblatt’s perceptrons and stated their belief that multi-layer perceptrons would not be able to overcome these limitations. The result of this book was that research into neural networks virtually disappeared during the 1970s and early 1980s.

There were a few exceptions. In 1982 Kohonen described ANNs that he called self-organising maps (SOMs) [67]. (A similar idea had been described by Willshaw and von der Marlsburg in 1976, but their model received less interest [68]). SOMs are examples of unsupervised ANNs that are used for detecting spatial organisation within input data. The neurons within a SOM are conceptually arranged in a grid, usually of 1 or 2 dimensions. Before training, neurons weights are initialised randomly. After training, each neuron responds strongly only to inputs within a particular region of input space and neurons that are ‘close’ to each respond to similar areas of the input space. The weights may therefore be viewed as centres that detect nearby inputs. A SOM therefore creates a topological map of its inputs. It is analogous to certain areas of the human brain that respond to sensory inputs [69, 70, 71]. For each input vector, the SOM training algorithm undergoes three phases-

- Competition. The neuron that has the weight vector with the smallest Euclidean distance from the input vector is selected.

- Co-operation. Within the conceptual grid, or ‘feature space’, a neighbourhood around the centre of the winning neuron is identified. This neighbourhood is described by a mathematical function that peaks at the neuron’s position within the grid and falls away with distance to reach zero at infinite distance. A typical function is the Gaussian function. As training proceeds, the width of the neighbourhood is reduced, usually in an exponential fashion.
- Adaptation. The weights (centres) of all neurons are adjusted to bring them closer to the input pattern. The neurons that are closest to the winning neuron (in feature space) will be affected to a greater extent due to the radial decline of the neighbourhood function.

Both the neighbourhood width σ and the learning rate η decrease as the number of epochs n increases. To achieve full convergence η must be kept at a constant rate of about 0.01 towards the end of training in order to achieve full convergence of the algorithm.

Once a SOM is trained, the weights of the network represent a ‘feature map’, in that they map the input space onto a conceptual feature space. The position of each neuron within feature space corresponds to a particular domain, or feature of the input domain. The SOM may be seen as an encoder, that encodes input space into feature space. The final position of the weight vectors gives useful information concerning the distribution of the input vectors. This may be used to classify inputs according to their corresponding positions in feature space or to identify the sources of variation within a population, in a manner analogous to principal component analysis.

The Hopfield network [72], introduced in 1982, is a recurrent network. It contains a single layer of neurons and the output of each neuron is fed back into the ANN as an input to all of the other neurons. The weight matrix is symmetric, i.e. $w_{ij} = w_{ji}$ for all pairs of neurons i and j . Other than these differences, Hopfield neurons act like perceptrons, giving outputs of +1 or -1 according to the sign of the weighted sum of the inputs.

Hopfield networks may be used as content addressable memory. After training they are able to retrieve a correct vector given a faulty or incomplete input vector. In training, or ‘storage’ as it is more accurately described, the weights of the network are set in a deterministic way, by the application of linear algebra. There is no need to use iterative methods such as gradient descent.

During retrieval an incorrect or partial vector is introduced to the network. A single neuron is selected randomly and its net input is calculated. If this is positive, the neuron

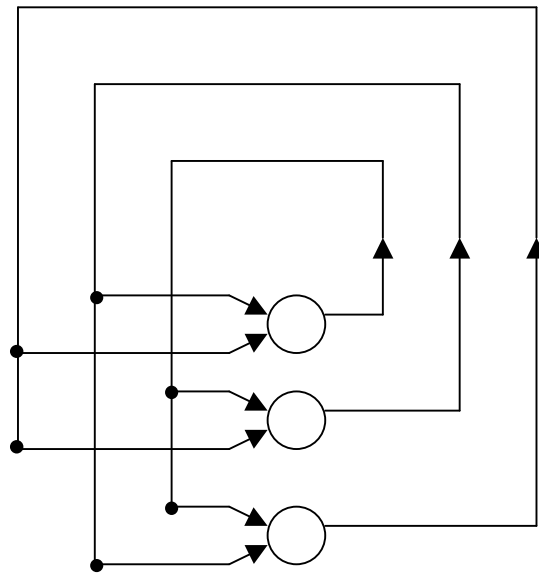


Figure 1.5: Hopfield network architecture

will switch its output to +1. Conversely, if the net input is negative, the output will be set to -1. If the net input is zero, the state of the neuron is not changed. The process is continued until there are no more changes. The output should then correspond to one of the patterns originally stored in the network.

1.5.4 Multi-layer Perceptrons

All of the networks considered in the last subsection contained direct links between input and output neurons, without a hidden layer. Most of them also used stepwise or linear transfer functions. In 1986 Rumelhart and McClelland [73] presented a training algorithm that would allow the use of one or more hidden layers of neurons and a variety of transfer functions. The only condition was that the transfer function was differentiable, i.e. one could calculate a gradient for the function at all points. This rules out stepwise functions and led to the use of more sophisticated transfer functions.

A step towards the use of more flexible transfer functions may be made by extending the least squares rule [74]. Widrow and Hoff considered the dependence of the net input on the weight vector. However, when using the sum of squared errors (SSE) as the objective function it is more appropriate to consider the effect of the weights on a neuron's output. It is possible to do this provided a differentiable transfer function is used. The output from the transfer function is dependent upon the net input v and may therefore be expressed as $f(v)$. However, the net input v is in turn a function of

the input weights w and the size of the inputs i . Using differentiation it is therefore possible to obtain the gradient of the output as a function of the input weights and the input vector.

When dealing with an ANN with more than one hidden layer, the inputs to a hidden layer may come from a further hidden layer. They are therefore dependent on the transfer functions of the previous layer. Provided these transfer functions are differentiable functions it is possible to apply the chain rule to the next layer of neurons. By successive application, the error is ‘back-propagated’ so that the dependence of the error on each of the weights in the network may be calculated. Having obtained partial gradients with respect to all of the weights within the network, the network’s weights may then be adjusted in the direction of steepest gradient descent. This method was first used by Werbos [75]. The name ‘back-propagation’, or simply ‘BP’, became widely used following the publication of ‘Parallel Distributed Processing’ by Rumelhart and Williams in 1986 [73]. This book led to a resurgence of research into ANNs that is still ongoing.

The back-propagation algorithm may be applied to any ANN containing neurons with differentiable transfer functions. However, it has been very closely associated with a family of functions collectively known as sigmoid functions, due to their S-shape. These functions accept a net input, which is a weighted sum of all inputs to the neuron and therefore describes a hyperplane through input space. An additional input called a ‘bias’ is also used. The use of a bias was first mentioned by Rosenblatt in his description of a single layer perceptron [64]. This is an input of fixed value (usually -1) and allows the hyperplane to take any possible position. Specifically, it is not constrained to pass through the origin. Sigmoid functions transform the net input, softening it towards its more extreme values so that the final output is constrained to lie between maximum and minimum values, as illustrated in figure 1.6.

All sigmoid functions share the following properties-

- They are differentiable at all points.
- They approach linear behaviour in their middle region.
- At their extremities they ‘level off’, approaching fixed values asymptotically.

They therefore combine the properties of linear and stepwise functions: they vary monotonically but have fixed maximum and minimum values. Sigmoid functions are described in more detail in section 2.1.

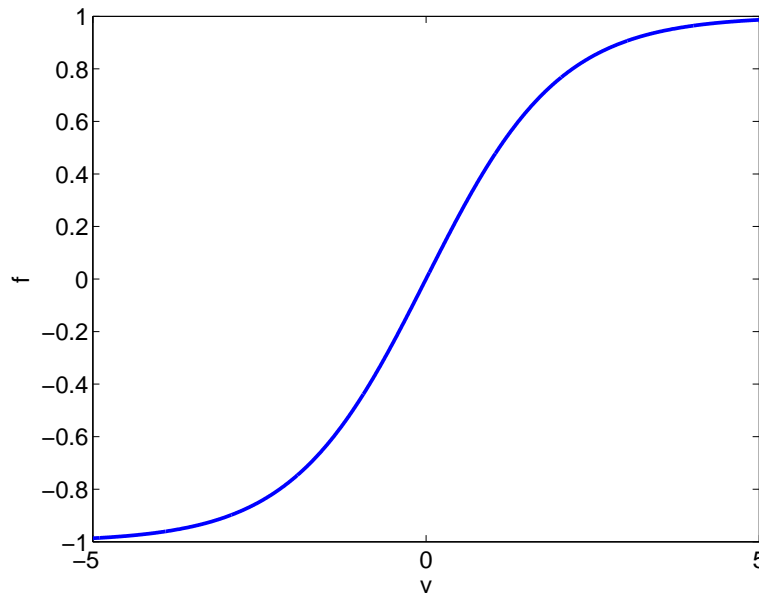


Figure 1.6: Graph of a bipolar sigmoid function

The term ‘multi-layer perceptron’ (MLP) is used to describe a particular type of network. It has the following properties-

- It is a ‘feedforward’ network. This means the network is made up of fixed layers - an input layer, an output layer and one or more hidden layers. There are no connections within a layer, but each layer is usually fully-connected to the subsequent layer.
- It is trained using the BP algorithm.
- MLP is usually assumed to refer to an ANN containing only sigmoid neurons, with the possible exception of the output layer which may contain linear neurons.

Despite the name, multi-layer perceptrons have little in common with Rosenblatt’s perceptron on the surface: they have a different transfer function and a different training algorithm. However, they are seen as being in the same tradition as their earlier cousin since they aim to predict output vectors given input vectors, via a knowledge base represented by stored weights that are adjusted with a supervised training algorithm.

The gradient of the error with respect to the weights varies continuously across the weight space. The direction of weight changes is tangential to the error surface at the point reached by the last iteration. Since the step size is not infinitesimally small, the BP algorithm must be an approximation to the method of steepest gradient descent.

By making the step size smaller the approximation is made closer. However, this is likely to result in slower learning. A larger step size, on the other hand, will lead to faster learning, but may cause the algorithm to oscillate without approaching an error minimum, or even diverge.

The obvious solution is to introduce a variable learning rate. Rumelhart, Hinton and Williams suggested an elegant way to achieve this by introducing a momentum term [76]. At the n th epoch this term adds a fraction of the $(n - 1)$ th update to the change in the weights. If the two updates are in a similar direction a larger step will be taken, whereas if they are in opposing directions a smaller step will be taken. The introduction of momentum causes the BP algorithm to converge quicker and reduces oscillation around a minimum gradient. A derivation of the BP algorithm with momentum was presented by Hagiwara in 1992 [77]. The algorithm is described mathematically in section 2.2.2.

A second aspect to be considered is the way in which weights are updated. Network weights may be adjusted after the presentation of each input pattern, a process known as ‘stochastic’ weight updates. Alternatively, during the ‘batch’ process, the calculated weight updates may be stored for each input but only summed and applied after the presentation of every item within the training set. Batch weight updates give more reliable gradient information than stochastic weight updates. However, stochastic weight updating has two advantages-

- The memory requirements are much smaller for stochastic weight updates.
- Stochastic updates are better able to avoid small local minima in the error surface.

Stochastic weight updates are therefore preferred for many datasets, particularly noisy ones. Several authors have shown that different learning rates should be applied to each layer of neurons, in order to give optimum training times. In particular, larger learning rates should be applied to the later than to the earlier layers [78, 35]. Some authors go further, suggesting that every weight in a network should have its own learning parameter and that learning parameters should vary from one epoch to the next [79].

Another factor in the effectiveness of BP is the initialisation of weights. Weights are usually set randomly to small values, such as $[-0.1, 0.1]$ before training. However, it has been shown [78] that the outcome of training is usually highly dependent on the

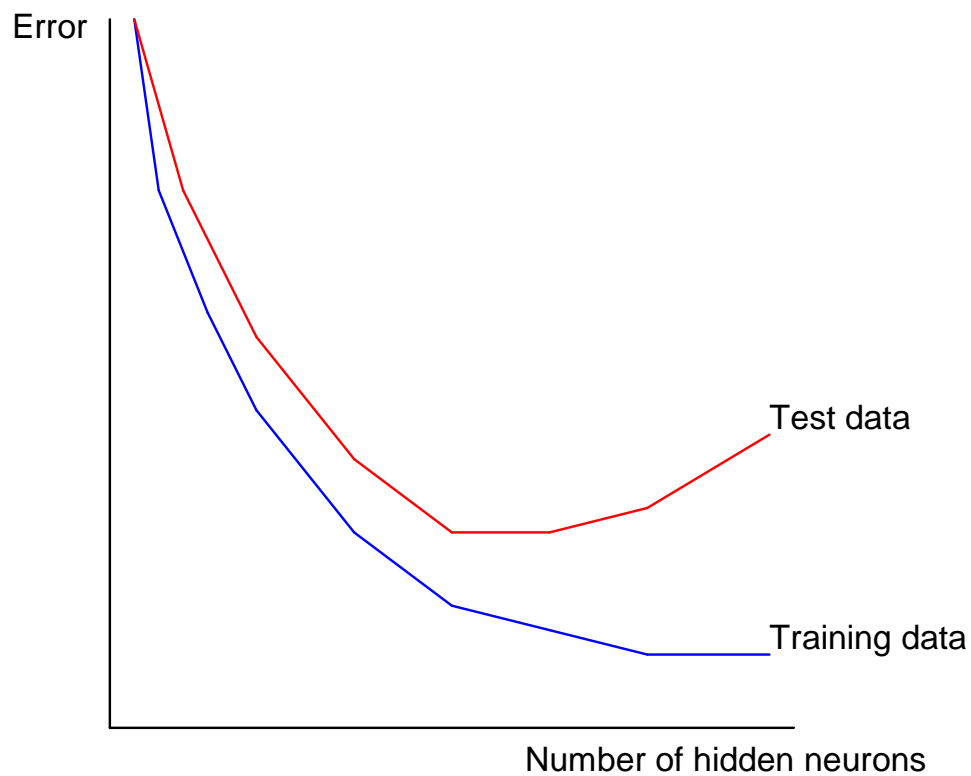


Figure 1.7: Overfitting caused by oversized networks

starting weights. Alternative means for the initialisation of weights may involve the inclusion of *a priori* knowledge.

The selection of network size (model selection) is also an important factor in BP training. The approach most commonly used is to train a series of ANNs containing different numbers of hidden neurons. As more neurons are added, the error on the training data generally falls monotonically. However, if tested with data not used in training, the assessed error is typically seen to fall as neurons are added, before starting to rise after a certain point, as illustrated schematically in figure 1.7.

This phenomenon is the result of ‘overfitting’ the underlying function. When there are too many free parameters within the network it starts to fit the errors within the training data in addition to the underlying function, with a resulting increase in the test error.

Another type of overfitting occurs if a network is trained for too many epochs. Again, the error on unseen data is seen to rise after a certain point (figure 1.8). One way to avoid this type of overfitting is to stop training early. This may be achieved with the use of a validation set. The available data is partitioned into training, validation and test sets. Training is stopped when an increase in the error for the validation set is

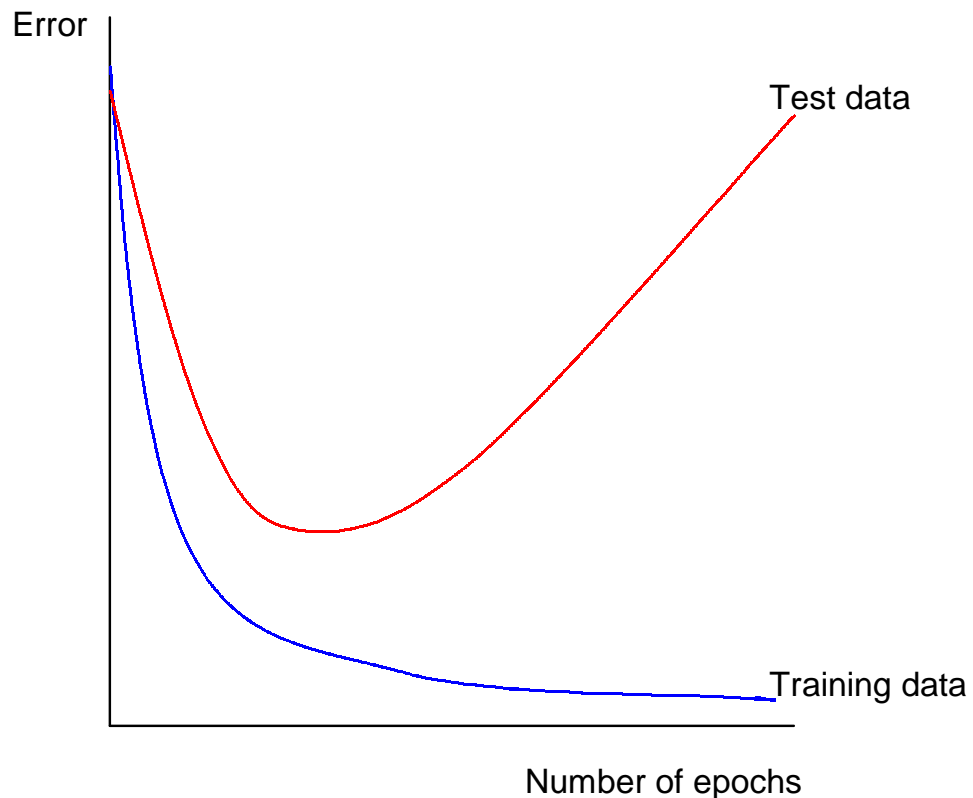


Figure 1.8: Overfitting caused by overtraining

observed. The generalisation properties of the network may then be assessed using the test dataset.

The need for early stopping depends upon the size of the MLP. Amari *et al.* [80] have shown that early stopping may be valuable in cases where the ratio of free parameters to training samples is greater than 30, but will not be beneficial otherwise. This issue is investigated further in section 4.2 in the context of the wave overtopping dataset.

Related to the early stopping method is the method of cross-validation, which is often used when the size of the available dataset is small. In this method, the data is first divided into training and test data and the test data is set aside. The training data is divided into a number of equal sized segments. ANNs are then trained on all but one of the segments. The remaining segment is used as a verification set to determine an early stopping point. It may also be used to assess the optimum ANN architecture, i.e. the optimum number of hidden neurons. Once training is complete, the test set is used to assess the performance of the network.

Overtraining may also be avoided through the use of weight decay [81]. In this

procedure a penalty is attached to the use of large weights, so smoothing the overall approximating function. Typically, the penalty is proportional to the sum of the square of the weights. Weight decay is underpinned by the theory of regularisation within the field of numerical optimisation.

Related to the technique of weight decay is the network pruning procedure. Rather than limiting the size of the network weights, this method aims to reduce the number of weights within a network. Training therefore takes place in two steps-

1. Create and train a fully-connected network, i.e. one in which each neuron is connected to every neuron in the layers both before and after it.
2. Remove connections between neurons that make little contribution to the final outputs of the network.

An effective algorithm is that of the ‘optimal brain surgeon’, due to Hassibi and Stork [82]. This algorithm takes a trained network and assesses the ‘saliency’ of each network connection. A Taylor series of the error with respect to the weights is constructed. Assuming that a fully trained network has an error gradient with respect to the weights of zero, the most important term is therefore deemed to be the second order term, involving the Hessian matrix $\frac{\partial^2 E}{\partial \mathbf{w}^2}$. The weight that gives the smallest increase in this term is set to zero, i.e. the connection is removed from the network, and the remaining weights recalculated. The process may be repeated until a large error increase is observed.

To summarise, BP training of MLPs may be adapted through the setting of a number of modes and parameters, including-

- Learning rates
- Momentum coefficients
- Stopping criteria
- Weight initialisation
- Choice of batch or stochastic weight updates
- Model selection
- Weight decay parameter
- Network pruning

Each of these affects the outcome of training, and the selection of these parameters is often intuitive rather than systematic. This has led some authors to search for more automated and predictable training methods, some of which are described in the next section.

1.5.5 Alternative training methods

Two problems with BP training are the dependence on initial weights and the tendency to become trapped in local minima. Simulated annealing is an attempt to overcome these difficulties. It uses a more stochastic approach than BP. Thus, rather than making weight changes in the direction of steepest descent, weight changes are assigned a probability. This probability is dependent on a notional ‘temperature’. At high temperatures, the ANN can jump out of valleys in the error surface, but at lower temperatures the weights become more constrained to lower ‘energy states’ [83]. If ΔE , the change in training error, is negative the weight change Δw will take place. If ΔE is positive the change would result in an error increase. The associated weight change may be made, with a probability dependent upon a ‘temperature’ T . The temperature is decreased exponentially as training proceeds. The probability of an error-increasing change therefore drops and training eventually settles into a minimum. If a sufficiently slow cooling schedule is chosen, the algorithm should visit the whole of the weight space sufficiently often that the global minimum is identified. The mathematical details of simulated annealing are given in section 2.2.3.

Cascade correlation is a constructive algorithm that grows ANNs to an optimum size [84]. Neurons are added one at a time and the efficiency of training is improved by ‘freezing’ the weights of existing neurons. The weights of the new neuron are then trained on the remaining error of the ANN. Each new neuron is added in a new layer and accepts outputs from preceding neurons as inputs, in addition to the original network inputs. This results in a deep network, in which each layer is a single neuron that identifies successively finer detail in the approximating function (see figure 1.9).

Training is usually stopped when a minimum error has been reached. It is usually performed with the ‘quickprop’ algorithm. This takes into account the gradient vector during the previous weight update and therefore results in a variable learning rate in a similar way to a momentum term. (See section 2.2.3 for mathematical details.)

An important alternative to basic gradient descent methods is the family of second-order gradient descent methods. These are derived from the field of numerical optimisation, in which the aim is to optimise an unknown non-linear function (the error) with

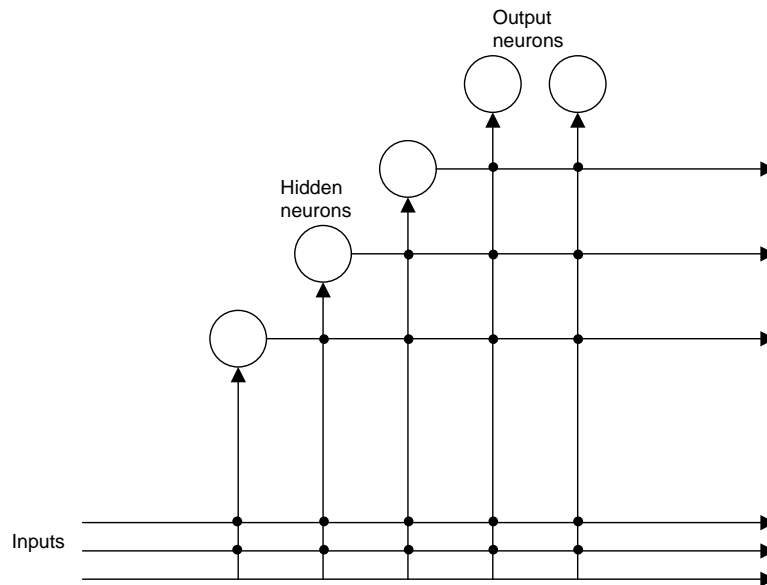


Figure 1.9: Cascade correlation architecture

respect to a number of unknown independent parameters, i.e. the network weights. The approach in these methods is to find a point at which the gradient of the error surface is zero. This point will correspond to a minimum, provided precautions are taken to avoid identifying a maximum or saddle point in the error surface.

In the conjugate gradient method [85, 86], each weight change is made in a direction that is conjugate to all previous steps, with respect to the approximating function. Further, the step size is optimised using line minimisation. For these reasons, this method is much more efficient than standard back-propagation. In order to find the conjugate direction it would usually be necessary to obtain the Hessian of the network weights with respect to the observed error. However, by making each weight adjustment a linear combination of the previous weight adjustment and the current gradient vector it is possible to obtain a new search direction without explicit calculation of the Hessian [85]. Further details are provided in Appendix B.

The conjugate gradient method avoids calculating the Hessian, due to the high computational cost involved. The Levenberg-Marquardt method, on the other hand, calculates an approximation of the Hessian matrix [86]. It then uses a combination of gradient descent and Newton's method to find the zero of the error gradient [87].

The Levenberg-Marquardt algorithm is a very effective heuristic, reaching satisfactory minima in a small fraction of the number of epochs required by basic gradient descent, for most datasets. However it has some drawbacks-

- The construction of the Hessian involves a large number of calculations for large ANNs, scaling with the number of training patterns and the square of the number of network weights. This can lead to long training times and large memory requirements in spite of the efficiency of the algorithm.
- The error measure used must be the mean square error. This is not considered to be a major limitation by most authors since this is the most commonly used error measure [87]. However, other error measures are considered in section 1.6.3.

A mathematical treatment of the Levenberg-Marquardt algorithm is given in section 2.3. The effectiveness of this algorithm is discussed in the context of the wave overtopping dataset in Chapter 4 and in the context of various other datasets in Chapter 7.

1.5.6 Global training methods

The Levenberg-Marquardt may be seen as a ‘global’ training method, since it takes into account the gradients of all weights within an ANN simultaneously. However, the individual gradients are local, the underlying assumption being that the error gradient of one weight is largely independent of the other weights. The Levenberg-Marquardt method therefore finds a local error minimum and the closeness of this minimum to the global minimum is dependent upon the (randomly chosen) starting point. The simulated annealing method is designed to search the whole weight space and therefore is potentially a global method. However, the method is inefficient, requiring several visits to each location in weight space in order to settle into the global minimum. In this section two further ‘global’ methods are explained, genetic algorithms and Bayesian data analysis. In both cases, they are global in two senses: they are capable of globally searching the weight-space of a particular neural network architecture. In addition they are able to search the space of possible architectures with the aim of identifying the most appropriate one.

Genetic algorithms

Genetic algorithms (GAs) are search algorithms based upon the process of natural selection. Invented by Holland in the 1960s [88], they were initially used as search and optimisation techniques. More recently they have been used within the machine learning paradigm, including neural network training.

The GA algorithm acts upon a population of ‘genotypes’. These are encodings for possible solutions to the problem in question, or ‘phenotypes’. Typically they are strings of numbers or binary digits. The population of genotypes are ‘evolved’ through a number of ‘generations’. This process is analogous to the traditional training of ANNs through a series of epochs. However, GAs operate on a whole population of neural networks, rather than on a single network. The production of each generation involves three steps-

1. **Reproduction.** In this step, genotypes are copied to the next generation. The number of offspring created will depend upon the fitness of the parent, as measured by an evaluation function. For ANNs this will typically be the mean square error of the corresponding phenotype.
2. **Crossover.** Pairs of genotypes are mated randomly. During this step the pairs of genotypes are split at a random position and each half combined with the appropriate half from the other genotype.
3. **Mutation.** This involves the random alteration of individual values (‘alleles’) within the genotypes. The probability of random mutation is typically very low - about 0.001.

Since success in the reproduction step is dependent upon an evaluation function, there will be a tendency to move towards phenotypes that are more fit for purpose. Since a large population is used the available search-space should be covered more effectively than with traditional ANN training, but the members of the population should converge towards an optimal solution, or solutions, given sufficient generations. [89]

The encoding used within GAs may include any properties selected by the researcher. Within the field of neural networks, the encoded properties fall mainly into three categories: connection weights, neural architectures and learning rules [90, 91]. There has been little research into the evolution of transfer functions. Some exceptions are included in section 1.5.8. When optimising network weights it is often found that GAs are effective at identifying regions that contain minima but less effective at performing local search within these regions. Several researchers have therefore adopted hybrid algorithms, in which GAs are followed by gradient descent training in order to pinpoint the location of minima [92].

Bayesian methods

Bayesian data analysis takes place in three steps [93]:

- Set up a full probability model for all observable and unobservable quantities in a problem.
- Condition on the observed data. This involves calculating the posterior distribution of the unobserved quantities in which we are interested, given the observed data and the chosen probability model.
- Evaluate the fit of the model. If one is dissatisfied with the model, the three steps may be run again, allowing the comparison of different models.

In the context of neural networks, a neural network architecture represents a statistical model, whose inputs are explanatory variables and whose outputs are data points which we hope to predict using the statistical model. The network weights are model parameters that may be adjusted to improve the ‘fit’ of the model. Within a Bayesian framework, it is possible to make inferences concerning the plausibility of a particular model (ANN architecture), a particular set of parameter values (network weights) or specific predictions made by the model. These inferences will be in the form of probability statements - usually probability density functions (pdfs). By making some or all of the inputs to a network the subject of Bayesian analysis, it is also possible to assess the usefulness of different inputs in terms of their explanatory power within the model [94].

The Bayesian approach has a number of advantages over more established methods for ANN training-

- It has a sound conceptual and mathematical basis.
- The assumptions made in constructing the explanatory model are stated explicitly.
- Bayesian methods automatically incorporate the principle of ‘Occam’s razor’. This means that they favour smaller networks with smaller weights, and there is no need to introduce smoothing terms such as weight decay parameters (section 1.5.4) or regularisation parameters (section 1.5.7) in an *ad hoc* way.
- It allows global optimisation of network parameters.
- It permits the comparison of different models, including the selection of network architectures and of network inputs.

- Error bars may be estimated, allowing confidence levels to be assigned to a particular model, to network weights or to network outputs.

1.5.7 Radial Basis Function networks

At a similar time to the re-emergence of MLPs following Rumelhart's exposition of the BP algorithm, the theory of RBF networks was developed. Powell surveyed early work on radial basis functions in 1985 [95]. They were originally conceived not as neural networks but as solutions to ill-posed hyper-surface reconstruction problems. Radially symmetric functions were used as a 'basis' with which to construct an unknown function. The purpose of the basis functions was to expand the inputs into a high-dimensional space, by using a number of functions greater than the dimensionality of the inputs. The reason for transforming the inputs into a higher dimensional space was Cover's theorem on the separability of patterns. In 1965 Cover [96] proved that a pattern-classification task was more likely to be linearly separable if cast in a high-dimensional space. The transformed high-dimensional inputs were then passed through a linear transformation to obtain the final outputs.

Broomhead and Lowe [97] placed the radial basis function approach within the context of neural networks. Each basis function now became the transfer function of a hidden layer neuron, while the output neurons had linear transfer functions. However, the neural network was seen within the context of function-fitting. From this viewpoint, network training is equivalent to hypersurface reconstruction and generalisation is equivalent to multivariable interpolation.

Broomhead and Lowe start from the perspective of strict interpolation, in which the overall approximating function is constrained to pass through all of the training points. For m training patterns this may be achieved by creating m hidden neurons, each with weights coinciding with one training point. Radial basis functions have the same number of free parameters as sigmoid functions. However, the input weight vector is usually interpreted as a 'centre' and the bias weight as a 'steepness'. This is because a radial basis function calculates its net input as the Euclidean distance between its input and its weights. This net input (or distance) is then input into a function that depends only on the bias (or steepness). The overall result is a radially symmetric function, as illustrated by the Gaussian function of figure 1.10. Alternative radial basis functions are described in section 2.1.

RBF networks invariably use a linear output neuron, which simply outputs a weighted sum of its inputs. As a result, the hidden-to-output weights that will give the lowest

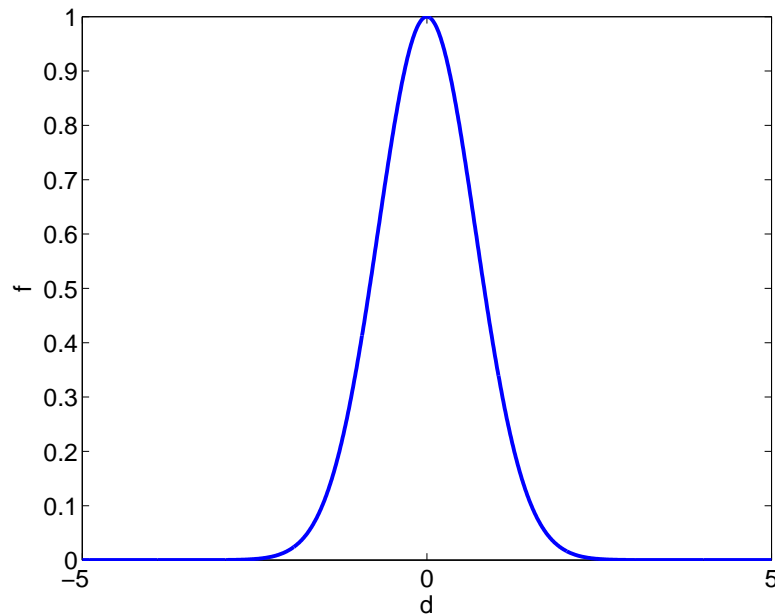


Figure 1.10: Graph of a Gaussian radial basis function

mean squared error may be calculated using linear algebra. There is no need to use the gradient descent techniques common to MLPs.

Broomhead and Lowe proceed to situations in which only a subset of the training patterns are used as RBF centres. The justification for doing this is to reduce the number of degrees of freedom and so reduce the risk of overfitting. In this situation the design matrix \mathbf{A} , which contains the outputs of all hidden neurons given each of the applied inputs, is non-square and the pseudo-inverse gives the least squares solution.

Broomhead and Lowe point out that the use of a subset of the training data as RBF centres results in a smoothing effect, or ‘regularisation’ [97, 98]. However, it has been common to introduce additional regularisation in order to reduce the size of the output weights obtained from the least squares solution. Regularisation was used by Tikhonov as a means of solving ill-posed problems [35, 99, 100]. The problem for ANNs is to approximate the input-output function. This problem is described as ‘ill-posed’ for two reasons-

- In areas of sparse data, there is insufficient information to uniquely identify the underlying function.
- Noise in the data means that distinct input or output values may not be identifiable and there may be apparent discontinuities in the function.

Ill-posed problems may be made well-posed through the introduction of *a priori*

assumptions. These assumptions are smoothness constraints, i.e. they assume that the function is not fully local, so that values at one point may act as a guide to values at nearby points. Constraints are included as an additional function. These smoothness constraints have a similar mode of operation and have a similar effect to the introduction of weight penalties in the weight decay technique discussed in section 1.5.4.

Poggio and Girosi [99] have provided an alternative justification for the use of regularisation with datasets of high dimensionality. Data necessarily becomes sparse when it is of high dimensionality. This is the well-known ‘curse of dimensionality’ [101]. The sparse data results in a large number of possible solutions and in order to choose between them it is necessary to make smoothness assumptions. Since RBF networks transform the input space into a high-dimensional space, regularisation may be particularly applicable to them. A mathematical treatment of regularisation is provided in section 2.5. The issue of regularisation is discussed in detail in the context of the results obtained using the overtopping dataset in Chapter 5.

While Broomhead and Lowe suggest the use of a subset of the training data as RBF centres, they give little guidance on the procedure for making a selection. Moody and Darken selected centres using k-means clustering [102, 103]. Further, they allowed the centres and steepnesses to alter as the result of gradient descent training.

In 1991 Chen *et al.* [104] introduced the forward selection (FS) algorithm for the selection of centres. This is a network growing technique. It introduces a computationally efficient method for calculating the errors of the series of networks created by successively adding a single neuron from the training set. The reduction in error after each addition may be calculated using an orthogonal least squares (OLS) procedure. An RBF network may therefore be grown one neuron at a time, with the neuron selected at each the stage being the ‘best’ choice at that time. Further details are provided in section 2.4.

In the late 1990s Kubat and Orr [105, 106] introduced methods for deriving RBF centres from regression trees. Regression trees partition data into hyper-rectangles. A RBF neuron may be derived from each hyper-rectangle, with its centre at the centroid of the hyper-rectangle and its width proportional to the volume of the hyper-rectangle. Orr considers nodes at different levels of the regression tree, which therefore have differing widths. Nodes are selected for inclusion in the RBF network, with preference given to wider nodes. As a result wider radial basis functions tend to be added early on, with more localised functions built into the system towards the end of training.

1.5.8 Hybrid Neural Networks

Hybrid neural networks, containing both sigmoidal and radial basis transfer functions in the same layer, have been suggested by Poggio and Girosi [99]. However, there have been only a few reported implementations of them.

Cohen and Intrator create hybrid networks which they call Perceptron Radial Basis Nets (PRBFN). Their approach is to cluster the data and then to choose a neuron, either sigmoidal or radial-based, that approximates the local function within each cluster [107]. Their method is compared with a novel algorithm introduced in this study in Appendix C.

Flake's 'square unit augmented, radially extended MLP' (SQUARE-MLP) approach may also be seen as a hybrid approach [108]. Flake uses each of the inputs to a network twice. One is used as a raw input, while the second uses the square of the original value. The hidden layer neurons all have sigmoidal transfer functions. However, when the sigmoidal functions are applied to the squared inputs, the final output functions are similar in form to the Gaussian functions commonly used as radial basis functions. A SQUARE-MLP is therefore similar to a hybrid sigmoid-RBF ANN, with the constraint that the number of RBF-type and sigmoidal neurons must be equal.

Genetic algorithms were introduced in section 1.5.6. They have specific application in the area of hybrid networks, since they are able to search the space of possible architectures, avoiding the need to trial all possible combinations of transfer functions. Liu and Yao used a GA to search through architectures with a single hidden layer containing both sigmoid and RBF neurons, applying their results to the classification of heart disease patients [109]. Jiang *et al.* used a similar method to model concrete stress [110]. Both studies used a GA only for the selection of architectures, with network weights determined by variants on gradient descent.

Hybrid neural networks may be contrasted with modular neural networks, popularised by the work of Jordan and Jacobs [111]. Their 'hierarchical mixture of experts' (ME) approach partitions data and creates separate networks for each partition. The outputs of the networks are then combined using a 'soft' gating function. While Jordan and Jacobs only consider MLP networks as candidates for the individual networks, subsequent authors have admitted the possibility of different types of neuron [112].

1.6 Artificial Neural Networks in Hydroinformatics

As described in section 1.4 numerical modelling incurs a large computational cost, while empirical curve-fitting incurs a large experimental outlay (section 1.3). Both methods have limited applicability, due to these costs. The curve-fitting approach is further limited by any particular choice of free parameters and mathematical function. For this reason there has been recently a growing interest in the use of ANNs as an alternative method of predicting various hydraulic parameters. Whereas the curve-fitting approach uses parametric regression, the ANN approach may be seen as method of non-parametric regression. The large number of free parameters means, in effect, that no assumption is made concerning the mathematical structure of the input-output relationship.

1.6.1 Freshwater Applications of ANNs

Within hydroinformatics, rainfall-runoff modelling is the area in which there has been the most interest in ANNs. The aim of modelling is to predict the level of runoff given known rainfall rates which may vary spatially or temporally, and hence to predict when flooding may occur [113]. This process has clear parallels with the prediction of wave overtopping rates, in terms of both the functional and causal relationships between the input and output variables.

Smith and Eli [114] demonstrated the feasibility of using an ANN to make such a prediction. They used simulated data from a 5-by-5 grid of cells to which were applied variable rainfall rates. They trained a back-propagation network to predict the time and level of peak discharge. Shamseldin [115] found that the inclusion of seasonal information as an input improved the performance of ANNs, across a range of catchment areas.

Hsu *et al.* [116] used a series of time-delayed rainfall measurements to predict runoff levels using measured data from the Leaf River Basin in Mississippi. They obtained results that are superior to those from back-propagation by using a two-step training algorithm, known as linear least squares simplex (LLSSIM). In this algorithm the input-hidden weights are determined first, using a multi-start simplex algorithm [116]. A linear output neuron is employed. In the second step, the hidden-output weights may therefore be determined using a linear least squares optimisation procedure, rather than with gradient descent. The LLSSIM algorithm has considerable similarity to the orthogonal least squares (OLS) algorithm commonly used to train

RBF networks (see section 1.5.7). It is found that this training technique gives results that are an improvement on back-propagation.

Mason *et al.* [117] used synthetic data that included rainfall intensity, cumulative rainfall and derivative of rainfall intensity as inputs. They found that RBF networks trained considerably faster than MLP networks. Fernando and Jayawardena [118, 119] found the same advantage in training speed when using RBF rather than MLP networks. Again the OLS method explains the observed training efficiency (section 1.5.7). Dawson and Wilby [120] found the same advantage in training speed. However, in their study MLPs gave greater prediction accuracy, with RBFs giving accuracy comparable to that from linear models.

Dibike *et al.* [121] compared the performance of various types of ANN on rainfall-runoff prediction. They found that RBF networks trained the fastest, but gave relatively high MSEs. MLPs gave reasonable results, with comparable results from three different methods: BP training with either a unipolar or bipolar sigmoid transfer function, and Levenberg-Marquardt training. A recurrent Elman network was found to train the slowest but also gave the lowest MSE.

Senthil Kumar *et al.* [122] recently compared the ability to model rainfall-runoff scenarios of RBF networks with MLP networks trained using LLSSIM. Their conclusions are quite complex, with the most effective network dependent on flow conditions: RBF networks give lower errors for medium and high flows, while MLPs give better results for low flow conditions.

Some attempts have been made to automate the search for an optimum layer size. Cascade-correlation ANNs have been applied to streamflow predictions by various authors [123, 124, 125, 126]. Thriumalaiah and Deo [124] observed that the cascade correlation algorithm gives much shorter training times than back-propagation, with comparable error values. Muttiah *et al.* [126] created an ANN that could predict peak discharges from a range of watersheds throughout the United States. Yang [127] used a method that combined a genetic algorithm, to perform a global search, with gradient descent, to locate local minima. This was applied to flood forecasting and to water quality estimation. Abrahart *et al.* [128, 129] used pruning techniques and genetic algorithms to find optimum sized networks for river flow forecasting.

Standard MLP networks have also been used for streamflow prediction by several researchers [130, 131, 132]. Generally it has been found that streamflow exhibits very different behaviour under high, low and medium flow-rate conditions, and traditional

ANNs have had difficulty predicting the three types of behaviour within a single network [113].

Hsu *et al.* [133] used a self-organising feature map (SOM) to group the inputs before using linear regression to make runoff predictions for each subset identified. In addition to giving reasonable prediction accuracy and short training times this method yielded information concerning the division of data into groups exhibiting different types of flow behaviour. Furundzic [134] also used a SOM to group data, before inputting the groups of data into three separate MLP networks. Temperature was used as an input variable, in addition to previous rainfall and runoff values. Prediction accuracy was found to be superior to that from linear regression.

See and Openshaw [135, 136, 137] have used a similar technique to group data with a SOM. After creating a trained MLP for each subset of the data, the outputs are recombined using fuzzy logic. A genetic algorithm was used to optimise the if-then rules used. The partitioning of data may therefore be seen as ‘soft’, and the separate ANNs may be seen as parts of an overall modular network.

Zhang and Govindaraju [138] used a mixture of experts (ME) approach [111] to create a modular network. Each module is an ANN that takes all of the inputs. The outputs of all networks are then combined as a weighted sum. The relative weights are determined individually for each datum by a ‘gating network’. Weights within each network are determined within a Bayesian theoretical framework. Rather than using gradient descent, the Bayesian approach is to maximise the posterior likelihood of a set of weights. The ME approach is an extension of the soft partitioning of See and Openshaw, in which the weights assigned to the outputs of each network vary from one input to another in a non-linear fashion.

Maier and Dandy [139] have investigated the effect of adjusting various training parameters in the back-propagation algorithm. They use the prediction of river salinity using time-lagged inputs as a test case. The most significant finding is that choice of transfer function has a strong effect on both training speed and generalisation ability, with the hyperbolic tangent function clearly superior to the unipolar sigmoid function and linear function.

Abrahart and others have investigated the choice of inputs in rainfall-runoff prediction using saliency analysis [140, 141]. He also considers the related problem of data selection. It is pointed out that rainfall-runoff data is very noisy and that any particular partitioning into training, validation and test sets therefore introduces considerable bias into the training process. Abrahart’s solution is to use a ‘bootstrap’ approach, in which

data is regularly resampled during the training process [142]. Finally, various error measures are discussed in addition to Mean Square Error (MSE) including maximum over-prediction and maximum under-prediction [141].

Hsieh and Tang [143] consider the problems of overfitting, data selection and knowledge extraction in the context of ANN models in meteorology and oceanography. Suggested solutions to overfitting are ensemble averaging, early stopping, network pruning and weight decay, while the number of inputs may be reduced through application of principal component analysis (PCA). Spectral analysis is suggested as a means of approximating the non-linear component of an input-output relationship.

Overall, studies on the use of ANNs in freshwater flow prediction have a number of features in common-

- The aim is invariably to predict future flow, or discharge, rates using past flow rates. Within wave overtopping prediction, this is the approach used by numerical simulation. It is not used within this study, which aims only to predict time-averaged overtopping rates, but it would be possible to apply a time-varying approach to wave overtopping prediction.
- Flow rates are seen to exhibit very different behaviour under various conditions. As a result, standard MLP networks often have difficulty making generic predictions across a range of conditions. It has become increasingly common to use a modular approach. Sub-networks are then assigned the task of reconstructing a particular part of the input-output function or a subset of the training data.
- Constructive algorithms, such as cascade correlation or OLS, generally give quicker training times than BP. The resultant test errors are roughly comparable to those obtained from BP, although results are highly problem-dependent.
- An optimum transfer function has yet to be identified. Different studies give preference to either sigmoid or RBF functions.
- Various studies have identified the importance of considering further criteria when designing ANNs for hydroinformatic simulation. These include stopping criteria, model size, input parameter selection and data selection. These issues are discussed further in section 1.6.3.

1.6.2 Coastal Applications of ANNs

Applications of neural networks in coastal hydroinformatics are much rarer than fresh-water studies. The first published paper in this area was by Mase *et al.* [144], published in 1995. This predicted the stability of rubble-mound breakwaters using structural parameters, such as a permeability parameter, as well as sea-state information including water depth and wave steepness.

A number of authors have used previous tide levels to predict future tidal levels, with methods similar to those used in streamflow predictions [145, 146]. ANN outputs have also been combined with harmonic analysis to produce long-term tidal predictions [147]. Deo and Sridhar Naidu [148] used previous wave heights to predict their future values. They found that the cascade correlation algorithm was superior to back-propagation in terms of accuracy and training time. Deo and Kiran Kumar [149] similarly found that the cascade correlation algorithm was efficient in interpolating between monthly mean wave heights to obtain weekly values.

Tsai *et al.* [150] predicted wave heights and periods at one coastal station using values from a series of other stations within Taichung harbour, Taiwan. Similarly, Huang and Murray [151] used water levels at a series of locations to predict tidal currents at an inlet of Long Island, New York. Both studies used a basic MLP. Makarynsky [152] predicted wave heights and periods at one location off the Irish coast using measurements at nearby locations.

Tangang *et al.* [153] used ensemble averaging to predict sea surface temperatures from previous sea level pressure values. In ensemble averaging a number of different networks are trained, with different starting weights [154, 155]. In the study by Tangang different samples of training data were used to train each network. The overall output of the system was then computed by averaging the outputs from all of the networks. The aim of ensemble averaging is to reduce the variance inherent in any single neural network [156].

Recently, attempts have been made to predict wave parameters using independent variables, rather than related measurements at earlier times or nearby locations. Deo *et al.* [157] predicted wave height and wave period by using wind speed over a previous period of time as inputs to a MLP. They found that different ANNs were required for fair weather and monsoon conditions. Deo and Jagdale [158] predicted the heights and period of breaking waves from values of the deep water wave height, deep water wave period and sea bed slope. They used laboratory measurements to train a MLP network. El-Shazly [159] used values of temperature, pressure and wind to predict monthly

mean sea levels at Alexandria over a 7-year period. Best results were obtained using a general regression neural network (GRNN). This is a type of variable width RBF network in which there is a neuron centred at every point within the training data [160].

The European CLASH project has compiled a large database of wave overtopping data, which is used within this study (see Chapter 3. Members of the CLASH project have published a number of relevant papers. Medina *et al.* [161] used an ANN to predict wave overtopping at Zeebrugge breakwater from the crest freeboard and a small number of hydraulic parameters. Training data was obtained using scale models, but testing with prototype data gave mixed results. The trained neural network was used to simulate new data, which was used to create a pseudo-empirical formula with exponential form, similar to equation 1.1. Verhaeghe *et al.* carried out an in-depth study of the data within the CLASH database [162, 163]. Pozueta *et al.* have very recently developed MLPs for predicting overtopping discharges using the database [164]. They use an ensemble approach, with bootstrap resampling from the data. Pozueta's study uses an updated version of the CLASH database that was not available at the time that the research reported in this thesis was performed. It is intended that the findings of this study will be tested further using the most recent database at a future time (see section 9.3).

Several authors have commented on the desirability of extracting symbolic information from neural networks or inserting known relationships into ANNs [165, 166]. Dibike *et al.* [167] showed that a neural network could 'learn' the partial differential equations representing wave behaviour, and that these equations could be reproduced from a trained ANN. The ANN is able to perform a time-step in a finite-difference scheme; given water depths arranged in a grid at time t , water depths at time $t+1$ may be predicted for a simple flume.

Overall, research on coastal applications has been more varied than that on fresh-water applications. There has been more attempt to predict dependent variables from independent variables, rather than using time-series data, and authors have been more adventurous in their choice of network architecture. Due to the sparsity of research in this area however, a consensus has yet to appear on the most effective architectures and training algorithms. The next section discusses these issues.

1.6.3 Design Issues

When designing neural networks, there are a number of issues that should be taken into account [35]. While there have not been sufficient studies in the area of coastal engineering to enable generalisations to be made, a number of authors have considered design issues in the context of ANN use for freshwater studies [38, 120, 122]. The issues fall into the following categories, which are discussed in turn:

- Performance criteria
- Data pre-processing
- Choice of inputs
- Determination of network architecture
- Choice of training algorithm
- Stopping and validation criteria

The most commonly used performance criteria are prediction accuracy and training speed, both of which depend primarily on the choice of training algorithm. Prediction accuracy is most commonly assessed as mean square error (MSE), although alternative measures such as maximum over-prediction and maximum under-prediction have been used on occasion [141, 168]. Even in cases where alternative error measures have been employed, the training algorithms used have minimised the MSE. While researchers within the ANN community usually assume that MSE is the best error measure, many coastal engineers are accustomed to using alternative error measures such as average error factor or average absolute error [31], which are likely to give more useful information when designing sea-walls. The approach used in this study is to use MSE as the error measure while developing of ANNs. This allows the use of standard training methods and makes comparisons with alternative ANN methods straightforward. After various types of neural network have been developed, they are compared using alternative error measures, in section 7.3.

Input data should be scaled so that different inputs have similar importance in training. When using sigmoid output functions, the output data should also be scaled to avoid training in the extreme ‘flat’ areas. In the case of bipolar sigmoid functions this means that a range such as $[-0.8, 0.8]$ should be used. Training may also be more effective if data with near-normal distribution is used.

Using too many inputs is likely to slow down training and lead to over-large networks [38]. It is therefore important to pre-select data before training. However, inputs must be carefully selected on the basis of their information content to avoid discarding useful information. Parameter selection methods include *a priori* knowledge, cross-correlation analysis and principal component analysis. The number of samples to be used in training is a related issue. Attempts have been made to estimate the optimum training-test ratio. Kearns has suggested that a fixed ratio of 80:20 gives reasonable results across a range of datasets [169, 35]. Amari [80] uses a formula that depends upon the size of the dataset relative to the number of free parameters within the ANN. Amari's theory is discussed in section 4.2.

Data needs to be carefully selected to ensure its reliability. While ANNs are tolerant to some degree of noise, they will have difficulty identifying an underlying function in the presence of very high variability. A particular problem occurs in the presence of systematic errors, which are likely to be incorporated into the approximating function since ANNs cannot distinguish between 'true' trends and systematic errors.

Network size is discussed in some detail by Maier and Dandy [38]. They point out that small networks have greater generalisation ability, require less storage space, train and respond more quickly and make rule extraction simpler. However, they also have a more complex error surface, with more local minima. Larger networks generally require fewer training epochs, can form more complex decision regions and are better able to avoid local minima. However, they are computationally expensive and require large training samples to give good generalisation ability. As seen in the last section, some authors have utilised pruning or constructive algorithms in order to automate the process of network size selection.

Senthil Kumar *et al.* [122] concern themselves with a comparison of MLP and RBF networks. Their conclusions are inconclusive. RBF networks are found to train faster and to give lower errors using some measures and particular flow regimes. However, MLP networks give better results using alternative measures.

MLPs have three alternative families of training algorithms : first order gradient descent, second order gradient descent and global methods. The advantages and disadvantages of these methods have been discussed in detail in sections 1.5.4 and 1.5.5. RBF networks have alternative algorithms available to them, as discussed in section 1.5.7. In addition, as we have seen, some authors have used alternative training algorithms, including cascade correlation and LLSSIM.

Amari *et al.* have proved that early stopping should not be required if networks are of reasonable size, i.e. if the ratio of training samples to connection weights exceeds 30 [80]. However, for smaller datasets it is expected that early stopping will improve generalisation performance. This issue is discussed further in section 4.2.

Consideration should be taken of the need to partition the available data into training and test data. Further partitioning of the training data into training and verification sets is also often performed, in order that a verification set is available to identify the optimum network architecture or the time at which to stop training. The manner and proportions in which the partitioning is performed is discussed with reference to the wave overtopping dataset in section 4.3

Chapters 3-5 describe the methods used to pre-process and select data, and to train and assess various types of ANN. One of the aims in these chapters is to demonstrate that the design issues raised in this section have been taken into consideration when conducting that research. It is to be hoped that the results reported in this study will reflect back on some of these design issues, providing evidence concerning the design of ANNs for hydroinformatics. Specifically, observations are made concerning

- the selection of data and its effect on network outputs
- the effectiveness of various training algorithms
- choice of transfer function and its effect on generalisation ability
- model size selection and its relationship to training algorithms and stopping criteria

1.7 CLASH and the development of a hybrid neural network

As shown in section 1.6.2, coastal applications of ANNs have been quite rare. The main reason for this is that large amounts of data are required to train ANNs and this data was not widely available at the time. However, from the second half of the 1990s onwards attempts were made to collect and collate coastal data, and to utilise it in ANN training. One such scheme is the European CLASH project [170, 171], which has collected large amounts of overtopping data from both model and prototype sites. This data covers a wide range of defensive structures and incident wave conditions

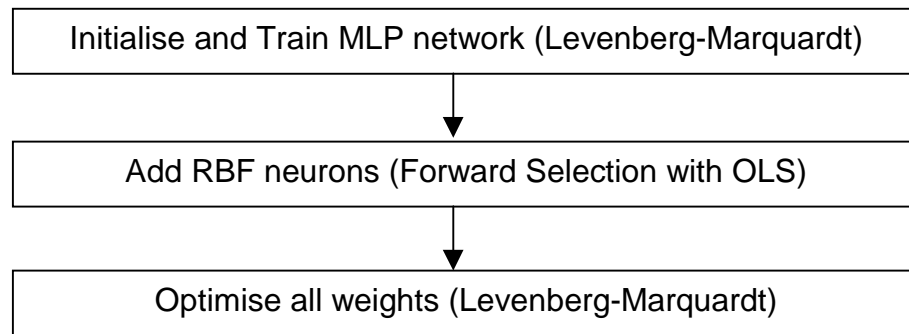


Figure 1.11: The three-step training process used by GL-ANNs

and therefore provides an ideal basis for the training of ANNs designed for generic prediction across a range of scenarios.

This study uses the CLASH database to train a wide range of ANNs, including MLPs and RBF networks. A variety of training algorithms have been used, including variants on gradient descent and least squares optimisation. From this investigation has emerged a hybrid network architecture with a corresponding hybrid training algorithm, which gives superior results to those from simpler architectures and training methods.

The hybrid ANN reported here contains both sigmoid and RBF neurons. The training method involves a three-step training algorithm (figure 1.11). First a MLP network is created and trained. RBF neurons are then selected for addition to the network. The output weights of both sigmoid and RBF neurons are chosen such that they minimise the mean square error of the network. Finally, all weights, including RBF centres and steepnesses, are optimised using gradient descent.

Sigmoid neurons are effective at identifying global features of an unknown function, whereas RBF neurons are able to represent more local variations within the function. The training process used by the hybrid networks therefore identifies the global aspects of the function before identifying the more local features. For this reason we call our networks ‘global-local artificial neural networks’ (GL-ANNs). The results reported in Chapter 7 suggest that GL-ANNs are able to estimate more accurately than either pure RBF networks or MLP networks the input-output relationship within the CLASH data. Further, GL-ANNs are seen to be parsimonious in their use of neurons, at least when compared to RBF networks.

GL-ANNs were also tested using a range of other datasets. Some of these are small, synthetic datasets with few inputs, while others are larger datasets with several inputs and, sometimes, large amounts of noise. From the results of these tests it has been possible to determine areas in which GL-ANNs perform well as well as some

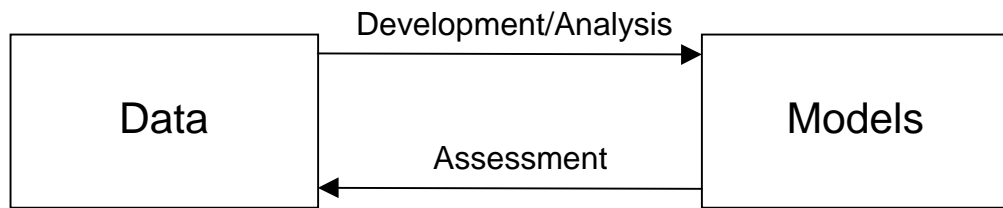


Figure 1.12: The interaction between data and models

of the limitations on their use. Some of the results also lead on to discussions of various issues related to the training of hybrid networks, RBF networks and of ANNs in general. These issues are:

- the need for regularisation when setting output weights
- the determination of optimum RBF steepnesses
- the value of mixed training algorithms involving both deterministic methods and gradient descent training
- criteria for model selection, i.e. the choice of type and number of neurons
- the choice of stopping criteria in training algorithms.

1.8 Overview

This section describes how the strands of wave overtopping and neural network theory are related within this thesis. The relationship may be seen as an interaction between data and the models used to analyse the data. This relationship has two elements: development (or ‘analysis’) and assessment (see figure 1.12). During development, the wave overtopping data acts as the stimulus for a study of neural network architectures. In this phase a narrow range of datasets is considered (just the CLASH dataset), but a wide range of ANN architectures. In the assessment phase, the process is reversed. A single architecture (GL-ANNs) is assessed in terms of its effectiveness in modelling various datasets with different characteristics. The aim of the assessment phase is to determine the strengths and limitations of the GL-ANN architecture and training process.

This thesis may be seen as being in four parts: background, development, assessment and conclusion. This chapter has provided a background to the fields of wave

overtopping prediction and artificial neural networks, as well as describing some previous research that brings together the two fields. Chapter 2 provides further background material, in the form of a number of mathematical methods used in neural network training.

Chapters 3-6 present the development/analysis phase of the research. Chapter 3 describes the CLASH dataset, including the pre-processing of data and selection of input parameters. The process of developing MLPs and the results of training MLPs with variations on gradient descent are described in Chapter 4. Chapter 5 includes the results of training RBF networks with the CLASH dataset and a discussion of these results. Chapter 6 describes in detail the GL-ANN training method and the theory behind it.

Chapters 7-8 describe the assessment phase. Chapter 7 gives the results of training GL-ANN with the CLASH data. Comparisons are made with RBF networks and with MLP networks trained with the Levenberg-Marquardt algorithm. Also included are extensive discussions of several issues arising from these results. Chapter 8 describes a number of benchmark datasets used to explore the applicability of the GL-ANN architecture and algorithm and reports the results of training MLP, RBF and GL-ANN networks with these datasets. Criteria are developed for determining whether the GL-ANN approach is likely to be fruitful for a particular dataset.

Finally, Chapter 9 concludes the thesis and makes suggestions for possible future areas of research.

Chapter 2

Mathematical Techniques for Neural Networks

This chapter describes in detail the mathematical methods used within this thesis. These are all techniques related to the training of ANNs. A preliminary section (section 2.1) introduces various transfer functions commonly used by neural networks. Section 2.2 introduces the algorithms and equations used to perform gradient descent optimisation, including back-propagation and several improvements to the basic BP algorithm. Section 2.3 describes the Levenberg-Marquardt algorithm and gives the equations utilised within the algorithm. Section 2.4 presents the Forward Selection (FS) procedure used to build RBF networks. This section includes detailed treatments of the Least Squares and Orthogonal Least Squares methods used to optimise the output weights during the FS procedure. Section 2.5 gives a mathematical treatment of regularisation within the context of FS.

2.1 Transfer Functions

As we have seen in Chapter 1 each neuron in an ANN has a transfer function. This is a simple mathematical function that takes a number of inputs and transforms them into a single output. Each transfer function has a number of adjustable parameters that correspond to the input weights of the neuron. This section describes two families of transfer functions: pseudo-linear transfer functions and radial basis transfer functions. Duch and Jankowski have provided a full survey of transfer functions, including combinations of pseudo-linear and radial based functions, for the interested reader [37].

2.1.1 Pseudo-linear transfer functions

The transformation performed by pseudo-linear transfer functions takes place in two steps. Firstly, the net input, v , is calculated as a weighted sum of the inputs, as given by equation 2.1. The sum starts with a suffix of 0 rather than 1 to allow for a fixed bias in addition to the variable inputs (see section 1.5.4).

$$v = \sum_{n=0}^p i_n w_n \quad (2.1)$$

A true linear function then passes on the net input unchanged (see figure 2.1a). However, if all the neurons in an ANN have linear transfer functions the overall output of a multi-layer network must be a linear combination of the inputs [172]. In order to make neural networks more versatile non-linearity must be introduced into some or all of the transfer functions. This non-linearity generally introduces limits on the possible outputs of the transfer function, usually $[0, 1]$ or $[-1, 1]$. This is often convenient mathematically, since the target function may have a limited range of possible outputs. It also has some biological validity, since the output of biological neurons is restricted in range [36]. Some commonly used linear and pseudo-linear transfer functions are defined by equations 2.2 - 2.7 and illustrated in figure 2.1.

Equations 2.3 and 2.4 introduce hard-limited thresholds to restrict the output range. These functions are illustrated in figures 2.1b and 2.1c. The remaining functions are sigmoid functions, so called because of their S-shape. These all have the advantage that they are differentiable at all points. This is essential to the operation of gradient descent methods (see sections 1.5.4 and 2.2).

Linear function:

$$f(v) = v \quad (2.2)$$

Threshold function:

$$f(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases} \quad (2.3)$$

Piecewise linear function:

$$f(v) = \begin{cases} 1 & \text{if } v \geq 1 \\ v & \text{if } -1 < v < 1 \\ -1 & \text{if } v \leq -1 \end{cases} \quad (2.4)$$

Logistic function:

$$f(v) = \frac{1}{1 + e^{-v}} \quad (2.5)$$

Hyperbolic tangent function:

$$f(v) = \tanh(v) \quad (2.6)$$

Bipolar sigmoid function:

$$f(v) = \frac{1 - e^{-v}}{1 + e^{-v}} \quad (2.7)$$

2.1.2 Radial basis transfer functions

As with pseudo-linear functions, the outputs of radial basis functions are calculated in two steps. The first step calculates the Euclidean distance, d , between the input and the neuron weights, according to equation 2.8. In order to find d , the two quantities have to be expressed as vectors \mathbf{i} and \mathbf{w} . The subscript j indicates the individual dimensions of the input.

$$d = \|\mathbf{i} - \mathbf{w}\| = \sqrt{\sum_{j=1}^k (i_j - w_j)^2} \quad (2.8)$$

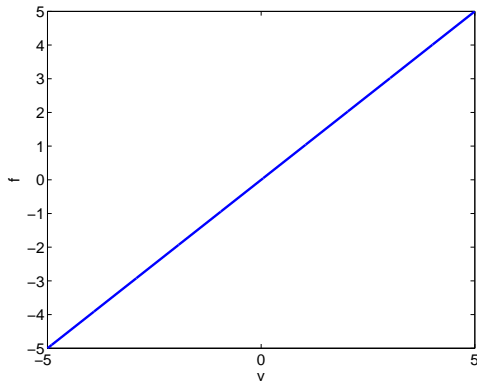
The Euclidean distance is then used as the net input to the neuron. The output of the neuron, y , is calculated as a function of this net input using a transfer function f , as in equation 2.9. Since the final output depends only upon the Euclidean distance d , it must be radially symmetric and centred upon the weight vector \mathbf{w} . For this reason, the weight vector is commonly described as a centre and the bias is often replaced with a ‘steepness’ parameter, σ , since it controls the steepness of the function f .

$$y(\mathbf{i}, \mathbf{w}) = f(d) \quad (2.9)$$

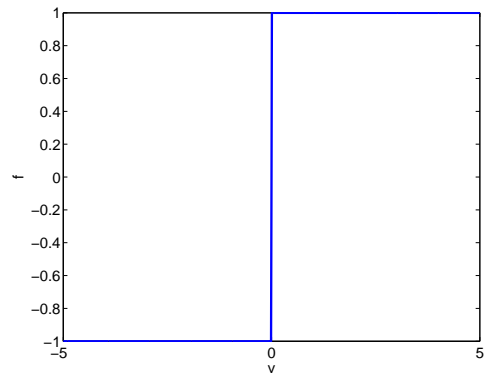
Various functions may be used in equation 2.9. Some commonly used functions are described by equations 2.10-2.15 and illustrated in figure 2.2 [35].

Triangular function:

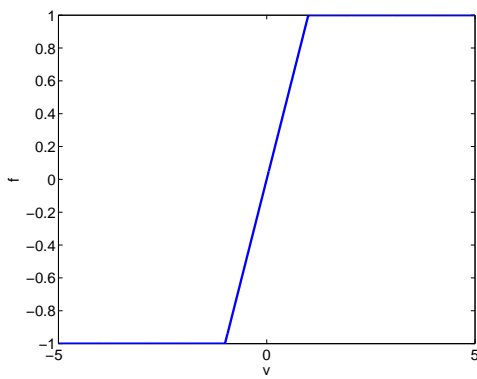
$$f(d) = \begin{cases} 0 & \text{if } d \leq -1 \\ 1 - |d| & \text{if } -1 < d < 1 \\ 0 & \text{if } d \geq 1 \end{cases} \quad (2.10)$$



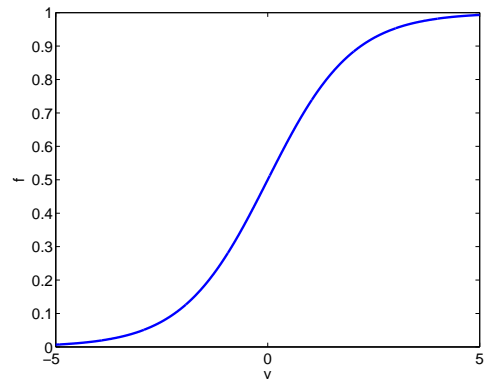
(a) linear



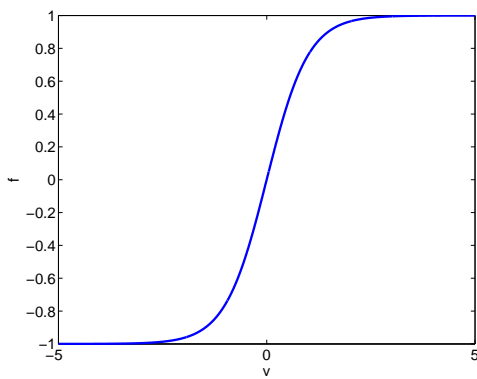
(b) threshold



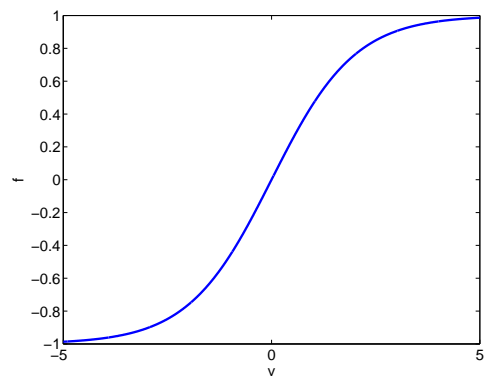
(c) piecewise linear



(d) logistic



(e) hyperbolic tangent



(f) bipolar sigmoid

Figure 2.1: Linear and pseudo-linear transfer functions

Thin plate spline:

$$f(d) = d^2 \ln |d| \quad (2.11)$$

Multiquadratic function:

$$f(d) = \sqrt{d^2 + \omega^2} \quad (2.12)$$

Inverse multiquadratic function:

$$f(d) = \frac{1}{\sqrt{d^2 + \omega^2}} \quad (2.13)$$

Gaussian function:

$$f(d) = e^{-\sigma^2 d^2} \quad (2.14)$$

Radial hyperbolic tangent function:

$$f(d) = 1 - \tanh(d^2) \quad (2.15)$$

2.2 Gradient Descent

2.2.1 Adaptive linear elements

During gradient descent training, the error gradient with respect to the weights in a network is calculated and weight changes are made in the direction of the error gradient. The error gradient is a vector quantity. It is therefore necessary to calculate the individual partial gradients with respect to each network weight. As long as the steps made during each weight update are small the direction of travel should be in the direction of the steepest gradient.

The simplest possible network contains neurons with linear activation functions and no hidden layer. Such networks have been described as ‘adaptive linear elements’ (ADALINE) [173]. The inputs pass directly to the output neurons and, for a particular input \mathbf{x} , the output of each output neuron is given by equation 2.16. Since ADALINES contain no hidden layer, the local input vector \mathbf{i} is identical to the input to the network, \mathbf{x} . Equation 2.16 is therefore identical to equation 2.1 except for the replacement of i by x .

$$y = \sum_{n=0}^p x_n w_n \quad (2.16)$$

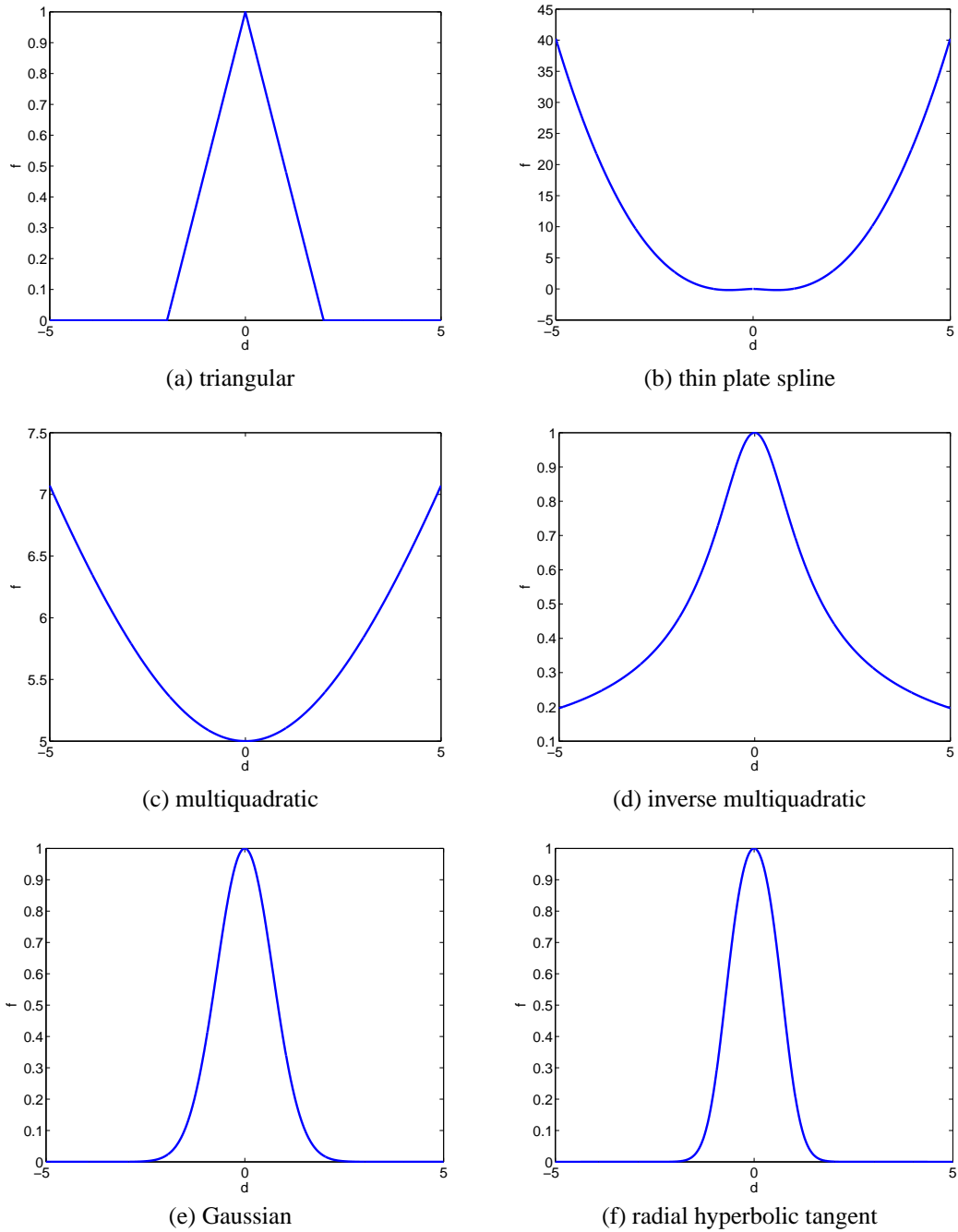


Figure 2.2: Radial basis transfer functions

The function to be minimised by ADALINE's is the squared error, E . If the target output for the output neuron is t , the squared error is defined by equation 2.17.

$$E = (t - y)^2 \quad (2.17)$$

Substituting for y and differentiating with respect to the weight vector \mathbf{w} gives the error gradient $\frac{dE}{d\mathbf{w}}$ of equation 2.18. The vector gradient may be separated into individual partial gradients $\frac{\partial E}{\partial w_j}$, given by equation 2.19.

$$\frac{dE}{d\mathbf{w}} = -2(t - y) \mathbf{x} \quad (2.18)$$

$$\frac{\partial E}{\partial w_j} = -2(t - y) x_j \quad (2.19)$$

In order to minimise the error, we wish to move in the opposite direction to the error gradient. If we introduce a learning rate η , the individual weight updates are then given by equation 2.20, in which η is a positive real number. The algorithm incorporating this weight update is known as the 'least squares rule' since it has been shown that it will lead to convergence to the least squares solution, given an appropriate choice of η [65].

$$\Delta w_j = \eta(t - y) x_j \quad (2.20)$$

The discussion so far has considered a single input vector. When considering m input patterns, the relevant error function is the sum of squared errors S , defined by equation 2.21. The overall error gradient is given by equation 2.22 and the weight updates by equation 2.23.

$$S(\mathbf{w}) = \sum_{i=1}^m (t_i - y_i)^2 \quad (2.21)$$

$$\frac{dS}{d\mathbf{w}} = -2 \sum_{i=1}^m (t_i - y_i) \mathbf{x}_i \quad (2.22)$$

$$\Delta w_j = \eta \sum_{i=1}^m (t_i - y_i) x_{ij} \quad (2.23)$$

The reader's attention is drawn to the difference between equation 2.20 and equation 2.23. The latter implies that the weight changes from all patterns should be

summed before being applied to the network. This process is known as ‘batch’ weight updating. The alternative procedure, represented by equation 2.20, is commonly described as ‘stochastic’ weight updating. It is to be expected that the batch update procedure will converge to the global minimum more quickly, due to superior gradient information. However, the stochastic method is often preferred when solving real problems. This issue has been discussed in section 1.5.4, as has the choice of learning rate.

2.2.2 Multi-layer perceptrons

Transfer functions in most MLPs are not restricted to linear functions. In order to extend gradient descent, the least squares rule must be extended to allow for a variety of transfer functions f . The gradient of the SSE with respect to the input weights of an output neuron is then expressed by equation 2.24, in which v_i is the net input to the given neuron given the i th input.

$$\frac{dS}{d\mathbf{w}} = 2 \sum_{i=1}^m (t_i - y_i) f'(v_i) \mathbf{x}_i \quad (2.24)$$

When compared to equation 2.22 it is seen that an extra term, $f'(v_i)$, has been introduced, to reflect the dependency of the outputs on the transfer function. When using stochastic weight updates, the individual weight adjustments may be expressed as in equations 2.25-2.26.

$$\delta = (t - y) f'(v) \quad (2.25)$$

$$\Delta w_j = \eta \delta x_j \quad (2.26)$$

Equation 2.25 shows that there is a factor, δ , common to the updates of all weights of a particular neuron. For this reason this training rule has become known as the ‘delta rule’. The weight updates are also seen to be proportional to the local input vector \mathbf{x} .

The greatest difficulty with the use of multiple layers was the problem of credit assignment, first identified by Minsky in 1961 [174]. The problem may be seen as one of identifying the extent to which a particular weight in a network is responsible for the final output of the network. This is required in order to assess the degree to which a particular weight should be adjusted during training. Output neurons have a direct effect on output values, and hence on SSEs. Hidden neurons have an indirect effect on

outputs, in that they affect the inputs to the output neurons. It is therefore more difficult to calculate their effect on the final output values.

Provided the hidden and output neurons have differentiable functions, there is a solution to the problem, using the chain rule. Equation 2.26 showed that output neuron weight updates that achieve steepest gradient descent are proportional to local gradients and to the local input vector. In a multi-layer network, this vector is no longer the same as the overall input to the network and is therefore denoted by \mathbf{i} rather than \mathbf{x} , leading to equation 2.27.

$$\Delta w_j = \eta \delta_j \quad (2.27)$$

\mathbf{i} is in turn dependent on the outputs from the previous layer of neurons. Application of the chain rule yields the dependence of the SSE on the hidden layer neurons, and hence the weight updates required for steepest gradient descent. These are given in equations 2.28-2.29, in which g and u are the transfer function and net input, respectively, of the j th hidden layer neuron.

$$\delta_j = g'(u) \sum_k \delta_k w_{kj} \quad (2.28)$$

$$\Delta w_{ji} = \eta \delta_j x_i \quad (2.29)$$

Δw_{ji} is the weight update for the i th weight of the j th neuron in the hidden layer. The k subscript refers to the neurons in the output layer. Thus the contribution of the hidden neuron to each of the output neurons is summed.

The δ values calculated using equation 2.28 may in turn be passed back to the previous layer if there are two hidden layers, and so on. For this reason, the training rule has been described as the generalised delta rule [76] and the training algorithm is commonly known as back-propagation of error or ‘BP’.

2.2.3 Modifications to back-propagation

This section gives a mathematical treatment of some modifications to BP, as described in sections 1.5.4 and 1.5.5.

Momentum

The introduction of momentum into gradient descent training speeds up convergence by introducing a variable learning rate. When successive updates are in the same direction the algorithm emphasises the weight changes. On the other hand, when successive weight changes are in opposing directions, the size of the updates is reduced. At the n th epoch, the weight updates are given by equation 2.30, in which $\Delta w_i(n)$ and $\Delta w_i(n-1)$ are the current and previous weight updates, respectively.

$$\Delta w_i(n) = \eta \delta_j(n) x_i(n) + \alpha \Delta w_i(n-1) \quad (2.30)$$

The momentum coefficient α is constrained such that $0 \leq \alpha < 1$. The effect of this term is therefore to add a fraction of the last update to the current update.

Weight Decay

Weight decay aims to reduce network overfitting by adding a penalty term to the error function [81]. The penalty term is commonly chosen to be the sum of squares of the network weights. The error function is then given by equation 2.31 and the error gradient for the i th weight is given by equation 2.32. The weight decay parameter, λ , controls the level of weight decay, or ‘regularisation’, applied during training.

$$E = E_0 + \frac{1}{2} \lambda \sum w_i^2 \quad (2.31)$$

$$\delta = \delta_0 - \lambda w_i \quad (2.32)$$

Simulated Annealing

Simulated annealing tries to avoid becoming trapped in local error minima by using a fairly high learning rate, but disallowing some weight changes. In order to encourage convergence a pseudo-temperature T is introduced. This parameter determines the probability of a weight change being made. The temperature is reduced during training and the probability of a weight change being performed is given by equation 2.33.

$$p(\Delta w) = e^{(-\Delta E/T)} \quad (2.33)$$

An annealing schedule must be introduced. This commonly includes an exponential drop in temperature, with training stopping when a fixed number of epochs have

led to no reduction in error [83].

Quickprop

Quickprop was introduced by Fahlman [84] and has been particularly associated with the cascade correlation algorithm. The weight updates are calculated according to equation 2.34, in which $\mathbf{g}(t)$ and $\mathbf{g}(t-1)$ are the current and previous values of the error derivative.

$$\Delta \mathbf{w}(t) = \frac{\mathbf{g}(t)}{\mathbf{g}(t-1) - \mathbf{g}(t)} \Delta \mathbf{w}(t-1) \quad (2.34)$$

Its operation is similar to that of gradient descent with momentum, with the previous weight update taken into account when calculating the current update.

2.3 Levenberg-Marquardt method

The Levenberg-Marquardt method is a second-order method [175, 86]. Rather than finding the error minimum directly it aims to locate the zero of the error gradient. The zero α of a univariate function f may be found using the Newton-Raphson method according to the iterative formula of equation 2.35.

$$\alpha_{n+1} = \alpha_n - \frac{f(\alpha_n)}{f'(\alpha_n)} \quad (2.35)$$

When extended to a multivariate function, α becomes a vector and the derivative of the function is now a vector derivative, as in equation 2.36.

$$\alpha_{n+1} = \alpha_n - \frac{f(\alpha_n)}{\nabla(\alpha_n)} \quad (2.36)$$

In the case of neural network optimisation, we wish to find the zero of the error gradient \mathbf{g} with respect to the network weights. Since \mathbf{g} is a vector quantity and is itself a derivative, we have to work with the Hessian matrix \mathbf{H} (equation 2.37).

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{\mathbf{g}(\mathbf{w}_n)}{\mathbf{H}(\mathbf{w}_n)} \quad (2.37)$$

Each element in the Hessian contains second derivatives of the error function, summed over all training patterns. However, the error measure E is related to the outputs and target outputs by equation 2.17. The elements within the Hessian therefore contain values like that in equation 2.38, summed across all training patterns.

$$\frac{\partial^2 E}{\partial w_i \partial w_j} = 2 \left(\frac{\partial y}{\partial w_i} \frac{\partial y}{\partial w_j} + (y - t) \frac{\partial^2 y}{\partial w_i \partial w_j} \right) \quad (2.38)$$

One can calculate local values of the first derivatives. These are the δ values used in the gradient descent method. The second derivatives in the above equation are disregarded when estimating the Hessian. This is a reasonable estimate since the error $(y - t)$ is expected to be small. Further, we expect the values of $(y - t)$ to have an approximately Gaussian distribution with mean zero. When summed over a large number of training patterns the second terms are therefore likely to cancel out to a large extent.

Having obtained an approximation of the Hessian, the Newton-Raphson method may be used to find the nearest zero of the error gradient. Two problems may arise. Firstly, the local Hessian estimation may not be an adequate representation of the underlying function. Secondly, the second-order algorithm by itself may approach a maximum or saddle point on the error surface, rather than a minimum. In order to avoid these problems, the Levenberg-Marquardt method includes an additional gradient descent term. The weight adjustment vector is then given by equation 2.39.

$$\Delta \mathbf{w} = (\mathbf{H} + \lambda \text{diag}(\mathbf{H}))^{-1} \mathbf{g} \quad (2.39)$$

The parameter λ adjusts the relative weighting given to Newton's method and to gradient descent. If the error falls after applying the weight adjustment, λ is decreased. If, on the other hand, the error increases, the weight changes are reversed, λ is increased and the weight changes are re-calculated.

2.4 RBF centre selection

Forward selection of centres (FS) is a method used to choose the centres to be used within hidden neurons of a RBF network. Candidate centres are restricted to the input vectors of the training set. The task is to add one centre at a time from the available centres, so as to give the greatest possible reduction in SSE after each addition. This section describes the mathematical underpinnings of the FS method.

2.4.1 Fully interpolated networks

Early work on RBF networks focused on fully-interpolated networks. These networks contained the same number of hidden neurons as there were elements in the training

set, with one centre corresponding to each input vector. The output layer performed a weighted sum of the radial basis function outputs, i.e. all output layer neurons had linear transfer functions. We may express the hidden layer outputs as a square matrix \mathbf{F} , whose elements F_{ij} represent the output of the j th neuron given the i th input. This matrix is commonly called the ‘design matrix’ [176]. As described in section 2.1.2, the outputs of radial basis functions depend upon the distance between the input and weight vectors. For this reason \mathbf{F} is necessarily symmetric. The final outputs are related to the hidden layer outputs by equation 2.40, in which w_j are the hidden-output weights.

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} F_{11} & \cdots & F_{1m} \\ \vdots & \ddots & \vdots \\ F_{m1} & \cdots & F_{mm} \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} \quad (2.40)$$

The output weights are then determined using equation 2.41, provided \mathbf{F} has an inverse.

$$\mathbf{w} = \mathbf{F}^{-1}\mathbf{y} \quad (2.41)$$

Micchelli [177] has proved that \mathbf{F} is necessarily non-singular for a number of functions, including multiquadratics, inverse multiquadratics and Gaussian functions (see figure 2.2), provided that none of the input vectors are identical. An exact solution to equation 2.41 must therefore exist.

2.4.2 Least squares solution

This section considers a situation in which only a subset of the available centres are used, so the number of hidden neurons is less than the size of the training data. One must now distinguish between the full design matrix \mathbf{F} and the design matrix for the network which we are considering, \mathbf{A} . In this situation \mathbf{A} is non-square and an exact solution is not possible: the actual output vector \mathbf{y} will not be identical to the target output vector \mathbf{t} . However, the pseudo-inverse gives the least squares solution to equation 2.40. This solution is given by equation 2.42 [176].

$$\mathbf{w} = (\mathbf{A}^T\mathbf{A})^{-1} \mathbf{A}^T\mathbf{t} \quad (2.42)$$

It would be possible to try all possible combinations of input vectors, calculate the output weights and hence the final outputs before choosing the network with the lowest

error function. However, there are efficient ways to calculate the reduction in error upon adding a single neuron. First, the projection matrix, \mathbf{P} , is calculated according to equation 2.43. This matrix is called the ‘projection matrix’ because it projects the vectors within \mathbf{F} , which for m input patterns are m -dimensional, into the space of the ANN model, which for n hidden neurons is n -dimensional.

$$\mathbf{P} = \mathbf{I}_m - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{t} \quad (2.43)$$

The SSE is then given by equation 2.44.

$$S = \mathbf{t}^T\mathbf{P}^2\mathbf{t} \quad (2.44)$$

2.4.3 Forward Selection

The use of the projection matrix does not in itself lead to an improvement in computational efficiency, since it is still necessary to invert an m -by- m matrix to obtain the SSE for a network. However, when neurons are added one at a time, there is an efficient method for updating the projection matrix, given by equation 2.45. In this equation \mathbf{f}_j is a column in the full design matrix \mathbf{F} .

$$\mathbf{P}_{n+1} = \mathbf{P}_n - \frac{\mathbf{P}_n\mathbf{f}_j\mathbf{f}_j^T\mathbf{P}_n}{\mathbf{f}_j^T\mathbf{P}_n\mathbf{f}_j} \quad (2.45)$$

Further, the reduction in SSE upon adding the neuron J to the network may be obtained as equation 2.46. By running through all possible centres (J values) it is possible to identify the one which will give the greatest reduction in SSE at each stage [176].

$$S_n - S_{n+1} = \frac{(\mathbf{t}^T\mathbf{P}_n\mathbf{f}_j)^2}{\mathbf{f}_j^T\mathbf{P}_n\mathbf{f}_j} \quad (2.46)$$

2.4.4 Orthogonal Least Squares

A further improvement in computational efficiency is achievable by factorising \mathbf{F} into an orthogonal matrix $\tilde{\mathbf{F}}$ and an upper triangular matrix [104]. Each time a neuron is added to the network an adjustment must be made to $\tilde{\mathbf{F}}$ according to equation 2.47 in order to keep the columns orthogonal to each other.

$$\tilde{\mathbf{F}}_{n+1} = \tilde{\mathbf{F}}_n - \frac{\tilde{\mathbf{f}}_J \tilde{\mathbf{f}}_J^T \tilde{\mathbf{F}}_n}{\tilde{\mathbf{f}}_J^T \tilde{\mathbf{f}}_J} \quad (2.47)$$

The benefit is that the reduction in SSE may now be found without the computation of the projection matrix, according to equation 2.48.

$$S_n - S_{n+1} = \frac{(\mathbf{t}^T \tilde{\mathbf{f}}_J)^2}{\tilde{\mathbf{f}}_J^T \tilde{\mathbf{f}}_J} \quad (2.48)$$

2.5 Regularisation

Regularisation is very commonly introduced into the FS algorithm. During regularisation, a penalty term is added to the error function in order to avoid overfitting, indicated by very large connection weights [98]. Thus, instead of minimising the SSE, one would minimise the function in equation 2.49. $P(f)$ acts as a ‘stabiliser’, smoothing the overall function f .

$$S = \sum_i (y_i - t_i)^2 + \lambda P(f) \quad (2.49)$$

λ is a regularisation parameter, and P may take different forms. A commonly used stabiliser is the sum of squared weights. The approach then has clear parallels with the introduction of weight decay into gradient descent training of MLPs (see section 1.5.4). The solution that minimises the cost function S is then given by equation 2.50, in which \mathbf{I}_n is the n -by- n identity.

$$\mathbf{w} = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}_n)^{-1} \mathbf{A}^T \mathbf{t} \quad (2.50)$$

The reduction in SSE upon adding a single neuron may be calculated according to equation 2.51 if using least squares or according to 2.52 within the orthogonal least squares paradigm.

$$S_n - S_{n+1} = \frac{(\mathbf{t}^T \mathbf{P}_m \mathbf{f}_J)^2}{\lambda + \mathbf{f}_J^T \mathbf{P}_n \mathbf{f}_J} \quad (2.51)$$

$$S_n - S_{n+1} = \frac{(\mathbf{t}^T \tilde{\mathbf{f}}_J)^2}{\lambda + \tilde{\mathbf{f}}_J^T \tilde{\mathbf{f}}_J} \quad (2.52)$$

2.6 Summary

This chapter has described a number of mathematical techniques that are involved in the training of ANNs. Some techniques have particular relevance to later chapters in this thesis:

- Back-propagation, the Levenberg-Marquardt algorithm and momentum terms are used in the training of MLP networks described in Chapter 4.
- Forward selection with orthogonal least squares and regularisation are particularly relevant to the training of RBF networks reported in Chapter 5.
- The Levenberg-Marquardt algorithm and Forward Selection with Orthogonal Least Squares provide the basis for the GL-ANN algorithm described in Chapter 6.

Chapter 3

The CLASH Dataset

CLASH is the acronym for ‘Crest Level Assessment of coastal Structures by full scale monitoring, neural network prediction and Hazard analysis on permissible wave overtopping’. It is a European Union funded project that includes thirteen partners in seven different countries. One of its objectives is to develop a generic method for the prediction of wave overtopping rates using artificial neural networks as a tool [171]. Overtopping rates are quoted as mean rates over the period of a storm (usually about 2 hours for full-scale measurements) for a unit length of wall. They therefore have units $m^3/s/m$.

3.1 Data collection

As a first step towards the generic method a database has been created. This database contains data from both laboratory scale-model tests and from full-scale measurements at operational sea-walls. Data falls into two categories, hydraulic and structural. Hydraulic data describes the observed sea-state in terms of wave heights, wave periods, wave steepnesses and angle of wave attack. In some cases data is available at the toe of the structure and in other cases for deep water near the structure. Structural data are a parameterised representation of the sea-wall in question. Individual variables are mostly dimensions of parts of the structure, such as R_c , the crest freeboard, or B_h , the berm width.

Much of the data in the database was collected before the start of the CLASH project and in some cases did not contain all of the required parameters. It has therefore been necessary to calculate estimates of unknown parameters from known ones [162]. There are three main gaps in the data. In some cases hydraulic data is only

available in deep water. It is then necessary to run a numerical simulation in order to obtain values at the toe of the structure. Processing by the ‘simulating waves nearshore’ (SWAN) program allows wave heights and periods at the toe of the structure to be estimated from their deep water counterparts plus information concerning the foreshore characteristics [163]. In some cases $T_{m-1,0,deep}$, the spectral wave period at deep water, is not available either but other measurements of wave period have been measured. This problem is solved more easily, since $T_{m-1,0,deep}$ and the peak wave period $T_{p,deep}$ are related approximately by the simple relationship of equation 3.1.

$$T_{m-1,0,deep} = \frac{T_{p,deep}}{1.1} \quad (3.1)$$

Finally, there are different ways of calculating the significant wave height. In some cases the significant wave height is quoted as $H_{1/3,toe}$ rather than $H_{m0,toe}$ ¹. The method of Battjes and Groenendijk [178] is then used to calculate the total variance of the water surface elevation, m_0 , from which $H_{1/3,toe}$ may be obtained using the simple relationship of equation 3.2.

$$H_{m0,toe} = 4 \sqrt{m_0} \quad (3.2)$$

As we have seen, parameters at the toe of the structure may be calculated from their deep water counterparts, but the opposite is not true. The deep water parameters therefore contain some gaps. For this reason, and also because deep water characteristics affect wave overtopping only indirectly, it was decided that hydraulic parameters at the toe of the sea-walls would be used in this study. This results in fifteen independent parameters, of which four are hydraulic and eleven are structural. In addition, there are thirteen composite parameters, which are combinations of some of the independent parameters. Finally each datum (set of variables) has a unique name, a ‘reliability factor’ and a ‘complexity factor’. The reliability and complexity factors measure the accuracy of the data. Data with a high reliability factor were measured using techniques with considerable variability. High complexity factors indicate a complex sea-wall structure that is not fully represented by the structural variables. Further details are provided in section 3.2. The available input variables are listed in table 3.1 and illustrated in figure 3.1.

The database is currently at an interim stage. Due to the technical difficulties in

¹ $H_{1/3,toe}$ is defined as the average height of the highest third of the waves within a random wave-train at the toe of the structure whereas $H_{m0,toe}$ is a wave height defined as four times the standard deviation of a random wave-train and is obtained from spectral analysis

Symbol	Variable description
$H_{m0,toe}$	significant wave height at the toe of the structure
$T_{m-1,0,toe}$	mean wave period at the toe of the structure
β	angle of wave attack relative to the normal
h	water depth at the toe of the structure
h_t	water depth over the toe of the structure
B_t	width of the toe of the structure
γ_f	roughness/permeability factor of the structure
$cot\alpha_u$	mean cotangent of the slope, upward of the berm
$cot\alpha_d$	mean cotangent of the slope, downward of the berm
R_c	crest freeboard of the structure
B_h	width of the berm
h_B	water depth over the berm
A_c	armour crest freeboard of the structure
G_c	width of the structure crest
$s_{m-1,0}$	wave steepness
RF	reliability factor
CF	complexity factor

Table 3.1: Variables in the CLASH database

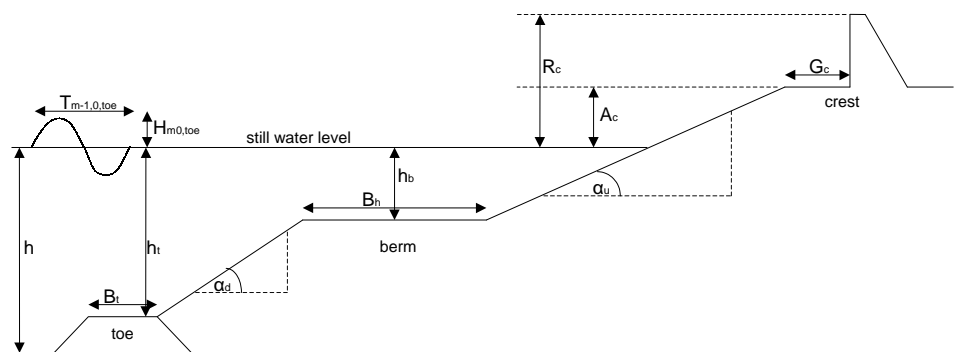


Figure 3.1: Cross-sectional view showing sea-wall structural parameters

collecting data, parameter values are often highly variable. In some cases this has resulted in very similar, or even identical, inputs having radically different measured overtopping rates. A further problem is the existence of ‘white spots’. These are areas of the data that represent viable structures, but for which little data is available. An objective of the CLASH project is to rectify the problems of faulty data and white spots by generating additional data. This will be achieved through numerical simulation and through laboratory tests and should result in a ‘cleaner’ dataset. However, the database as it stands is very ‘noisy’.

3.2 Data selection and pre-processing

Since some data is from small-scale models while other data is from full-scale prototypes, there is a very large variation in raw parameter values. In order to prevent this large variation from obscuring functional relationships, composite parameters were used in ANN training. Lengths were converted to their ‘dimensionless’ counterparts by dividing by the wave height at the toe of the structure, $H_{m0,toe}$. For this reason, this parameter was omitted from training.

Wave period and overtopping rate were converted to dimensionless quantities using appropriate powers of $H_{m0,toe}$ and the acceleration due to gravity, g . The process of creating dimensionless quantities is well-established and is commonly known as ‘Froude’s law scaling’, since it is based on Froude’s law of similarity. This states that two systems A and B have dynamic similarity if the ratio of the inertia force to the gravitational force is equal for the two systems [27]. This law is usually stated in terms of the Froude number, F , which is a constant (for dynamically similar systems) with the value given in equation 3.3, in which u is a characteristic velocity, l is a characteristic length and g is the gravitational acceleration.

$$F = \frac{u}{\sqrt{lg}} \quad (3.3)$$

The velocity in system A is then related to the velocity in system B by equation 3.4.

$$\frac{u_A}{\sqrt{l_A g_A}} = \frac{u_B}{\sqrt{l_B g_B}} \quad (3.4)$$

Given that velocity=length/time, the characteristic times in the two systems, τ , are related according to equation 3.5.

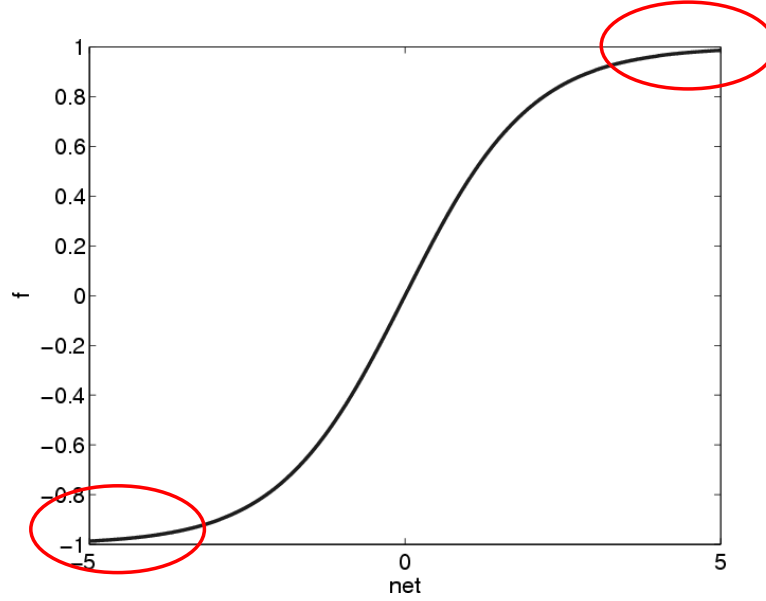


Figure 3.2: Saturation in a sigmoid transfer function

$$\sqrt{\frac{g_A}{l_A}}\tau_A = \sqrt{\frac{g_B}{l_B}}\tau_B \quad (3.5)$$

There have been some concerns as to the validity of this law when applied to freshwater laboratory tests and full-scale seawater scenarios (section 1.3). In particular, viscosity and surface tension do not scale with physical dimensions. However, research suggests that errors incurred from Froude law assumptions are likely to be small [27]. From this point onwards, the subscript 0 is used to indicate a dimensionless quantity. For example R_0 is the dimensionless crest freeboard, equal to $\frac{R_c}{H_{m0,toe}}$.

Equation 3.5 implies that, if lengths are scaled according to some characteristic length, times should be scaled proportional to the square root of that length divided by \sqrt{g} . The non-dimensional mean wave period T_0 is therefore given by the quantity $T_{m-1,0,toe} \sqrt{\frac{g}{H_{m0,toe}}}$.

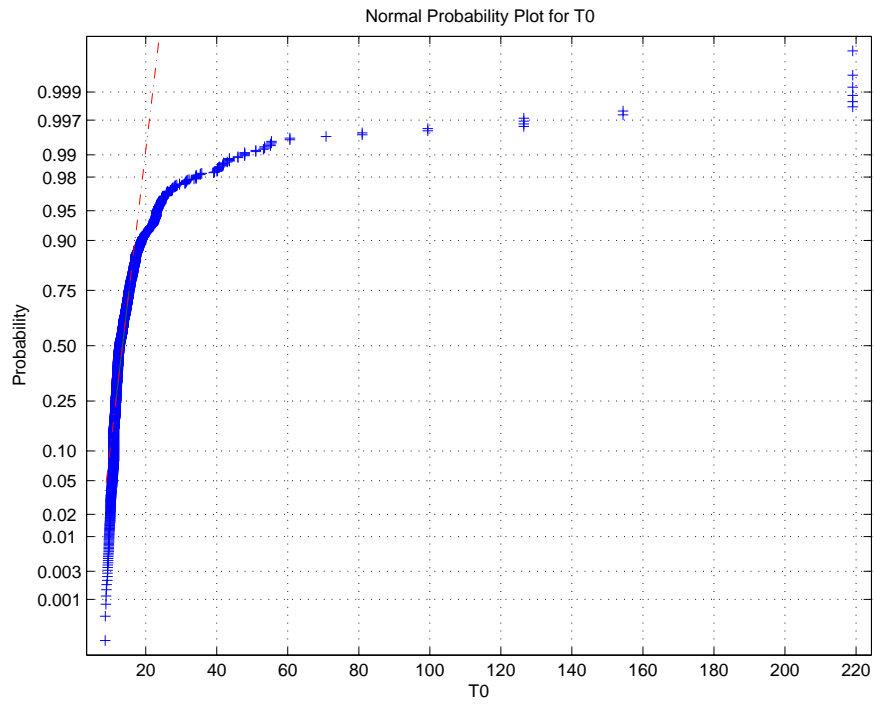
The wave overtopping rate has units $m^2 s^{-1}$ and must therefore have a scaling factor that is the product of the scaling factors for a length and for a velocity, i.e. $\sqrt{l^3 g}$. The non-dimensional wave overtopping rate q_0 is therefore given by $\frac{q}{\sqrt{gH_{m0,toe}^3}}$.

Some of the training variables were transformed before training in order to achieve near-Normal distributions, since it is known that training is more effective with Normally distributed data [179]. Transformations considered were inverse and natural

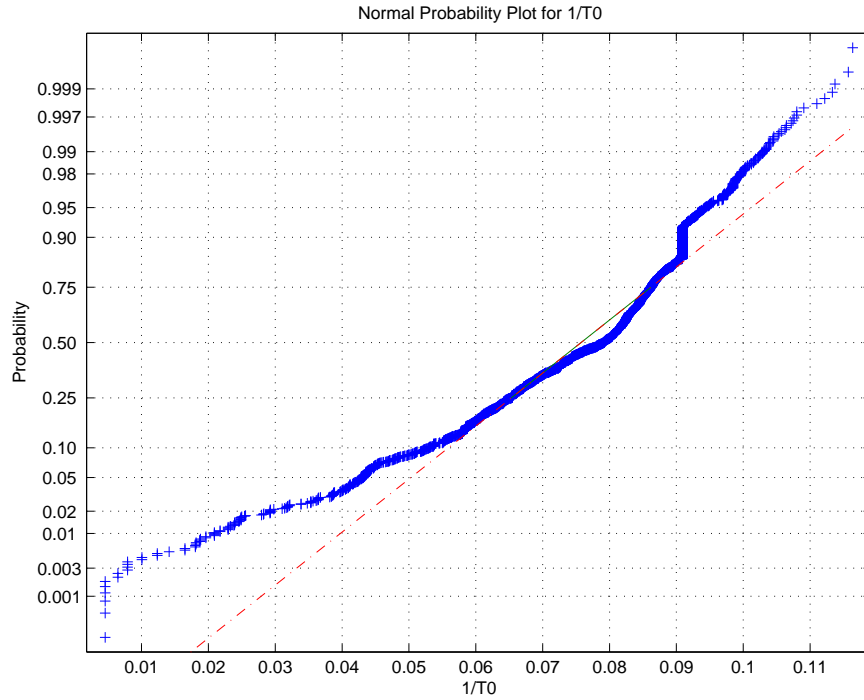
logarithm. Visual inspection was used to determine whether to use one of these transformations, resulting in the use of $1/T_0$, $\ln(h_{t0})$ and $\ln(B_{t0})$ as inputs and $\ln(q_0)$ as an output. Normal probability plots for raw and transformed variables are given in figures 3.3-3.6.

All variables were linearly transformed ('normalised') in order to give them a range of $[-0.8, 0.8]$. The result is that all input variables have a similar effect on weight adjustments, rather than inputs with large values dominating training [180]. In addition this technique avoids 'saturation' [179]. This phenomenon occurs as a result of the shape of sigmoid functions. These functions have the largest gradients when the net input has a moderate value (close to zero). At more extreme values the gradient is very low as illustrated by the red ellipses in figure 3.2. If training moves any of the neurons' outputs into this area there is likely to be a reduction in training speed. In some cases a neuron's weights becomes completely stuck in one area, leading to a reduction in predictive accuracy.

In order to reduce redundancy in the data and hence speed up training, not all of the potential fifteen inputs were used in neural network training. The wave steepness, $s_{m-1,0}$ was not measured directly but was calculated from other parameters (see equation A.6 in Appendix A). It therefore contains no additional information and was not used in training.

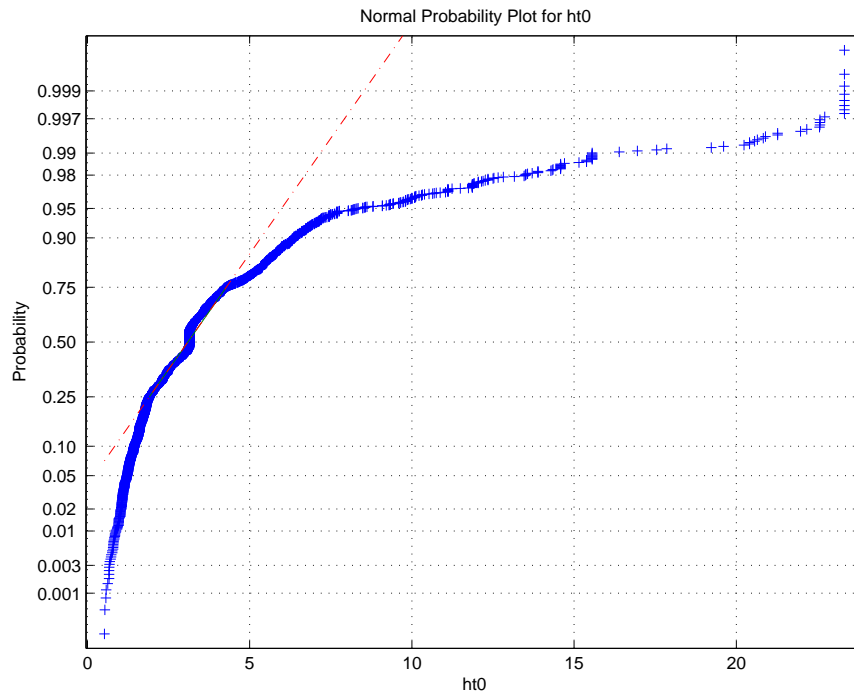


(a) T_0

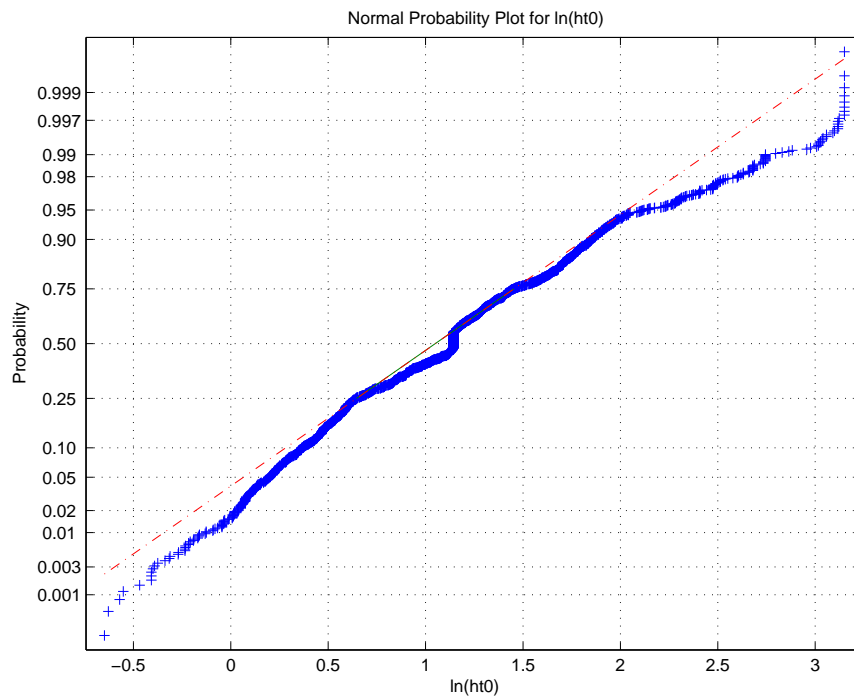


(b) $1/T_0$

Figure 3.3: Raw and transformed Normal probability plots for T_0

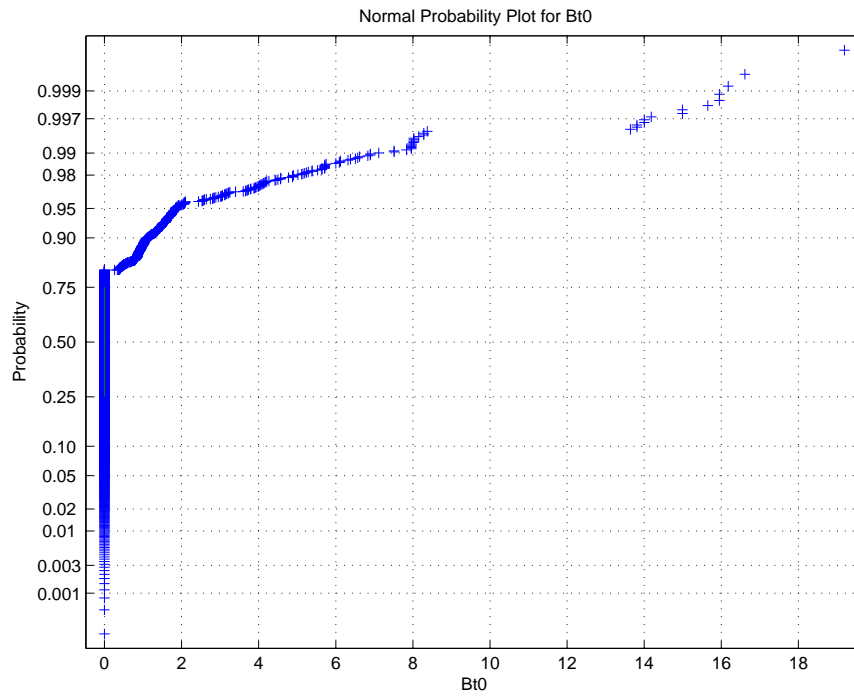


(a) h_{t_0}

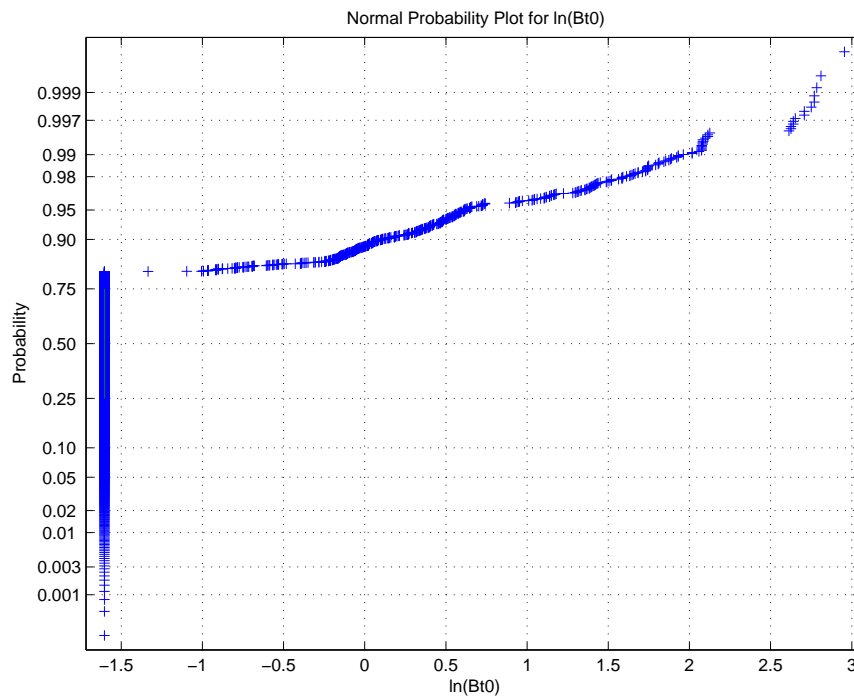


(b) $\ln(h_{t_0})$

Figure 3.4: Raw and transformed Normal probability plots for h_{t_0}

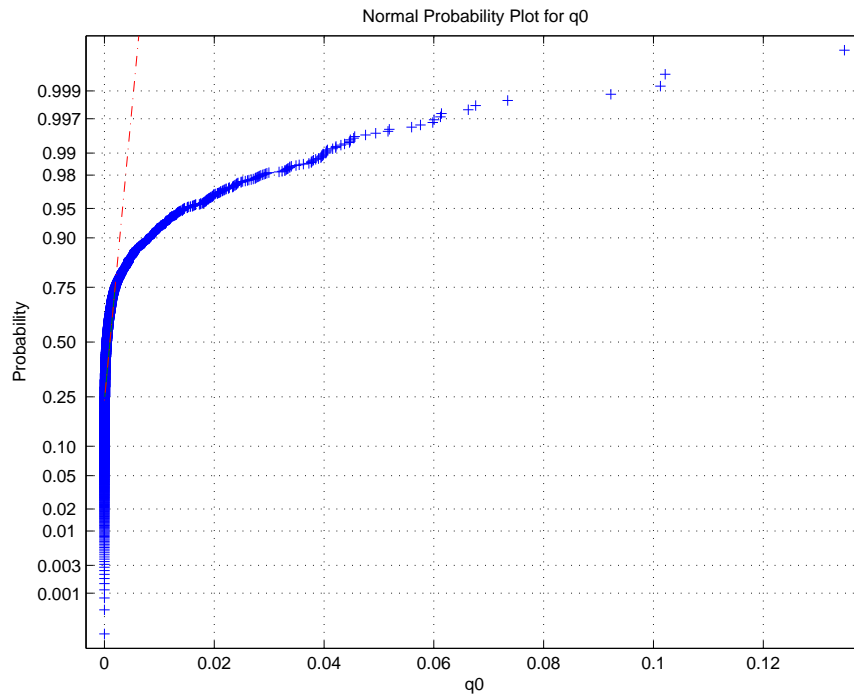


(a) B_{t0}

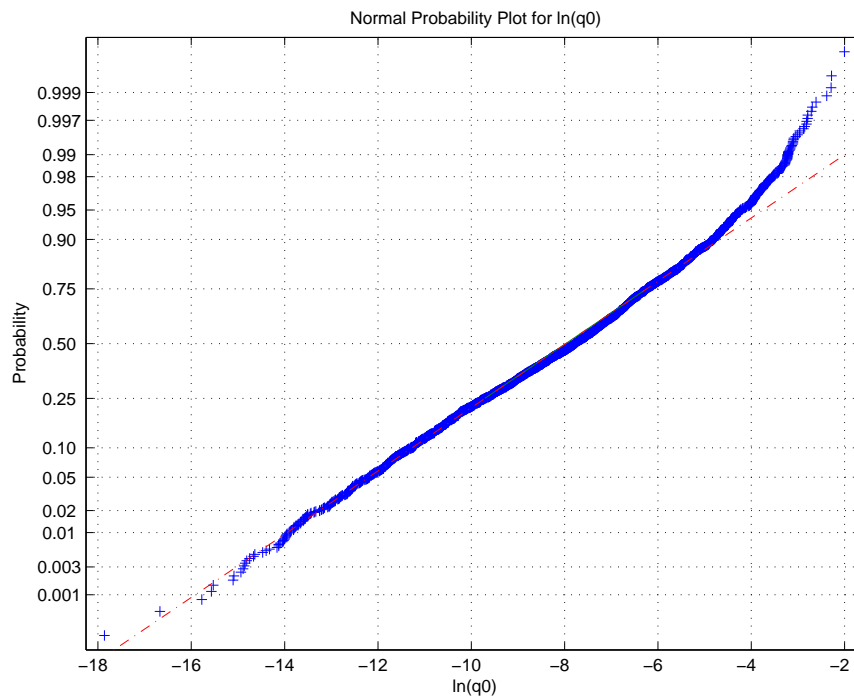


(b) $\ln(B_{t0})$

Figure 3.5: Raw and transformed Normal probability plots for B_{t0}



(a) q_0



(b) $\ln(q_0)$

Figure 3.6: Raw and transformed Normal probability plots for q_0

	β	$\ln(h_{t0})$	$\ln(B_{t0})$	h_{b0}	B_{h0}	G_{c0}	R_0	$1/T_0$	γ_f	$\cot(\alpha_d)$	A_{c0}	$\cot(\alpha_u)$	h_0	$\ln(q_0)$
β	1.000	0.166	-0.064	-0.005	-0.011	-0.108	0.001	0.036	0.157	0.032	0.027	0.047	0.155	-0.012
$\ln(h_{t0})$	0.166	1.000	-0.327	0.130	0.107	-0.188	0.002	0.170	0.344	0.262	-0.034	0.182	0.857	0.095
$\ln(B_{t0})$	-0.064	-0.327	1.000	0.017	-0.122	0.216	0.088	0.049	-0.316	-0.190	0.126	-0.074	-0.064	-0.263
h_{b0}	-0.005	0.130	0.017	1.000	0.423	-0.029	-0.098	0.024	0.181	0.043	-0.076	0.039	0.200	0.169
B_{h0}	-0.011	0.107	-0.122	0.423	1.000	-0.173	-0.094	-0.006	0.223	0.161	-0.049	0.035	0.100	0.020
G_{c0}	-0.108	-0.188	0.216	-0.029	-0.173	1.000	-0.159	0.045	-0.636	-0.057	-0.244	0.009	-0.142	-0.280
R_0	0.001	0.002	0.088	-0.098	-0.094	-0.159	1.000	-0.186	0.163	-0.097	0.928	-0.082	0.030	-0.429
$1/T_0$	0.036	0.170	0.049	0.024	-0.006	0.045	-0.186	1.000	-0.066	-0.138	-0.191	-0.138	0.084	-0.162
γ_f	0.157	0.344	-0.316	0.181	0.223	-0.636	0.163	-0.066	1.000	0.156	0.206	0.129	0.261	0.381
$\cot(\alpha_d)$	0.032	0.262	-0.190	0.043	0.161	-0.057	-0.097	-0.138	0.156	1.000	-0.081	0.820	0.197	0.097
A_{c0}	0.027	-0.034	0.126	-0.076	-0.049	-0.244	0.928	-0.191	0.206	-0.081	1.000	-0.084	0.031	-0.360
$\cot(\alpha_u)$	0.047	0.182	-0.074	0.039	0.035	0.009	-0.082	-0.138	0.129	0.820	-0.084	1.000	0.162	0.104
h_0	0.155	0.857	-0.064	0.200	0.100	-0.142	0.030	0.084	0.261	0.197	0.031	0.162	1.000	0.067
$\ln(q_0)$	-0.012	0.095	-0.263	0.169	0.020	-0.280	-0.429	-0.162	0.381	0.097	-0.360	0.104	0.067	1.000

Table 3.2: Correlation coefficients for the CLASH data parameters

Input parameters	Output parameter
$T_{m-1,0,toe}$ $\sqrt{g/H_{m0,toe}}$ β $h_t/H_{m0,toe}$ $B_t/H_{m0,toe}$ γ_f $\cot\alpha_d$ $R_c/H_{m0,toe}$ $B_h/H_{m0,toe}$ $h_b/H_{m0,toe}$ $G_c/H_{m0,toe}$	$\frac{q}{\sqrt{gH_{m0,toe}^3}}$

Table 3.3: Parameters used in ANN training

Three pairs of structural parameters were highly correlated with each other ($\sigma > 0.8$) and only one member of each pair was therefore included in training. These were R_c and A_c , h and h_t , and $\cot(\alpha_d)$ and $\cot(\alpha_u)$ (see table 3.2). The remaining parameters are given in table 3.3.

One other interesting piece of information that may be extracted from the correlation coefficients is the correlation with the output $\ln(q_0)$. Predictive variables which correlate highly with the output are likely to have greater predictive power. The input variables come in the following order, with the variables with the largest values of $|\sigma_{q_0}|$ first-

$$R_0 > \gamma_f > A_{c0} > G_{c0} > \ln(B_{t0}) > h_{b0} > 1/T_0 > \cot(\alpha_u) > \cot(\alpha_d) > \ln(h_{t0}) > h_0 > B_{h0} > \beta$$

This information should be treated with care, since correlation coefficients only measure the degree of linear correlation between variables. There may be non-linear dependencies that are not revealed by this measure.

The complexity factor (CF) reflects the extent to which the parameterised representation within the database is an accurate description of the physical structure. Approximations have been made in the process of parameterisation. For example, the berm is assumed to be horizontal. In cases where the berm is not horizontal, an approximation, or ‘schematisation’, is made: the sloping berm is replaced by a horizontal berm, with the slopes above and below the berm adjusted such that the positions of the crest and toe of the wall are unchanged, as illustrated in figure 3.7 [163].

In cases where the structure is simple and is accurately described by the database parameters, the data is assigned a complexity factor (CF) of 1. However, when approximations have been made during the parameterisation, CF may take values of 2, 3 or

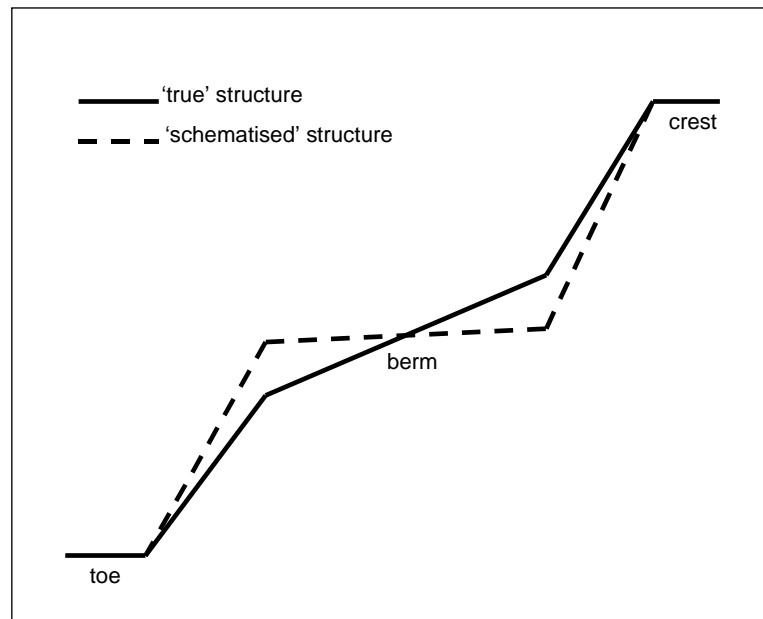


Figure 3.7: Schematisation of a structure with a non-horizontal berm

4.

The reliability factor (RF) reflects the technique used to measure overtopping volumes. For practical reasons this measurement may encounter considerable difficulties, particularly for prototype measurements. RF may also take values between 1 and 4 inclusive. The detailed determination of CF and RF parameters is a complex process and goes beyond the scope of this study. Further details are available in [163].

Data with high RF or CF factors have greater variability and are therefore less useful in neural network training. For the purposes of neural network training and testing, only data with RF values of 1 or 2 and CF values of 1 have therefore been used.

The aim of this study is to predict overtopping rates. For this reason, data with a zero recorded overtopping rate has not been included. There has been some debate concerning the meaning of ‘zero overtopping’, with some researchers treating values below $10^{-6}m^3/m/s$ as zero. One reason for doing this is that q values near or below this rate are particularly difficult to measure accurately. However, $10^{-6}m^3/m/s$ is considered to be the cutoff rate above which overtopping is considered dangerous to high-speed vehicles and may cause damage to buildings (see figure 1.1). Given that the primary practical use of wave overtopping prediction is in hazard warning systems it seems foolish to exclude data in this region. All data with a recorded overtopping rate above zero has therefore been included, although this increases the variability of

the data.

Filtering the data such that $CF = 1$, $RF = 1$ or 2 and $q_0 > 0$ removes approximately half of the data within the database. This leaves 3053 items of data, which is still a large sample for ANN training and testing. Due to the high variability and high-dimensionality of the data, however, a large sample size is seen to be very valuable.

3.3 The nature of the CLASH dataset

The preceding two sections have considered the collection, selection and pre-processing of the data within the CLASH dataset. This section considers what the final dataset is like. Firstly, the distribution of the data is considered. Section 3.3.1 examines the marginal distributions with respect to individual parameters. Then section 3.3.2 investigates the overall multivariate distribution of the data - particularly the degree of clustering within the data. The remaining subsections explore relationships between the input and output parameters. Section 3.3.3 considers the extent to which this relationship may be approximated by a simple exponential relationship, while section 3.3.4 assesses how accurately the input-output relationship as a whole may be described by linear regression, as a means of assessing the linearity (or non-linearity) of the data.

3.3.1 Marginal distributions

Despite the transformations described in section 3.2, there remain some irregular features of the dataset. An examination of the marginal distributions reveals the following-

- 95 % of the data has a wave attack angle, β , of 0° . If neural networks are trained with this data, their accuracy in predicting oblique wave attack is likely to be quite poor.
- 78 % of the data has a zero toe width, B_t . Again, prediction for structures with significant toe width is likely to be inaccurate.
- 85 % of the data has equal gradients above and below the berm. This explains the high correlation coefficient for the two parameters, mentioned in the last section. Only the cotangent of the slope below the berm, $cot(\alpha_d)$, has been used in ANN training and assessment.
- The roughness/permeability coefficient γ_f takes a limited range of values: 56% of the data have $\gamma_f=1.0$, 21% have $\gamma_f=0.55$, 15% have $\gamma_f=0.4$ and only 8%

have other values (between 0.55 and 1.0). Intermediate values of γ_f represent a ‘white spot’ in the data, like the area of non-zero β and non-zero B_t . An aim of the CLASH project is to fill in white spots in the data, by running additional laboratory scale tests. It is to be expected that the final database will therefore contain a fuller set of data, allowing more accurate prediction within these areas [181, 182].

3.3.2 Data clustering

The distribution over marginal distributions has been considered in section 3.3.1. It is more difficult to describe the multi-variate distribution as a whole. Lawrence *et al.* estimate the overall spread of the data by plotting k-nearest neighbour (K-NN) density estimates for different datasets [183]. The K-NN technique takes an integer parameter k . For each point within a dataset it finds the k nearest points. It then finds the volume of the sphere required to contain these points, V , and estimates the data density around each point as k/V [184].

Histograms of the data density may be plotted to indicate the spread of densities. For evenly distributed data we expect such a graph to show a sharp peak, as the data density is equal at all points in data-space. On the other hand, data that is clumped into localised clusters with sparse areas between the clusters will show a wide variation in data densities. The histograms of figure 3.8 are typical of highly clustered data. The data densities have been scaled to have a median of 1.0. However, there is considerable spread of densities between the values of 0.0 and 1.7, and a large tail of densities, with over a third of the data having densities above 1.7.²

A physical interpretation of the clustering behaviour is possible. Data has been collected from a wide variety of defensive structures, with sets of data usually collected for each structure, or family of structures. We might therefore expect clusters to appear, each one representing a different type of structure, e.g. smooth near-vertical wall, rubble mound breakwater, etc.

Lawrence *et al.* use the spread of the data to predict whether a dataset is more appropriately modelled using neural networks with ‘local’ or ‘global’ transfer functions. They cite the interquartile range as a summary measure, with values over 1.2 favoured by local functions and lower values favoured by global functions. The interquartile

²The large bar on the righthand side of the graphs (more visible on the linear graph than on the logarithmically scaled graph) indicates data densities with values of 10 or above. This data has been aggregated in order to fit it into the graphs while retaining a reasonable scale.

range for the CLASH dataset is 2.4, which indicates that it is strongly favoured by local methods. This prediction is tested in the next two chapters, which model the CLASH dataset using, respectively, global functions in the form of MLP networks and local functions in the form of RBF networks.

3.3.3 Exponential relationships

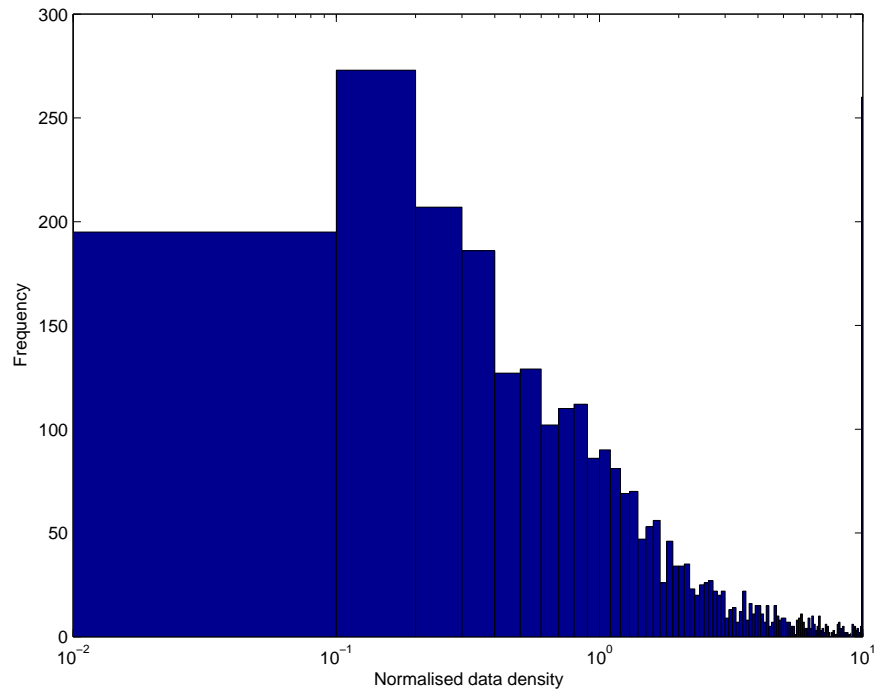
When designing neural networks, it is often useful to build prior knowledge into the training process, so that the network can concentrate on learning unknown structure in the data. From past experience it is known that there are relationships between some of the data variables. In particular, it is known that an exponential function gives a reasonable fit to the dependence of q_0 on R_0 and T_0 , represented by the Besley equation described in section 1.3 and restated as equation 3.6 for convenience.

$$q_0 = AT_0 \exp\left(\frac{-BR_0}{T_0}\right) \quad (3.6)$$

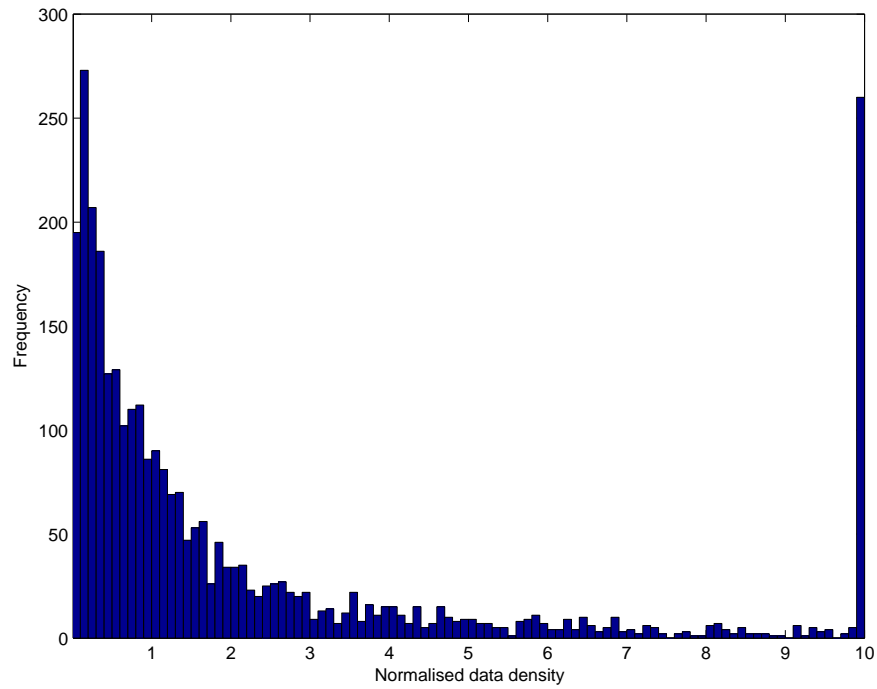
This relationship gives further justification to the process of taking the logarithm of q_0 and the inverse of T_0 . These transformations were introduced in section 3.2 as a means of achieving a near-Normal distribution. However, they could also be seen as a way of building *a priori* information into the training process.

Figure 3.9 plots the predictions of the Besley equation against measured values of q_0 . It is seen that an exponential function gives a reasonable estimate of the relationship between R_0/T_0 and q_0 , although the particular values of A and B tend to give a conservative estimate of the overtopping rate, i.e. the overtopping rates predicted are usually slightly higher than the measured rates. The remaining variance in the predictions may be due to three factors-

- non-linearities in the functional relationship between the main variables (R_0 , T_0 and q_0).
- effects due to the other variables. In particular, we might expect the detailed geometries of the defensive structures to have an effect that would appear as if overlaid over the gross feature of the structure, i.e. the dimensionless crest freeboard R_0 .
- variability in the measurements due to imprecision in measuring techniques and the effects of factors not included in the parameterisation used.



(a) logarithmic scale



(b) linear scale

Figure 3.8: k-nearest neighbour density estimates for the CLASH dataset

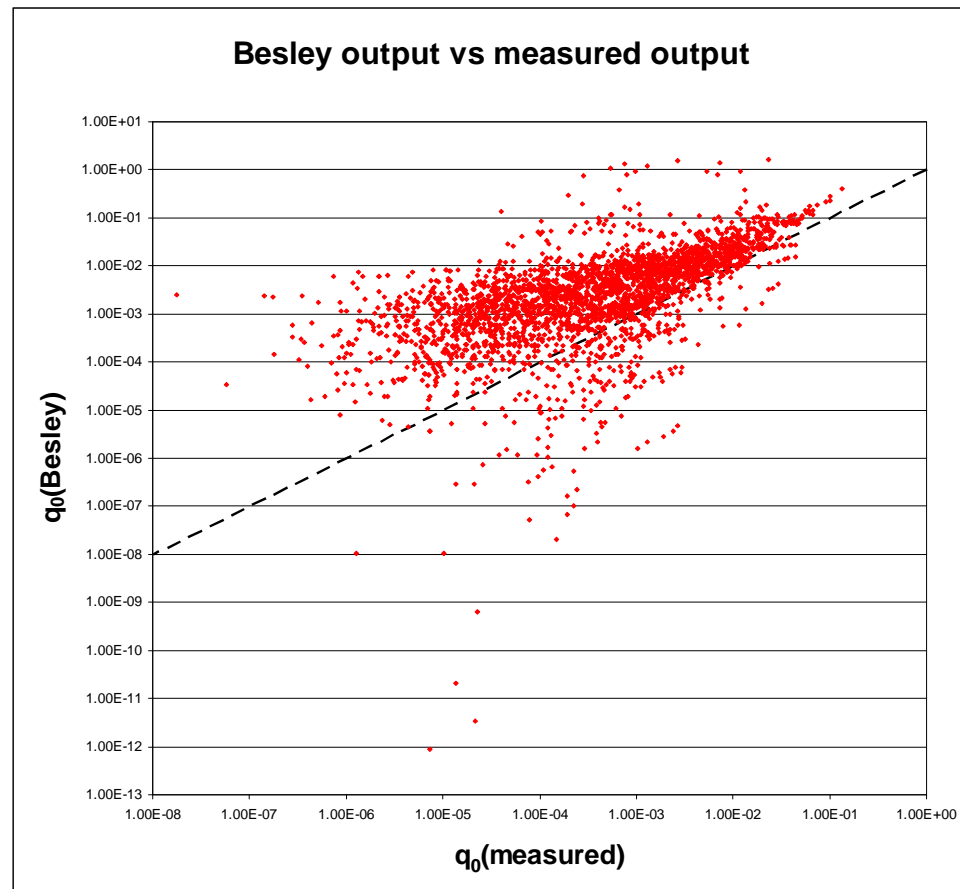


Figure 3.9: Besley predictions compared to measured overtopping rates

In using an ANN to model overtopping it is hoped that the first two features will be identified due to the inherent non-linearity of the ANN approach. Further, the effects of variability in the data should be minimised due to the ability of neural networks to interpolate between noisy data.

3.3.4 Linearity of the CLASH dataset

The last section identified a relationship between $\ln(q_0)$ and R_0/T_0 , while recognising that there are additional factors that affect the input-output relationship. This section aims to go further, by putting a value on the extent of non-linearity within the underlying function and by considering the effect of all of the input variables.

Linear regression analysis was performed on the CLASH dataset. The data used was selected and pre-processed as described in section 3.2. The resulting studentised residuals are plotted against the fitted outputs, i.e. estimates of normalised $\ln(q_0)$, in

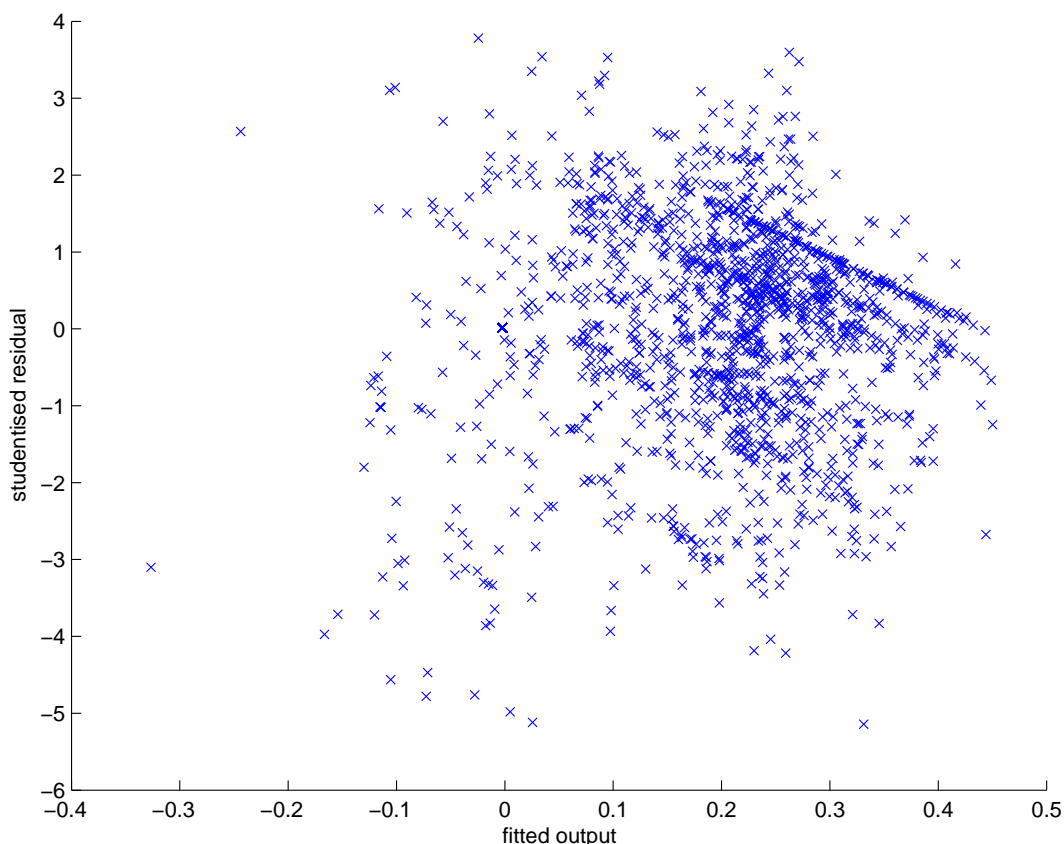


Figure 3.10: Plot of studentised residuals vs. estimated q_0 after linear regression on the CLASH dataset

figure 3.10. For datasets which have an approximately linear input-output relationship we would expect the distribution of residuals to be independent of $\ln(q_0)$. For the CLASH dataset this is seen to be partly the case. However, the residuals are seen to be higher for low $\ln(q_0)$ values. The R^2 value obtained from regression analysis is 0.42. This may be interpreted as implying that about 42% of the variance in the data may be ‘explained’ using linear combinations of the independent parameters, leaving a considerable non-linear component of the function to be explained.

3.4 Summary

The treatment of the CLASH dataset in this chapter may be summarised as follows. Data has been selected to remove the less reliable data and the variables with the most explanatory power have been selected. All data has been normalised to allow for variability in the scale of different parameters and to allow comparisons between structures

at widely different scales. Some parameters have also undergone mathematical transformations in order to give near-Normal marginal distributions.

The characteristics of the CLASH dataset may be summarised as follows. The CLASH dataset is currently noisy and has a number of gaps, or ‘white spots’. There is considerable clustering of the data. Some evidence points to a degree of linearity within the data. However, there are also considerable non-linear elements in the relationship between the independent parameters and $\ln(q_0)$.

Chapter 4

CLASH prediction using MLP Networks

4.1 Introduction

The theory behind the training of MLP networks has been explained in detail in sections 1.5.4 and 1.5.5. As explained in section 1.6.3 there are a number of design issues that have to be taken into account when creating neural networks. It is becoming increasingly accepted that there is not a single ‘best’ approach to designing a neural network, and that the optimum approach depends upon the dataset under consideration [36]. This chapter therefore considers how MLP networks may best be applied to the specific problem of wave overtopping prediction.

Section 4.2 reports the results of a number of pilot studies, which were designed to identify the optimum training parameters to be used in gradient descent training. Section 4.3 describes the technique used to select the optimum architecture, while section 4.4 describes the overall method used to train the networks. The results are reported and discussed in section 4.5.

4.2 Gradient Descent Pilot Studies

The results of BP training are known to be dependent upon various training parameters. In general, the optimum values of these parameters are different for each dataset and cannot be determined from theory alone. A series of pilot studies were therefore performed in order to identify the most effective parameters prior to the main study. Each pilot study involved a series of tests aimed at narrowing down optimum, or near

optimum, values for one or more of the following parameters-

- mode of weight updates (batch or stochastic)
- learning rate
- stopping criterion and maximum number of training epochs
- weight initialisation ranges
- linear vs. sigmoid output neuron
- momentum coefficient

4.2.1 Pilot study 1: Learning rates and weight update mode

In the first set of tests MLPs containing between 5 and 20 hidden neurons were created. 10 random splits of the data were made to create separate, equally sized training and test sets. For each architecture a variety of learning rates were applied to the networks and each network was trained 10 times. Each of the 10 runs had the same starting weights but used a different selection of the training data. All network weights were initialised to small random values in the range $[-0.1, 0.1]$. Training was continued for 5000 epochs and both training and test errors were recorded every 100 epochs.

Figures 4.1 and 4.2 show the MSEs on the test and training datasets, respectively. A range of learning rates were applied in stochastic mode, and each learning rate is represented by a different line on the graphs. MSE values are quoted in terms of the normalised values of q_0 , which have a range of $[-0.8, 0.8]$ (see section 3.2). The best results are obtained with a learning rate of 0.02. The optimum hidden layer size is 13, with a resultant test MSE of 0.0136 (averaged over all 10 runs). This average error, if applied equally to all datapoints, is equivalent to an error factor in the actual (unnormalised) value of q_0 of 3.2.

When comparing figures 4.1 and 4.2 it is clear that the test and training errors track each other very closely. The lines on the two graphs are therefore almost exactly parallel, with a difference between the training and test errors of approximately 0.002. This suggests that the training error is a good guide to the test error. It also implies that overtraining is not occurring to a great extent: if overtraining was occurring, we would expect networks that result in a low training error to have a high test error, as a result of overfitting the training data.

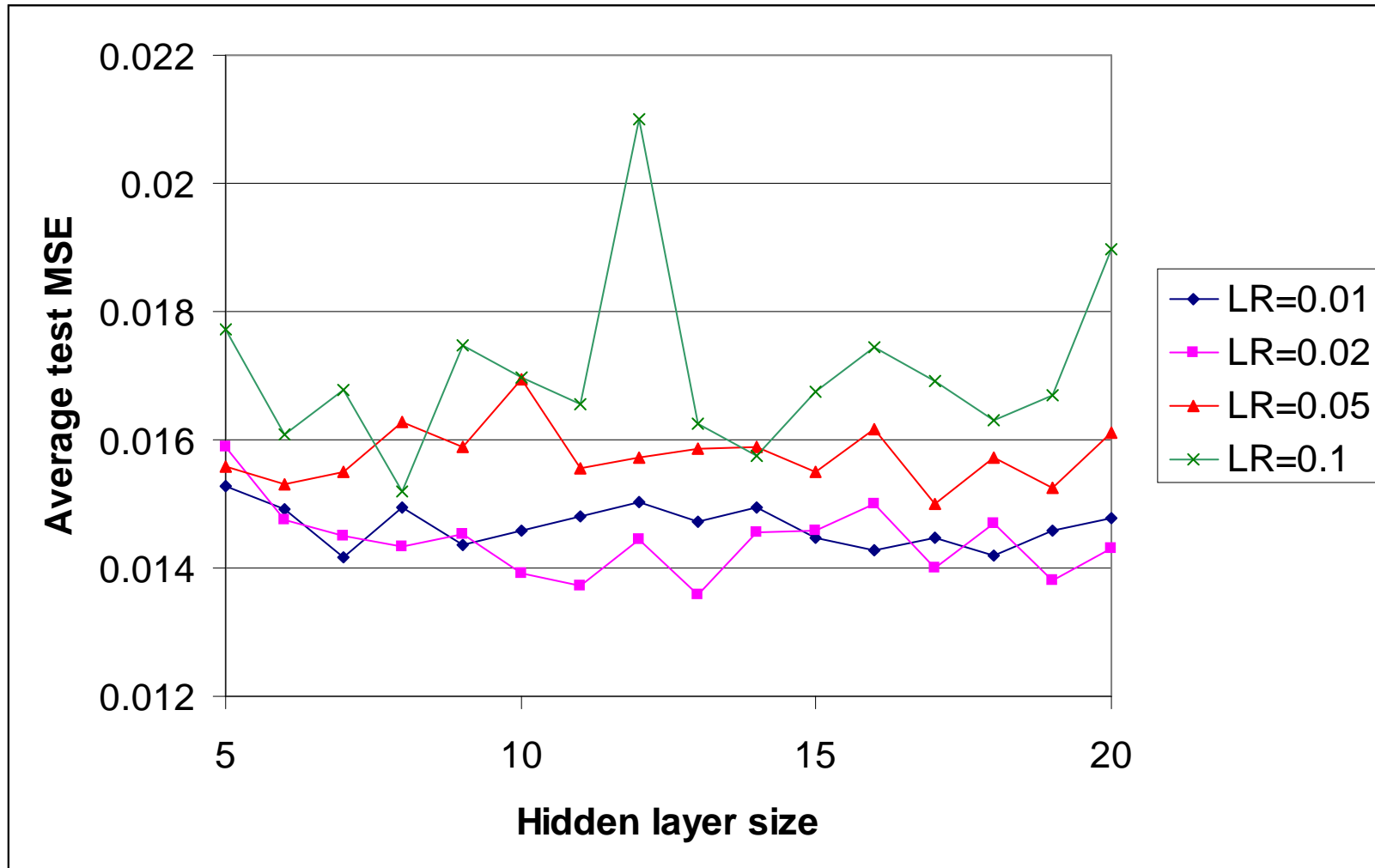


Figure 4.1: Average test MSEs for networks trained with stochastic weight updates and 4 different learning rates

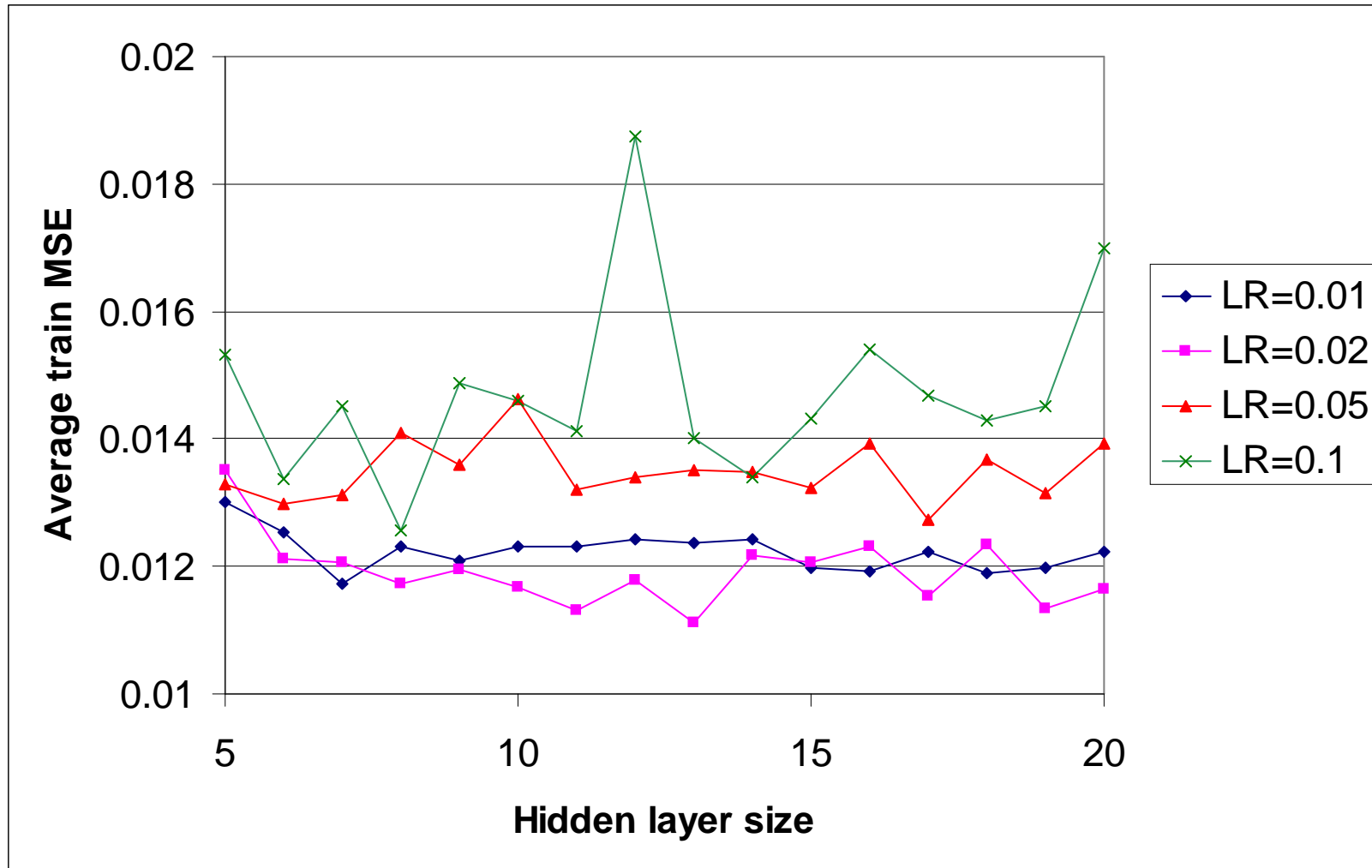


Figure 4.2: Average training MSEs for networks trained with stochastic weight updates and 4 different learning rates

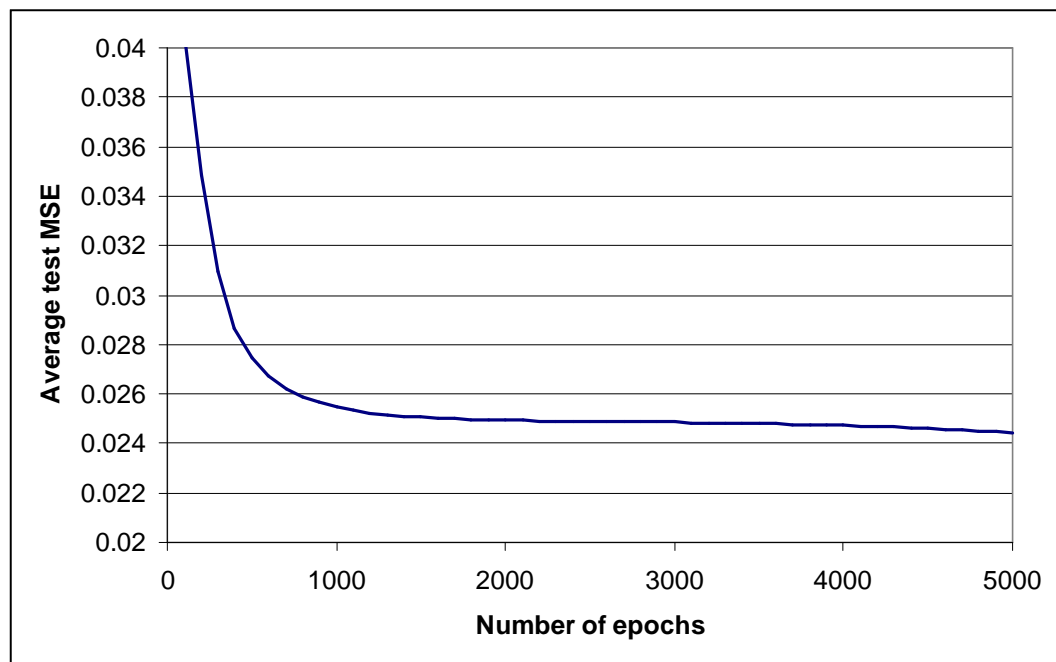


Figure 4.3: Progression in test errors with batch weight updates

In order to test whether stochastic weight updates were the best choice, batch updates were also investigated. However, it proved difficult to find a satisfactory learning rate. During batch updates, the weight updates calculated for each input vector are accumulated and applied after all training vectors have been presented. Since there are approximately 1500 training vectors, the accumulated values could be up to 1500 times the size of the stochastic weight updates. In order to achieve stability one might expect the learning rate to be of the order of 1500 times smaller than that for stochastic weight updates. However, there is substantial cancellation of gradient vectors when accumulated across all of the training vectors and the highest stable learning rate is found to be 0.0002, about 100 times lower than that for stochastic weight updates. Higher values of η are seen to cause instability.

Unfortunately, $\eta = 0.0002$ results in slow learning. The progression in test errors is shown in figure 4.3 for networks with 7 hidden neurons and is typical of the results achieved. After 5000 epochs, the test errors are those shown in figure 4.4. These are much higher than those achieved with stochastic weight updates, for all hidden layer sizes. It was therefore decided that stochastic weight updates would be used in BP training.

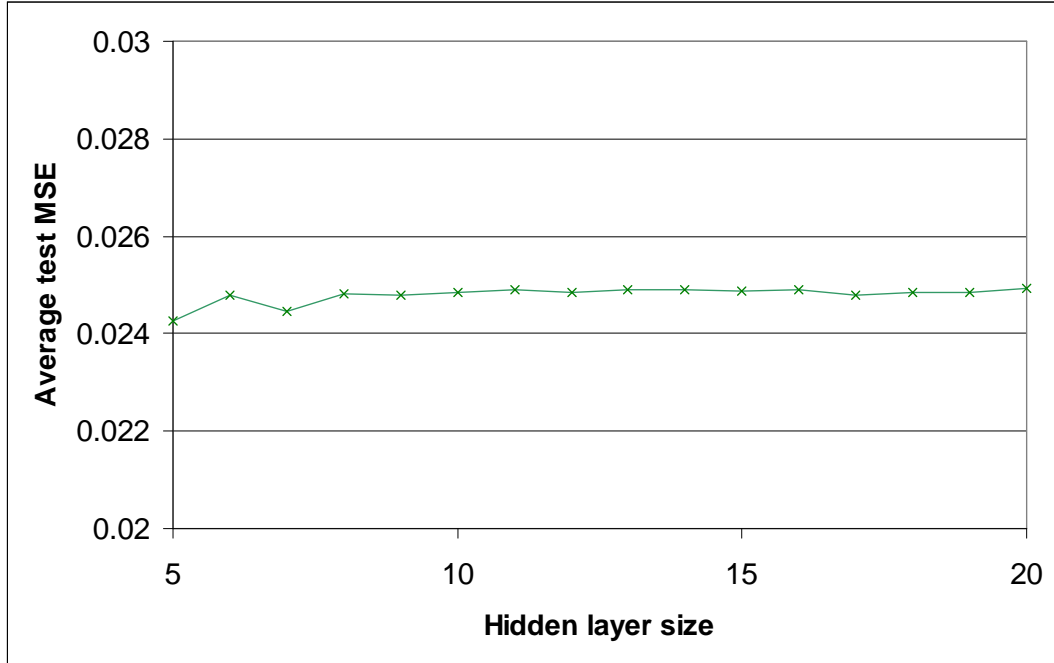


Figure 4.4: Test errors achieved with batch weight updates and $\eta = 0.0002$

4.2.2 Pilot study 2: Stopping criterion

It is important to introduce a stopping criterion, in order to avoid the phenomenon of ‘overfitting’ illustrated in figure 1.8. Amari *et al.* [80] have shown that overfitting does not occur when ANNs are in ‘asymptotic’ mode. The condition for this mode is given by equation 4.1. In this equation m is the number of free parameters (weights) which, for a MLP network with n inputs, h hidden layer neurons and p outputs is given by equation 4.2. t is the number of training items.

$$t > 30m \quad (4.1)$$

$$m = (n + 1)h + (h + 1)p \quad (4.2)$$

In the case of the CLASH data, t is approximately 1500, n is 10 and p is 1. The asymptotic condition should therefore hold only if the number of hidden neurons is 4 or less. The networks considered here are all larger than this and will therefore not be in asymptotic mode.

Since the networks are not in asymptotic mode a stopping criterion is important. Three different stopping criteria were considered-

- cross-validation using a verification dataset.
- introduction of a convergence criterion that assesses whether the test error has converged, without reference to a verification dataset.
- setting a maximum number of epochs after which training is terminated.

Cross-validation has the disadvantage that it requires the setting aside of data from the training set, so less data is available for training purposes. Cross-validation is a useful technique when the training error is a poor guide to the test error. However, figures 4.1 and 4.2 show that this is not the case. For these reasons, cross-validation was ruled out as a stopping criterion.

The next approach considered was the introduction of a convergence criterion. Convergence was defined as the point at which 5 previous measurements of training error (MSE), equivalent to 500 epochs, showed no reduction in the error greater than 1% of the overall error. Table 4.1 shows the percentage of networks that converged after 5000 epochs or less, given different learning rates and hidden layer sizes. Table 4.2 shows the average number of epochs required to achieve convergence. For the purposes of this table, the convergence epoch of networks that have not converged within 5000 epochs has been set as 5100. In general it is seen that convergence occurs quicker when a higher learning rate is applied and the hidden layer size is small.

In order to assess the effectiveness of the convergence criterion, it is necessary to find out whether it is a good guide to the *test error*, i.e. the error achieved with data not used in training. The test error was therefore recorded at the same times as the training error, and its progress evaluated. The results are given in tables 4.3 and 4.4.

The results using the test sample of the dataset indicate that convergence occurs at an earlier point for the test than for the training sample, with an average gap between training convergence and test convergence of approximately 700 epochs. The effect of this phenomenon is illustrated in figure 4.5. This shows the average number of epochs required to achieve convergence for networks with between 5 and 20 hidden neurons with the optimum learning rate (0.02).

The disparity between the convergence of training and test errors implies that the convergence criterion may not be an effective predictor of minimum test error. An investigation into the detailed behaviour of the training and test errors suggests a further problem. Figure 4.5 suggests that the test error converges after approximately 2000 epochs and the training error after approximately 2500 epochs. However, care must be taken not to terminate training too early. In many cases the test error is seen to fall off

hidden layer size	$\eta = 0.01$	$\eta = 0.02$	$\eta = 0.05$	$\eta = 0.1$
5	100	100	100	100
6	100	100	100	100
7	100	100	100	100
8	90	90	100	100
9	100	100	100	100
10	100	100	100	100
11	80	100	100	100
12	90	90	100	100
13	100	100	100	100
14	100	100	100	100
15	100	90	100	100
16	90	100	100	100
17	90	100	100	100
18	80	100	100	100
19	90	90	100	100
20	100	100	100	100

Table 4.1: Percentage of training errors that have converged within 5000 epochs

hidden layer size	$\eta = 0.01$	$\eta = 0.02$	$\eta = 0.05$	$\eta = 0.1$
5	2910	1900	1620	1490
6	2730	2370	1400	1470
7	3320	2880	1620	1380
8	3330	2610	1810	1550
9	3020	2030	1630	1570
10	3150	2160	1670	1100
11	3350	2530	1660	1190
12	3140	2630	1410	1370
13	3250	2680	2020	1360
14	2700	1900	1470	1450
15	3430	2990	1730	1140
16	3560	2350	1490	1420
17	3340	2070	1760	1370
18	3570	1910	1600	1130
19	3820	3020	1500	1140
20	2640	2650	1540	1300

Table 4.2: Number of epochs required for convergence in training error, averaged over 10 runs

hidden layer size	$\eta = 0.01$	$\eta = 0.02$	$\eta = 0.05$	$\eta = 0.1$
5	100	100	100	100
6	100	100	100	100
7	100	90	100	100
8	90	100	100	100
9	90	100	100	100
10	80	100	100	100
11	90	90	100	100
12	100	100	100	100
13	100	100	100	100
14	100	100	100	100
15	100	100	100	100
16	100	100	100	100
17	100	100	100	100
18	100	100	100	100
19	90	90	100	100
20	100	100	100	100

Table 4.3: Percentage of test errors that have converged within 5000 epochs

hidden layer size	$\eta = 0.01$	$\eta = 0.02$	$\eta = 0.05$	$\eta = 0.1$
5	2250	1610	1290	1170
6	1910	1790	1170	1300
7	2270	1820	1300	1080
8	2310	1900	1280	1250
9	2450	1810	1400	1390
10	2630	1850	1520	1010
11	2490	1990	1440	1020
12	2610	1940	1180	1280
13	2700	1780	1660	1000
14	2420	2040	1240	1180
15	2790	1960	1200	1090
16	2840	1870	1260	1260
17	2510	1790	1320	1120
18	2520	1840	1530	1110
19	2980	1880	1310	1110
20	2310	2180	1460	1040

Table 4.4: Number of epochs required for convergence in test error, averaged over 10 runs

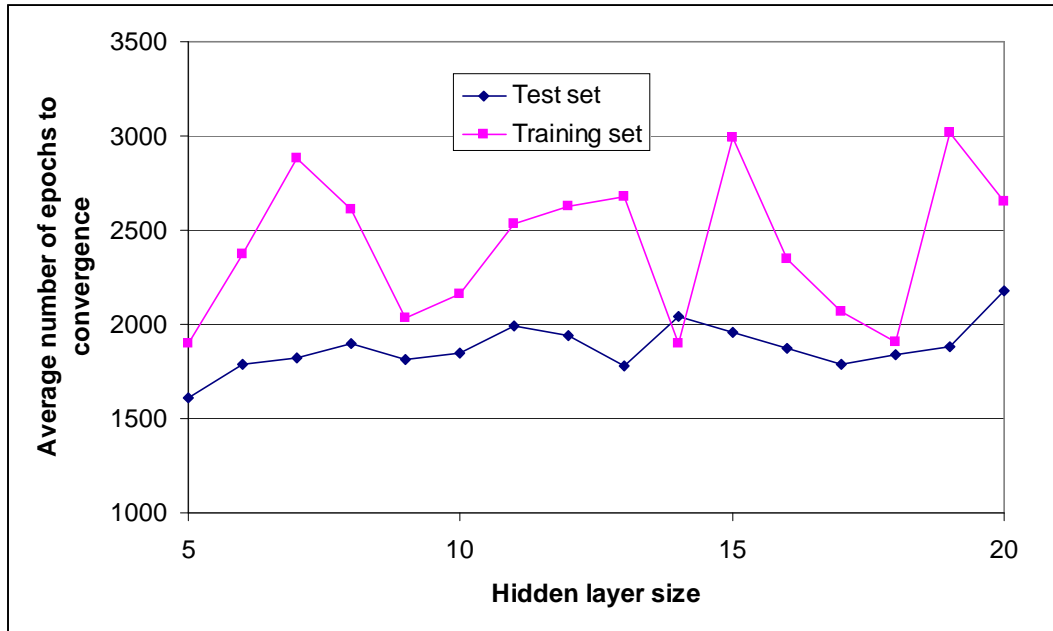


Figure 4.5: Number of epochs required for convergence, assuming optimum learning rate and stochastic weight updates

training speed	hidden layer size	η
slow	20	0.01
medium	13	0.02
fast	5	0.1

Table 4.5: Training parameters for slow, medium and fast convergence

after convergence has first occurred. Figures 4.6-4.8 show the progression in training and test errors for three different scenarios, described by the parameters in table 4.5.

The three scenarios correspond to slow, medium and fast training regimes. They illustrate the difference between different learning rates: higher learning rates lead to quicker convergence but less stability in the training process, whereas lower learning rates result in lower convergence rates but greater stability. Despite these differences in learning behaviour, the graphs share the following common features-

- Although convergence occurs before 5000 epochs for nearly all of the individual networks, the average training and test errors continue to show a downward tendency for some time.
- The error curves for both training and test datasets are seen to level off towards 5000 epochs.

- The overfitting area of training, indicated by a sharp increase in the test error, does not appear to have been reached within 5000 epochs.

These results suggest that training for a fixed number of epochs gives satisfactory results, provided the number of epochs is appropriate. Training for 5000 epochs appears to be effective, with test errors appearing to have reached a plateau region without going beyond this region into an overfitted regime. This technique was therefore used to train all MLP networks.

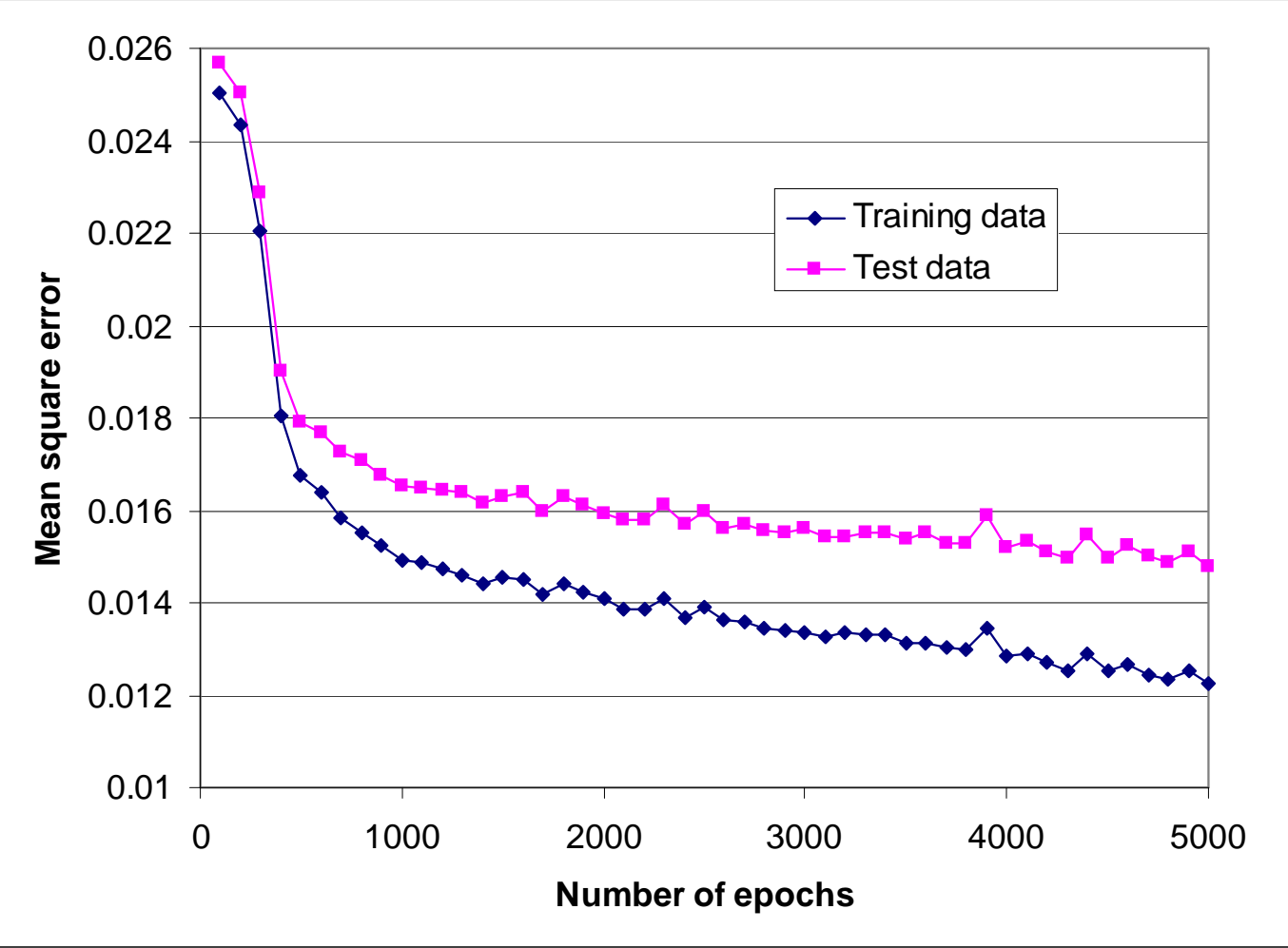


Figure 4.6: Training progression within a slow training regime

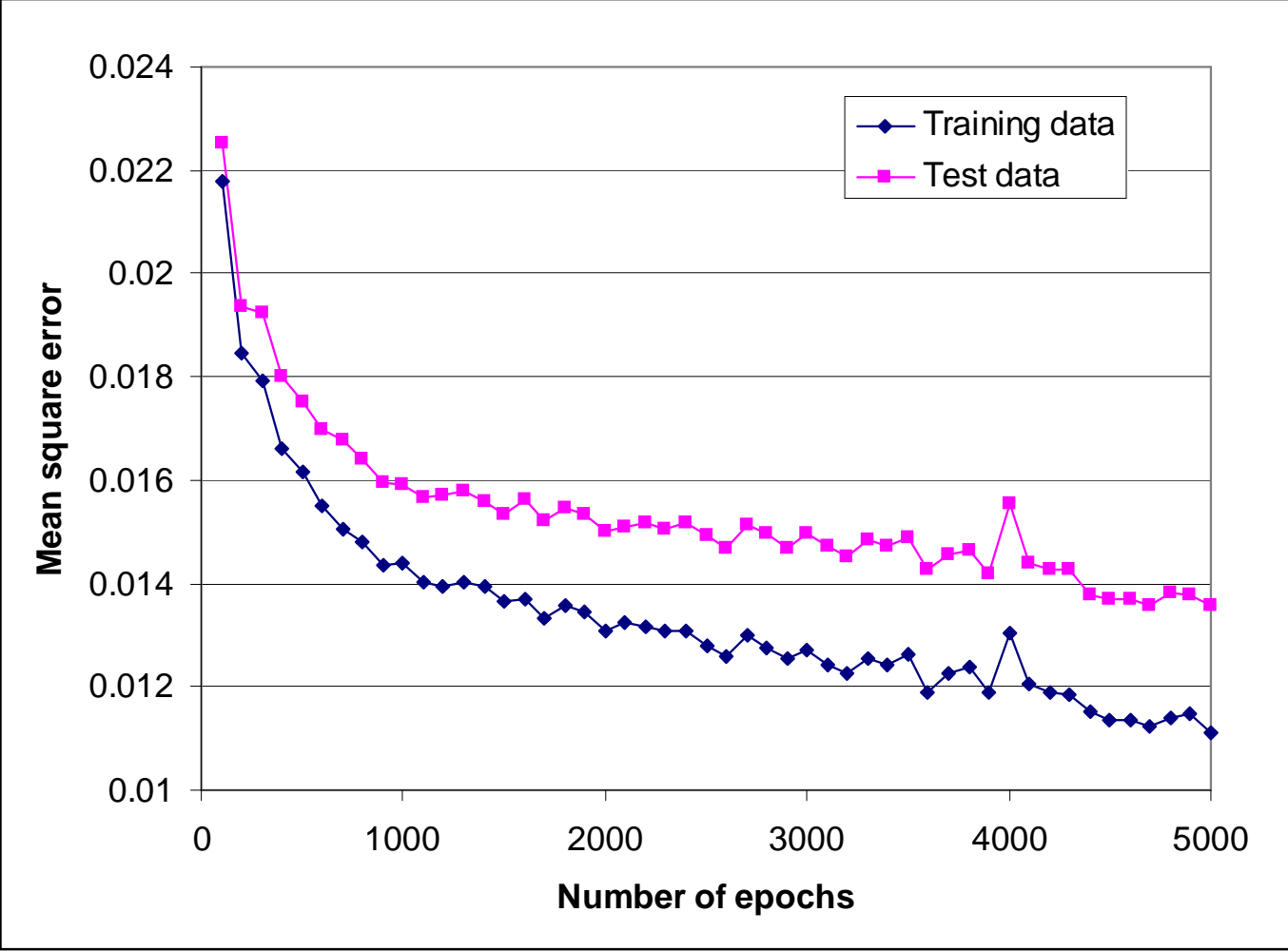


Figure 4.7: Training progression within a medium training regime

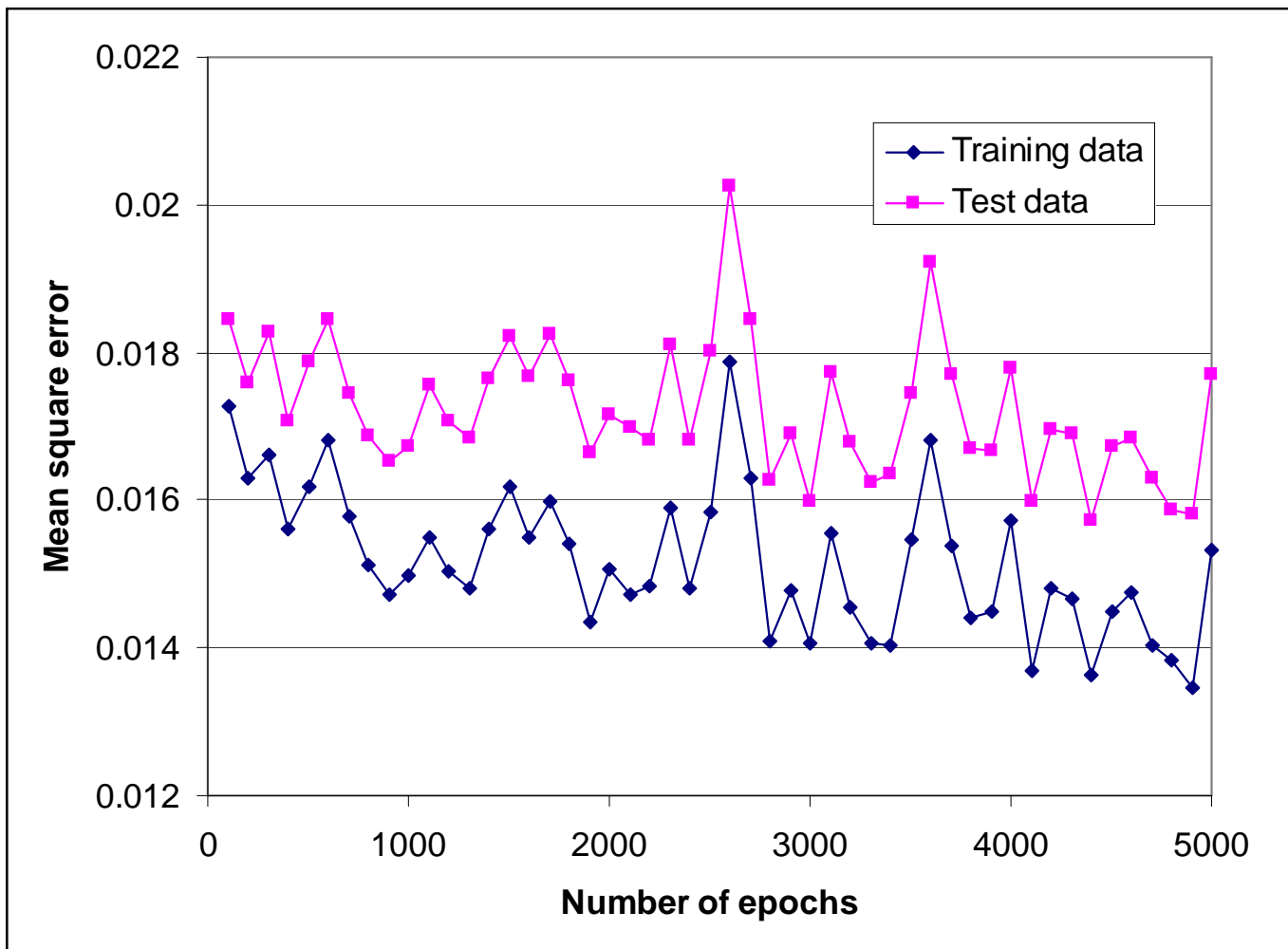


Figure 4.8: Training progression within a fast training regime

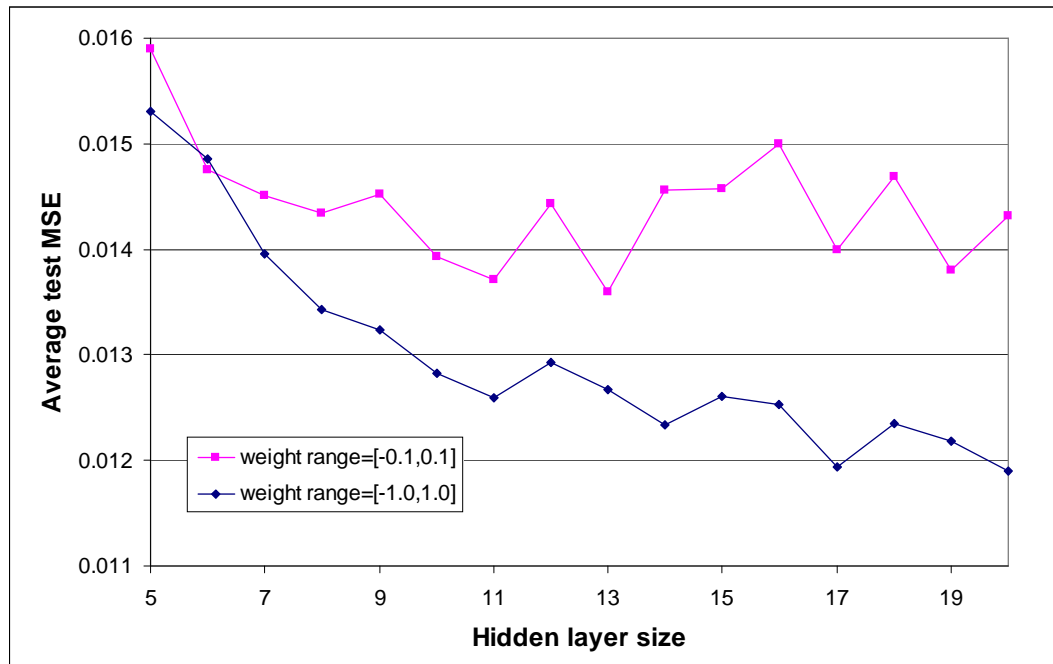


Figure 4.9: Average test MSEs for different weight initialisation ranges

4.2.3 Pilot study 3: weight initialisation range

Having ascertained the optimum learning rate, weight update mode and stopping criterion, the issue of weight initialisation was addressed. It is known that the initial range of weights can have a large effect on training efficiency. If very large initial weights are used there is a danger that neurons will be driven into saturation (see figure 3.2). On the other hand, if very small weights are used the outputs will all be very close to zero, leading to very low average gradients, and therefore slow training and poor coverage of the available weight space [35].

All tests so far used small initial weights within the range $[-0.1, 0.1]$. To see whether larger weights would lead to quicker convergence and/or lower MSEs, weights were initialised to values in the range $[-1.0, 1.0]$. Test MSE values are shown in figure 4.9, for a range of architectures. It is seen that the ANNs with a larger range of initial weights generally gave lower test MSEs.

A disadvantage in using a wider range of initial weights is that convergence occurs more slowly, as illustrated by figure 4.10. One explanation for this observation is that larger initialisation ranges result in a search through a greater proportion of the weight space. This takes slightly longer but results in lower MSE values. Due to the improved generalisation performance, it was decided that the wider range of initial

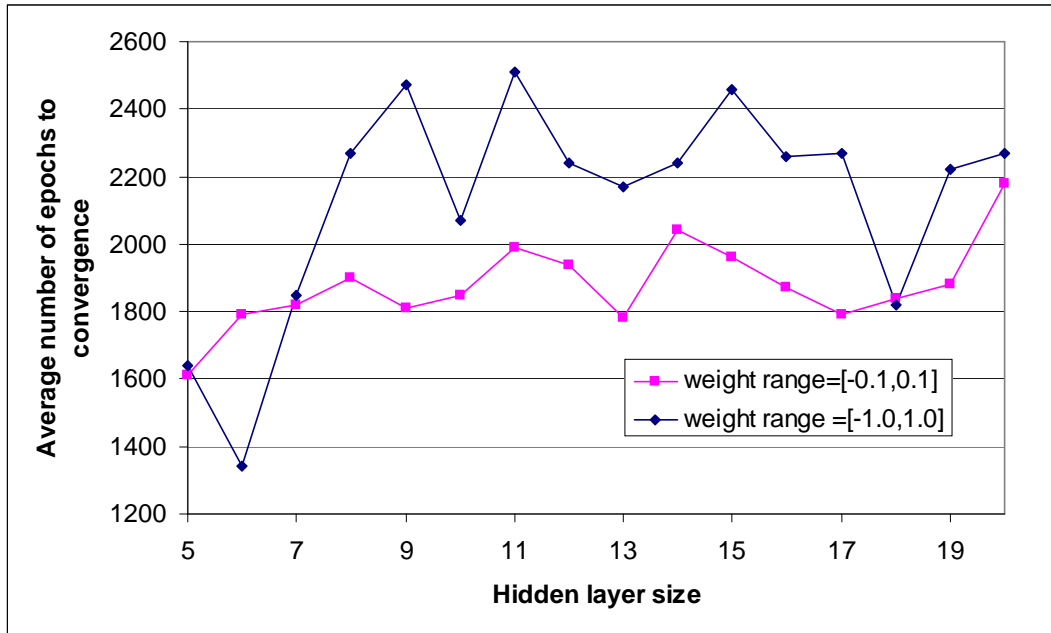


Figure 4.10: Number of epochs required to achieve convergence for different weight initialisation ranges

weights would be used for all ANNs.

The optimum range of initial weights is in line with the findings of Wessels and Barnard [185]. They found that the net inputs to all neurons should ideally be initialised to have a mean of zero and standard deviation of 1. Assuming that the inputs are randomly and uniformly distributed across the range $[-1.0, 1.0]$, this may be achieved by using uniform random weights within the range $[-3/\sqrt{f_i}, 3/\sqrt{f_i}]$, where f_i is the number of inputs (fan-in) to unit i [172]. For the ANNs created here, the number of inputs to the hidden layer neurons is 10 and the number of inputs to the output neuron varies between 1 and 20. Taking the fan-in as 10 gives an ideal range of $[-0.95, 0.95]$, very close to the range used in the above tests.

4.2.4 Pilot study 4: Output neuron transfer function

There are two reasons for preferring a linear to a sigmoid output neuron. Firstly, linear output neurons are commonly used for regression problems in order to allow extrapolation into areas outside the range of the training data. Secondly, the RBF networks used in this study have linear output functions (see Chapter 5), so comparisons between the two types of network have increased validity if the MLPs also have linear output functions.

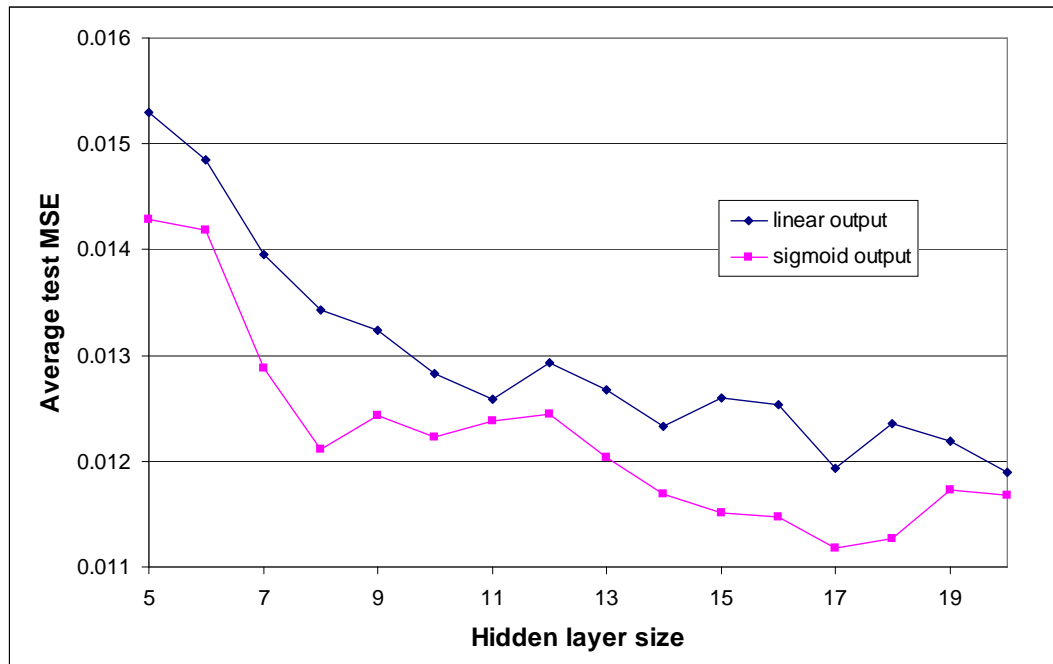


Figure 4.11: Average test MSEs for different output neuron transfer functions

However, in order to test whether a sigmoid transfer function could give a better approximation to the underlying function the linear output neuron was replaced with a sigmoid neuron. The results are illustrated in figure 4.11 and show that a sigmoid output function gives improved generalisation performance, as measured by test MSE.

It was decided that the main study would include investigations using both sigmoid and linear output functions, in order to give the best possible results for a MLP (sigmoid function) and a fair comparison with other types of ANN (linear function).

4.2.5 Pilot Study 5: Momentum coefficient

The introduction of momentum into gradient descent training is intended to speed up training and allow a more accurate determination of the error minimum (see sections 1.5.4 and 2.2.3). When networks contain only linear neurons it is possible to calculate optimum values for the learning rate parameter η and the momentum coefficient α [186]. However, when sigmoid transfer functions are used, η and α are usually determined experimentally. It was decided that a full search across all possible values of η and α was impractical, and the value of η was therefore fixed at its optimal value without momentum, 0.02. In order to retain stability, α cannot exceed 1.0. The trial values were therefore chosen as 0.1, 0.5 and 0.9.

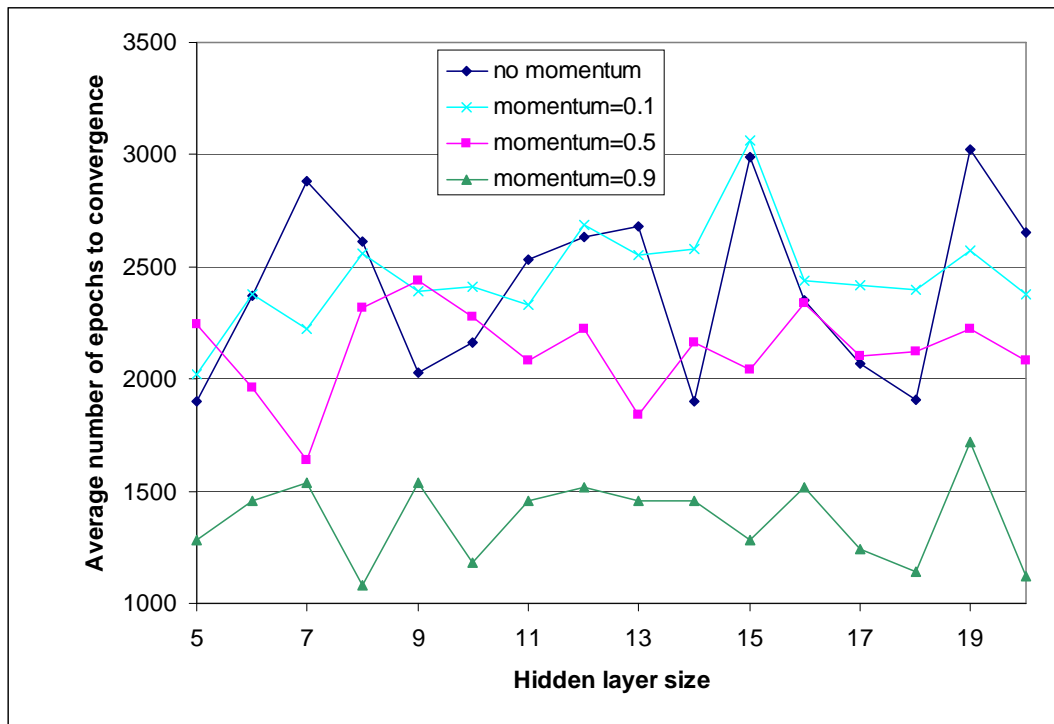


Figure 4.12: Number of epochs required to achieve convergence with and without momentum

Figure 4.12 shows that the introduction of higher values of α generally led to quicker convergence, although the results are variable. The resultant test MSEs are shown in figure 4.13, with the corresponding results without momentum included for comparison. It is seen that lower values of α gave better results, with $\alpha = 0.1$ giving results which are similar to those without momentum.

Overall, the introduction of momentum does not appear to be useful. Although convergence occurs more quickly, the increased stepsize appears to make it more difficult to approach a reasonable error minimum. In the case of $\alpha = 0.9$ the algorithm verges on instability. Since a satisfactory value of α could not be found, it was decided that momentum would not be used in the training of MLPs.

4.2.6 Pilot Study 6: Levenberg-Marquardt method

The Levenberg-Marquardt method uses second-order gradient information in order to perform more efficient gradient descent (see sections 1.5.5 and 2.3). One of the advantages of this technique is that it does not require the setting of as many parameters as first-order gradient descent methods. A pilot study was run to see whether use of the

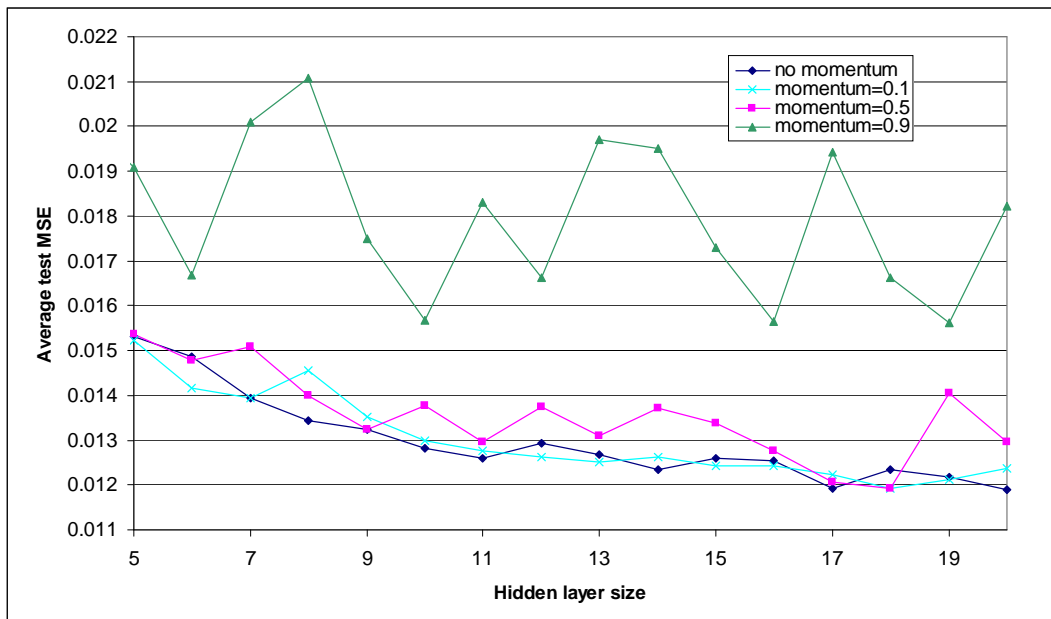


Figure 4.13: Average test MSEs with and without momentum

L-M algorithm would lead to quicker convergence as expected, and whether lower test MSEs could be achieved through its use. Figure 4.14 shows the average training and test MSEs achieved for ANNs of different sizes, with a linear output neuron, trained for 200 epochs with the L-M algorithm. A range of hidden layer sizes are seen to result in test errors averaging below 0.0120, suggesting that the L-M algorithm does generally result in better performance than the BP algorithm. The latter only gave such low errors for two hidden layer sizes.

Figure 4.15 shows the progress of the test errors, averaged over 10 networks each containing 8 hidden layer neurons. Similar graphs are obtained for networks of different sizes. They show similar features to those obtained for gradient descent training (see figure 4.7), with errors falling off during early epochs, before levelling out. It is seen that the L-M algorithm leads to much quicker convergence, with errors levelling out after about 150 epochs, rather than after approximately 3000 epochs with the BP algorithm. There is a danger that the test error could start to increase if training is continued for some time after this level region. For this reason, L-M training was terminated after 200 epochs.

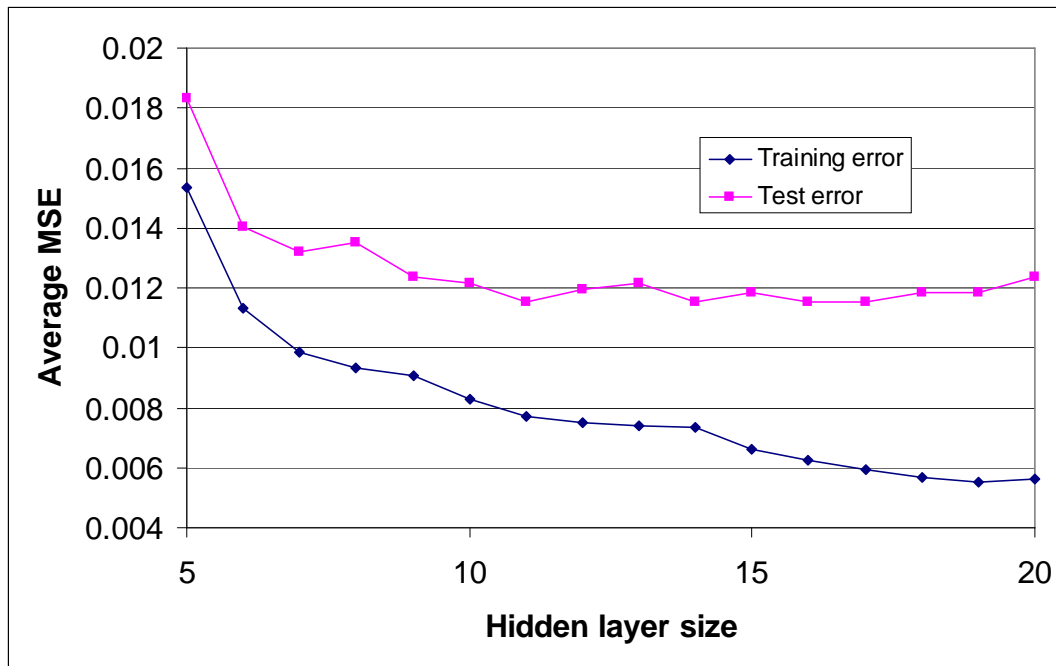


Figure 4.14: Average training and test MSEs with Levenberg-Marquardt algorithm

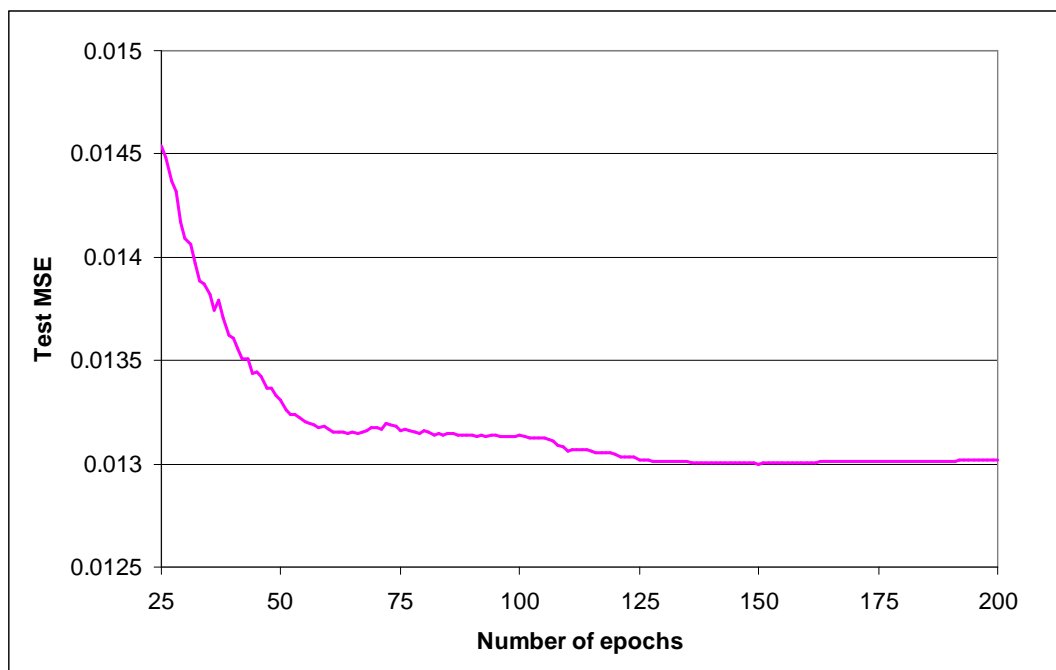


Figure 4.15: Progression in test errors during Levenberg-Marquardt training of a MLP network containing 8 neurons

4.3 Model selection method

This section considers the method used to select the ‘best’ network architecture. In particular the task is to identify the optimum number of hidden neurons. This process of model selection is closely intertwined with three design issues that are also discussed-

- data division
- stopping criteria
- assessment criteria

We can see how these issues are related when we consider the method of cross-validation (see section 1.5.4 and [187]). A widely used variant of cross-validation is the method of multi-fold cross-validation. In this method a fixed subset of the data is used for test purposes. The remaining data are divided into n equal subsets. A series of n networks are then trained for each potential architecture using $n - 1$ of the subsets for training and the remaining subset for verification. Stone [187] originally used subsets containing just one item of data. However, this creates large computational demands, particularly for large datasets. Recent studies commonly split the data into 10 equal subsets. The optimum model is identified by averaging the verification errors across all networks and the test error must also be averaged across all networks with the optimum architecture.

Cross-validation requires a three-fold division of data into training, validation and test subsets [35]. The validation subset may have two purposes: locating a stopping point and determining the optimum model size. The test subset is used to assess the trained networks. Multi-fold cross-validation is commonly applied to small datasets. It has been shown that as the data size tends towards infinity, the fraction of data used in training must tend towards 1, i.e. an infinite amount of data must be used for training, with a diminishing number held back for verification [188]. For large datasets the method may therefore require excessive computational requirements. A further problem with multi-fold cross-validation is that it tends to produce over-complex models.

This study has used a method known as ‘Monte Carlo cross-validation’ (MCCV) [189]. In this method test data is set aside and the remaining data are randomly partitioned into training and verification subsets. Partitioning is repeated several times and network training is performed on each one [190, 188], as in multi-fold cross-validation. It is seen experimentally that using a relatively high proportion of the data for verification purposes usually gives optimum results [190]: approximately equal sized training

and verification sets are effective in most cases. Further, theory demonstrates that as the data size tends towards infinity the ratio of training to verification data has to tend towards 0.

The MCCV method appears to be better adapted to the particular requirements of the CLASH dataset. As mentioned in Chapter 3, this dataset is very noisy. In order to see the effect of the noise on the generalisation properties of the ANNs, a further pilot study was performed. This involved training sets of networks with the same hidden layer size and using the same training-test split. Hidden layer sizes were chosen as 6, 10 and 14. These architectures were chosen because they give average test MSEs that are different from each other. For each network size and dataset, 5 networks were trained. The starting weights for each network were individually randomised and every network was trained for 200 epochs using the L-M algorithm. The resultant test MSEs are shown in table 4.6.

These results were analysed using a two-way ANOVA [191] in order to identify the source of variance in the test errors. Two sources of the variance were identified, both with p-values below 0.01: the choice of model (hidden layer size) and an interaction between the model and the dataset splits. The first finding is unsurprising, since we expect the model size to have an effect on the observed error. The second finding implies that certain dataset splits give better results with particular network sizes, while other splits might be better suited to different network sizes.

The effect of the choice of data appears to lie in the high noise level of the data. When using a single test set for assessment, the assessed error is highly dependent on the items within the test dataset. Often a small number of data can have a drastic effect on the calculated error. In this case the calculated error is unlikely to reflect the error of the population as a whole.

In order to reduce the effect of dataset selection, it was decided that the data would be split in 30 different ways and a separate network trained with each data split. The errors may then be averaged over the 30 networks. An advantage of the MCCV method is that it can be adapted so that not only the training and verification sets but also the test sets can be randomly assigned. When the test error is averaged across all ANNs with a particular architecture, it is then less likely to be biased as the result of ‘rogue’ items within a particular test set.

The variation within a particular condition seen in table 4.6 is due to the initialisation of network weights. In order to minimise this effect as much as possible, the

data split	6 hidden neurons	10 hidden neurons	14 hidden neurons
1	0.01308	0.01249	0.01117
	0.01386	0.01158	0.01142
	0.01309	0.01117	0.01127
	0.01587	0.01218	0.01086
	0.01370	0.01137	0.01049
2	0.01455	0.01213	0.01167
	0.01713	0.01068	0.01151
	0.01699	0.01090	0.01057
	0.01385	0.01315	0.01136
	0.01704	0.01171	0.01107
3	0.01268	0.01257	0.01113
	0.01296	0.01185	0.01222
	0.01394	0.01266	0.01172
	0.01384	0.01216	0.01144
	0.01394	0.01198	0.01272
4	0.01335	0.01126	0.01192
	0.01267	0.01202	0.01123
	0.01423	0.01194	0.01173
	0.01385	0.01242	0.01162
	0.01470	0.01241	0.01216
5	0.01352	0.01179	0.01086
	0.01344	0.01370	0.01202
	0.01341	0.01341	0.01230
	0.01363	0.01631	0.01244
	0.01683	0.01320	0.01193

Table 4.6: Test MSEs for different hidden layer sizes and training-test splits

weights are initialised to different weights for every network and the results are therefore averaged across different weight initialisations as well as different data splits.

The MCCV method used within this study may be summarised as follows-

- The available data is randomly split 30 times into training, verification and test sets, in the ratio 50:25:25.
- A series of networks with different architectures are created. For each architecture, training is performed 30 times, once with each training set.
- The alternative network architectures are assessed. The objective function is the MSE on the verification data, averaged across all 30 networks.
- The test error is obtained. This is the average MSE on the test data.

4.4 Method

This section summarises the method used to train and assess the ANNs. The reasons for various design choices have been explained in the preceding sections.

30 random splits of the data were made, to give training, verification and test sets in the ratio 50:25:25. All networks were created with weights initialised to random values in the range $[-1.0, 1.0]$. Training was performed for 5000 epochs in the case of simple gradient descent and 200 epochs for Levenberg-Marquardt training.

ANNs were created and trained with varying hidden layer sizes, starting with 5 neurons and increasing one at a time until the verification error averaged across all 30 networks showed a consistent increase. The bipolar sigmoid function of equation 2.7 was used for all hidden layer neurons. Separate networks were created with linear transfer functions and sigmoid transfer functions for the single output neuron.

Verification errors were obtained for each network in order to identify the best performing networks. The final performance measure was the error obtained using the ‘unseen’ test data. The results are given in the next section.

4.5 Results and Discussion

This section gives the results upon training MLP networks with various architectures and using either the back-propagation or Levenberg-Marquardt algorithms. Having

obtained the results for each individual network, they may be averaged in one of two ways-

- The results for a particular architecture and network size may be averaged across all 30 training-test splits. This process allows the identification of the individual architecture that is most effective at generalising the underlying function.
- For each training-test split and family of networks, the optimum hidden layer size may be identified. This allows the comparison of the ‘best’ result for a particular training method and network family. For example, the effectiveness of the L-M algorithm may be compared with that of the BP algorithm. In later sections such comparisons will be extended to a comparison with other types of network, such as RBF networks.

The two methods may be summarised by stating that the first method compares networks with a ‘fixed’ architecture, while the second allows a ‘variable’ architecture. In this study, results are quoted in both ways, in order to identify the best architecture and the best method.

4.5.1 Back-propagation

Linear output neuron

Table 4.7 shows the training, verification and test errors for a series of networks containing a linear output neuron and between 1 and 30 hidden neurons. All results are averaged across 30 different networks, each trained with a different split of the data, i.e. using a fixed architecture. The minimum verification error occurs with 26 hidden neurons and results in a MSE for the test data of 0.01199.

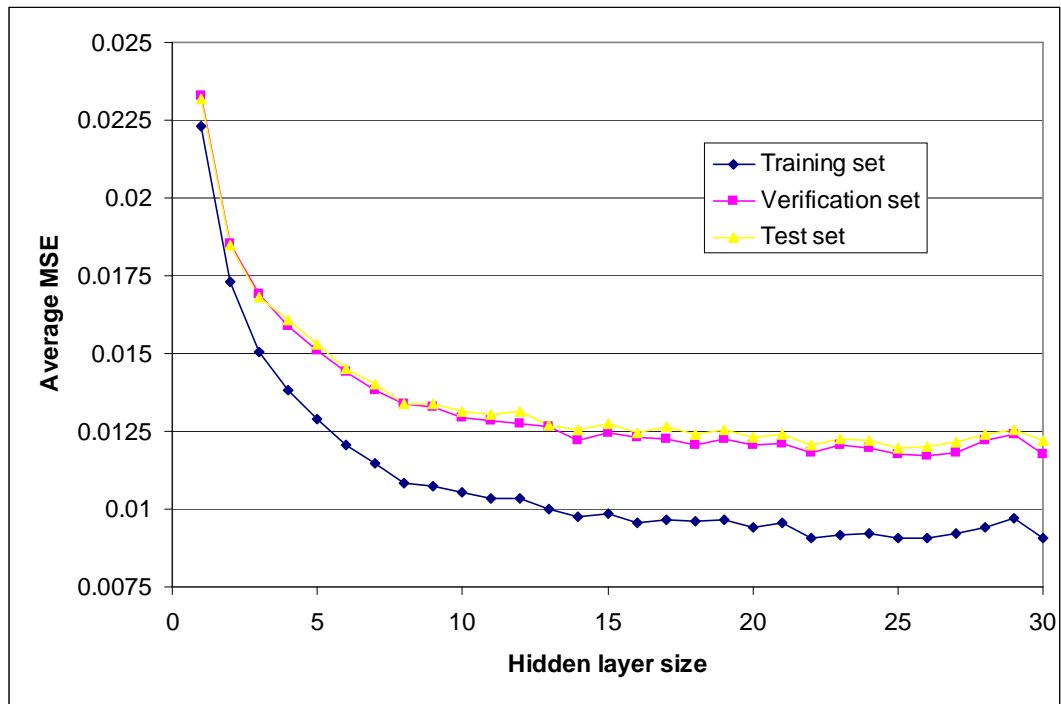
Figure 4.16a displays the same results. This graph has features commonly seen in ANN training, with the errors falling up to a certain layer size, before starting to rise as a result of overfitting. It is clear from the graph that the verification error is a good guide to the test error.

Table 4.8 presents the same results from the point of view of the dataset splits. For each split, the network resulting in the lowest verification error is identified and the performance of that network is assessed using the test dataset, i.e. a variable architecture is allowed.

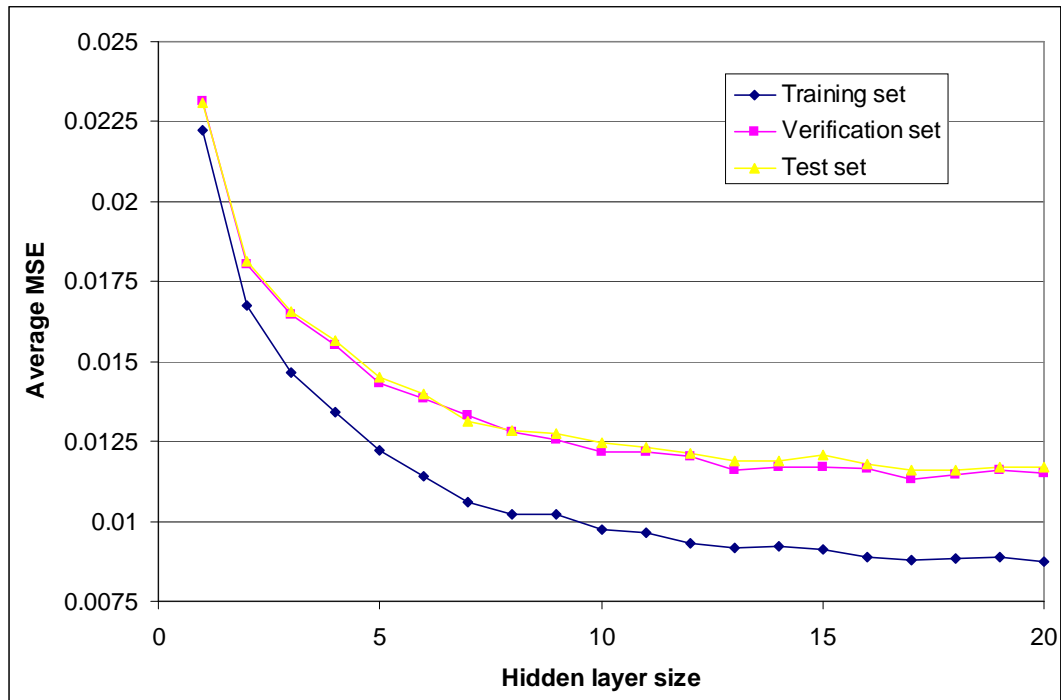
Since the optimum network size has been chosen each time, the average verification error is lower than it is when averaged across networks of any particular network

Hidden layer size	Training error	Verification error	Test error
1	0.02230	0.02326	0.02317
2	0.01731	0.01852	0.01850
3	0.01507	0.01689	0.01683
4	0.01383	0.01588	0.01607
5	0.01288	0.01510	0.01527
6	0.01206	0.01440	0.01451
7	0.01145	0.01382	0.01401
8	0.01081	0.01339	0.01340
9	0.01076	0.01329	0.01339
10	0.01052	0.01292	0.01315
11	0.01033	0.01282	0.01303
12	0.01032	0.01277	0.01312
13	0.01001	0.01266	0.01272
14	0.00976	0.01220	0.01257
15	0.00986	0.01247	0.01276
16	0.00955	0.01229	0.01244
17	0.00964	0.01228	0.01265
18	0.00962	0.01207	0.01240
19	0.00963	0.01227	0.01256
20	0.00941	0.01207	0.01231
21	0.00955	0.01211	0.01239
22	0.00909	0.01179	0.01208
23	0.00916	0.01208	0.01223
24	0.00920	0.01195	0.01219
25	0.00906	0.01176	0.01198
26	0.00907	0.01171	0.01199
27	0.00919	0.01182	0.01214
28	0.00941	0.01220	0.01243
29	0.00972	0.01240	0.01257
30	0.00906	0.01175	0.01218

Table 4.7: Average errors for MLP with linear output neuron and BP training



(a) Linear output neuron



(b) Sigmoid output neuron

Figure 4.16: Training, verification and test errors after BP training

Data split	Verification error	Test error	Optimum hidden layer size
1	0.01154	0.00974	21
2	0.01127	0.00986	19
3	0.01101	0.01183	25
4	0.01138	0.01037	27
5	0.01074	0.01277	17
6	0.01073	0.01107	29
7	0.00972	0.01281	30
8	0.01069	0.01101	20
9	0.01142	0.01195	27
10	0.01099	0.01120	27
11	0.01070	0.01214	28
12	0.01071	0.01162	22
13	0.01155	0.01401	19
14	0.01069	0.01196	27
15	0.01011	0.01037	25
16	0.01031	0.01234	24
17	0.00954	0.01131	29
18	0.01187	0.01031	22
19	0.01155	0.01102	29
20	0.00981	0.01136	25
21	0.01182	0.01043	22
22	0.01138	0.00954	29
23	0.01008	0.01113	30
24	0.01155	0.01154	26
25	0.01236	0.01137	15
26	0.01049	0.01314	24
27	0.01038	0.01311	27
28	0.01103	0.01099	30
29	0.01145	0.01070	24
30	0.01008	0.01348	23
average	0.01090	0.01148	24.7

Table 4.8: Optimum errors for MLP with linear output neuron and BP training

Hidden layer size	Training error	Verification error	Test error
1	0.02223	0.02313	0.02311
2	0.01676	0.01803	0.01814
3	0.01463	0.01645	0.01655
4	0.01339	0.01552	0.01566
5	0.01223	0.01433	0.01449
6	0.01143	0.01382	0.01399
7	0.01059	0.01332	0.01312
8	0.01022	0.01278	0.01283
9	0.01019	0.01256	0.01275
10	0.00972	0.01219	0.01244
11	0.00964	0.01219	0.01230
12	0.00933	0.01204	0.01213
13	0.00916	0.01160	0.01189
14	0.00920	0.01171	0.01187
15	0.00913	0.01171	0.01207
16	0.00890	0.01163	0.01179
17	0.00879	0.01132	0.01161
18	0.00882	0.01144	0.01161
19	0.00888	0.01158	0.01170
20	0.00873	0.01152	0.01169

Table 4.9: Average errors for MLP with sigmoid output neuron and BP training

size. The test error is seen to be correspondingly enhanced, with an average value of 0.001148.

Sigmoid output neuron

The results with a sigmoid transfer function in the output neuron may be treated in the same way. Table 4.9 gives the errors averaged across data splits (fixed architecture) and table 4.10 gives the optimum errors for each dataset (variable architecture). The optimum hidden layer size (the one with lowest average verification error) is found to be 17 neurons, with a corresponding average test error of 0.01161. The optimum test error when a variable architecture is permitted is 0.01116 (averaged across the 30 datasets).

In comparison with the results using a linear output neuron, the optimum layer size is smaller and the optimum achievable error is lower. It can be concluded that, when using BP training, the sigmoid transfer function is more effective at fitting the underlying function.

Data split	Verification error	Test error	Optimum hidden layer size
1	0.01118	0.00993	13
2	0.01101	0.00916	20
3	0.01097	0.01122	13
4	0.01205	0.01083	13
5	0.01025	0.01179	18
6	0.01059	0.01107	19
7	0.00976	0.01174	13
8	0.00979	0.01184	20
9	0.01063	0.01248	17
10	0.01097	0.00963	16
11	0.01068	0.01185	13
12	0.01080	0.01152	18
13	0.01099	0.01265	13
14	0.01100	0.01166	19
15	0.01022	0.00974	18
16	0.00985	0.01087	20
17	0.00991	0.01217	15
18	0.01124	0.01058	20
19	0.01110	0.01086	13
20	0.01001	0.01131	20
21	0.01262	0.01023	17
22	0.01155	0.01019	17
23	0.00956	0.01123	17
24	0.01180	0.01129	13
25	0.01151	0.01052	17
26	0.00981	0.01210	18
27	0.00992	0.01283	20
28	0.01114	0.01027	19
29	0.01164	0.01054	18
30	0.00908	0.01271	20
average	0.01072	0.01116	16.9

Table 4.10: Optimum errors for MLP with sigmoid output neuron and BP training

Hidden layer size	Training error	Verification error	Test error
5	0.01351	0.01612	0.01635
6	0.01130	0.01425	0.01428
7	0.01006	0.01322	0.01326
8	0.00944	0.01341	0.01305
9	0.00973	0.01292	0.01318
10	0.00828	0.01197	0.01225
11	0.00782	0.01218	0.01193
12	0.00859	0.01277	0.01287
13	0.00719	0.01236	0.01177
14	0.00705	0.01150	0.01168
15	0.00665	0.01203	0.01184
16	0.00650	0.01199	0.01215
17	0.00590	0.01251	0.01169
18	0.00580	0.01224	0.01173
19	0.00577	0.01236	0.01169
20	0.00556	0.01182	0.01149

Table 4.11: Average errors for MLP with linear output neuron and L-M training

4.5.2 Levenberg-Marquardt training

The results using the L-M algorithm have been analysed in the same manner as those from BP training. Again results are quoted in terms of MSEs averaged across networks with the same hidden layer size and then as optimum MSEs for each dataset, allowing the network size to vary.

Linear output neuron

The optimum sized network when using a linear output neuron and L-M training is seen from table 4.11 to be 14, with a corresponding average test error of 0.01168. The L-M algorithm has therefore achieved lower errors using smaller networks than the BP algorithm (see section 4.5.1).

From table 4.12 the optimum test MSE, allowing variable architectures, is 0.01107. Again this result is lower than that achieved using BP training.

Sigmoid output neuron

The average errors using a sigmoid output neuron and L-M training are a slight improvement on those using a linear output neuron. The minimum verification MSE

Data split	Verification error	Test error	Optimum hidden layer size
1	0.01118	0.01016	11
2	0.01067	0.01105	17
3	0.01060	0.01178	12
4	0.01053	0.01140	14
5	0.01156	0.01366	20
6	0.00973	0.01148	20
7	0.00990	0.00911	18
8	0.00927	0.01110	18
9	0.00939	0.01124	17
10	0.01123	0.01015	13
11	0.01141	0.01269	17
12	0.01087	0.00986	19
13	0.00879	0.00897	20
14	0.01145	0.00945	20
15	0.01117	0.00927	16
16	0.00972	0.01076	12
17	0.01138	0.01118	12
18	0.01087	0.01247	16
19	0.00956	0.01127	14
20	0.00928	0.01220	20
21	0.01126	0.01050	18
22	0.01121	0.00984	15
23	0.01015	0.01101	19
24	0.00948	0.01304	10
25	0.01100	0.01101	17
26	0.01002	0.01238	16
27	0.00990	0.01099	15
28	0.00991	0.01286	14
29	0.00959	0.01080	16
30	0.01002	0.01050	18
average	0.01037	0.01107	16.1

Table 4.12: Optimum errors for MLP with linear output neuron and L-M training

Hidden layer size	Training error	Verification error	Test error
1	0.02464	0.02561	0.02565
2	0.01856	0.01979	0.01976
3	0.01538	0.01711	0.01739
4	0.01297	0.01537	0.01522
5	0.01160	0.01435	0.01440
6	0.01082	0.01369	0.01375
7	0.00995	0.01295	0.01300
8	0.00917	0.01267	0.01258
9	0.00851	0.01230	0.01209
10	0.00809	0.01177	0.01214
11	0.00773	0.01150	0.01208
12	0.00724	0.01136	0.01143
13	0.00699	0.01102	0.01139
14	0.00675	0.01120	0.01129
15	0.00640	0.01115	0.01142
16	0.00613	0.01109	0.01099
17	0.00611	0.01114	0.01133
18	0.00571	0.01095	0.01104
19	0.00549	0.01077	0.01114
20	0.00518	0.01106	0.01131

Table 4.13: Average errors for MLP with sigmoid output neuron and L-M training

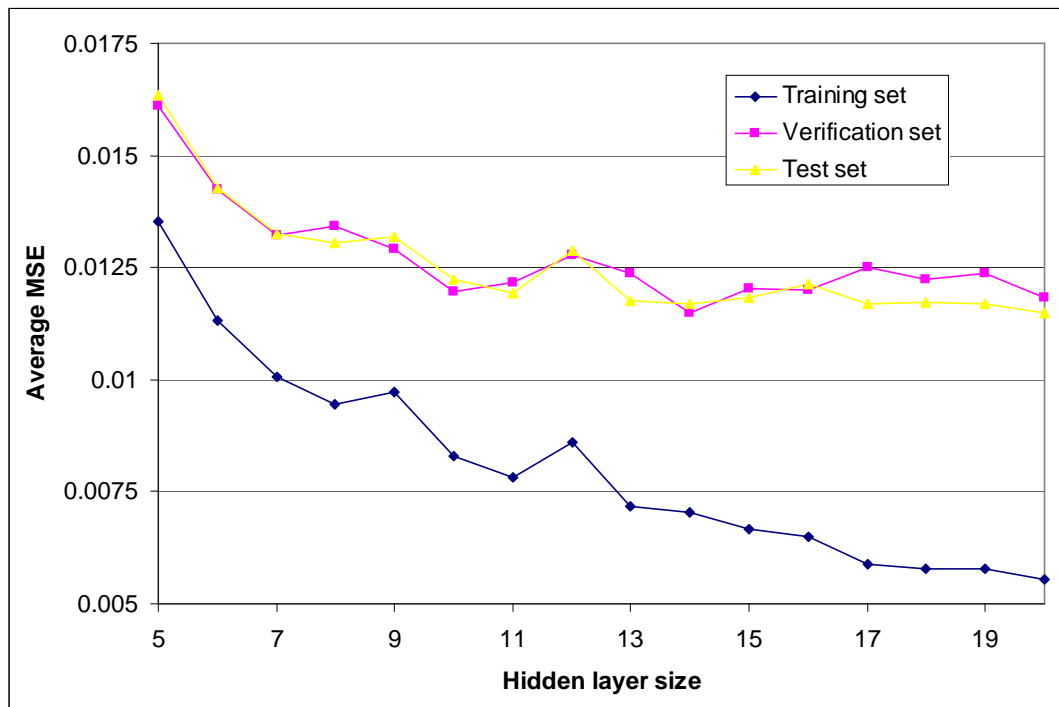
occurred with 19 hidden layer neurons, resulting in a corresponding test error of 0.01114 (see table 4.13). Again the verification error was seen to be a fairly good guide to test error, as illustrated in figure 4.17b.

The results when variable network sizes are allowed are reported in table 4.14. The average test MSE, 0.01071, is better than that achieved using any other methods investigated so far.

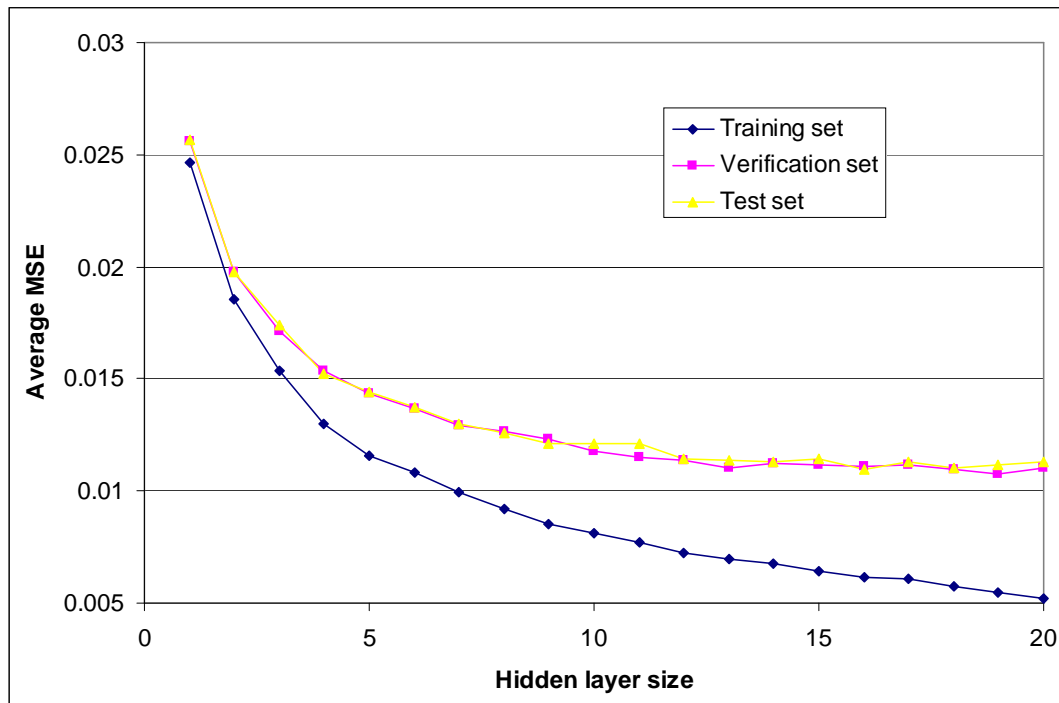
4.5.3 Results summary

Table 4.15 summarises the results of this chapter. The parameters used in the table are defined as follows-

- Optimum layer size. The number of hidden layer neurons in the networks that give the lowest verification error, when averaged over 30 networks, i.e. fixed architecture.
- Best average error. The average test error from the networks identified as having



(a) Linear output neuron



(b) Sigmoid output neuron

Figure 4.17: Training, verification and test errors after L-M training

Data split	Verification error	Test error	Optimum hidden layer size
1	0.01041	0.00919	17
2	0.01013	0.01042	16
3	0.00920	0.01208	20
4	0.01011	0.01176	13
5	0.01064	0.01063	19
6	0.01097	0.01175	17
7	0.00887	0.00979	17
8	0.00966	0.01029	14
9	0.00913	0.01106	17
10	0.00992	0.00889	17
11	0.01040	0.01067	18
12	0.00940	0.00922	18
13	0.00906	0.00894	13
14	0.01040	0.00986	13
15	0.01091	0.00970	16
16	0.00905	0.01096	15
17	0.01015	0.00977	16
18	0.01061	0.01131	11
19	0.00964	0.01186	16
20	0.00963	0.01052	20
21	0.01056	0.01035	11
22	0.01071	0.01009	14
23	0.01009	0.01097	19
24	0.00962	0.01219	12
25	0.01057	0.01035	14
26	0.00987	0.01261	13
27	0.01020	0.01089	17
28	0.00895	0.01054	20
29	0.00985	0.01302	13
30	0.00990	0.01149	13
average	0.00995	0.01071	15.6

Table 4.14: Optimum errors for MLP with sigmoid output neuron and L-M training

Training regime	Optimum layer size	Best average error	Optimum error
Linear output (BP)	26	0.01199	0.01148
Sigmoid output (BP)	17	0.01161	0.01116
Linear output(L-M)	14	0.01168	0.01107
Sigmoid output (L-M)	19	0.01114	0.01071

Table 4.15: Summary statistics for MLP training

the optimum layer size.

- Optimum error. The average test error for the set of networks that give the lowest verification errors for each dataset. This set may include networks with different hidden layer sizes, i.e. variable architecture.

Overall it is seen that the L-M algorithm gives superior results to the BP algorithm and that a sigmoid output function is superior to a linear transfer function.

4.5.4 Speed and memory comparisons

Both the BP and L-M algorithms have been implemented in the Java programming language. It is known that the speed of the BP algorithm scales approximately linearly with network size, whereas the L-M algorithm is approximately proportional to the square of the network size. It is therefore difficult to compare the speeds of the two algorithms. The time taken to train a set of networks was measured. The set used was the same in each case: 20 networks containing between 1 and 20 hidden layer neurons.

The time taken to train a set of networks using the L-M algorithm was found to be approximately 45 minutes, running on a personal computer (PC) containing an AMD Athlon 2100+ chip with a clock-speed of 1.74 GHz. The time taken to train a set of networks with the BP algorithm on the same PC was 110 minutes. The total development time for the BP algorithm was also increased by the need to perform pilot studies in order to ascertain training parameters including learning rate, weight update mode and momentum coefficient. The development of the L-M algorithm was relatively straightforward.

The L-M algorithm requires considerably greater quantities of working memory (RAM) than the BP algorithm. However, it was found that the algorithm could be implemented in the Java programming language without allocating any memory in addition to the default maximum of 64 MB. This level of memory allocation is unlikely

to present a problem, even when running the program on an average PC.

Chapter 5

CLASH prediction using RBF networks

This chapter investigates the use of RBF networks with the CLASH dataset. Following a pilot study to identify the optimum RBF steepness (section 5.1) and a description of methods (section 5.2), section 5.3 reports the results of training various types of RBF network with the CLASH data. In addition to the standard method of forward selection, training is performed using forward selection with regularisation and forward selection with gradient descent optimisation. The results are summarised and compared with those obtained using MLP networks in section 5.4.

5.1 Width Pilot Study

The method used to train RBF networks was Forward Selection with Orthogonal Least Squares (FS-OLS). This method has been described in section 2.4. Unlike the BP algorithm it does not require the setting of various training parameters. The only parameter that needs to be chosen is the width of the radial basis functions used in the hidden layer neurons. The steepness parameter, σ , was introduced in section 2.1. Rather than dealing with this parameter directly, it is convenient to think in terms of the ‘width’ or ‘spread’ of a neuron. This is defined as the distance from the centre of the neuron that will give an output value of 0.5 and is related to the steepness parameter by equation 5.1, in which s is the spread of a neuron.

$$s = \frac{\sqrt{-\ln(0.5)}}{\sigma} \quad (5.1)$$

In order to find the optimum RBF width, 10 runs were carried out using different data splits for widths between 0.2 and 1.2 (in units of 0.2). At this stage, only the verification errors were assessed - the test errors were not used in the determination of optimum RBF width. The variation in verification error with hidden layer size is shown in figure 5.1. This figure shows that the error declines quicker for wider spreads, but that very large spreads lead to an increase in verification error at an earlier point. These observations may be explained by considering what occurs during the training process. When using wider radial functions, each function covers a wider range of input values, so small numbers of neurons can give a reasonable estimate of the outputs. As training proceeds, additional neurons attempt to identify increasingly local features in the underlying function, which affect fewer and fewer input vectors. Wider functions are less able to identify local variation in the function and therefore do less well at this stage. It is expected that there is an optimal spread value for the CLASH dataset, and from figure 5.1 it appears to be 0.4 or 0.6.

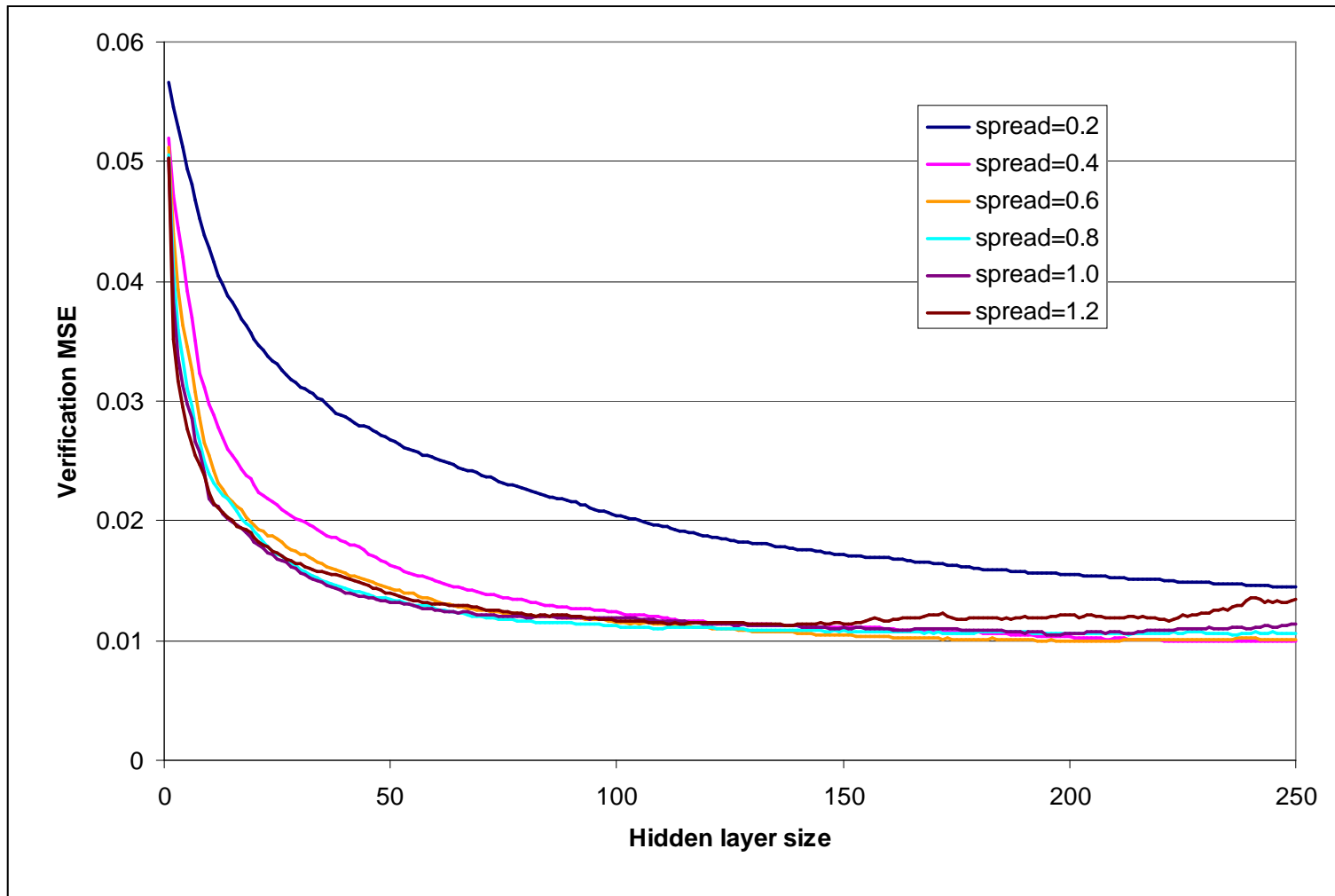


Figure 5.1: Variation in verification error with hidden layer size for RBF networks with various spread values

RBF width	Optimum layer size	Best average error	Optimum error
0.2	250	0.01448	0.01446
0.4	247	0.00992	0.00973
0.6	202	0.00990	0.00955
0.8	237	0.01052	0.00998
1	197	0.01048	0.00993
1.2	139	0.01128	0.01035

Table 5.1: Verification errors for networks containing RBFs of various widths, averaged for networks of the same size

Table 5.1 summarises the results of this pilot study. The statistics are the same as those reported for MLP networks. ‘Optimum layer size’ is the best network size for each spread and the corresponding errors, averaged across all 10 networks, are recorded as the ‘best average error’. The ‘optimum error’ is found by allowing the network size to vary across datasets and averaging the corresponding test errors across all 10 datasets.

Figures 5.2 and 5.3 display this information graphically. The first graph indicates that a spread of 0.6 results in the lowest verification errors, with $s = 0.4$ giving slightly higher errors. Given a considerable degree of variation in the results, the difference in average results may not be statistically significant. A decision was taken to train families of networks with spreads of both 0.4 and 0.6. Figure 5.3 shows that the optimum layer size generally decreases as the spread increases. As the radial basis functions have larger spheres of influence fewer of them are needed to cover the input space.

5.2 Method

The same 30 data splits were used to train RBF networks as were used for MLP networks (see section 4.4), in order to allow fair comparisons between the two types of network. All RBF networks were trained until they contained 250 RBF neurons. Resultant MSEs were obtained for all intermediate networks, so that ANNs of different sizes could be compared. Although verification errors were used to identify the best networks, or family of networks, the performance measures in the next section refer to errors obtained with the ‘unseen’ test data.

In addition to training networks with the basic FS-OLS algorithm, some networks were trained with the inclusion of regularisation. This technique, as described in section 2.5, reduces the extent of overfit by introducing a penalty term which is linked to

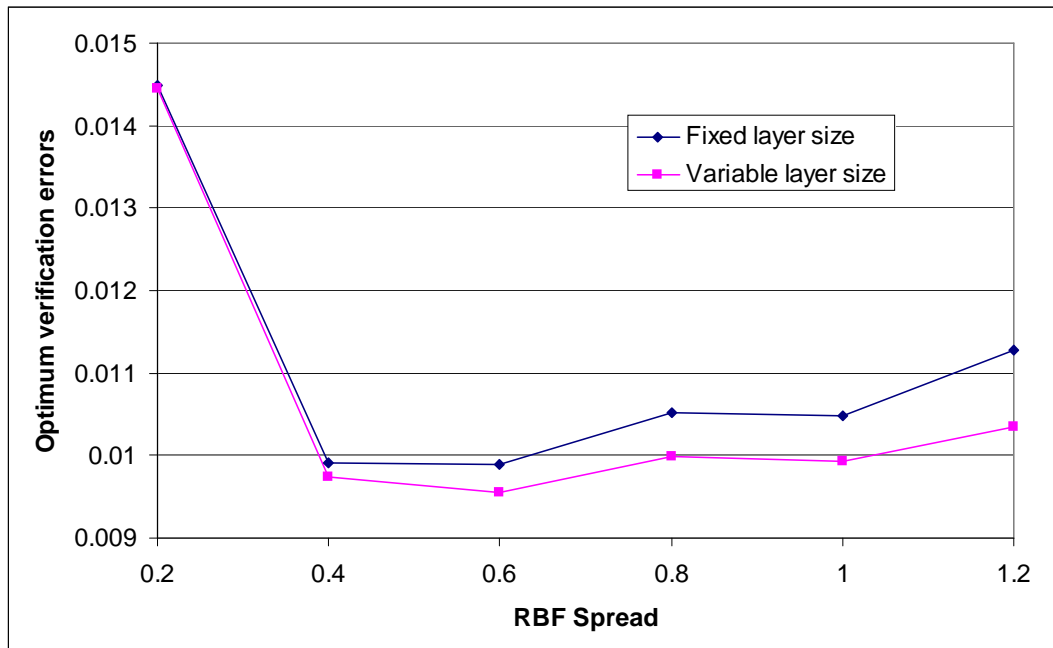


Figure 5.2: Dependence of verification errors on spread parameter

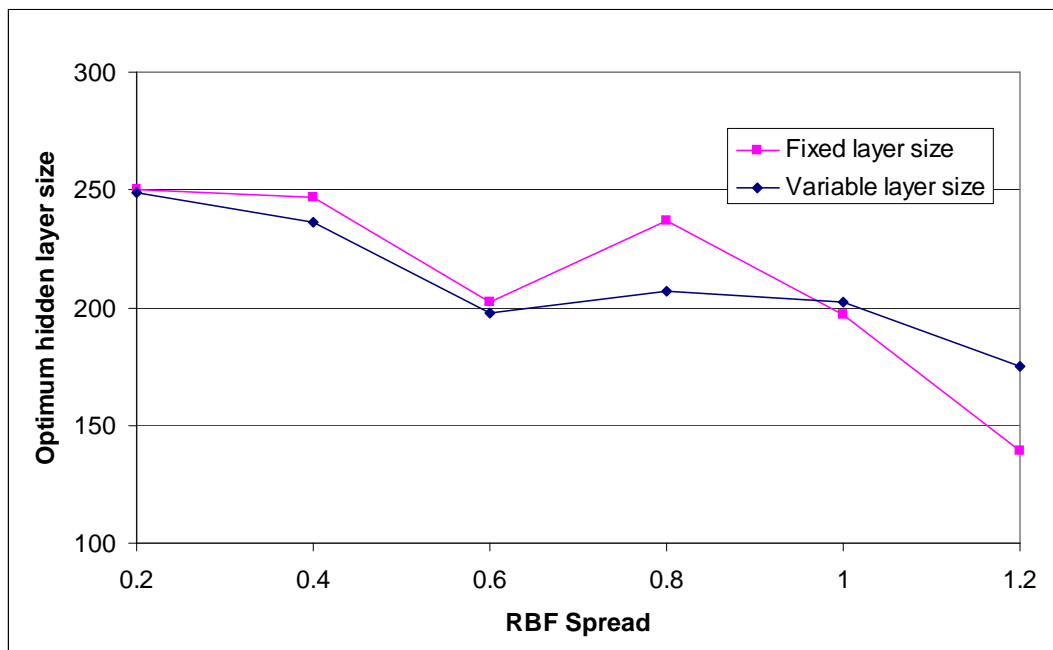


Figure 5.3: Dependence of hidden layer size on spread parameter

a regularisation parameter, λ . This term has the effect of reducing the weights between hidden and output neurons and so smoothing the overall function. A variety of λ values were used, varying exponentially between 10^{-10} and 1, in units of 10^2 . The results obtained using Forward Selection with Regularisation are given in section 5.3.2.

Finally, section 5.3.3 gives the results of training RBF networks with a combined training algorithm. This involves two steps: the construction of networks using FS-OLS followed by a gradient descent optimisation step. The optimisation step was performed using the Levenberg-Marquardt algorithm, as described in section 2.3. Training at this stage was performed for 50 epochs, which was found to be sufficient to achieve convergence. Networks trained with the FS-OLS algorithm are expected to have near-optimum weights, so the gradient descent step is quick compared to the training of pure MLP networks. The latter have randomly initialised weights and require approximately 200 epochs to converge (see section 4.2.6).

The improvement in results brought about by the introduction of a gradient descent optimisation step into RBF training has been pointed out by Schwenker *et al* [192]. A similar technique is used in the training of GL-ANNs, as described in Chapter 6. One reason for including the technique here is to allow a comparison between the results for networks containing just RBF transfer functions and those for hybrid networks reported in Chapter 7.

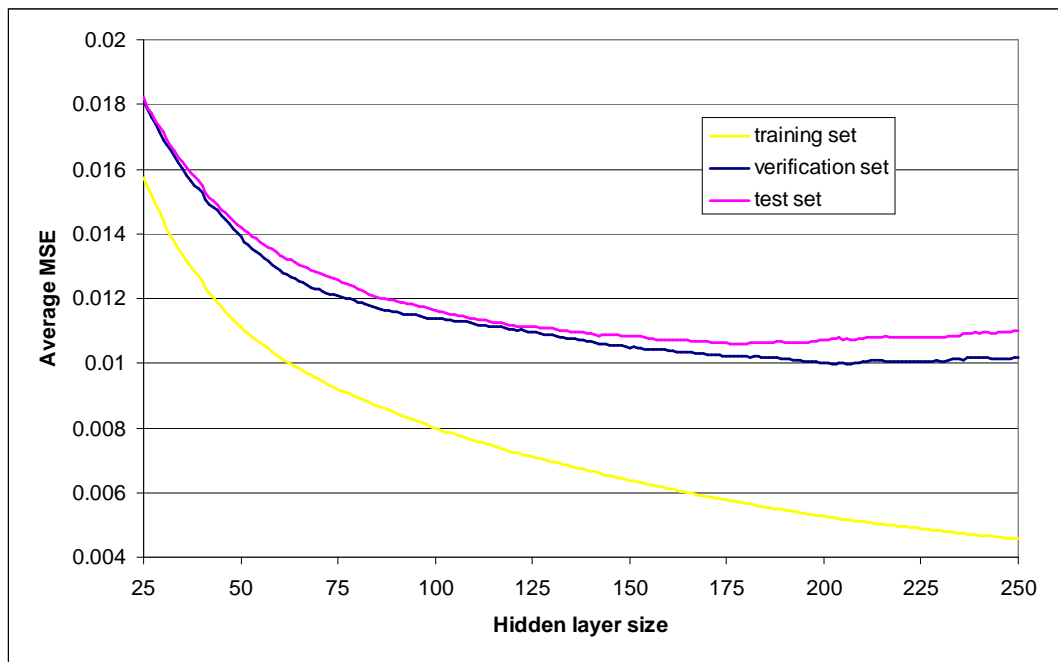
5.3 Results and Discussion

5.3.1 Results without regularisation

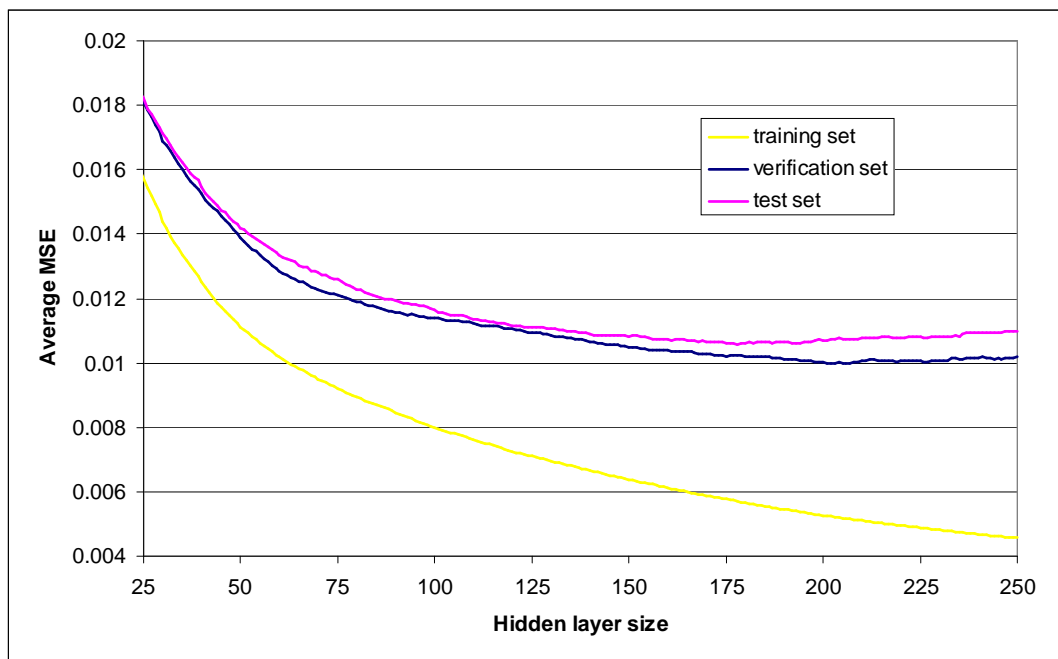
The progression in average errors as RBF neurons are added to the networks is shown in figure 5.4. In order to maintain a reasonable scale, results start at 25 hidden neurons. Spreads of both 0.4 and 0.6 result in a minimum verification error at approximately 200 hidden neurons, after which overfitting occurs.

The best results achieved, based on verification errors, with networks containing transfer functions with the two different spreads are given in table 5.2. This shows that the lowest test errors are obtained when the spread value is 0.4.

It is noticeable when comparing the results for the two spreads that the verification errors are a better guide to the test errors for the narrower spread. Since neither set was ‘seen’ by the networks during training there is no obvious reason for this observation, and must lie in the particular selections of data-splits. The implication is that averaging



(a) Spread=0.4



(b) Spread=0.6

Figure 5.4: Training, verification and test errors for RBF networks with 2 different spreads

Spread	Averaging technique	Verification MSE	Test MSE	Hidden layer size
0.4	Fixed layer size	0.01023	0.01060	198
	Variable layer size	0.00972	0.01021	227.9
0.6	Fixed layer size	0.00998	0.01075	203
	Variable layer size	0.00958	0.01050	206.9

Table 5.2: Best errors achievable with RBF networks

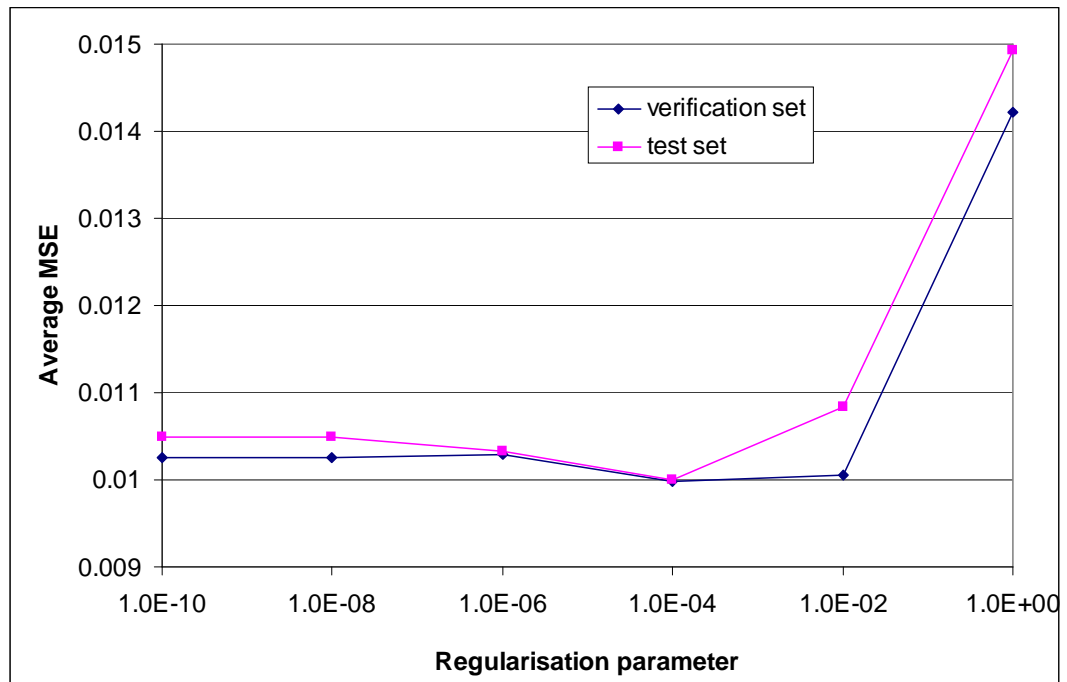
over 30 different data-splits has not completely eliminated the interactions between data-splits and model performance mentioned in section 4.3.

5.3.2 Results with regularisation

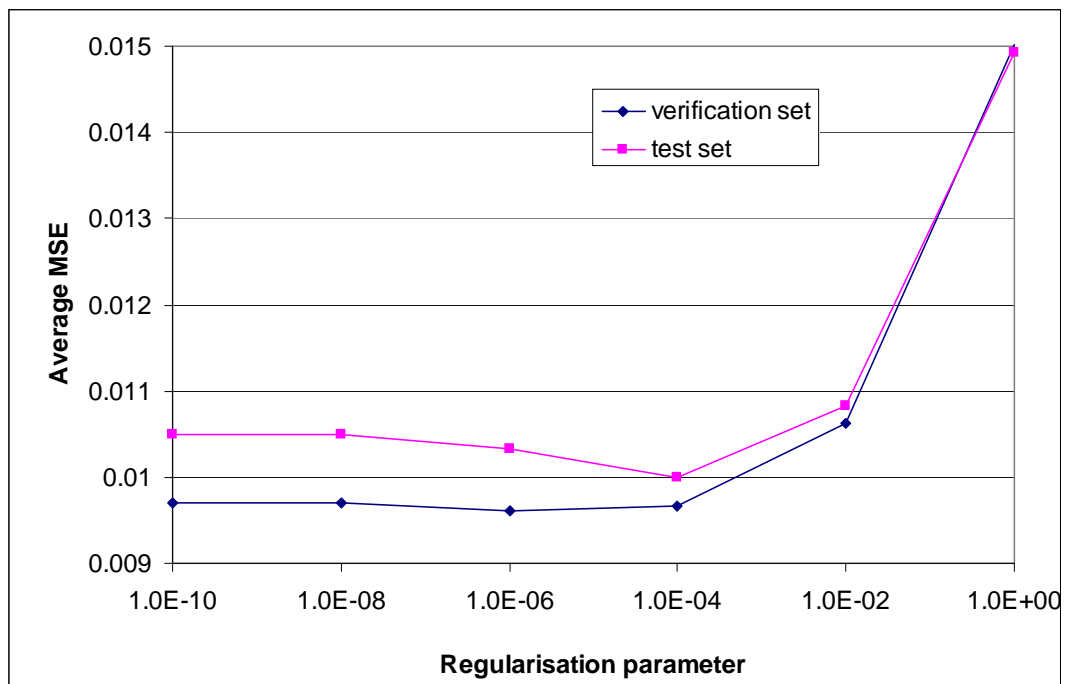
When regularisation was applied, networks were initially created with up to 200 neurons, since this was the approximate size of the optimum sized networks without regularisation. Figure 5.5 shows the dependence of the minimum error, averaged across fixed sized networks, for various regularisation parameters. The minimum error is achieved using $\lambda = 10^{-4}$, for widths of both 0.4 and 0.6. However, the minimum verification error is seen to occur in both cases with 200 neurons, or very close to this number. This suggests that the minimum error has not yet been achieved, so training was continued further with $\lambda = 10^{-4}$.

Figures 5.6 and 5.7 show the progression in errors, averaged across 30 networks, for networks containing radial basis functions with widths of 0.4 and 0.6, respectively, and a regularisation parameter of 10^{-4} . Training was stopped when 450 neurons, approximately 30% of the available centres, had been added, because the training had become very slow. At this stage the verification and test errors are levelling out, although they have not yet reached a minimum. The average test MSEs obtained at this point are 0.00938 and 0.00930 for widths of 0.4 and 0.6, respectively.

The effect of regularisation is that the addition of extra neurons does not lead to overfitting, even with extremely large networks, if an appropriate choice of λ is made. This makes it difficult to identify an optimum sized network, when averaging across all networks. However, when looking at networks trained with each data-split individually, it is possible to identify network sizes which result in minimum verification errors. These are seen to occur with somewhat fewer than 450 hidden layer neurons, as shown in tables 5.3 and 5.4. The optimum test errors achieved when variable architectures are allowed are 0.00939 with $s = 0.4$ and 0.00930 with $s = 0.6$.



(a) Spread=0.4



(b) Spread=0.6

Figure 5.5: Verification and test errors of RBF networks as a function of regularisation parameter

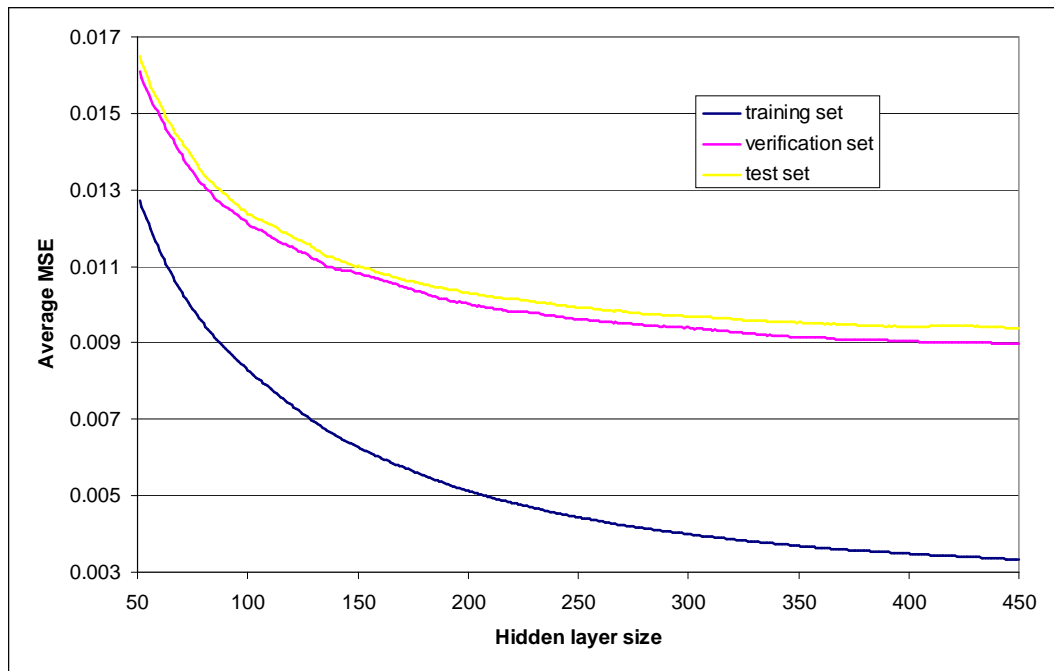


Figure 5.6: Error progression for RBF network with spread 0.4 and $\lambda = 10^{-4}$

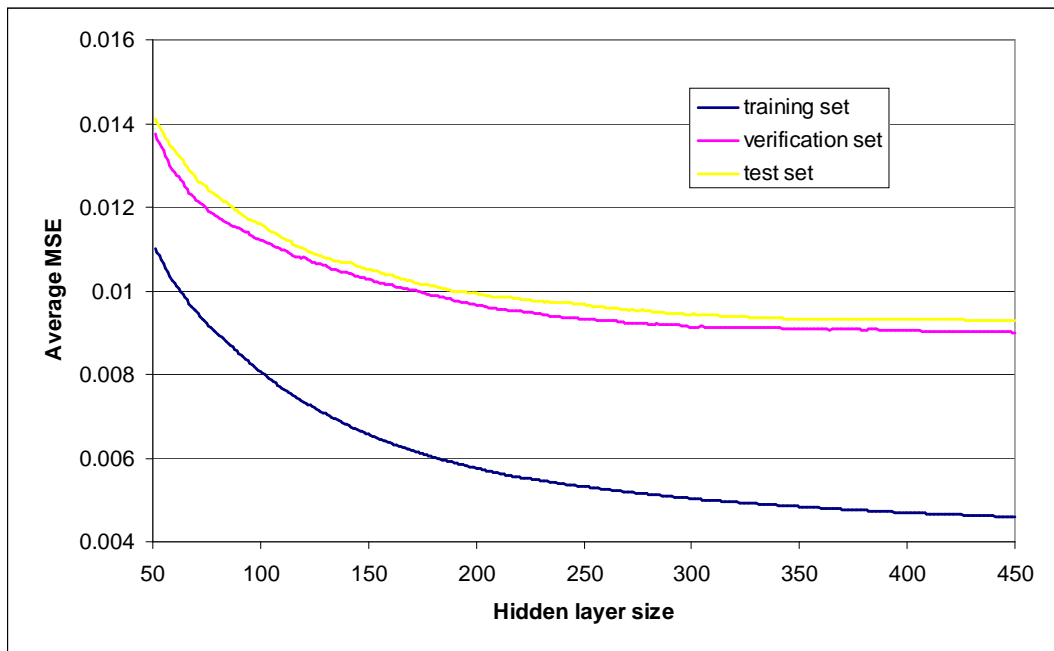


Figure 5.7: Error progression for RBF network with spread 0.6 and $\lambda = 10^{-4}$

Data split	Verification error	Test error	Optimum hidden layer size
1	0.00921	0.00765	434
2	0.00937	0.00917	402
3	0.00932	0.00947	447
4	0.00994	0.00915	446
5	0.00791	0.00978	375
6	0.00826	0.00892	442
7	0.00791	0.01002	437
8	0.00955	0.01086	370
9	0.00836	0.01007	404
10	0.00856	0.00886	445
11	0.00854	0.01012	405
12	0.00992	0.00944	448
13	0.01062	0.01113	413
14	0.00996	0.00933	410
15	0.00755	0.00857	405
16	0.00852	0.00971	380
17	0.00821	0.00920	420
18	0.00902	0.00884	383
19	0.01054	0.01030	360
20	0.00791	0.00869	438
21	0.00956	0.00905	446
22	0.00885	0.00703	425
23	0.00743	0.00938	438
24	0.00976	0.00884	360
25	0.00918	0.01018	371
26	0.00841	0.01002	441
27	0.00727	0.01010	437
28	0.00910	0.00881	407
29	0.00982	0.00799	373
30	0.00735	0.01106	448
Average	0.00886	0.00939	413.7

Table 5.3: Optimum errors for RBF networks with spread=0.4 trained with regularisation

Data split	Verification error	Test error	Optimum hidden layer size
1	0.00942	0.00818	316
2	0.01023	0.00828	319
3	0.00924	0.00900	398
4	0.00944	0.00897	389
5	0.00778	0.00963	381
6	0.00804	0.00942	357
7	0.00822	0.01020	279
8	0.00950	0.01062	418
9	0.00808	0.01000	449
10	0.00879	0.00936	446
11	0.00848	0.00985	346
12	0.00999	0.00942	296
13	0.00991	0.01192	417
14	0.00970	0.01026	241
15	0.00794	0.00808	330
16	0.00834	0.00844	421
17	0.00786	0.00884	314
18	0.00936	0.00928	241
19	0.00954	0.00976	449
20	0.00797	0.00791	450
21	0.01015	0.00915	422
22	0.01022	0.00767	416
23	0.00730	0.00961	419
24	0.00916	0.00855	301
25	0.00911	0.00946	321
26	0.00808	0.00961	450
27	0.00774	0.01053	449
28	0.00919	0.00870	342
29	0.00957	0.00768	320
30	0.00733	0.01071	442
Average	0.00886	0.00930	371.3

Table 5.4: Optimum errors for RBF networks with spread=0.6 trained with regularisation

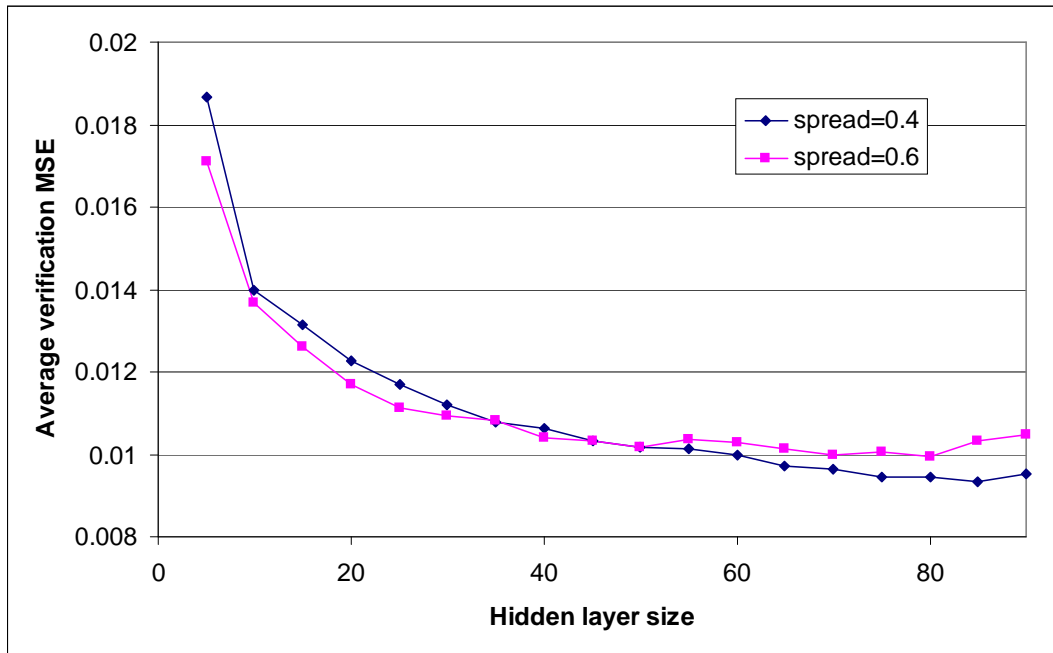


Figure 5.8: Verification errors for RBF networks trained with an optimisation step

5.3.3 Results with gradient descent optimisation

Gradient descent optimisation was found to be a slow process. For this reason it was impractical to optimise all possible RBF networks. Initially 10 networks were optimised for network sizes up to 90 hidden neurons, in steps of 5 neurons, with starting RBF spreads of 0.4 or 0.6. The resulting verification errors are shown in figure 5.8. From this graph it is apparent that gradient descent optimisation has a particularly favourable impact upon smaller networks, resulting in a shift in the optimum network size from approximately 200 neurons without optimisation down to 85 neurons with optimisation. The best results appear to be achieved with a starting spread of 0.4.

For a hidden layer size of 85 and spread of 0.4, all 30 networks were optimised. When averaged across all 30 networks, the resulting test error was 0.00956. These results are a considerable improvement on the results without optimisation. Further, they are achieved using much smaller networks.

5.3.4 Speed and memory comparisons

Like the algorithms used to train MLP networks, the FS-OLS algorithm was implemented in Java and run on a PC containing an AMD Athlon 2100+ chip with a clock-speed of 1.74 GHz. The time taken to train a series of RBF networks containing between 1 and 250 neurons using the FS-OLS algorithm was approximately 23 minutes. This is about half the time taken to train a set of MLP networks using the L-M algorithm and approximately a fifth of the time taken by the BP algorithm.

Introduction of a regularisation parameter had no appreciable impact on training speeds. However, larger networks were created when regularisation was introduced. Since the training time increases approximately proportionally to the square of the network size, this led to considerable costs in training times. Training a series of networks containing 450 hidden layer neurons took approximately 95 minutes. This is less than the amount of time taken to train a series of MLP networks using BP, but more than the time taken by the L-M algorithm.

The FS-OLS algorithm requires greater RAM allocation than the L-M or BP algorithms. However, the requirement does not exceed 100 MB, and therefore appears to pose few problems if running the algorithm on a modern computer.

Gradient descent optimisation of large networks using the L-M algorithm is slow. For networks of the optimum size, containing 85 hidden neurons, it takes approximately 38 minutes to optimise each network. It is therefore necessary to be selective when choosing which networks to optimise (see section 5.3.3). An alternative would be to use a different algorithm to perform optimisation when large networks are involved. First-order gradient descent, as described in section 2.2.2, is a possible choice. However, the conjugate gradient method (see Appendix B) might be a more effective algorithm, and is considerably quicker than the L-M algorithm for large networks. Gradient descent optimisation does not require the allocation of memory in addition to the 64 MB automatically allocated by the Java virtual machine.

5.4 Summary and a comparison with MLP networks

Table 5.5 summarises the results achieved using RBF networks. Also included are the results for MLP networks trained with the L-M algorithm, obtained in chapter 4, for comparison. The best results are obtained using transfer functions with a spread of 0.6 and a regularisation parameter of 10^{-4} and they are a considerable improvement on

those obtained with MLP networks. These results may be regarded as the best achievable using variants of the forward selection algorithm on the CLASH dataset. The results obtained using this algorithm are not subject to the setting of various training parameters, as are those obtained for MLP networks. It may be that better optimisation of training parameters could improve the results for MLP networks somewhat. However, it seems unlikely that they would improve to such an extent that they could compete with the results obtained with RBF networks.

RBF networks generally require many more neurons than MLP networks to achieve comparable results. The reason for this is that each neuron only has a significant effect upon a small volume of the input-space. A large number of neurons are therefore required to cover this input-space particularly when there are a large number of inputs, as there are here.

In terms of training time the network size does not present a problem, with total training times comparable for MLP and RBF networks. However, from the point of view of function approximation the RBF approach appears to have some drawbacks. Firstly, the function created has an extremely complex functional form. From a Bayesian point of view such a function has a low 'likelihood'. Instinctively, one feels that the highly complex functions produced by the RBF networks cannot represent the 'true' underlying function.

There are also practical reasons for preferring a simpler function. One use of the research reported in this study would be to derive symbolic meaning from the neural network weights, possibly in the form of a regression tree. If the produced ANNs are extremely complex, the derived tree is unlikely to provide useful symbolic information. A further reason for having reservations concerning RBF networks is that it is known that they are poor at interpolating between clusters of data into empty areas of input space. As shown in section 3.3.2 the CLASH data is highly clustered. One would like to be able to interpolate between clusters so that predictions may be made concerning previously unknown structures. For this reason, it may be valuable to incorporate information from MLP networks.

Chapter 6 describes an algorithm that creates a hybrid MLP-RBF network, with the aim of combining the advantages of MLP networks - small network size and extrapolation ability - with the advantages of RBF networks - local function approximation and accurate output prediction. The training algorithm for these hybrids incorporates a gradient descent optimisation step. Section 5.3.3 showed that the introduction of this step results in a substantial reduction in optimum hidden layer size and is therefore

beneficial for the reasons given above.

Training technique	Best hidden layer size	Best test error with fixed architecture	Best test error with variable architecture
FS-OLS, $s = 0.4$	198	0.01060	0.01021
FS-OLS, $s = 0.6$	203	0.01075	0.01050
FS-OLS, $s = 0.4, \lambda = 10^{-4}$	450	0.00938	0.00939
FS-OLS, $s = 0.6, \lambda = 10^{-4}$	449	0.00930	0.00930
FS-OLS, then L-M optimisation	85	0.00956	-
MLP L-M, linear	14	0.01168	0.01107
MLP L-M, sigmoid	19	0.01114	0.01071

Table 5.5: Summary of the results of training RBF networks with the CLASH dataset

Chapter 6

GL-ANN theory and algorithm

6.1 Background

The need for machine learning techniques to identify global and local features separately has been recognised for some time. Minsky and Papert noted in 1969 that [66]

the appraisal of any particular scheme of parallel computation cannot be undertaken rationally without tools to determine the extent to which the problems to be solved can be analyzed into local and global components.

This chapter describes a scheme for developing global-local artificial neural networks (GL-ANNs). GL-ANNs have an architecture containing neurons with both sigmoidal and RBF transfer functions. Associated with this hybrid architecture is a training algorithm which is designed to give good generalisation properties and rapid training.

The aim of the GL-ANN method is to separate the global and local features of an unknown multivariate function. Recent support for such a separation comes from three main areas: mathematical analysis, cognitive psychology and developments within computer science.

6.1.1 Mathematics

Donoho and Johnstone [193] have shown that kernel-based and projection-based functions have complementary properties. In particular, they show that ‘ancillary smoothness’ in the target function may be used to reduce the effective dimensionality of the

data. They define an angularly smooth function as one that varies slowly with angle, while a function with radial smoothness shows small local variations in value. Projection-based functions are seen to respond well to angular smoothness while kernel-based functions respond well to radial smoothness. For complex, high-dimensional functions one expects to find aspects of both types of smoothness. In order to achieve optimum results with the smallest possible network it therefore seems advisable to use neurons with both projection-based and kernel-based functions.

6.1.2 Cognitive psychology

As well as having a sound mathematical basis hybrid networks may have more biological validity than pure multi-layer perceptron (MLP) networks [99, 194]. There is considerable evidence that the human brain processes information in a modular way [195]. For example, global and local aspects of visual stimuli are processed by different parts of the brain, suggesting the specialisation of neurons for these different purposes [196, 197]. Further, brain development often occurs in stages, with each stage dependent upon the completion of previous stages [198]. The architectural structure of GL-ANNs is similarly reflected in a stepwise training algorithm [195].

6.1.3 Computer Science

As computing power increases computer scientists are dealing with larger, higher-dimensional datasets and, presumably, more complex underlying functions. Hrycej believes that there is a need to use more complex models such as modular ANNs in order to satisfactorily model these functions [195]. Each module within a network may then be assigned a different task, or sub-task, according to the particular architecture of that module or the training method applied to it. One advantage in using a stepwise modular approach is that the effectiveness of each step may be assessed individually, enabling some information to be extracted from the ‘black box’ of ANN training.

Poggio and Girosi have suggested the use of networks containing both Gaussian and other functions in a single layer. These networks are extensions of traditional RBF networks called ‘HyperBFs’ [199]. They contain a single hidden layer containing Gaussian functions of variable width and additional non-radial functions. Girosi *et al.* [200] have demonstrated mathematically the close relationship of HyperBFs to regularisation theory. GL-ANNs may be seen as an implementation of HyperBFs, with a particular emphasis on the separation of global and local variations in the regression

function.

Moody has highlighted the difficulty in identifying both the coarse structure and the fine detail of an input-output relationship [201]. His multi-resolution technique uses RBF neurons of differing widths to solve this scaling problem. The GL-ANN approach builds on this work, allowing extra flexibility in the choice of RBF widths and the addition of sigmoid functions to map features of the function that are more suited to this geometric form.

GL-ANNs have similarities with the hybrid and modular approaches described in section 1.5.8 such as PRBFNs and mixtures-of-experts. They also share some features with Orr's regression tree derived RBF (RT-RBF) approach (section 1.5.7). However PRBFNs, mixtures-of-experts and RT-RBFs all cluster the training data prior to network training. The GL-ANN approach uses all training data in all phases of training, keeping the variance low [202]. It also avoids a number of known problems with clustering, namely-

- Clustering may reflect the distribution of the available data rather than the underlying functionality.
- Clustering generally reflects the distribution of the input data, but does not take into account the distribution of the output data [180]. This is a problem for highly non-linear data such as the wave overtopping data, for which small changes in the inputs sometimes cause large changes in the output.
- Unsupervised clustering can lead to very large, and therefore overfitted, networks [203].

One hybrid approach that does not use clustering is the genetic algorithm approach of Yang [127], described in section 1.5.8. Yang uses GAs to search model space for the optimum sigmoid-RBF hybrid architecture. His work concentrates on the choice of model, with basic Levenberg-Marquardt training used to train each network. This study may be seen as complementary to that of Yang. It uses a fairly 'brute force' approach to model selection, creating series of networks for all possible architectures, but employs a fairly sophisticated method of training individual networks.

6.2 The ideas behind GL-ANNs

MLP and RBF networks have complementary properties. While both are theoretically capable of approximating a function to arbitrary accuracy using a single hidden layer

[204, 205], their operation is quite different [35]. MLP networks have a fixed architecture and are usually trained using a variant of gradient descent, as described in sections 2.2 and 2.3. They invariably incorporate neurons with sigmoid activation functions. Their response therefore varies across the whole input space and weight training is affected by all training points. RBF networks, on the other hand, are most commonly created using a constructive algorithm. Gradient descent training is usually replaced by deterministic, global methods such as Forward Selection of Centres with Orthogonal Least Squares (FS-OLS). This method has been described in detail in section 2.4.

Whereas MLPs are effective at identifying global features of the underlying function, RBF networks have the capacity to identify local variation in the function [195, 180, 206]. MLPs are more distributive in their representation of the input-output relationship, since little meaning can be attached to the weights of any individual neuron. For this reason they may be seen as more ‘emergent’ and opaque [195, 207].

On the other hand RBF centres are deliberately selected, often from the training set, as representatives, or prototypes, of the entire training set. Since each neuron within a RBF network may be seen as a prototype for the whole dataset, RBF networks are slightly more transparent and are easier to interpret symbolically than are MLP networks [195].

The training of RBF networks is generally faster, as seen in Chapter 5. The main reason for this is that RBF networks generally contain linear output neurons and fixed hidden layer neurons. The optimisation algorithms used therefore involve the solving of linear rather than non-linear equations [208, 176]. However, RBF networks often contain many more neurons than the corresponding MLP networks, partly offsetting the advantage in computational efficiency [206], as reported in section 5.3.4.

A hybrid ANN containing both sigmoidal and radial neurons may have the advantages of both RBF and MLP ANNs, i.e. computational efficiency, good generalisation ability and a compact network architecture. GL-ANNs approximate on a global level first using a MLP and then add RBF neurons using FS-OLS, in order to add local detail to the approximating function. Identifying coarse structure before fine detail makes sense from a computational point of view [201]. This sequential process may also mirror the operation of biological brains: there is considerable evidence from cognitive psychology that humans identify global features of visual stimuli before local features [209] and that the global features affect the interpretation of the local features [210]. The training process is completed with an optimisation step that adjusts the weights of all neurons, including RBF centres and widths.

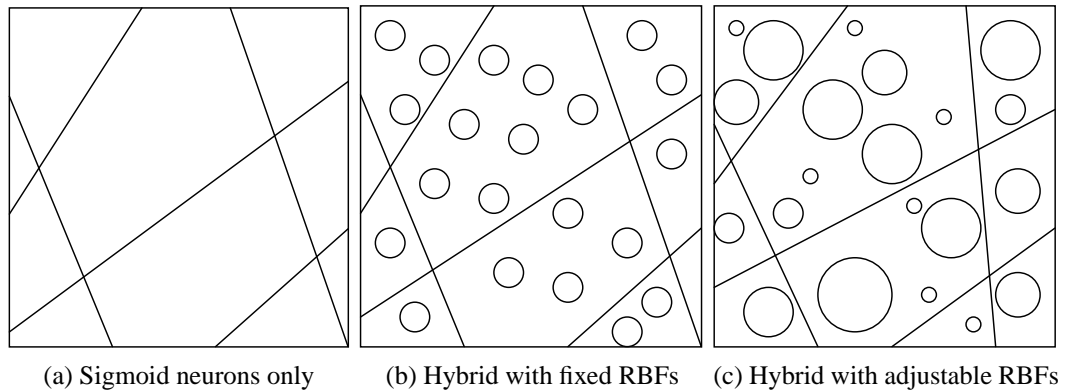


Figure 6.1: Diagrammatic representation of the GL-ANN training process

The three-step training process is illustrated in figure 6.1. After the first step, an ANN containing just sigmoid neurons is created. The sigmoid functions approximate a stepwise function (see figure 2.1) and therefore partition the input space into regions. After the second step, detail has been added over the top of these partitions, using RBF functions. Finally, the positioning of the sigmoid functions and the locations and sizes of the RBF functions are optimised, allowing RBFs of variable widths.

6.3 GL-ANN Algorithm

At each stage of GL-ANN training attempts have been made to select a training method that is efficient in terms of computational power, given the architecture of the network. Chapter 4 indicated that the Levenberg-Marquardt method is an efficient means of training MLP networks containing up to approximately 20 hidden neurons, and this method is used in the first stage of training GL-ANNs. In order to use this procedure local partial derivatives are first calculated for the input weights (including bias weight) using equation 6.1. Local inputs and weights are given by i_k and w_k , respectively and y is the pertinent neuron's output. The Hessian matrix may then be approximated as described in section 2.3.

$$\frac{\partial y}{\partial w_k} = \frac{1 - y^2}{2} i_k \quad (6.1)$$

In the second stage RBF neurons are added using a variant of the FS-OLS algorithm described in section 2.4. The RBF neurons employ symmetrical radial functions with fixed widths at this stage. The FS-OLS requires some modifications to make it applicable to hybrid networks. If the training data contains m items, each is regarded as

a potential RBF centre and the full design matrix, \mathbf{F} , is an m -by- m matrix containing the outputs of each RBF neuron given each input. The design matrix for a network containing p RBF centres, \mathbf{A} , is a m -by- p matrix containing columns selected from \mathbf{F} . If the target outputs are given by \mathbf{t} the optimal output weights, \mathbf{w} , may then be determined from equation 6.2, giving the minimum least square error.

$$\mathbf{w} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{t} \quad (6.2)$$

An efficient method for solving this problem, first reported by Chen *et al.* [104], has been described fully in section 2.4. It requires that \mathbf{F} is factorised into an orthogonal matrix $\tilde{\mathbf{F}}$ and an upper triangular matrix. The columns in $\tilde{\mathbf{F}}$ must be kept orthogonal to each other whenever a RBF neuron is added to the network. If the column vector in $\tilde{\mathbf{F}}$ corresponding to that neuron is denoted by $\tilde{\mathbf{f}}_j$, the alteration may be stated as equation 6.3.

$$\tilde{\mathbf{F}}_{n+1} = \tilde{\mathbf{F}}_n - \frac{\tilde{\mathbf{f}}_j \tilde{\mathbf{f}}_j^T \tilde{\mathbf{F}}_n}{\tilde{\mathbf{f}}_j^T \tilde{\mathbf{f}}_j} \quad (6.3)$$

In GL-ANNs the hidden layer contains RBF and sigmoidal neurons, both of which provide outputs that are passed on to the output neuron. The outputs of both the sigmoid and the RBF neurons must be ‘orthogonalised’ when calculating the error reduction. This requires the following modifications:

- The addition of extra columns to the full design matrix, to represent the outputs of the sigmoid neurons. \mathbf{F} is therefore non-square, containing, for m training items and n sigmoid neurons, m rows and $m + n$ columns.
- Before any RBF neurons are added, the design matrix must be orthogonalised by carrying out the orthogonalisation of equation 6.3 for each existing sigmoid neuron, so ensuring that only the components orthogonal to the existing neurons’ outputs are considered.

In the final training stage all weights, including hidden layer weights and each RBF steepness, are optimised using L-M training. The local partial derivatives for RBF weights (centres) and steepness are given, respectively, by equations 6.4 and 6.5. i_k , w_k and y are used as in 6.1 while d is the distance between the input vector \mathbf{i} and the weight vector \mathbf{w} (see equation 2.8). σ is the RBF steepness.

For RBF input weights (centres):

$$\frac{\partial e}{\partial w_k} = 2y\sigma^2 (i_k - w_k) \quad (6.4)$$

For RBF bias weight (steepness):

$$\frac{\partial e}{\partial w_k} = -2\sigma d^2 y \quad (6.5)$$

Given the local partial derivatives, the Hessian may be estimated as described in section 2.3 and second-order gradient descent performed on the hybrid network.

6.4 Summary

This chapter has described the background to the GL-ANN algorithm. It has been shown that support for the use of hybrid networks exists in the areas of mathematical optimisation, cognitive psychology and within computer science. The key idea behind GL-ANNs is the combination of sigmoid and RBF neurons. Associated with the hybrid architecture is a hybrid training algorithm that combines gradient descent training with forward selection. The algorithm has been described in detail in section 6.3 and is illustrated in figure 6.2. The aim in using this algorithm is to separately and sequentially identify global and local components of an unknown function. Chapters 7 and `cha:benchmarkDatasets` look at the effectiveness of the algorithm in modelling, respectively, the behaviour of the CLASH data and of a number of benchmark datasets.

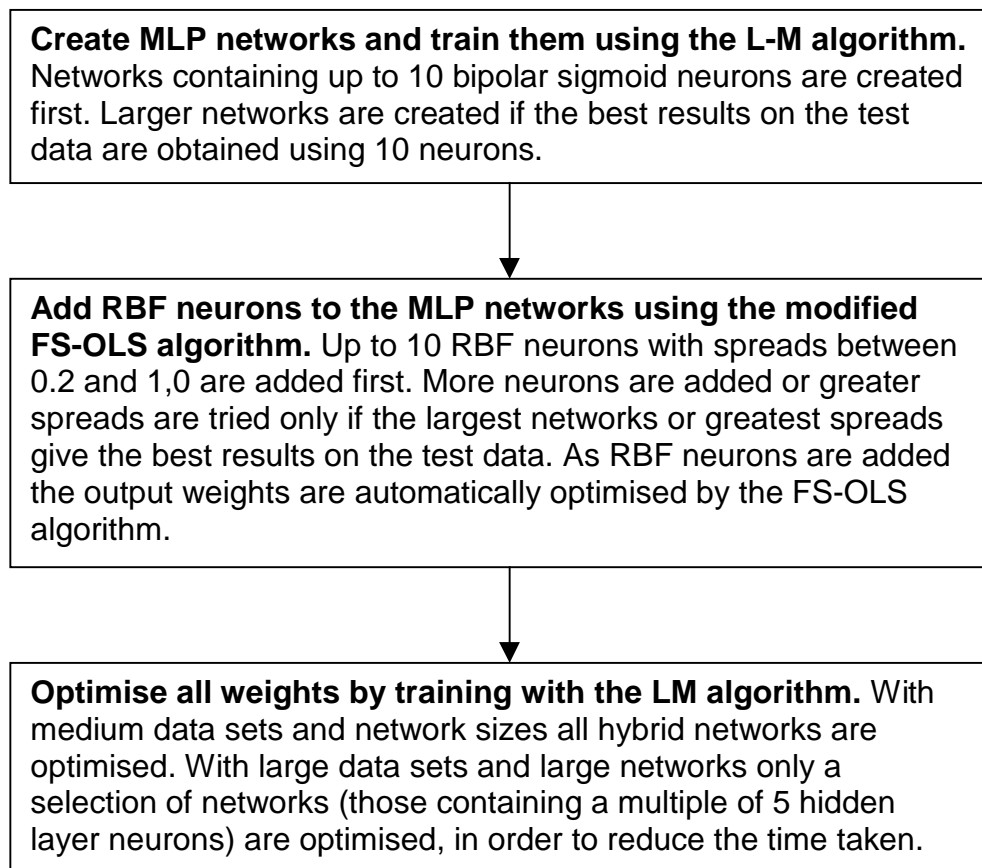


Figure 6.2: Flow chart summarising the GL-ANN algorithm

Chapter 7

CLASH prediction using GL-ANN algorithm

This chapter reports the results of training GL-ANN networks with the CLASH dataset. Section 7.1 describes the method used to train the networks. Section 7.2 gives the results of training two-step GL-ANNs, three-step GL-ANNs and hybrid networks trained with regularisation. Comparisons are made with the corresponding RBF networks and between the three types of hybrid network. Section 7.3 summarises the results.

7.1 Method

This section describes the method used to train series of networks to map the underlying function within the CLASH dataset with the GL-ANN algorithm. As described in Chapter 6 this is a three-step algorithm.

The first step involves the training of MLP networks. The results of this step have been reported in Chapter 4. Some of the networks described in that chapter were used as starting networks in the second training stage. However, only networks containing a linear output neuron were used. A linear output function is required, since the second step involves the use of the FS-OLS algorithm, which can set the hidden-output weights, but only if the output neuron has a linear transfer function. 30 different splits of the data were used, as reported in chapter 4.

In the second training step, RBF neurons were added to the trained MLP networks. Up to 250 RBF neurons were added to the MLP networks. Initially only 10 networks were trained for each architecture. The most promising architectures were then trained with all 30 data splits and the test errors averaged across all 30 networks.

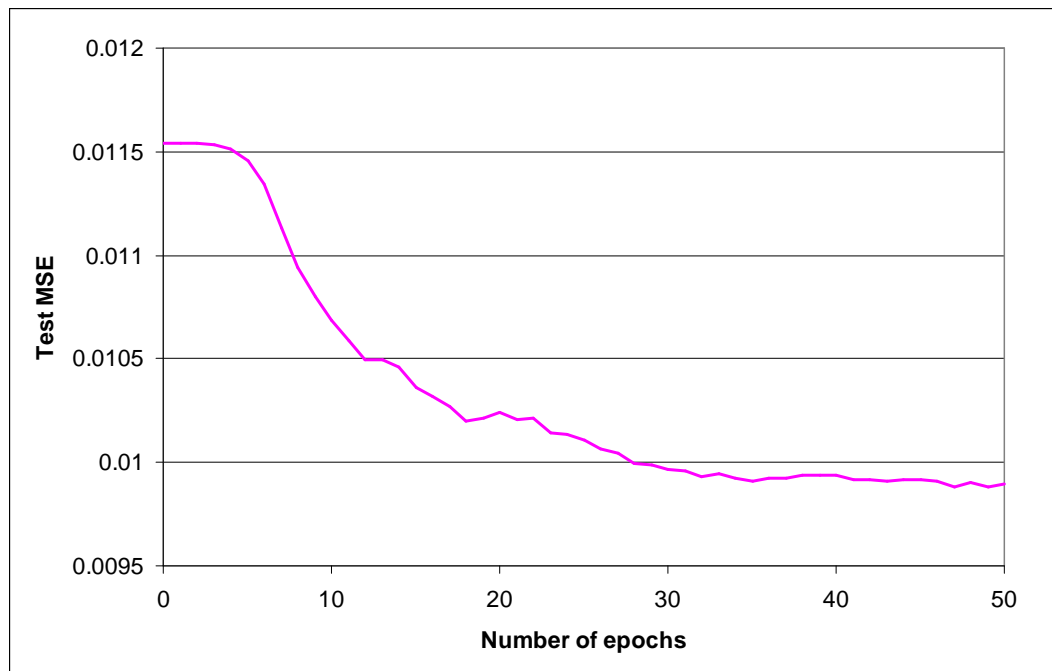


Figure 7.1: Progression in test error during the optimisation of hybrid networks containing 5 sigmoid neurons and 85 RBF neurons

The networks created with the CLASH dataset were very large and the optimisation step was slow. Networks were therefore selectively optimised, following a search procedure designed to locate the optimum network architecture. In this stage, only 10 networks were trained for each architecture. The results were used to successively narrow down the optimum architecture and only the optimum architecture was tested using all 30 data splits.

Due to the time taken to train the large networks, gradient descent optimisation was only carried out for 50 epochs, rather than the 200 epochs used to train MLP networks (see section 4.2.6). The starting networks have weights that are fairly close to their optimum values, since they have been produced by least squares optimisation. This contrasts with the situation during MLP training, when weights are initialised randomly. 50 epochs was therefore seen to be sufficient to achieve a levelling-off in the test error, as illustrated by figure 7.1. This shows the error progression for networks containing 5 sigmoid and 85 RBF neurons, averaged across 10 runs. Similar patterns of behaviour are seen for alternative architectures.

As an alternative to gradient descent optimisation, regularisation was introduced to the training of hybrid networks. This technique has also been used in the training of RBF networks, as described in Chapter 5. Again the regularisation parameter, λ , was

set to values between 1.0 and 10^{-10} , and RBF neurons with spreads of 0.4 or 0.6 were added.

In order to allow a fair comparison with pure RBF networks the networks created after the second training step are compared with RBF networks trained with FS-OLS, in section 7.2.1. The results of training with the full three-step algorithm are compared with RBFs trained with a two-step algorithm, including a gradient descent optimisation step, in section 7.2.2. Finally, the results of introducing regularisation are discussed in section 7.2.3. Again comparisons are made with pure RBF networks.

7.2 Results

7.2.1 Two-step algorithm

In creating hybrid networks, information obtained from the training of RBF networks was used to guide the choice of networks to create. For this reason attention was focussed upon RBF neurons with spreads of 0.4 or 0.6.

In the first stage, networks were trained with 6, 10 or 14 sigmoid neurons and up to 250 RBF neurons. The results from these architectures suggested that networks with fewer sigmoid neurons gave lower MSEs. Further hybrid networks were therefore created containing 5, 7, 8 and 9 sigmoid neurons. Again the networks contained up to 250 RBF neurons. The results averaged over 10 networks are illustrated in figures 7.2-7.5. The first two figures show the results with RBF spreads of 0.4 and, respectively, fixed and variable hidden layer sizes. Figures 7.4 and 7.5 show the corresponding results with a spread of 0.6.

The best results are seen to occur with a spread of 0.4 and with 6 or 8 sigmoid neurons. All 30 networks were trained with these architectures. The results are given in table 7.1. They show that the best architecture contains 6 sigmoid neurons and 207 RBF neurons, resulting in a test error of 0.00999. The best results obtained when the number of RBF neurons is allowed to vary for different data splits are obtained with networks containing 8 sigmoid neurons and an average of 198.8 RBF neurons. The resultant test MSEs average 0.00992.

These results are an improvement on those obtained using pure RBF networks or pure MLP networks. A comparison of the hidden layer sizes of hybrid networks obtained from the two-step GL-ANNs and pure RBF networks shows that they are of similar size (see section 5.3)

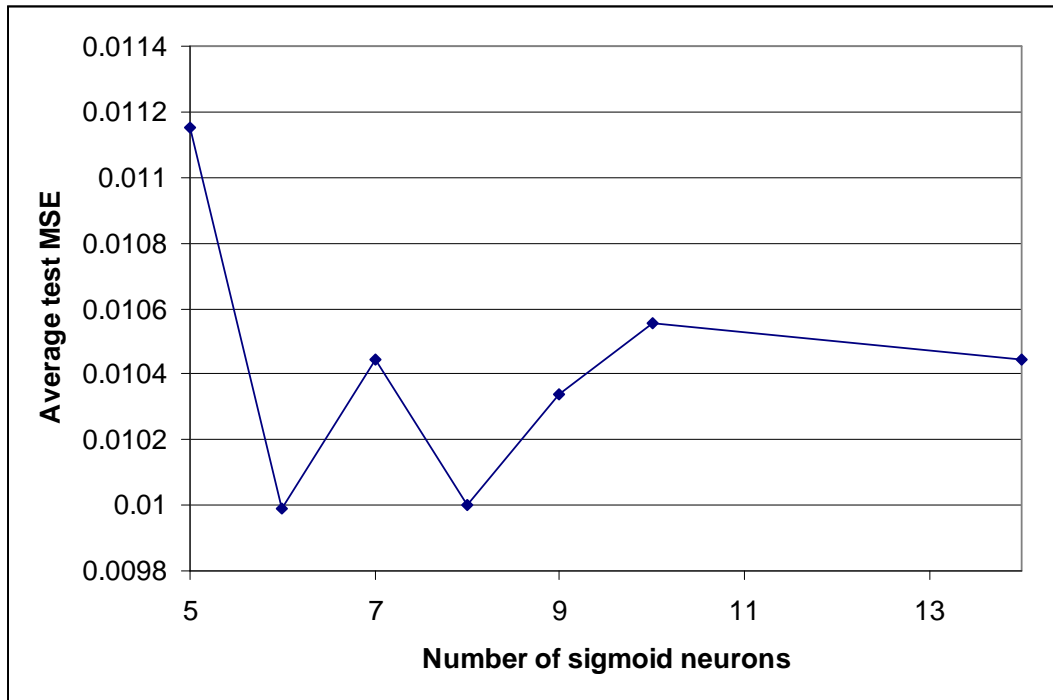


Figure 7.2: Errors for hybrid networks with spread 0.4 averaged across fixed architectures

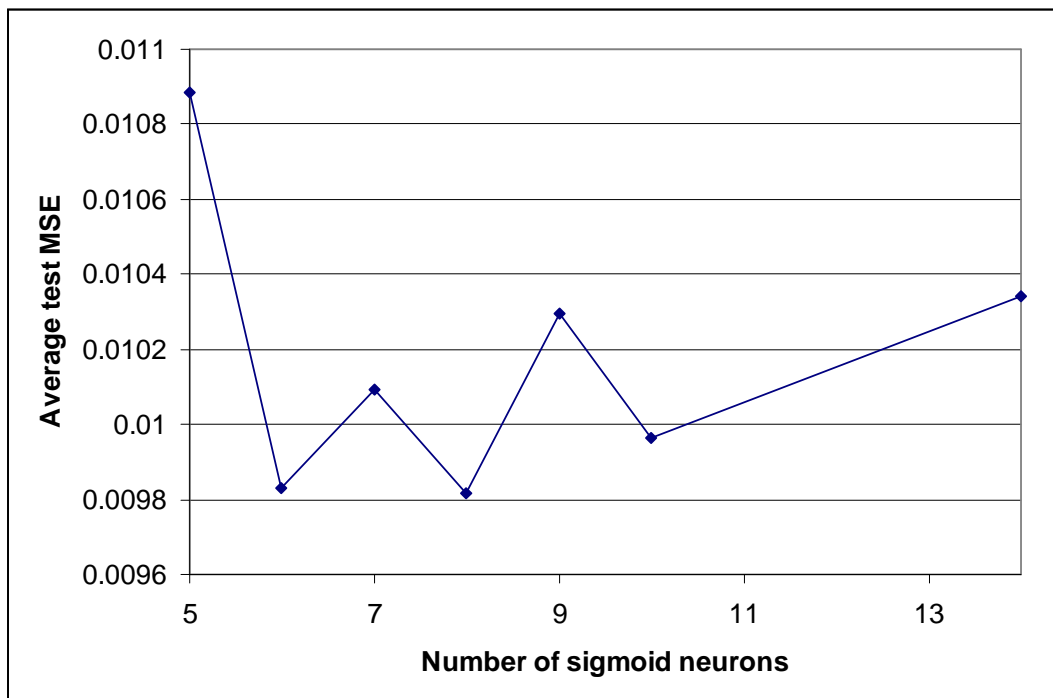


Figure 7.3: Errors for hybrid networks with spread 0.4 averaged across variable architectures

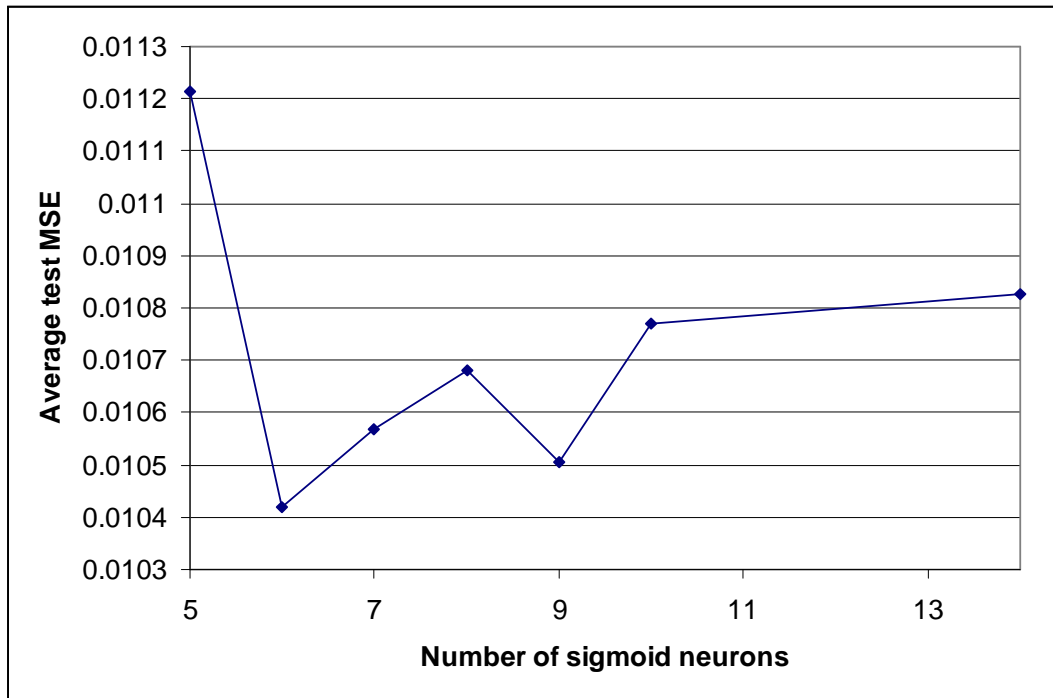


Figure 7.4: Errors for hybrid networks with spread 0.6 averaged across fixed architectures

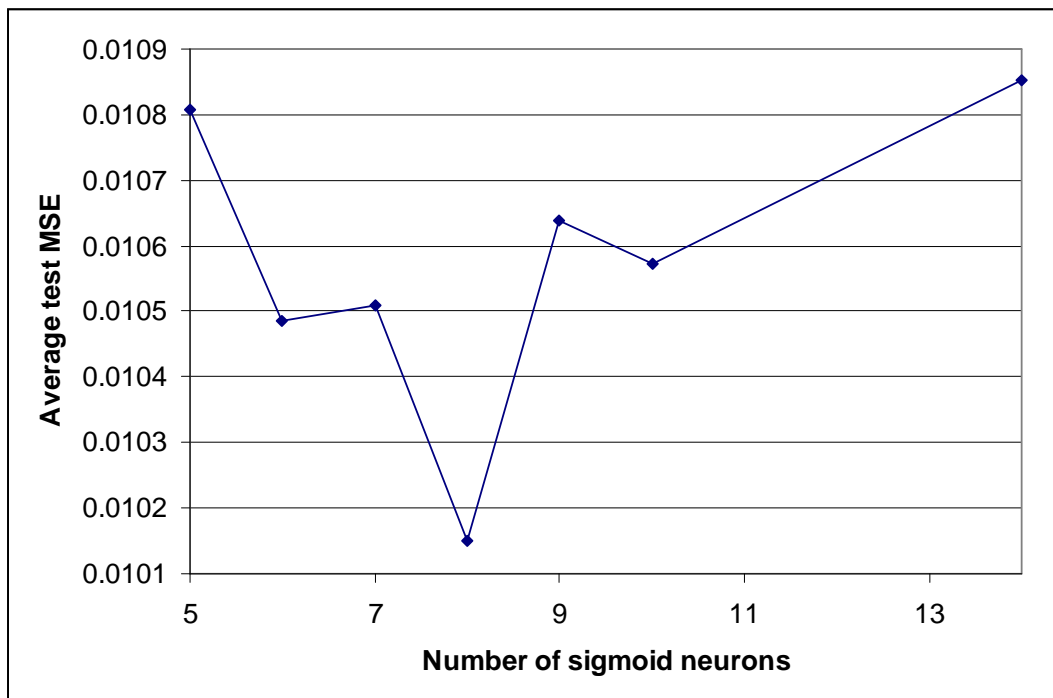


Figure 7.5: Errors for hybrid networks with spread 0.6 averaged across variable architectures

Number of sigmoid neurons	Averaging technique	Test MSE	Number of RBF neurons
6	Fixed layer size	0.00999	207
	Variable layer size	0.00993	211.2
8	Fixed layer size	0.01028	250
	Variable layer size	0.00992	198.8

Table 7.1: Best errors achievable with two-step GL-ANN

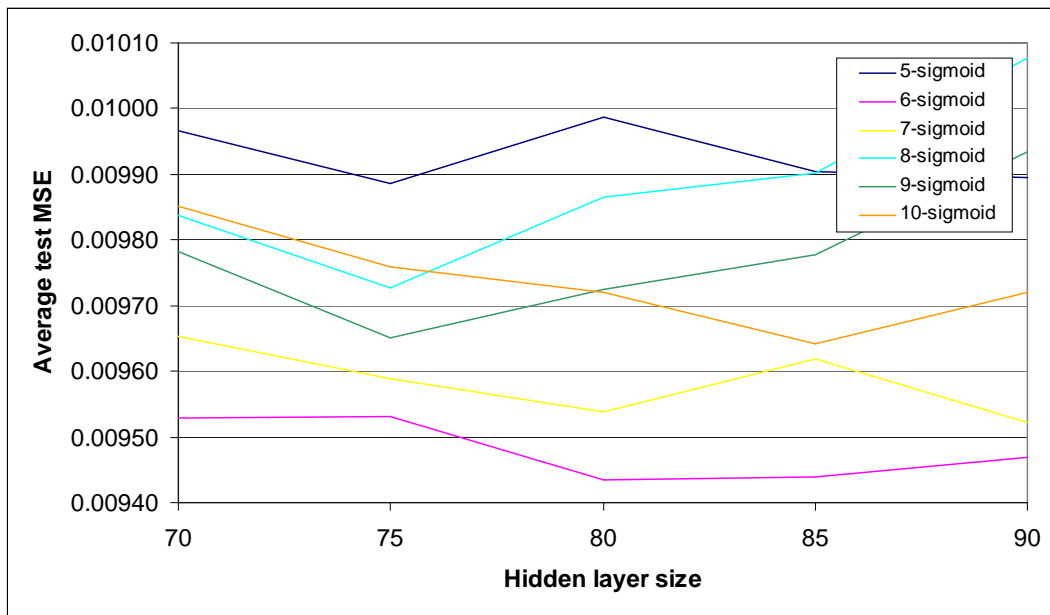


Figure 7.6: Errors for three-step GL-ANNs with near optimum architectures

7.2.2 Three-step algorithm

In selecting which hybrid networks to optimise, the results from the training of RBF networks were again used as guidance. The optimum hidden layer size for pure RBF networks was reduced by gradient descent optimisation from about 200 to 85. It was assumed that optimisation of hybrid networks would similarly reduce the optimum size of the networks. Only networks with up to 100 hidden layer neurons (in steps of 5) were therefore optimised. The number of sigmoid neurons in the GL-ANNs was varied between 5 and 10 inclusive and the starting RBF width was 0.4. The results are illustrated in figure 7.6. This figure focuses upon the architectures that gave the lowest test MSEs, which contained between 70 and 90 hidden neurons.

The best results are seen to be obtained with 6 sigmoid neurons and 80 hidden neurons, i.e. 6 sigmoid neurons and 74 RBF neurons. The test error obtained with this

architecture is 0.00952.

A comparison between the results of the two-step and three-step algorithms shows that the introduction of gradient descent leads to further improvements in performance as well as a substantial reduction in network size. The good performance of GL-ANNs may be attributed in part to the hybrid architecture, but also in part to the hybrid algorithm. For the CLASH dataset it appears that a combination of the deterministic method of FS-OLS and the more stochastic process of gradient descent leads to effective generalisation. When compared with the results obtained using RBF networks optimised with gradient descent (section 5.3.3), there is seen to be only a small reduction in error upon using the hybrid architecture. This suggests that the hybrid training method accounts for most of the improvement in the performance of three-step GL-ANNs, with the hybrid architecture playing a lesser role.

The errors obtained with three-step GL-ANNs are almost as low as those obtained when pure RBF networks are trained with regularisation. The effect of regularisation on hybrid networks is reported in the next section.

7.2.3 Hybrid networks trained with regularisation

Investigation of the effect of regularisation on hybrid networks focused upon the architectures most likely to yield effective networks, i.e. those with 6 sigmoid neurons and a RBF spread of 0.4. As with the training of pure RBF networks with regularisation, networks were originally trained with up to 200 RBF neurons. The results are shown in figure 7.7. As with pure RBF networks, the best results are obtained with $\lambda = 10^{-4}$, and again the verification errors are still seen to be falling after the addition of 200 neurons (compare section 5.3.2). As with pure RBF networks, training was continued until 450 RBF neurons had been added, using the optimum regularisation parameter, i.e. 10^{-4} .

The minimum verification MSE was achieved with 439 RBF neurons, equivalent to a total of 445 hidden neurons, and the test error, averaged across 30 networks, was 0.00936. The results obtained when each data split is allowed to ‘choose’ its own preferred architecture (number of RBF neurons) are given in table 7.2. The results are very close to those achieved with pure RBF networks. Since the optimum architectures contain very large numbers of RBF networks, they dominate the networks and the sigmoid neurons have little effect on the network size or the generalisation ability of the networks. The combination of regularisation and hybrid networks does not therefore appear to be a useful technique.

Data split number	Verification error	Test error	Optimum number of RBF neurons
1	0.00856	0.00779	448
2	0.00909	0.00985	445
3	0.00886	0.00890	400
4	0.00900	0.00833	302
5	0.00726	0.00985	337
6	0.00884	0.00920	420
7	0.00827	0.01027	385
8	0.00887	0.01011	438
9	0.00863	0.01020	346
10	0.00865	0.00813	373
11	0.00881	0.01138	391
12	0.00918	0.00894	334
13	0.00941	0.01059	445
14	0.01113	0.00934	450
15	0.00768	0.00843	345
16	0.00889	0.00854	400
17	0.00777	0.00905	348
18	0.01021	0.00791	309
19	0.00970	0.00892	450
20	0.00803	0.00792	441
21	0.00942	0.00927	449
22	0.00971	0.00850	338
23	0.00787	0.00997	324
24	0.00938	0.00837	301
25	0.00811	0.00908	291
26	0.00800	0.00954	418
27	0.00745	0.01151	378
28	0.00904	0.00921	430
29	0.01013	0.00723	439
30	0.00714	0.01290	450
Average	0.00877	0.00931	387.5

Table 7.2: Optimum errors for hybrid networks with spread=0.4 containing 6 sigmoid neurons trained with regularisation

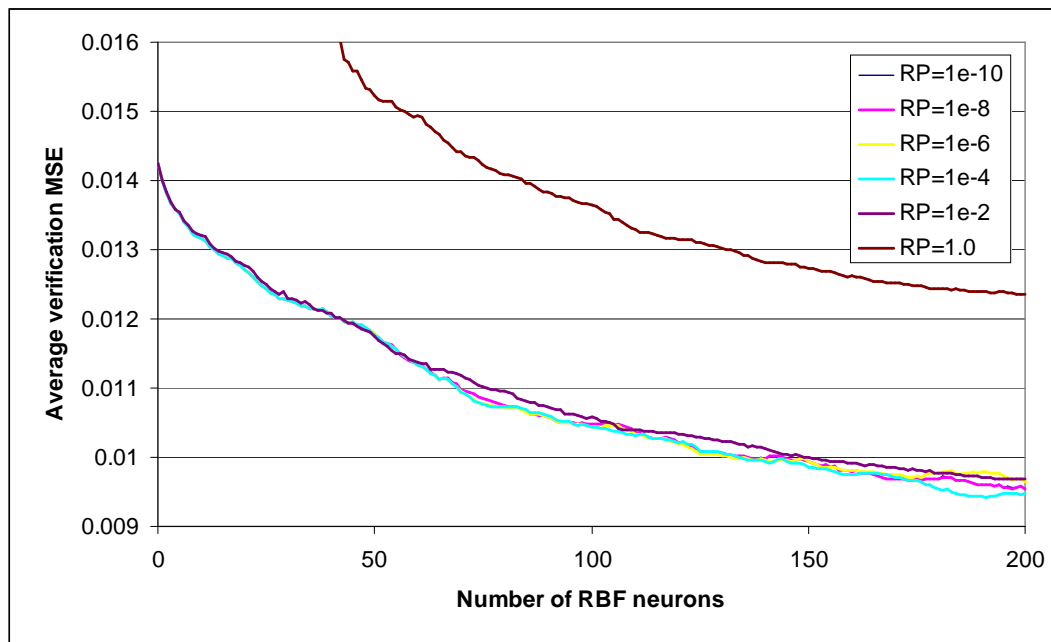


Figure 7.7: Verification errors for hybrid networks trained with regularisation

The effect of regularisation may be compared with that of gradient descent optimisation. Both techniques aim to improve upon the results obtained using the basic FS-OLS algorithm, and the best MSEs obtained using the two techniques are similar: 0.00936 with regularisation and 0.00952 with gradient descent optimisation. The most significant difference between the two techniques is the size of the networks created. The best results are obtained with regularisation by increasing the size of the networks (compared to the optimum size without regularisation). On the other hand, gradient descent optimisation appears to favour much smaller networks.

Examination of the network weights between the hidden and output layer suggests that GL-ANNs automatically incorporate a degree of regularisation. The average weight between the RBF neurons and the output neuron in an optimally sized pure unregularised RBF network is 18.8. The corresponding value for the RBF neurons in the most effective two-step GL-ANNs is 4.00 and for three-step GL-ANNs is just 1.29. Since the *modus operandi* of the regularisation procedure is to reduce the size of the hidden-output weights it appears that regularisation is not needed for GL-ANNs. This observation may be explained by considering the process of function-fitting. When RBF neurons are added to a hybrid network, an approximate input-output function is already simulated within the network via the sigmoid neurons. The difference between this approximate function and the ‘true’ function is fairly small and it is this difference

that the RBF neurons are intended to approximate. Since the RBF neurons have a relatively minor role in reducing the MSE they are likely to be assigned small weights by the FS-OLS algorithm which always sets the hidden-output weights to produce the lowest possible MSE.¹

7.2.4 Speed and memory comparisons

The FS-OLS procedure used to create two-step GL-ANNs and regularised hybrid networks is the same as that used to build pure RBF networks and therefore runs at the same speed. However, the required size of the networks is much smaller, if gradient descent optimisation is to be performed. To create a series of hybrid networks containing up to 100 RBF networks takes approximately $3\frac{1}{2}$ minutes when running on a PC containing an AMD Athlon 2100+ chip with a clock-speed of 1.74 GHz. This compares favourably with the 23 minutes required to produce a series of pure RBF networks containing up to 250 neurons.

The gradient descent step is much slower. To optimise a network containing 80 hidden neurons takes 23 minutes. It is therefore necessary to be selective in choosing which networks to optimise, as described in section 7.2.2. In the future it might be wise to replace the L-M algorithm with the conjugate gradient algorithm (Appendix B), which has much lower computational cost.

Both the L-M and FS-OLS procedures have substantial memory requirements, but these never exceed 100 MB and do not therefore present a difficulty to a modern computer.

7.3 Summary

This section aims to sum up all of the research involving the use of ANNs with the CLASH dataset, including the results from Chapters 4 and 5 as well as this chapter. Comparisons are also made with traditional methods of predicting overtopping rates.

Table 7.3 summarises the results obtained using MLP, RBF and hybrid networks. Also included are the best results from training RBF networks with regularisation and with gradient descent optimisation. In addition to the average normalised test MSEs, the average error factor and average absolute error are given. Both of these values

¹Orr has observed that regularisation is generally not useful when adding neurons with narrower spreads to those with wider spreads [211]. This observation is similar to that made here concerning the addition of RBF neurons to hybrid networks.

Network type	Normalised MSE	Error factor	Absolute error
MLP (linear output)	0.01168	6.99	2.54×10^{-3}
MLP (sigmoid output)	0.01114	8.93	1.52×10^{-3}
RBF	0.01060	5.61	1.29×10^{-3}
RBF with regularisation	0.00930	6.03	1.15×10^{-3}
RBF with gradient descent	0.00956	6.15	1.37×10^{-3}
GL-ANN	0.00952	5.59	1.51×10^{-3}

Table 7.3: Performance indicators for MLP, RBF and GL-ANN networks, averaged across 30 networks

are obtained using the de-normalised values of q_0 , i.e. the normalisation process described in section 3.2 has been performed in reverse. The error factor, EF, is defined by equation 7.1.

$$EF = \begin{cases} q_{0,predicted}/q_{0,target} & \text{if } q_{0,predicted} > q_{0,target} \\ q_{0,target}/q_{0,predicted} & \text{if } q_{0,predicted} \leq q_{0,target} \end{cases} \quad (7.1)$$

It is interesting to note that the ‘best’ architecture, determined from the average normalised test MSE, does not correspond to the most effective architecture on all performance measures. RBF and GL-ANN networks out-perform MLP networks on all measures, but the relative performance of RBF networks and GL-ANN networks vary according to performance measure. RBF networks give a lower average error, but GL-ANNs give lower normalised MSEs and lower error factors.

With the MLP networks, a sigmoid output neuron performs best on the first two measures, whereas a linear output neuron gives a lower error factor. The introduction of regularisation or gradient descent optimisation into RBF training reduces the measured MSE, but increases the error factor.

In order to make comparisons with traditional methods of predicting overtopping rates and in an attempt to analyse the results in more detail, a single network was chosen for each architecture. In each case this was the network which gave the lowest normalised test MSE. The results for these individual networks are given in table 7.4.

Also included are the equivalent figures using numerical simulation. These results are taken from Hu *et al* [28]. They used a high-resolution, finite-volume model to solve the non-linear shallow water equations (see section 1.4). Data used included 40 items of laboratory test data using regular waves overtopping smooth walls with slopes between 1:3 and 1:5. Also included were 11 tests using a laboratory test model of Great Yarmouth outer harbour, under varying sea conditions. The data used by Hu is taken

Network type	Normalised MSE	Average error	Error factor
MLP	0.00842	1.49×10^{-3}	2.60
RBF	0.00871	1.10×10^{-3}	2.75
GL-ANN	0.00739	0.92×10^{-3}	2.48
Numerical modelling	-	2.29×10^{-3}	1.89

Table 7.4: Performance indicators for the best performing MLP, RBF and GL-ANN networks

from a narrower range of structures and sea conditions than the CLASH data. However, their work is representative of the type of approach used in numerical modelling, and is provided here as a guide to the accuracy attainable with such methods.

When assessed in terms of average error, the ANNs perform better than numerical simulation, even though they have a much wider range of applicability. When the error factors are considered, numerical simulation out-performs the ANNs, with the GL-ANN coming closest to the simulated results in accuracy.

Figures 7.8-7.10 show the ratio $q_{0,predicted}/q_{0,target}$ for the individual data items within the CLASH dataset for the best performing individual ANNs with, respectively, MLP, RBF and GL-ANN architectures. As expected from the error factors in table 7.4, the GL-ANN gives results which lie closest to ideal, i.e. $q_{0,predicted}/q_{0,target} = 1$, although the difference in error factors between the networks is small.

All three networks show a tendency to overpredict low overtopping rates and to underpredict high overtopping rates. This may be partly a result of the ANN technique, which favours areas in which data density is high. In areas of low data density, the results are therefore likely to be less accurate. There may be a second factor operating at low overtopping rates. When overtopping rates are low, it is difficult to obtain accurate measurements, so the training data is likely to show a high degree of variability in this region.

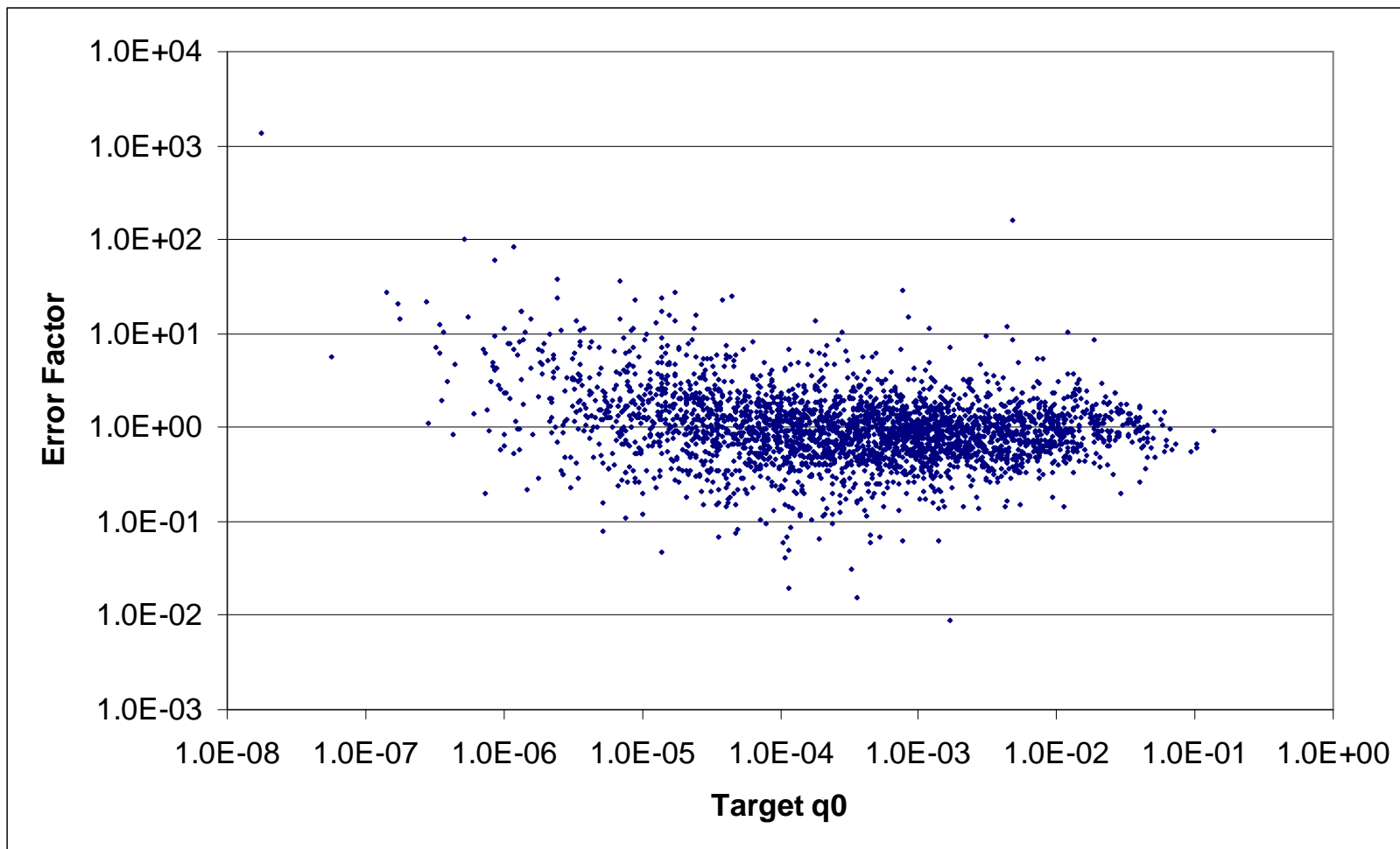


Figure 7.8: $q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing MLP network

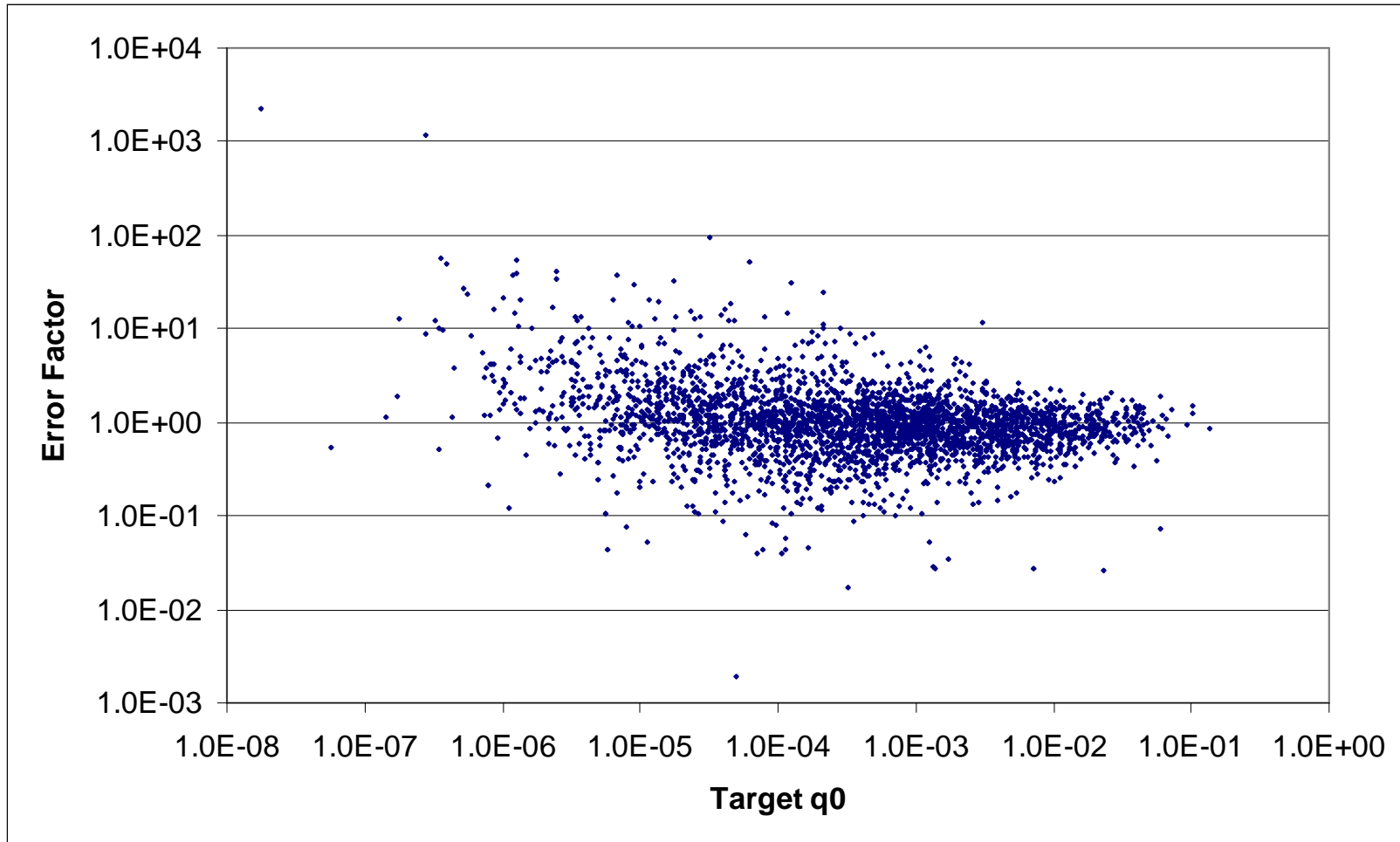


Figure 7.9: $q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing RBF network

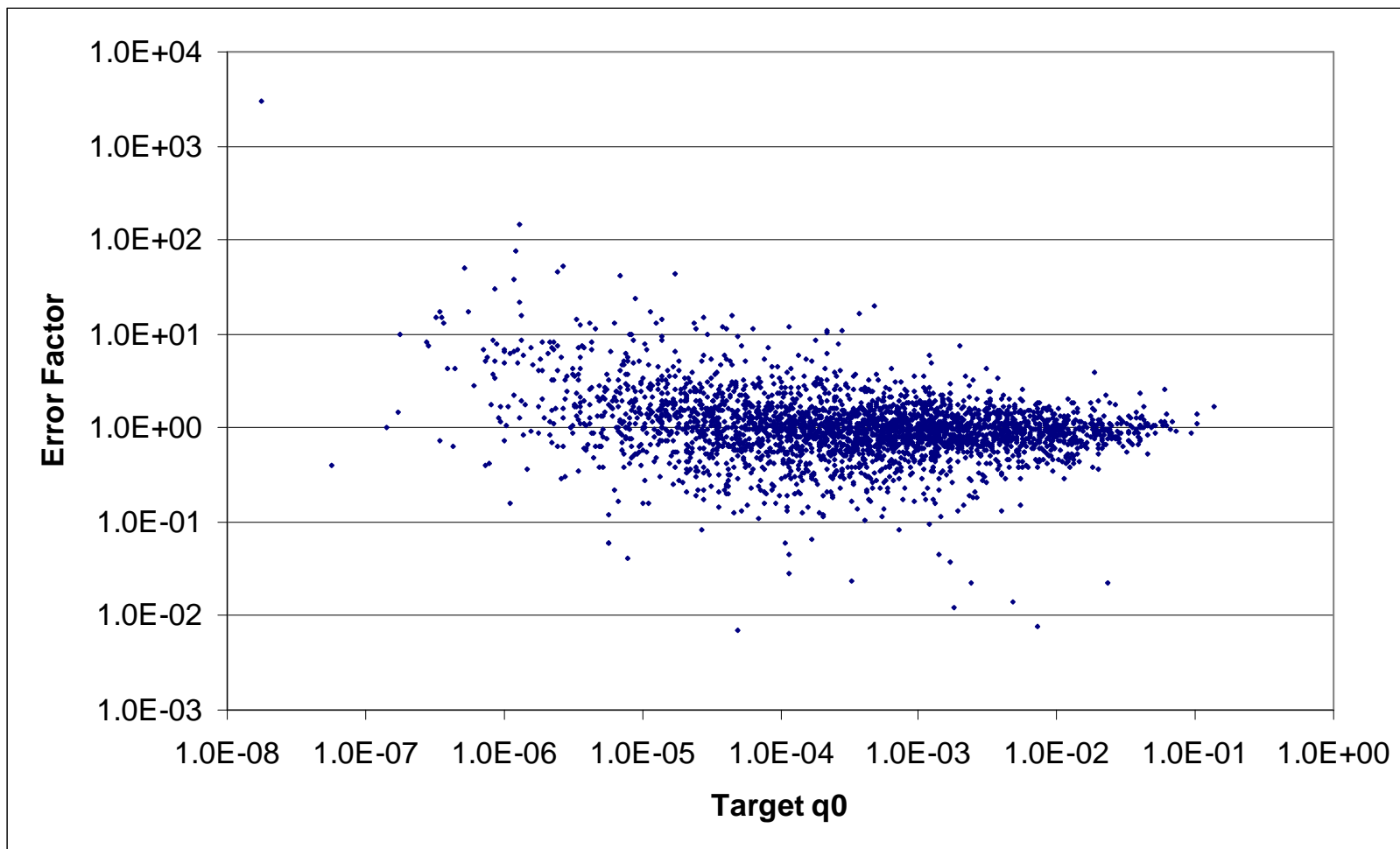


Figure 7.10: $q_{0,predicted}/q_{0,target}$ vs. target q_0 for the best-performing GL-ANN network

Chapter 8

GL-ANN Evaluation using Benchmark Datasets

This chapter introduces a number of benchmark datasets that have been used to assess the GL-ANN architecture and algorithm. The first aim in using these datasets is to find out whether the GL-ANN method is especially useful when applied to certain types of data. On the other hand, the use of benchmark datasets may identify types of dataset for which GL-ANNs are not a useful tool.

The second aim is to find out about the nature of the architectures created by the GL-ANN process. Architectures are differentiated in terms of number of hidden neurons and optimum RBF spread. Comparisons are made with the corresponding MLP and RBF networks.

The datasets used are all readily available and they have therefore been used previously by other researchers. In many cases it is therefore also possible to compare the results of the GL-ANN method with additional techniques such as regression trees.

Section 8.1 describes the datasets, using tools to assess the linearity and clustering behaviour of the datasets. Section 8.2 explains the methods used to train various types of network using these datasets. Section 8.3 reports the results of this training. Section 8.4 summarises this chapter.

8.1 Description of the benchmark datasets

The datasets used for benchmarking fall into two categories: synthetic and measured. Each of the synthetic datasets is generated using some mathematical function, usually with random noise added to the output. They have two main advantages. Firstly

the amount of noise may be easily altered, allowing the effect of noise on different approaches to be compared. Secondly an exact solution is available, since ‘clean’ samples are easily generated. The extent to which different techniques approach a ‘correct’ solution is therefore easily assessed.

The measured datasets have been obtained from a number of ‘real’ scenarios. For this reason they contain unknown quantities of noise and the concept of a ‘correct’ solution is less easily defined. However, they represent more realistic problems than the synthetic datasets. They contain data of higher dimensionality and the regression solution is generally more complex in mathematical form than for the synthetic data.

8.1.1 Synthetic Datasets

Four synthetic benchmark datasets were employed. As with the CLASH dataset, the target values are real-valued. The aim of ANN training is therefore to achieve accurate function approximation in each case. The tests are taken from Cohen and Intrator’s 2002 paper[107], in which comparisons are made with a number of other approaches. For this reason the treatment varies between the different datasets. While this creates some inconsistency it allows the consideration of a variety of datasets and permits comparison with a number of alternative methods. The approach used by Cohen and Intrator is to use separate training and test sets, but no verification set. This makes it necessary to use the test set both to identify the optimum architecture and as the final performance measure. In order to be consistent with earlier work, the same procedure is followed in this study.

The first function is the 1-D sine wave of equation 8.1, with x randomly selected from $[0, 1]$ and $f(x)$ corrupted by Gaussian noise with standard deviation (s.d.) of 0.1 and a mean of 0. The training and test sets both contain 50 samples[176]. Given the noisy data, there is a theoretical minimum value for the test MSE, equal to 0.01.

$$f(x) = \sin(12x) \quad (8.1)$$

The second function is the 2D sine wave of equation 8.2, with $x_1 = [0, 10]$ and $x_2 = [-5, 5]$. The training data is made up of 200 randomly selected items, again corrupted with Gaussian noise of standard deviation 0.1 and mean 0. However, clean data is used for testing purposes, arranged in a 20 by 20 grid to cover the entire input space. The test set therefore contains 400 data items.

$$f(x) = 0.8 \sin\left(\frac{x_1}{4}\right) \sin\left(\frac{x_2}{2}\right) \quad (8.2)$$

The third function is a simulated alternating current used by Friedman in the evaluation of multivariate adaptive regression splines (MARS) [212]. It is given by equation 8.3, in which Z is the impedance, R the resistance, ω the angular frequency, L the inductance and C the capacitance of the circuit. The input ranges are $R = [0, 100]$, $\omega = [40\pi, 560\pi]$, $L = [0, 1]$ and $C = [1 \times 10^{-6}, 11 \times 10^{-6}]$. 200 random samples, with Gaussian noise of standard deviation 175 and zero mean applied to Z , are used for training. 5000 random clean samples are used for testing.

$$Z(R, \omega, L, C) = \sqrt{R^2 + (\omega L - 1/\omega C)^2} \quad (8.3)$$

The fourth function is the Hermite polynomial of equation 8.4, with x randomly selected from $w = [-4, 4]$. 100 random samples corrupted by Gaussian noise of standard deviation 0.1 and zero mean are used for training purposes. 100 clean samples are used for testing. This function was first used by Mackay[94].

$$f(x) = 1 + (1 - x + 2x^2)e^{-x^2} \quad (8.4)$$

The synthetic datasets have been analysed using two tools, k-nearest neighbour data density estimates and linear regression analysis. Both tools are described in detail in section 3.3.

The data density estimates for the four datasets are shown in figures 8.1-8.4. The data densities for the sine 2D dataset display a sharp peak. This indicates a homogeneous distribution that is likely to favour MLP networks. The Hermite dataset, in contrast, shows a wide distribution of data densities, indicating substantial clustering. This dataset would be expected to perform better with RBF networks. The remaining two datasets show a moderate variation in data densities. The interquartile ranges of the data densities are 1.6 for the sine 1D dataset and 1.4 for the impedance dataset. These values are just above the boundary of 1.2 used by Lawrence[183] and indicate a slight preference for RBF networks.

Figures 8.5-8.8 present an analysis of linear regression. They plot studentised residuals against estimates of the target outputs in order to give an indication of the linearity, or non-linearity of the datasets. Figure 8.6 suggests that the sine 2D dataset may be partially fitted by linear regression, with some deviation from linearity. The graphs for all of the other datasets indicate that the size of the residuals depends strongly upon

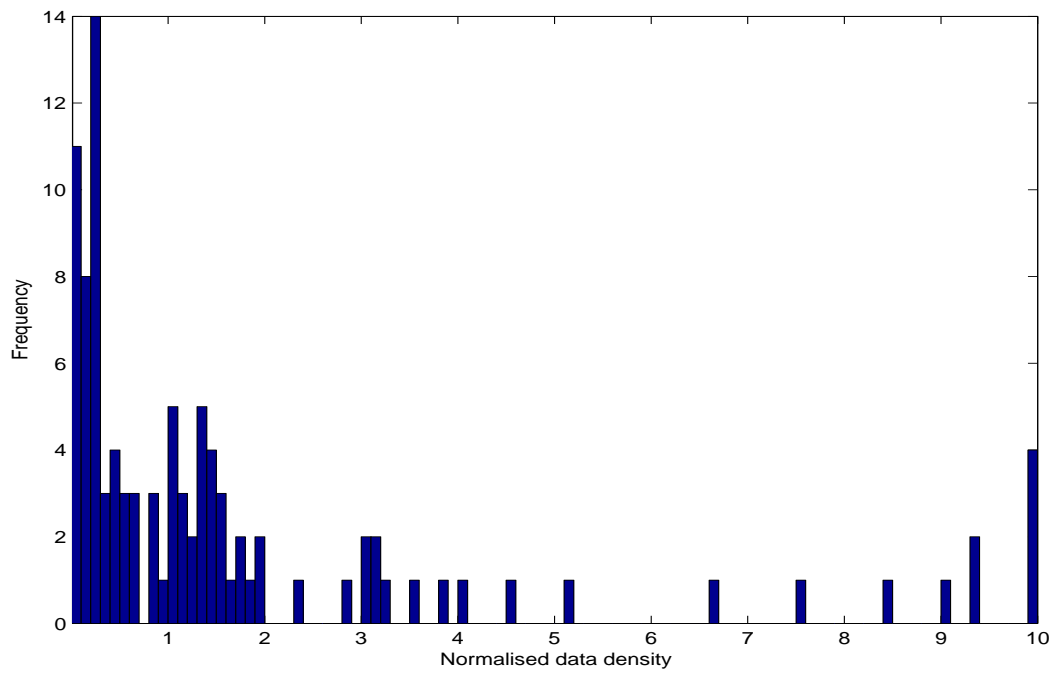


Figure 8.1: Data densities for the sine 1D dataset

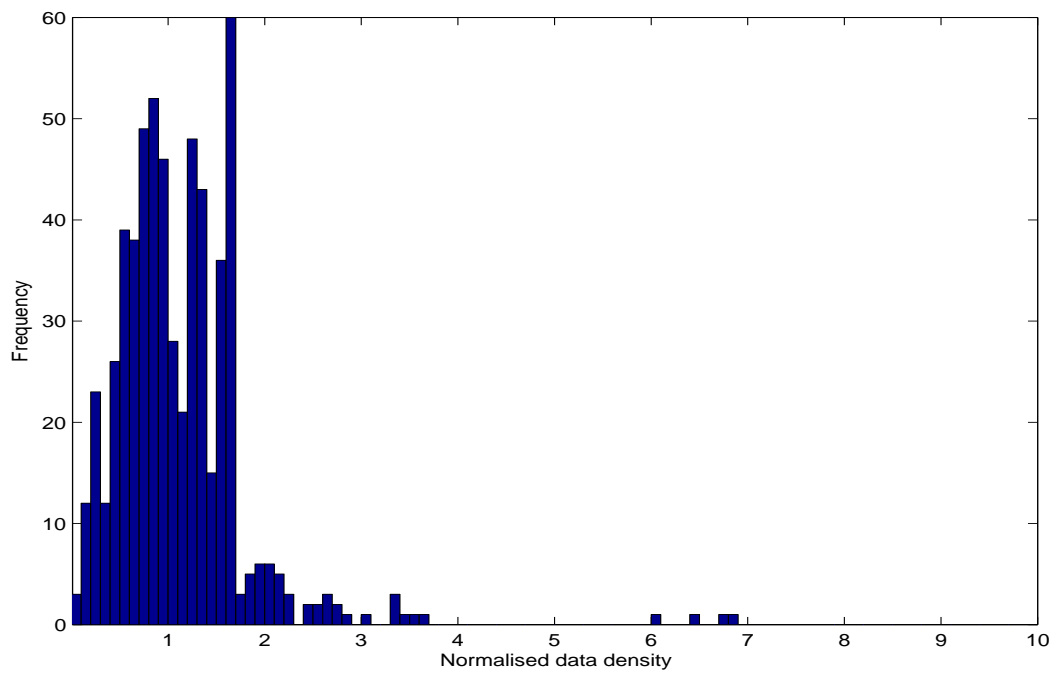


Figure 8.2: Data densities for the sine 2D dataset

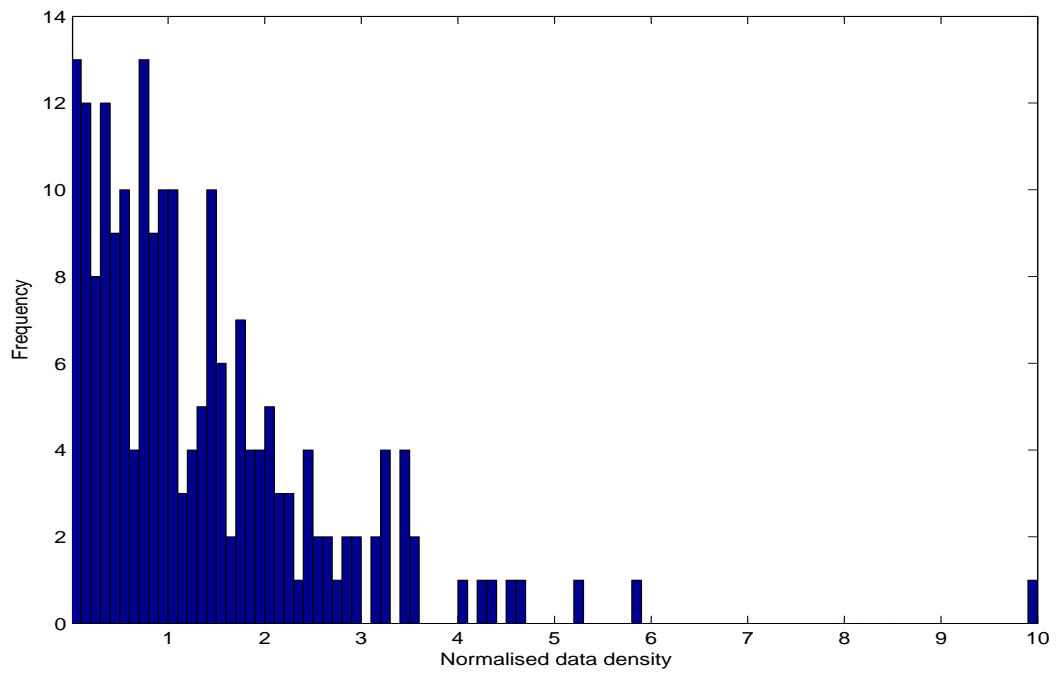


Figure 8.3: Data densities for the impedance dataset

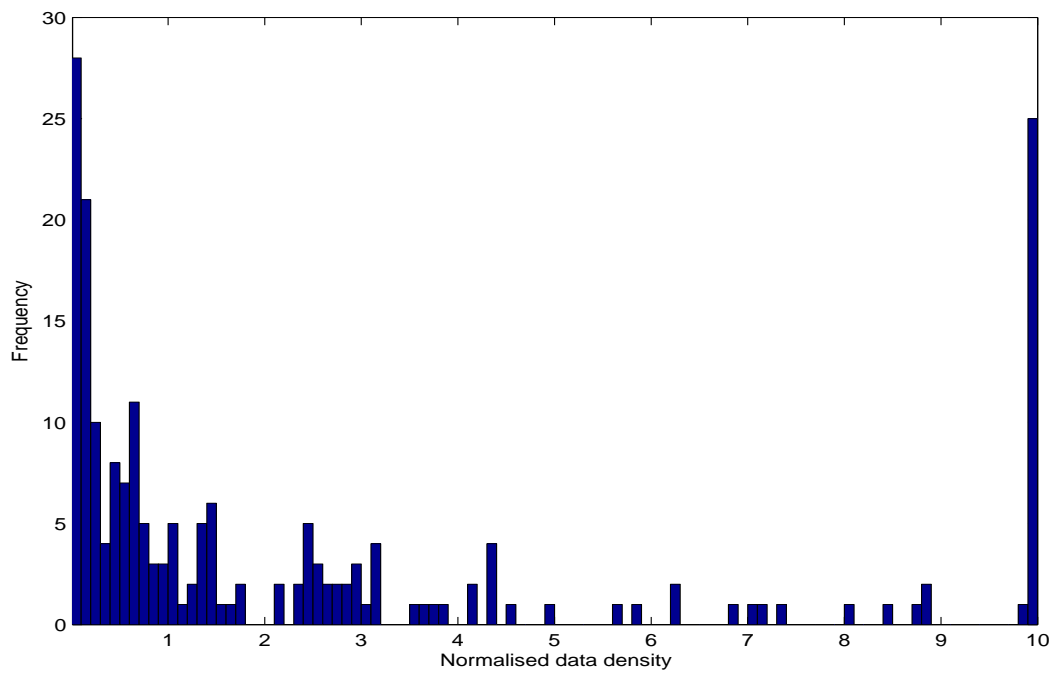


Figure 8.4: Data densities for the Hermite dataset

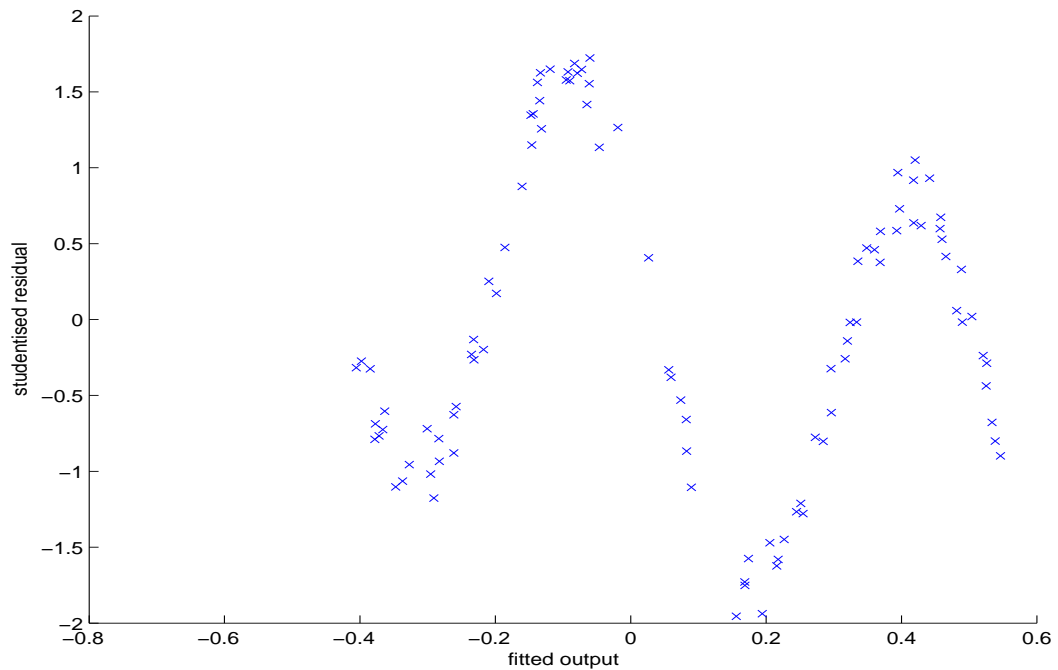


Figure 8.5: Plot of studentised residuals vs. estimated q_0 for the sine 1D dataset

the output value, and that the input-output relationship is therefore highly non-linear. This is confirmed by the R^2 values given in table 8.1. This table summarises various properties of the synthetic datasets used, including interquartile ranges (IQRs) of the data densities and R^2 statistics from regression analysis.

8.1.2 Measured Datasets

The measured datasets were all obtained from the University of California, Irvine (UCI). This university maintains a substantial number of datasets in a ‘machine learning repository’. These datasets are useful because they present non-linear, noisy data that are suitable for machine learning tasks. They may be downloaded from the Internet and have been widely used as benchmark tests when assessing ANNs and other machine learning techniques.

The first dataset, ‘housing’, is a compilation of house prices and factors that may affect these prices. The task is to predict median house prices of suburbs in the Boston area from the values of 13 independent variables, including the following:

- per capita crime rate
- proportion of residential land zoned for lots over 25,000 ft^2

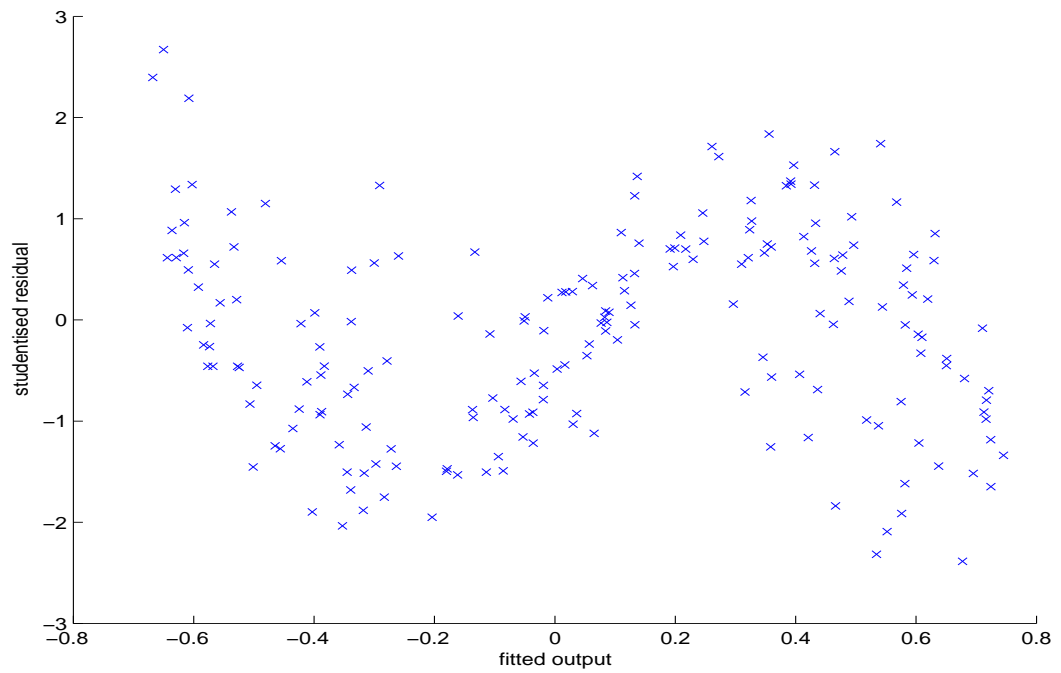


Figure 8.6: Plot of studentised residuals vs. estimated q_0 for the sine 2D dataset

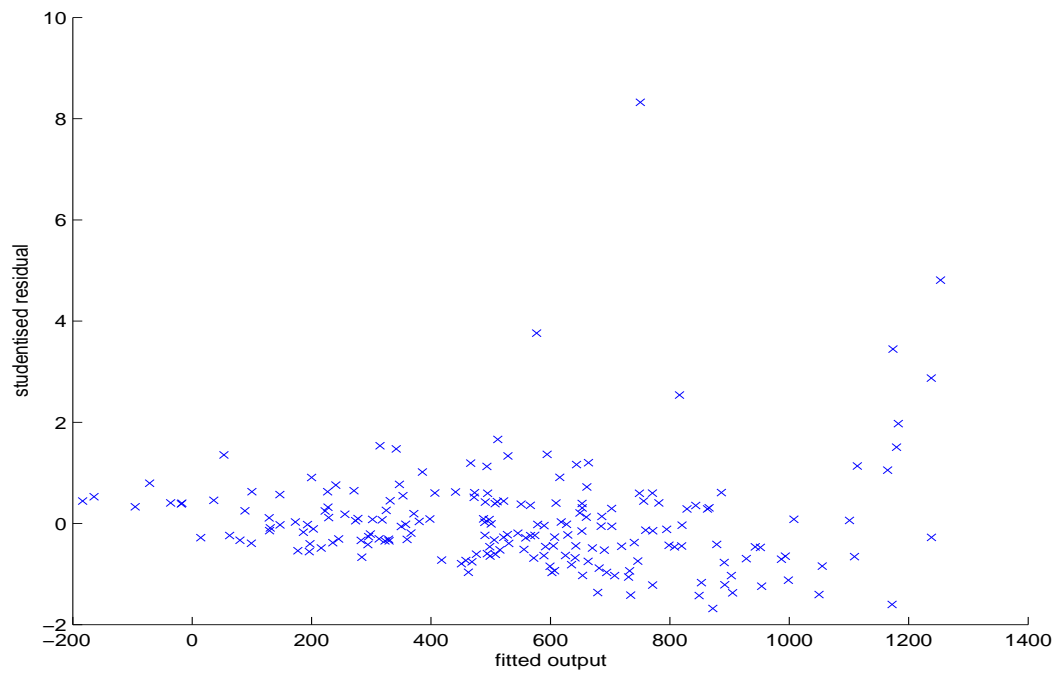


Figure 8.7: Plot of studentised residuals vs. estimated q_0 for the impedance dataset

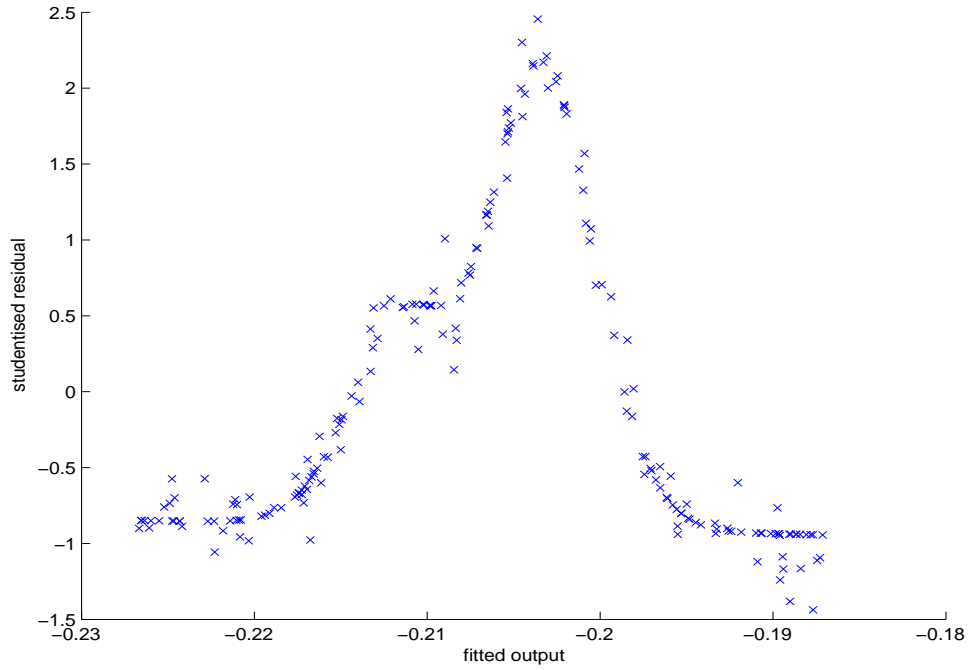
Figure 8.8: Plot of studentised residuals vs. estimated q_0 for the Hermite dataset

Table 8.1: Summary of the synthetic benchmark datasets

Name	Sine 1D	Sine 2D	impedance	Hermite
Source	Orr/Cohen	Orr/Cohen	Friedman/Cohen	Mackay/Cohen
Number of inputs	1	2	4	1
Training data size	50	200	200	100
Test data size	50	400	5000	100
Training noise s.d.	0.1	0.1	175	0.1
Test noise s.d.	0.1	0.0	0.0	0.0
IQR of density	1.6	0.76	1.4	3.4
R^2 value	0.17	0.71	0.19	0.001

- nitric oxide concentrations (in parts per million)
- pupil-teacher ratio
- percentage of the population of 'lower status'
- weighted distances to five employment centres
- average number of rooms per dwelling
- proportion of homes built before 1940

It contains 506 data items, and was first used by Harrison and Rubinfeld[213] in 1978.

The second dataset, 'servo', was first used by Quinlan[214] in 1992. It aims to predict the rise time of a servomechanism as a function of two gain settings and two discrete choices of mechanical linkages. There are therefore 4 independent variables in total, and the dataset contains 167 instances.

The third dataset, 'cpu', contains the following information for a number of computers: vendor name, machine cycle time, minimum and maximum main memory size, cache memory, and minimum and maximum channels. From this information the task is to predict the published relative performance. The size of the dataset is 209, and for the purposes of this study the vendor name was not used, so the number of inputs was 6. This dataset was first used by Ein-Dor and Feldmesser in 1987[215].

The final dataset, 'auto-mpg' contains information concerning car models including number of cylinders, year of manufacture, horsepower, weight and acceleration. From these parameters the aim is to predict the petrol consumption, in miles per gallon. Like the 'servo' dataset, it was first used by Quinlan[214].

Figures 8.9-8.12 show density distributions for the measured benchmark datasets. The housing and cpu datasets exhibit considerable clustering behaviour, with a wide range in data densities. On the other hand, the servo dataset is spread more evenly and shows a sharp peak around a normalised density of 1. The auto-mpg dataset is intermediate, showing some variation in data densities. The interquartile ranges of the data densities for each dataset are given in table 8.2, along with other summary statistics for the measured datasets.

Figures 8.13-8.16 illustrate the degree of linearity of the measured datasets. Table 8.2 includes R^2 values, which also give an indication of the degree of linearity. Of the four datasets, only 'servo' has a R^2 value considerably less than 1. Study of figure 8.14

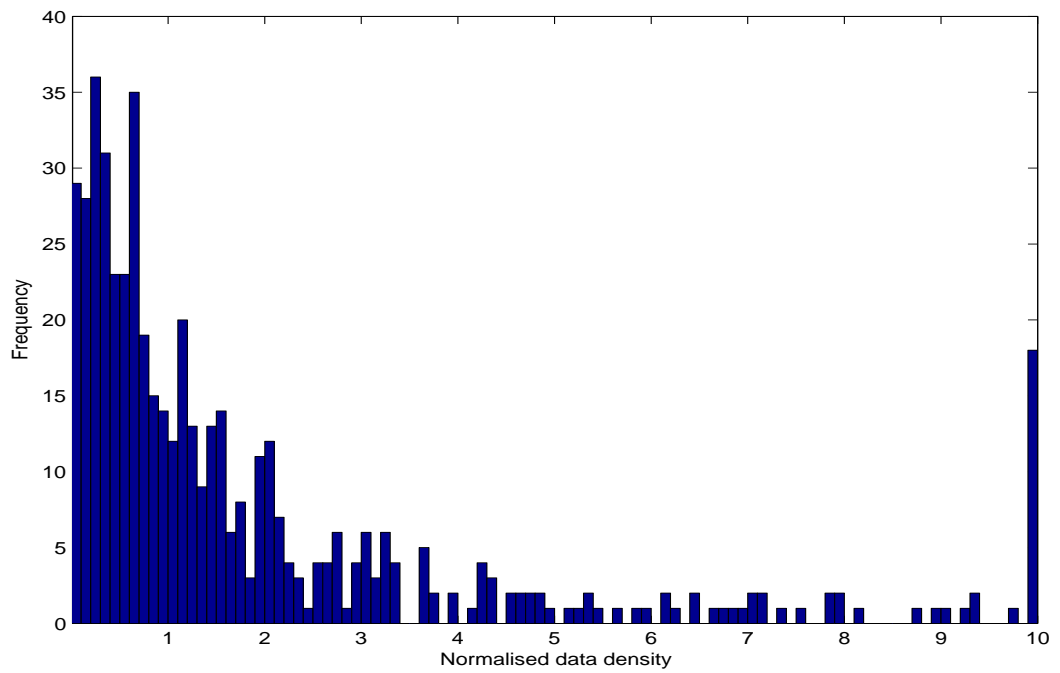


Figure 8.9: Data densities for the housing dataset

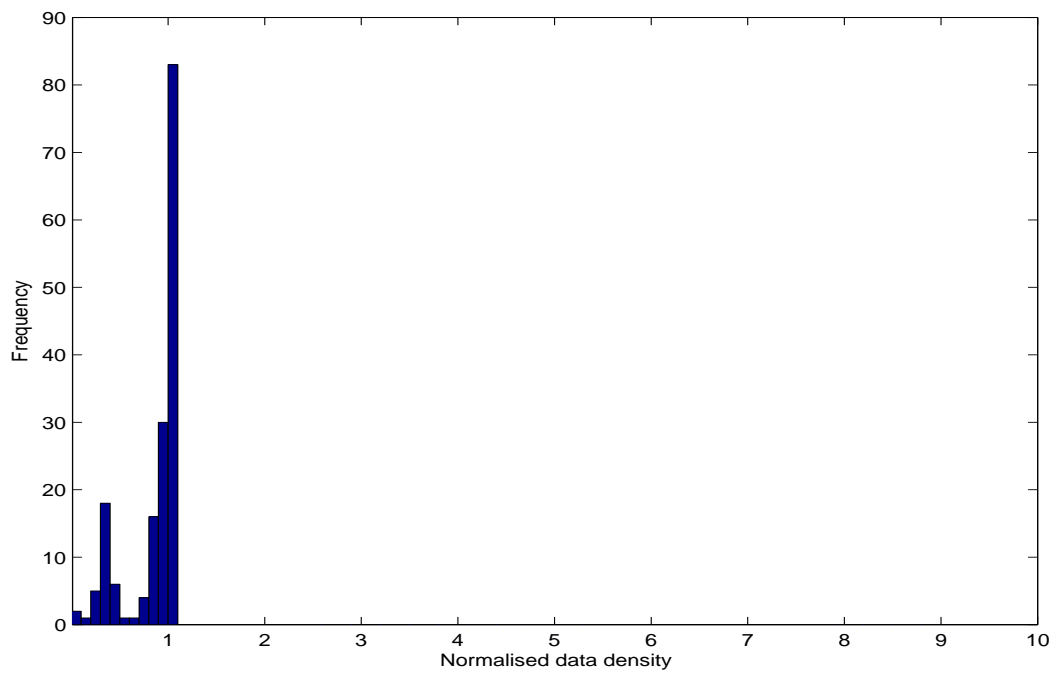


Figure 8.10: Data densities for the servo dataset

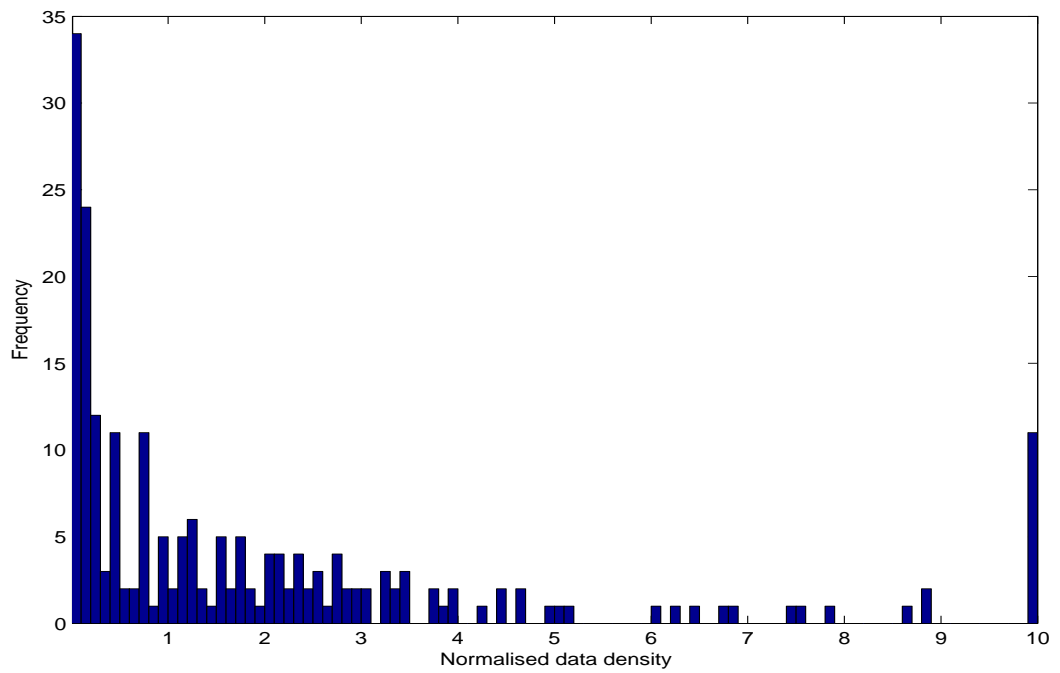


Figure 8.11: Data densities for the cpu dataset

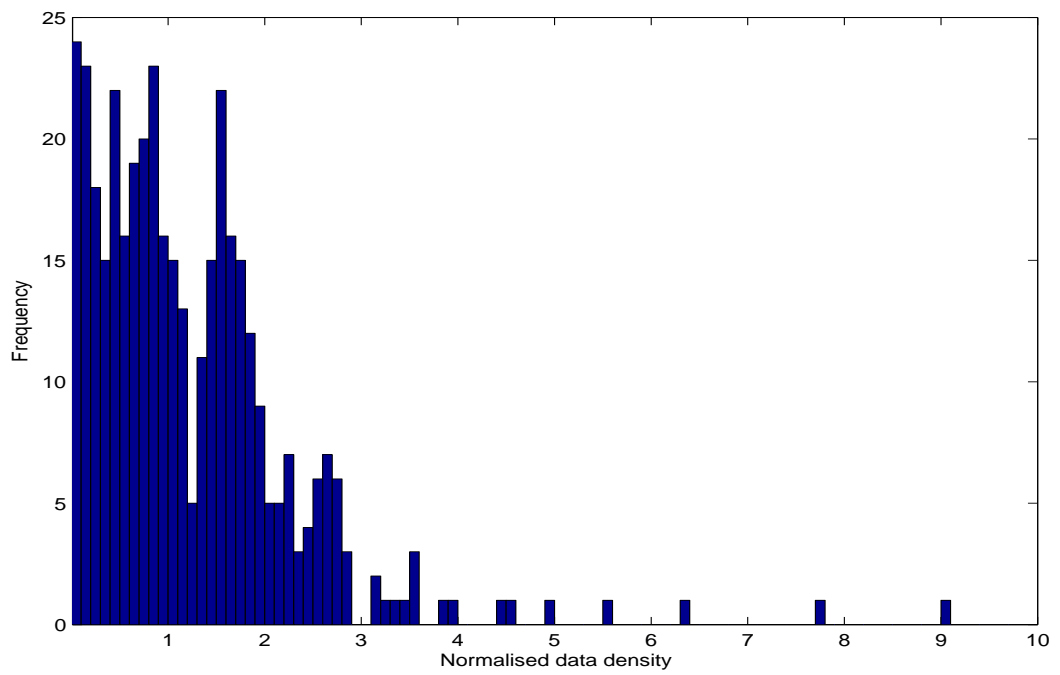


Figure 8.12: Data densities for the auto-mpg dataset

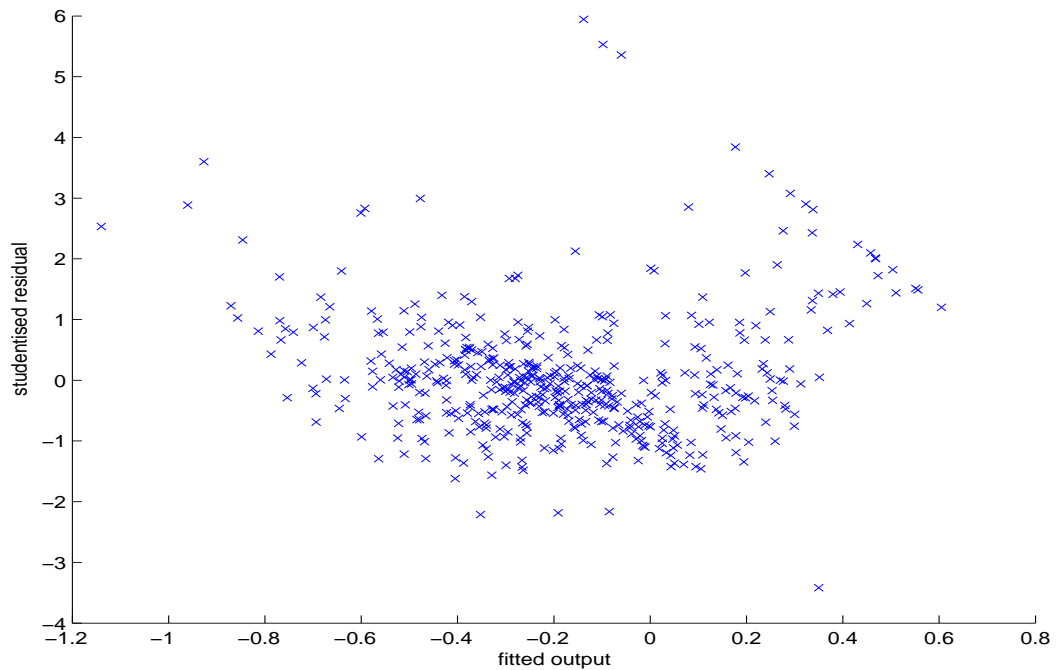


Figure 8.13: Plot of studentised residuals vs. estimated q_0 for the housing dataset

shows that there are clusters of data which deviate considerably from linear behaviour for this dataset. This observation is interesting since it appears to contradict the findings of the data density analysis. While data density analysis focusses primarily on distribution of the input data, regression analysis considers the relationship between the input and output values. The servo dataset has evenly distributed inputs, but the responses produced behave in a highly non-linear way.

Table 8.2 shows that all of the measured datasets have high dimensionality. One can also assume that they have a fairly high noise level, since they are taken from ‘real’ situations which involve a large number of independent parameters, only some of which are included in the data representation used.

All of the datasets were split into equal sized training and test sets. In order to simplify interpretation, and to maintain a uniform approach between the synthetic and measured datasets, a verification set was not used.

8.2 Method

The datasets trained were the following-

- 1D sine

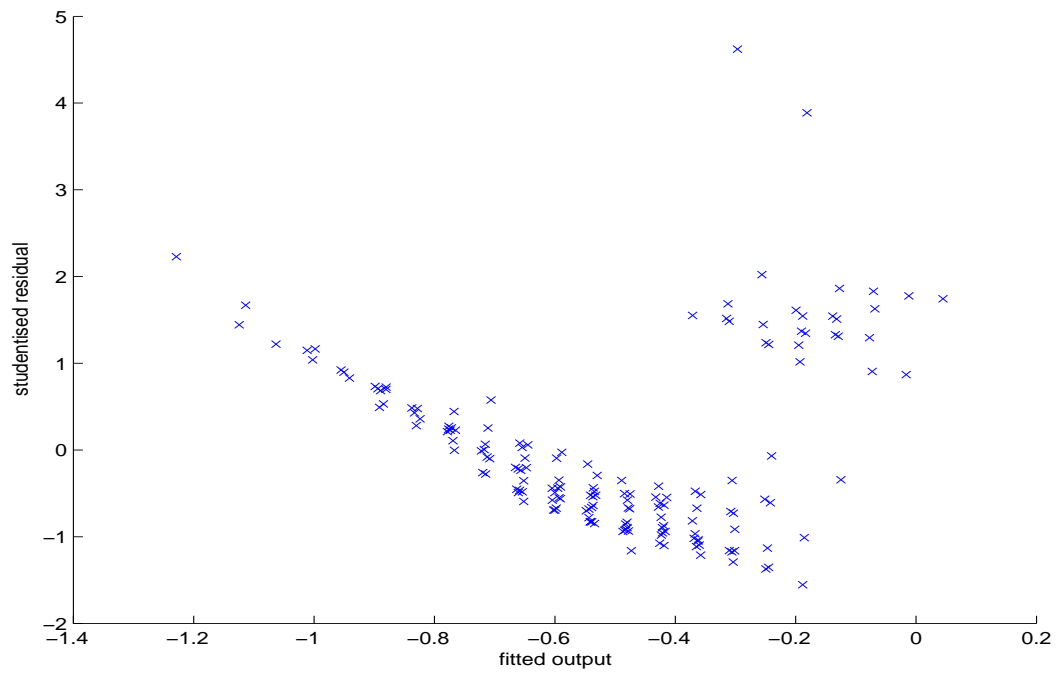


Figure 8.14: Plot of studentised residuals vs. estimated q_0 for the servo dataset

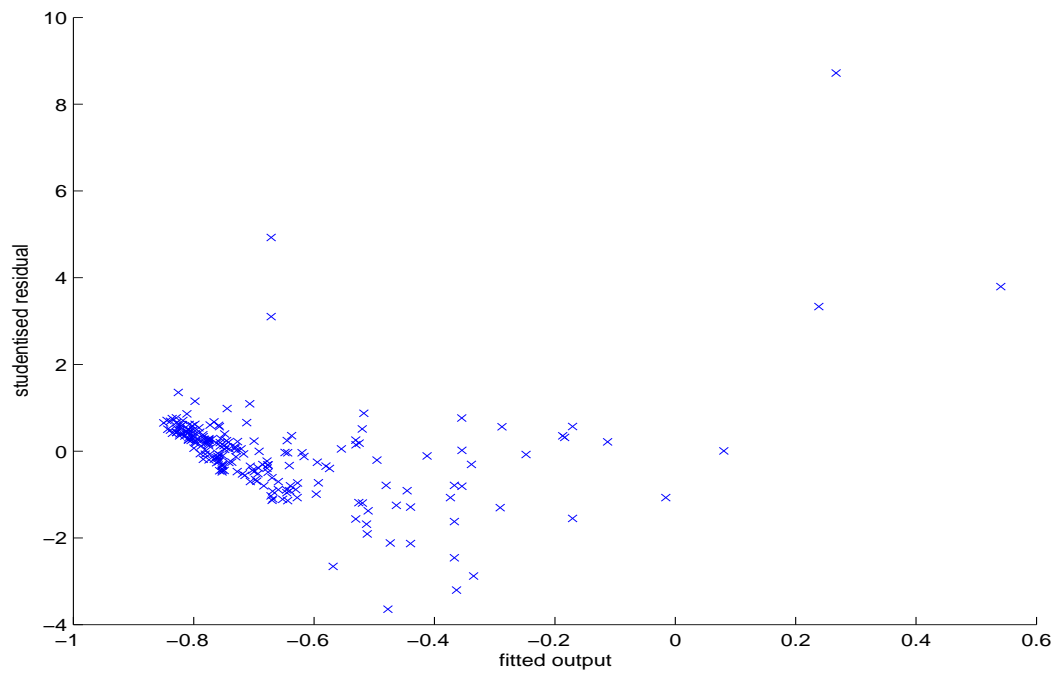


Figure 8.15: Plot of studentised residuals vs. estimated q_0 for the cpu dataset

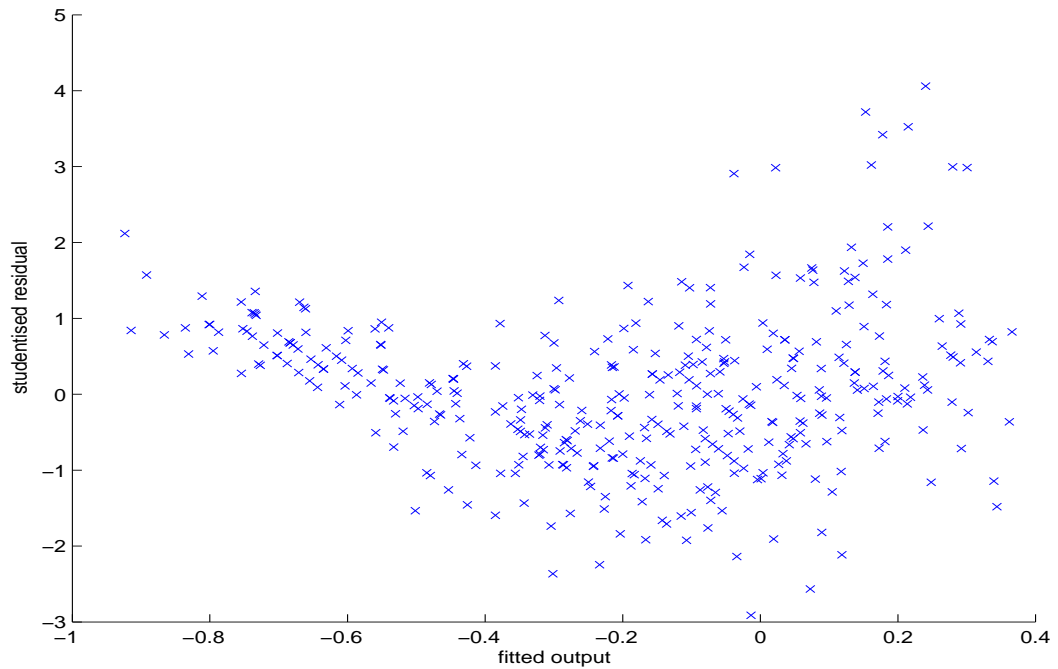


Figure 8.16: Plot of studentised residuals vs. estimated q_0 for the auto-mpg dataset

Table 8.2: Summary of the measured benchmark datasets

Name	housing	servo	cpu	auto-mpg
Source	UCI/Harrison	UCI/Quinlan	UCI/Ein-Dor	UCI/Quinlan
Number of inputs	13	4	6	7
Training data size	253	83	104	196
Test data size	253	84	105	196
IQR of density	1.8	0.15	2.6	1.2
R^2	0.74	0.50	0.86	0.82

- 2D sine
- impedance
- Hermite
- housing
- servo
- cpu
- auto-mpg

GL-ANNs were created using the three-step process described in Chapter 6.

In the first step MLP networks were trained with the L-M algorithm. All networks had bipolar sigmoid functions in the hidden layer and a linear function in the output layer. Initially, networks were trained containing between 1 and 10 hidden layer neurons, but in cases where the minimum error was achieved with 10 neurons, larger networks were also trained. In each case 10 different splits of the data were made.

In the second step up to 10 RBF neurons were added initially, and more were added if the results indicated that 10 RBF neurons gave the best results. Different widths of RBF were used. Widths between 0.2 and 1.0, in steps of 0.2, were tried first. If a width of 1.0 was seen to give the best results, greater spreads were tried.

In the third step, nearly all of the hybrid networks created were optimised using the Levenberg-Marquardt algorithm. An exception was made with the impedance dataset. The optimum sized hybrid networks were large for this dataset and gradient descent performed slowly. Only networks with a hidden layer size that was a multiple of 5 were therefore optimised.

When making comparisons with MLP networks, the networks produced by step 1 were considered. Separate RBF networks were created using the FS-OLS algorithm. Again networks with up to 10 RBF neurons were trained first and larger networks were only built if the lowest MSEs were achieved with 10 neurons. Similarly, networks with spreads between 0.2 and 1.0 were trained first, and larger spreads were used only if $s = 1.0$ gave the lowest MSEs.

The optimum networks were selected based on the lowest test MSEs, averaged over all 10 data-splits. Only fixed architectures were considered - datasets were not permitted to 'choose' the most favourable architecture individually.

Table 8.3: Mean square errors for the synthetic benchmark datasets

	1D sine	2D sine	impedance	Hermite
MLP	0.0175	0.00128	0.102	0.00214
RBF	0.0119	0.00095	0.152	0.00141
GL-ANN	0.0120	0.00111	0.098	0.00130

8.3 Results

8.3.1 Test errors

The best test MSEs obtained with the synthetic benchmark datasets are summarised in table 8.3. In the case of the ‘impedance’ dataset, the MSEs obtained have been divided by the variance of the test data. This practice was introduced by Friedman when he first used the dataset[216] and allows easier comparison with other methods.

The MSEs indicate that the GL-ANN algorithm is a useful tool for the impedance and Hermite datasets. These datasets have high dimensionality and are highly non-linear. The impedance dataset also has high noise levels. Both datasets display some level of clustering, particularly the Hermite dataset.

The 2-D sine function gives best results with a pure RBF network, while the 1-D sine function gives comparable results with pure RBF networks and GL-ANNs. The good performance of RBF networks in mapping the sine functions is perhaps unsurprising when one considers the similarity in shape between sine and Gaussian functions, illustrated in figure 8.17. In this graph the sine function has been translated to give a maximum at $x = 0$ and the width of the Gaussian function has been chosen such that the outputs of the functions coincide at $f(x)=0.5$.

The GL-ANN algorithm seems to find it difficult to approximate functions which are purely ‘radial’ in nature. The reason for this may be that the GL-ANN algorithm starts with a MLP containing sigmoid neurons. The function present within this network is likely to be an obstruction when radial functions are added in step 2.

On the other hand, functions which are very well described by MLP networks do not present a problem for GL-ANNs. In cases where radial functions can make little contribution the output weights from the RBF neurons will be set to low values by the FS-OLS algorithm, and it will be apparent that the addition of RBF neurons is not reducing the network error, so training will cease.

The degree of non-linearity within a dataset appears to be a good guide to the

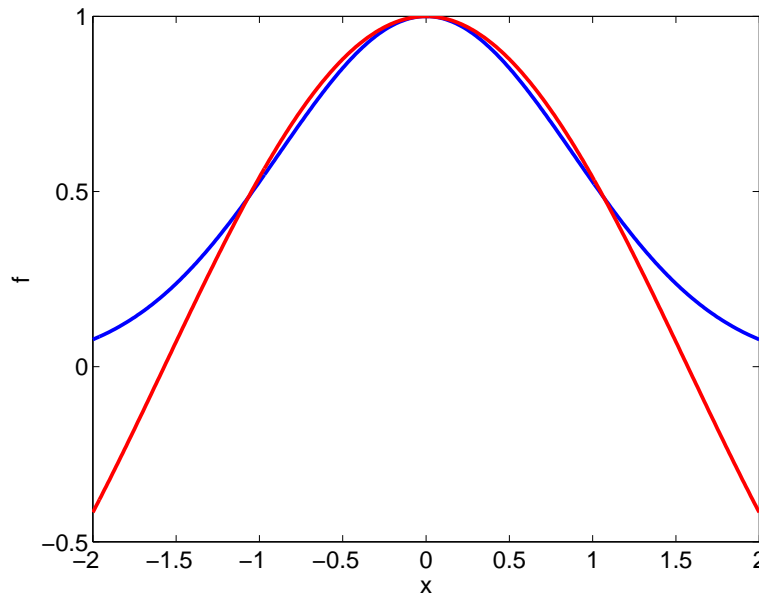


Figure 8.17: Graph showing a Gaussian function (blue) and a shifted sine curve (red)

Table 8.4: Mean square errors for the measured benchmark datasets

	housing	servo	cpu	auto-mpg
MLP	21.6	0.590	5561	8.54
RBF	15.2	0.664	3316	7.94
GL-ANN	15.9	0.512	5153	8.38

effectiveness of the GL-ANN algorithm. The sine 2D dataset is described quite well by a linear model ($R^2 = 0.71$) and does not perform well with GL-ANNs, whereas the remaining datasets have much lower R^2 values and perform relatively well with GL-ANNs.

Table 8.4 gives test MSEs for networks trained with the measured benchmark datasets. Two of the datasets, housing and cpu, give much better results with RBF than with MLP networks. This could have been predicted from the high interquartile range of the data densities, suggesting that the data is highly clustered. Neither of these datasets give particularly good results with the GL-ANN algorithm. This observation may be compared with that with the sinewave datasets: for datasets that are very well described by radial functions, the presence of sigmoid functions is an obstruction.

On the other hand, the GL-ANN algorithm gives good results for the servo dataset. This dataset gives slightly better results with MLP than RBF networks - again this would have been predicted from the interquartile range of data densities which is low,

Table 8.5: Number of hidden layer neurons for the synthetic benchmark datasets

	1D sine			2D sine			impedance			Hermite		
	S	R	T	S	R	T	S	R	T	S	R	T
MLP (L-M)	12	0	12	5	0	5	5	0	5	14	0	14
FS-OLS	0	6	6	0	13	13	0	48	48	0	7	7
GL-ANN	1	6	7	1	16	17	3	3	6	1	2	3

Table 8.6: Number of hidden layer neurons for the measured benchmark datasets

	housing			servo			cpu			auto-mpg		
	S	R	T	S	R	T	S	R	T	S	R	T
MLP (L-M)	2	0	2	10	0	10	2	0	2	1	0	1
FS-OLS	0	50	50	0	16	16	0	9	9	0	11	11
GL-ANN	1	39	40	3	26	29	2	3	5	1	4	5

implying a homogeneous data distribution. The results with the hybrid architecture are superior to those from either pure network, suggesting that the RBF neurons are able to add substantial detail to the function identified by the sigmoid neurons.

The auto-mpg dataset has an intermediate range of data densities, indicating little preference for MLP or RBF networks. Further, the R^2 value does not indicate a high degree of non-linearity, which would favour GL-ANNs. The MSEs for this dataset are similar for the 3 types of network.

As with the synthetic datasets, the degree of linearity within the datasets is a good indication of the relative performance of pure and hybrid architectures. The datasets which have R^2 values above 0.6 perform better with pure networks, whereas the only dataset with a lower R^2 value, servo, gives a lower MSE with GL-ANNs.

8.3.2 Optimum architectures

Table 8.5 gives the number of hidden layer neurons in optimally sized MLPs, RBF networks and GL-ANNs for the synthetic benchmark datasets, while table 8.6 gives the corresponding information for the measured datasets. In all cases ‘S’ refers to the number of sigmoid neurons, ‘R’ to the number of RBF neurons and ‘T’ to the total number of hidden layer neurons.

In most cases the optimum size of the GL-ANN networks is smaller than the corresponding size for pure RBF networks. In the case of the 1D sine and Hermite datasets it is also smaller than the optimum size for a MLP network.

With the sinewave and housing datasets the GL-ANNs are unable to improve upon the RBF networks. However, they imitate the RBF networks by using the smallest possible number of sigmoid neurons, i.e. 1.

For the more complex synthetic functions the GL-ANNs perform better than the RBF networks and create significantly different networks. The GL-ANN uses just 3 hidden neurons to reproduce the Hermite function and 6 for the impedance function. In the case of the servo dataset, the GL-ANN also discovers a novel hybrid function.

With the cpu dataset, the GL-ANN appears unable to imitate the high-performing RBF architecture. Instead it adopts an architecture similar to the best performing MLP network - with 2 sigmoid neurons - with the addition of a small number of RBF neurons. The unusual results with this dataset are discussed further in section 8.3.3.

These results confirm the observation made in Chapter 7 that GL-ANNs are parsimonious in their use of hidden neurons. They also show that they are able to discover types of function that are not available to pure RBF or MLP networks when they are advantageous, but will imitate pure networks when a hybrid function cannot reduce the MSE.

8.3.3 RBF spreads

Tables 8.7 and 8.8 give the spreads of the RBF neurons used in the most successful RBF and GL-ANN networks when trained with the synthetic and measured datasets, respectively. In the case of the GL-ANN networks, these are the average finishing spreads, after alteration by the third training step. These results suggest two trends-

- The spreads generally increase as the dimensionality of the input data increases. This is to be expected, since greater spreads are required to cover a higher-dimensional space.
- The GL-ANNs usually have comparable or narrower spreads than the RBF networks. This confirms the idea that the presence of the sigmoidal neurons frees the RBF neurons to concentrate on local variation in the input-output function.¹

¹The phenomenon of reduced RBF spread is seen when a fixed bias is introduced into RBF networks, since the radial functions do not have to fit the global bias, only local detail.[211]. The observation made here concerning hybrid networks may be seen in the same way.

Table 8.7: Synthetic datasets: optimum RBF spreads for pure RBF and hybrid networks

	1D sine	2D sine	impedance	Hermite
FS-OLS	0.45	0.8	2.4	0.1
GL-ANN	0.4	0.9	1.0	0.17

Table 8.8: Measured datasets: optimum RBF spreads for pure RBF and hybrid networks

	housing	servo	cpu	auto-mpg
FS-OLS	1.6	0.8	3.6	1.4
GL-ANN	1.2	0.4	0.4	0.22

As we have seen, GL-ANNs performed particularly poorly with the cpu dataset. One noticeable feature of the results for this dataset is the very large spread value (3.6) for the optimum RBF networks. A possible explanation is that the RBF neurons have a different mode of working with this dataset than is usual. The very wide spreads suggest that the RBF neurons are acting over a much wider region than is common and therefore map the global features of the function. In the GL-ANN this option is not available to the RBF neurons, since the sigmoid neuron, or neurons, present in the network have already adopted that role. The best result is obtained by adding a small number of RBF neurons with narrow spread. These cause a slight reduction in MSE compared to that obtained by MLPs, but the generalisation abilities of the hybrid cannot approach those of the pure RBF network.

8.3.4 RBF output weights

Table 8.9 shows the average weights between RBF and output neurons in the most successful networks, for the datasets that gave a better performance with GL-ANNs than with pure RBF networks. It is seen that the weights are generally much smaller for the GL-ANNs than the RBF networks. The same observation was made regarding the

Table 8.9: Output weights of RBF neurons in pure RBF and GL-ANN networks

	impedance	Hermite	servo
FS-OLS	2700	0.55	1.60
GL-ANN	54.2	0.88	0.27

CLASH dataset in section 7.2.3, where it was suggested that GL-ANNs automatically incorporate a degree of regularisation. The Hermite dataset is an exception. For this dataset, the GL-ANN contains wider RBF neurons than the pure RBF network, suggesting that those neurons have not been relegated to their usual role in describing only local variations in the function. They have greater importance in this network than in most GL-ANNs and the weights connecting these neurons to the output are therefore larger than usual.

8.4 Summary

The main aims of the studies reported in this chapter were

- to identify the strengths and weaknesses of the GL-ANN algorithm. In particular the objective was to define criteria that could be used to identify datasets likely to give low MSEs with a GL-ANN, compared to MLP or RBF networks.
- to find out more about the architectures created by the GL-ANN process.

The findings may be summarised as follows.

The results using synthetic datasets indicate that higher-dimensional, noisy datasets perform well with the GL-ANN algorithm. However the measured datasets are all high-dimensional and noisy, but the performance of GL-ANNs varies substantially between them. These datasets may be differentiated by their relative performances with MLP and RBF networks, and by their degree of non-linearity.

Datasets that show a strong preference for RBF over MLP networks, as evidenced by test MSEs, do not tend to perform well with the GL-ANN algorithm. These datasets are indicated by the spread (interquartile range) of the data densities. Higher interquartile ranges indicate more clustered data, which is likely to be fitted better by RBF networks.

However, this measure should not be relied upon too heavily. The CLASH dataset has a high interquartile range of data densities (see section 3.3.2), but test MSEs with MLPs are almost as low as those from RBF networks. A possible explanation for this is that there are quite strong interactions between different clusters of data, approximated by the exponential relationship between crest freeboard and overtopping rate (see section 3.3.3). The CLASH dataset therefore has some features that are modelled well by MLP networks as well as other features that are modelled well by RBF networks.

The degree of non-linearity within a dataset seems to be the best available indicator of performance with GL-ANNs. If the R^2 value obtained from linear regression is below 0.6, the dataset appears to perform well with the GL-ANN algorithm.

GL-ANNs are generally smaller than the corresponding RBF networks. In some cases they are also smaller than the optimum MLP networks. In situations where GL-ANNs give much lower MSEs than networks containing a single type of transfer function in the hidden layer, it is generally the result of identifying a novel function that is not available to 'pure' networks. When such a function is not available, the best-performing GL-ANN is usually seen to imitate a pure network as closely as it can.

RBF spreads within GL-ANN networks are generally similar to or less than those for pure RBF networks. The weights connecting RBF neurons to output neurons in GL-ANNs are also generally less than the corresponding weights in pure RBF networks. This confirms the idea that the RBF neurons in GL-ANNs are mainly confined to identifying local features within the input-output function.

Chapter 9

Conclusions and further work

9.1 Summary and conclusions

The findings of this research may be summarised under four headings:

- the nature of the CLASH dataset (Chapter 3)
- the results of training various neural networks to approximate the wave overtopping rate through training with the CLASH dataset (Chapters 4, 5 and 7)
- methods for identifying datasets for which the GL-ANN method would be beneficial (Chapters 7 and 8)
- description of the architectures created by the GL-ANN algorithm and of the manner in which the GL-ANN method operates (Chapters 7 and 8)

In addition, background material has been provided in the form of:

- a review of previous research in the areas of hydroinformatics, artificial neural networks and the links between the two (Chapter 1)
- a description of the relevant mathematical methods used in neural network training (Chapter 2)
- a description of the novel algorithm used for training Global-Local Artificial Neural Networks (Chapter 6)

The nature of the CLASH dataset may be summarised thus. It is a large, highly noisy dataset with considerable redundancy in the data. There are substantial ‘white

spots' in the data used in this study, although this should be remedied in later versions of the dataset. The dataset can be made more homogeneous using Froude scaling and mathematical transformations of some input parameters. However, even with these transformations, the relationship between the independent parameters and the wave overtopping rate is highly non-linear. Nevertheless, there is evidence of some relationships that hold globally throughout the data, in particular an approximately linear relationship between R_0 , T_0 and $\ln(q_0)$.

The training of MLP networks with the CLASH dataset revealed a range of information. Stochastic weight updates were much more effective than batch weight updates. Sigmoid output neurons were found to give slightly better results than linear output neurons and the Levenberg-Marquardt algorithm performed better than back-propagation. The introduction of momentum into the latter was found not to be beneficial.

RBF networks trained with the FS-OLS algorithm were found to give lower errors than MLP networks, although they require substantially more hidden layer neurons. Further improvements in performance were seen to occur with the introduction of either regularisation or a gradient descent optimisation step. The former produces the best results using networks that are larger than standard RBF networks, whereas the latter produces the best results using smaller networks.

GL-ANNs were seen to give errors comparable to those obtained from RBF networks trained with regularisation, with the CLASH data. However, the former use substantially fewer neurons than the latter. A comparison of hybrid networks trained with a two-step and a three-step algorithm suggests that the good performance of GL-ANNs is partly due to their hybrid architecture and partly due to their hybrid training algorithm.

Datasets which are likely to benefit from use of a GL-ANN have certain characteristics. They generally have high-dimensional inputs and are corrupted by high levels of noise. They are also likely to be highly non-linear. The R^2 statistic obtained from linear regression appears to be a good guide to non-linearity, with values below 0.6 performing well with the GL-ANN technique. The clustering behaviour of datasets (measured as the interquartile range of the estimated data densities) gives an indication of their relative performances with MLP and RBF networks. This may then be used as a guide to performance under GL-ANNs: networks that show a strong preference for RBF networks are unlikely to perform well with GL-ANNs.

GL-ANNs are generally smaller than the corresponding RBF networks. They usually have narrower RBF spreads and lower hidden-output weights. The optimum hidden layer size and spread for a RBF network may be used as a guide when creating GL-ANNs since they represent an upper bound on the corresponding parameters in GL-ANNs.

GL-ANNs usually operate by identifying coarser features of the input mapping before finer details. This process leads to an automatic regularising effect, as shown by the size of network weights.

Overall, it appears that datasets that perform well with GL-ANNs have some inter-parameter relationships that operate on a global level, and are therefore benefitted by the use of sigmoid neurons and gradient descent training, and others that operate on a local level, and are benefitted by radial basis functions and a deterministic selection of centres.

GL-ANNs are not appropriate for all datasets, but they appear to be a useful tool for datasets with highly complex relationships. A suggested course of action when trying to create an ANN for a previously unseen dataset is the following-

1. Assess the clustering behaviour of the data, as described in section 3.3.2. Highly clustered data (roughly that with an interquartile range greater than 1.2) is likely to be favoured by RBF networks.
2. Assess the linearity of the data, as described in section 3.3.4. Strongly non-linear data ($R^2 < 0.6$) is likely to perform well with GL-ANNs, although highly localised data, as indicated by the previous step, may create problems for the GL-ANN algorithm.
3. Decide upon candidate architectures.
4. Train the candidate networks. If RBF networks are trained first, their optimum network parameters may be a useful guide to the training of GL-ANNs. When training GL-ANNs, MLP networks must be created as an intermediate step, as must hybrid networks (two-step GL-ANNs). If the performance is satisfactory at one of these intermediate stages, training may be stopped early.

9.2 Original contributions

The original contributions made by this research may be summarised thus:

- A detailed study has been carried out into the efficacy of different architectures and training algorithms in fitting the underlying function within the CLASH dataset. To the best of my knowledge, RBF networks have not previously been trained with this dataset. I believe that comparisons between linear and sigmoid output neurons and between the back-propagation and Levenberg-Marquardt algorithms have also not been performed previously with this dataset. I believe that the comparisons between different RBF algorithms, including forward selection with regularisation and forward selection with gradient descent optimisation, are further new areas of study with respect to the CLASH dataset.
- An algorithm that combines gradient descent training with forward selection of centres has been developed. This algorithm results in the creation of hybrid networks containing pseudo-linear and radial basis function neurons in a single hidden layer. I have called these networks ‘global local artificial neural networks’ (GL-ANNs). To the best of my knowledge this algorithm is previously unreported.
- Criteria for predicting the efficacy of GL-ANN training have been developed. These use a variety of information, including performance with pure networks, interquartile ranges of data densities and R^2 values from linear regression.
- Typical properties of GL-ANNs have been assessed and described in terms of network architecture, radial basis function spread values and hidden-output weight sizes.

9.3 Further work

As pointed out in Appendix A, many curve fitting approaches suffer from the drawback that they cannot predict zero overtopping discharges for any finite crest freeboard. Since logarithmic values of q_0 are used to train the neural networks in this study, this research has the same difficulty. One solution that is likely to be investigated in the future is to introduce a filtering network. This would act on the input data and make a decision whether zero overtopping would be likely to occur with the given inputs. If zero overtopping was identified the inputs would not then be fed into the main neural network.

Future work could involve the use of a greater range of overtopping data. The less reliable data in the CLASH database has been excluded in this study. Other workers

have included all available data, but weighted the data, so that data with low RF and CF values are presented to the networks more than once [164]. A similar approach could be investigated.

The final version of the CLASH database has recently become available. It is intended that future research will incorporate the data from this database. Two approaches are possible. The first is to use the data that is newly available in the updated database as a ‘blind’ input to the networks trained in this study. Since the new database contains some data that fills in white spots in the data record, this would give evidence concerning the abilities of the networks to interpolate into sparsely populated areas of the input space. A second approach would be to train a new set of networks using the updated data. The new data is ‘cleaner’ than the data used in this study, since the results of some unreliable tests have been replaced with more reliable data. One would therefore expect the performance of all networks to be improved.

A related development would be the use of GL-ANNs with further data from other subject areas. Since Chapter 8 has shown that the GL-ANN method may be usefully applied to datasets other than the CLASH dataset, one would expect this to be a fruitful area of research. Data that is highly non-linear and noisy, such as weather conditions or stock market fluctuations, should be the target for future investigations.

Improvements to the GL-ANN algorithm may be available. The gradient descent optimisation step is often slow, as the networks created are large. The possibility of replacing Levenberg-Marquardt training with back-propagation (section 2.2.2) or conjugate gradient training (Appendix B) should be investigated.

A further problem is encountered in the identification of the optimum size of networks. After the second stage of training, very large networks give the lowest errors. However, after the third step much smaller networks often perform better. For this reason, the training of large networks in step two often proves to be wasteful. A method that can approximate the optimum size of GL-ANN networks before conducting gradient descent optimisation would therefore be very useful.

An important area for further research is the extraction of symbolic information from the networks created, possibly through the construction of regression trees. This is a popular area of research generally amongst the ANN community. However, GL-ANNs may have a specific role to play, since they are parsimonious in their use of neurons. For this reason, any symbolic information extracted from them is likely to be easier to interpret than information extracted from RBF networks.

It would be valuable if confidence levels could be attributed to the predictions made

by neural networks. A frequentist approach has been used by Pozueta *et al* [164]. They trained 500 networks with identical architectures and averaged the outputs in order to predict wave overtopping rates. The range of outputs from the 500 networks was then used to create confidence levels for each prediction, resulting in error bars to indicate, for example, 95% confidence levels. An alternative would be to take a Bayesian approach (see section 1.5.6). This would involve the explicit modelling of data distributions and would allow the comparison of different architectures, choices of inputs and training methods in a unified way.

The research described in this thesis has concentrated on the development of a novel algorithm for the training of individual networks. The search for the optimum architecture has generally been performed in a fairly crude manner. It might be advantageous to combine the GL-ANN method with a global search procedure such as a genetic algorithm or Bayesian analysis. The global procedure could then perform a search across architectures while the GL-ANN algorithm would optimise the individual networks.

Bibliography

- [1] Halcrow Group Ltd, HR Wallingford and John Chatterton Associates. National Appraisal of Assets at Risk from Flooding and Coastal Erosion, including the potential impact of climate change. Technical report, Department for Environment Food and Rural Affairs, UK, 2001. URL <http://www.defra.gov.uk/environ/fcd/policy/naarmaps.htm>.
- [2] Flood Management Division, DEFRA. National Assessment of Defence Needs and Costs for flood and coastal erosion management (NADNAC). Technical report, Department for Environment Food and Rural Affairs, UK, 2004. URL <http://www.defra.gov.uk/environ/fcd/policy/naarmaps.htm>.
- [3] K. Kimura, T. Fujiike, K. Kamikubo and K. Ishimoto. Damage to vehicles on a coastal highway by wave action. In *Proceedings of the 3rd Conference on Coastal Structures*, pp. 1009–1016. Balkema, Rotterdam, Santander, Spain, June 7–9, 1999.
- [4] W. Allsop, T. Bruce, J. Pearson, J. Alderson and T. Pullen. Violent wave overtopping at the coast, when are we safe? In *International Conference on Coastal Management 2003*, volume 5. Thomas Telford, Brighton, United Kingdom, October 15–17, 2003.
- [5] DEFRA. The Appraisal of Human-Related Intangible Impacts of Flooding. Technical Report FD2005/TR, Department for Environment Food and Rural Affairs, UK, 2004. URL <http://www.rpaltd.co.uk/tools/downloads/reports/intangiblestechreport.pdf>.
- [6] T. Hedges and M. Reis. Random wave overtopping of simple sea walls: a new regression model. *Proceedings of the ICE. Water Maritime and Energy*, 130, pp. 1–10, 1998.

- [7] Anglo-Saxon Chronicles, November 2005. URL <http://sunsite.berkeley.edu/OMACL/Anglo/part3.html>.
- [8] I. West. Chesil Beach: Storms and Floods. Geology of the Wessex Coast of Southern England, November 2005. URL <http://www.soton.ac.uk/~imw/chestorm.htm>.
- [9] E. Bryant and S. Haslett. Was the AD 1607 Coastal flooding event in the Severn estuary and bristol channel (UK) due to a tsunami? *Archaeology in the Severn Estuary*, 13, pp. 163–167, 2002.
- [10] British Broadcasting Corporation (BBC). Internet, November 2005. URL <http://news.bbc.co.uk/1/hi/wales/4397679.stm>.
- [11] TW Blog. Internet, November 2005. URL <http://www.severnsolutions.co.uk/twblog/archive/2005/01/06/greatflood1606>.
- [12] E. Coke. *The Case of the Isle of Ely*, 1609.
- [13] D. Defoe. *The Storm*. Penguin Books, 1704.
- [14] R. Muir-Wood. December 1703 Windstorm; 300-year Retrospective. In *The Great Storm*. Royal Meteorological Society, National Maritime Museum, Ramsgate, Kent, UK, November 29–30, 2003. URL www.rms.com/Publications/1703.Windstorm.pdf.
- [15] British Broadcasting Corporation (BBC). Internet, November 2005. URL http://www.bbc.co.uk/weather/features/understanding/1703_storm.shtml.
- [16] A. Hart-Davis and E. Troscianko. *Henry Winstanley and the Eddystone Lighthouse*. Sutton Publishing, Cambridge, MA, USA, 2002.
- [17] Environment Agency. Internet, November 2005. URL http://www.environment-agency.gov.uk/subjects/flood/826674/882909/426221/?lang=_e.
- [18] British Broadcasting Corporation (BBC). Internet, November 2005. URL http://www.bbc.co.uk/weather/features/understanding/1953_flood.shtml.

- [19] Meteorological Office, UK. Internet, November 2005. URL <http://www.metoffice.com/corporate/pressoffice/anniversary/floods1953.html>.
- [20] MAFF. A Strategy for Flood and Coastal Defence in England and Wales. Technical Report PB1471, Ministry for Agriculture, Fisheries and Food (MAFF), Environment Agency, UK, 1993.
- [21] DEFRA. Making space for water: Taking forward a new Government strategy for flood and coastal erosion risk management in England. Technical Report PB10516, Department for Environment Food and Rural Affairs, UK, 2005. URL <http://www.defra.gov.uk/environ/fcd/policy/strategy/1stres.pdf>.
- [22] New Scientist. Internet, November 2005. URL <http://www.newscientist.com/channel/earth/tsunami>.
- [23] New Scientist. Internet, November 2005. URL <http://www.newscientist.com/article.ns?id=dn7940>.
- [24] M. Owen. Design of seawalls allowing for wave overtopping. Technical Report EX924, HR Wallingford, Environment Agency, UK, 1980.
- [25] L. Franco, M. de Gerloni and J. van der Meer. Wave overtopping on Vertical and Composite Breakwaters. In B. Edge (editor), *Proceedings of the 24th International Conference on Coastal Engineering*, pp. 1030–1045. ASCE, New York, Kobe, Japan, October 23–24, 1994.
- [26] P. Besley. Overtopping of Seawalls: Design and Assessment Manual. Technical Report W178, HR Wallingford, Environment Agency, UK, 1999.
- [27] G. Bullock, A. Crawford, P. Hewson, M. Walkden and P. Bird. The influence of air and scale on wave impact pressures. *Coastal Engineering*, 42, pp. 291–312, 2001.
- [28] K. Hu, C. Mingham and D. Causon. Numerical simulation of wave overtopping of coastal structures using the non-linear shallow water equations. *Coastal Engineering*, 41, pp. 433–465, 2000.

- [29] J. Stoker. *Water Waves; The Mathematical Theory with Applications*. John Wiley, New York, USA, 1992.
- [30] S. Richardson, D. Ingram, C. Mingham and D. Causon. On the Validity of the Shallow Water Equations for Violent Wave Overtopping. In B. L. Edge and J. M. Hemsley (editors), *Ocean Wave Measurement and Analysis*, pp. 1112–1125. ASCE, 2002.
- [31] J. Shiach, C. Mingham, D. Ingram and T. Bruce. The Applicability of the Shallow Water Equations for Modelling Violent Wave Overtopping. *Coastal Engineering*, 51, pp. 1–15, 2004.
- [32] J. Zhou, D. Causon, D. Ingram and C. Mingham. Numerical Solutions of the Shallow Water Equations with Discontinuous Bed Topography. *International Journal for Numerical Methods in Fluids*, 38, pp. 769–788, 2002.
- [33] P. Troch, T. Li, J. de Rouck and D. Ingram. Wave Interaction with a sea dike using a VOF Finite Volume Method. In J. Chung, M. Prevosto, N. Mizutani, C. Kim and S. Grilli (editors), *Proceedings of the 13th International Offshore and Polar Engineering Conference*, pp. 325–332. Princeton, NJ, USA, May 25–31, 2003.
- [34] L. Qian, D. Causon, D. Ingram and C. Mingham. Cartesian Cut Cell Two-Fluid Solver for Hydraulic Flow Problems. *Journal Of Hydraulic Engineering*, 129(9), pp. 1683–1695, 2003.
- [35] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, USA, second edition, 1999.
- [36] J. Andersen. *An Introduction to Neural Networks*. MIT Press, Cambridge, MA, USA, 1995.
- [37] W. Duch and N. Jankowski. Survey of Neural Transfer Functions. *Neural Computing Surveys*, 2, pp. 163–212, 1999.
- [38] H. Maier and G. Dandy. Neural networks for the prediction and forecasting of water resources variables: a review of the modelling issues and applications. *Environmental Modelling and Software*, 15, pp. 101–124, 2000.

- [39] Aristotle. On Memory and Reminiscence, 500 B.C. URL <http://classics.mit.edu/Aristotle/memory.html>.
- [40] C. Schommer. Incremental discovery of association rules with dynamic neural cells. In L. Cojocaru and C. Martn-Vide (editors), *Proceedings of the ECAI 2004 Workshop on Symbolic Networks*. Universidad Politcnica de Valencia, Valencia, Spain, 2004.
- [41] R. Belew. *Adaptive information retrieval: Machine learning in associative networks*. Ph.D. thesis, University of Michigan, Michigan, 1986.
- [42] T. Doszkocs, J. Reggia and X. Lin. Connectionist models and information retrieval. *Annual Review of Information Science and Technology*, 25, pp. 209–260, 1990.
- [43] C.-H. Chen and V. Honavar. A Neural-Network Architecture for Syntax Analysis. *IEEE Transactions on Neural Networks*, 10(1), pp. 94–114, 1999.
- [44] J. Austin. *The Design and Application of Associative Memories for Scene Analysis*. Ph.D. thesis, Brunel University, 1986. URL <http://www.cs.york.ac.uk/arch/NeuralNetworks/references.htm>.
- [45] S. Ullman. *High level vision: object recognition and visual cognition*. MIT Press, Cambridge, MA, USA, 1996.
- [46] J. L. Blue, G. T. Candela, P. J. Grother and C. Wilson. Evaluation of pattern classifiers for fingerprint and OCR applications. *Pattern Recognition*, 27, pp. 485–501, 1994.
- [47] C. Wilson. Evaluation of character recognition systems. In C. Kamm (editor), *Neural Networks for Signal Processing III.*, pp. 485–496. IEEE Signal Processing Society, New York, USA, September 6–9, 1993.
- [48] S.-L. Chen, L.-S. Chen, Y.-C. Chung and S.-W. Chen. Automatic License Plate Recognition. *IEEE Transactions on Intelligent Transportation Systems*, 5(1), pp. 42–53, March 2004.
- [49] A. Broumandnia and M. Fathy. Application of pattern recognition for Farsi license plate recognition. In *ICGST International Conference on Graphics, Vision and Image Processing*. Cairo, Egypt, December 19–21, 2005.

- [50] T. Sejnowski and C. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1(1), pp. 145–168, 1987.
- [51] R. Brause. Medical Analysis and Diagnosis by Neural Networks. In J. Crespo, V. Maojo and F. Martin (editors), *2nd International Symposium on Medical Data Analysis*, pp. 1–13. Madrid, Spain, October 8–9, 2001. URL citeseer.ist.psu.edu/brause01medical.html.
- [52] M. Handzic, F. Tjandrawibawa and J. Yeo. How Neural Networks Can Help Loan Officers to Make Better Informed Application Decisions. In E. Cohen (editor), *Informing Science and Information Technology Education*, pp. 97–109. Pori, Finland, June 24–27, 2003. URL <http://proceedings.informingscience.org/IS2003Proceedings>.
- [53] L. Giles, S. Lawrence and A.-C. Tsoi. Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine Learning*, 44, pp. 161–183, 2001.
- [54] P. Kershaw and P. Rossini. Using Neural Networks to Estimate Constant Quality House Price Indices. In *Fifth Annual Pacific-Rim Real Estate Society Conference*, pp. 97–109. Kuala Lumpur, Malaysia, January 26–30, 1999.
- [55] R. Calvo. Neural network prediction of solar activity. *The Astrophysical Journal*, 444, pp. 916–921, 1995.
- [56] J. Haberl and S. Thamilsaran. The great energy predictor shootout II. Measuring retrofit savings. *ASHRAE*, 40(1), pp. 49–56, 1998.
- [57] W. Miller, R. Sutton and P. Werbos (editors). *Neural Networks for Control*. MIT Press, Cambridge, MA, USA, 1990.
- [58] G. W. Irwin, K. Warwick and K. J. Hunt (editors). *Neural network applications in control*, volume 53 of *IEE control engineering series*. Institution of Electrical Engineers, London, UK, 1995.
- [59] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, pp. 115–133, 1943.
- [60] D. Hebb. *The Organization of Behaviour: A Neuropsychological Theory*. John Wiley, New York, USA, 1949.

- [61] N. Rochester, J. Holland, L. Haibt and W. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, IT-2, pp. 80–93, 1956.
- [62] T. Sejnowski. Strong covariance with nonlinearly interacting neurons. *Journal of mathematical biology*, 4, pp. 303–321, 1977.
- [63] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In D. Shannon and J. McCarthy (editors), *Automata Studies*, pp. 43–98. Princeton University Press, Princeton, NJ, USA, 1956.
- [64] F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, pp. 386–408, 1958.
- [65] B. Widrow and M. Hoff. Adaptive Switching Circuits. *IRE WESCON Covention Record*, 38, pp. 96–104, 1960.
- [66] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, USA, 1969.
- [67] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43, pp. 59–69, 1982.
- [68] D. Willshaw and C. von der Marlsburg. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London Series B*, 194, pp. 431–445, 1976.
- [69] J. Kaas, M. Merzenich and H. Killackey. The reorganization of somatosensory cortex following peripheral nerve damage in adult and developing mammals. *Annual Review of Neurosciences*, 6(2), pp. 325–356, 1983.
- [70] D. Hubel and T. Wiesel. Functional architecture of macaque visual cortex. *Proceedings of the Royal Society B*, 198, pp. 1–59, 1977.
- [71] N. Suga. The extent to which bisonar information is represented in the bat auditory cortex. In G. Edelman, W. Gall and W. Cowan (editors), *Dynamic Aspects of Neocortical Function*, pp. 653–695. Wiley, New York, USA, 1985.

- [72] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79, pp. 2554–2558, 1982.
- [73] D. Rumelhart and J. McClelland (editors). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Cognition*. MIT Press, Cambridge, MA, USA, 1986.
- [74] S. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16, pp. 299–307, 1967.
- [75] P. Werbos. *Beyond Regression: New tools for prediction and analysis in the behavioural sciences*. Ph.D. thesis, Harvard University, Cambridge, MA, USA, 1974.
- [76] D. Rumelhart, G. Hinton and R. Williams. Learning Internal Representations by Error Propagation. In D. Rumelhart and J. McClelland (editors), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Cognition*, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986.
- [77] M. Hagiwara. Theoretical derivation of momentum term in back-propagation. In *International Joint Conference on Neural Networks*, volume 1, pp. 682–686. Baltimore, MD, USA, June 7–11, 1992.
- [78] D. McLean, Z. Bandar and J. O’Shea. The Evolution of a Feed Forward Neural Network trained under Back-Propagation. In *ICANN’97*, pp. 418–422. Springer-Verlag, 1997.
- [79] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1, pp. 295–307, 1988.
- [80] S. Amari, N. Murata, K.-R. Muller, M. Finke and H. Yang. Statistical Theory of Overtraining - Is cross-validation asymptotically effective? In M. Mozer, M. Jordan and T. Petsche (editors), *Advances in Neural Information Processing Systems*, volume 8, pp. 176–182. MIT Press, Cambridge, MA, USA, 1996.
- [81] A. Krogh and J. Hertz. A Simple Weight Decay can Improve Generalization. In J. Moody, S. Hanson and R. Lippmann (editors), *Advances in Neural Information Processing Systems*, volume 4, pp. 950–957. Morgan Kaufmann, San Mateo, CA, USA, 1995.

- [82] B. Hassibi and D. Stork. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. *Advances in Neural Information Processing Systems*, pp. 164–171, 1992.
- [83] S. Kirkpatrick, C. Gelatt and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220, pp. 671–680, 1983.
- [84] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2, pp. 524–532. Morgan Kaufmann, San Mateo, CA, USA, 1990.
- [85] J. Shewchuk. An introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, August 1994.
- [86] T. Masters. *Advanced Algorithms for Neural Networks: C++ Sourcebook*. John Wiley, Frankfurt, Germany, 1995.
- [87] A. Ranganathan. The Levenberg-Marquardt Algorithm, November 2005. URL <http://www.cc.gatech.edu/~ananth/lmtut.pdf>.
- [88] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, London, UK, 1975.
- [89] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Wokingham, 1989.
- [90] K. Balakrishnan and V. Honovar. Evolutionary design of neural architectures - a preliminary taxonomy and guide to literature. Technical Report CS-TR 95-01, Artificial Intelligence Research Group, Iowa State University, Iowa, USA, 1995.
- [91] X. Yao. A Review of Evolutionary Artificial Neural Networks. *International Journal of Intelligent Systems*, 8, pp. 539–567, 1993.
- [92] R. Belew, J. McInerney and N. Schraudolph. Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. Technical Report CS90-174, Cognitive Computer Science Research Group, University of California at San Diego, CA, USA, 1990.
- [93] A. Gelman, J. Carlin, H. Stern and D. Rubin. *Bayesian data analysis*. Texts in Statistical Science. Chapman and Hall, London, UK, 1995.

- [94] D. MacKay. Bayesian Interpolation. *Neural Computation*, 4, pp. 415–447, 1992.
- [95] M. Powell. Radial Basis Functions for Multivariable Interpolation: A Review. In J. Mason and M. Cox (editors), *Algorithms for Approximation*, pp. 143–167. Oxford University Press, Oxford, UK, 1985.
- [96] T. Cover. Geometrical and Statistical Properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14, pp. 326–334, 1965.
- [97] D. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2, pp. 321–355, 1988.
- [98] M. Orr. Regularisation in the Selection of Radial Basis Function Centres. *Neural Computation*, 7, pp. 606–623, 1995.
- [99] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. A.I. Memo 1140, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, July 1989.
- [100] A. Tikhonov. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Math Doklady*, 4, pp. 1035–1038, 1963.
- [101] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, USA, 1961.
- [102] S. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, IT28, pp. 127–135, 1982.
- [103] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observation. In L. LeCun and J. Neyman (editors), *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pp. 281–297. University of California Press, Berkeley, CA, USA, 1967.
- [104] S. Chen, C. Cowan and P. Grant. Orthogonal least squares Learning for radial basis function networks. *IEEE Transactions on Neural Networks*, 2, pp. 302–309, March 1991.
- [105] M. Kubat. Decision Trees can Initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5), pp. 813–821, 1998.

- [106] M. Orr, K. Takezawa, A. Murray, S. Ninomiya, M. Oide and T. Leonard. Combining Regression Trees and Radial Basis Function Networks. *International Journal of Neural Systems*, 10(6), pp. 453–465, 2000.
- [107] S. Cohen and N. Intrator. A Hybrid Projection-based and Radial Basis Function Architecture: Initial Values and Global Optimisation. *Pattern Analysis and Applications*, 5, pp. 113–120, 2002.
- [108] G. Flake. Square unit augmented, radially extended, multilayer perceptrons. In *Neural Networks: Tricks of the Trade*, pp. 145–163. Springer-Verlag, 1998.
- [109] Y. Liu and X. Yao. Evolutionary Design of Artificial Neural Networks with different nodes. *Proceedings of the IEEE International Conference*, pp. 670–675, 1996.
- [110] N.Jiang, Z. Zhao and L. Ren. Design of structural modular neural networks with genetic algorithm. *Advances in Software Engineering*, 34, pp. 17–24, 2003.
- [111] M. Jordan and R. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6, pp. 181–214, 1994.
- [112] K. Chen, L. Yang, X. Yu and H. Chi. A self-generating modular neural network architecture for supervised learning. *Neurocomputing*, 16, pp. 33–48, 1997.
- [113] A. T. C. on Application of Artificial Neural Networks in Hydrology. Artificial Neural Networks in Hydrology 2: Hydrologic Applications. *Journal of Hydrologic Engineering*, 5(2), pp. 124–137, April 2000.
- [114] J. Smith and R. Eli. Neural-Network Models of Rainfall-Runoff Process. *Journal of Water Resources Planning and Management*, 121(6), pp. 499–508, 1995.
- [115] A. Shamseldin. Application of a neural network technique to rainfall-runoff modelling. *Journal of Hydrology*, 199, pp. 272–294, 1997.
- [116] K. Hsu, H. Gupta and S. Sorooshian. Artificial neural network modeling of the rainfall-runoff process. *Water Resources Research*, 31(10), pp. 2517–2530, 1995.
- [117] J. Mason, R. Price and A. Tem'me. A neural network model of rainfall-runoff using radial basis functions. *Journal of Hydraulic Research*, 34(4), pp. 537–548, 1996.

- [118] A. Fernando and A. Jayawardena. Runoff forecasting using RBF networks with OLS algorithm. *Journal of Hydrologic Engineering, ASCE*, 3(3), pp. 203–209, 1998.
- [119] A. Jayawardena and A. Fernando. Use of Radial Basis Type Artificial Neural Networks for Runoff Simulation. *Computer-Aided Civil and Infrastructure Engineering*, 13, pp. 91–99, 1998.
- [120] C. Dawson and R. Wilby. A comparison of artificial neural networks used for rainfall-runoff modelling. *Hydrology and Earth Systems Sciences*, 3(4), pp. 529–540, 1999.
- [121] Y. Dibike, D. Solomatine and M. Abbott. On the encapsulation of numerical-hydraulic models in artificial neural network. *Journal of Hydraulic Research*, 37(2), pp. 147–161, 1999.
- [122] A. S. Kumar, K. Sudheer, S. Jain and P. Agarwal. Rainfall-runoff modelling using artificial neural networks: comparison of network types. *Hydrological Processes*, 19, pp. 1277–1291, 2005.
- [123] N. Karunanithi, W. Grenney, D. Whitley and K. Bovee. Neural Networks for River Flow Prediction. *Journal of Computing in Civil Engineering*, 8(2), pp. 201–220, 1994.
- [124] K. Thirumalaiah and M. Deo. Real-Time Flood Forecasting Using Neural Networks. *Computer-Aided Civil and Infrastructure Engineering*, 13, pp. 101–111, 1998.
- [125] K. Thirumalaiah and M. Deo. River Stage Forecasting Using Artificial Neural Networks. *Journal of Hydrologic Engineering*, 3(1), pp. 26–32, January 1998.
- [126] R. Muttiah, R. Srinivasan and P. Allen. Prediction of two-year peak stream discharges using neural networks. *Journal of the American Water Resources Association*, 33(3), pp. 625–630, 1997.
- [127] R. Yang. *Application of neural networks and genetic algorithms to modelling flood discharges and urban water quality*. Ph.D. thesis, University of Manchester, John Rylands Library, Manchester, UK, 1997.

- [128] R. Abrahart, L. See and P. Kneale. New tools for neurohydrologists: using ‘network pruning’ and ‘model breeding’ algorithms to discover optimum inputs and architectures. In *Geocomputation*, volume 3. September 17–19, 1998.
- [129] R. Abrahart, L. See and P. Kneale. Using network pruning and model breeding algorithms to optimise neural net architectures and forecasting inputs in a rainfall-runoff model. *Journal of Hydroinformatics*, 1(2), pp. 103–114, 1999.
- [130] M. Markas, J. Salas and H.-K. Shin. Predicting streamflow based on neural networks. In *Proceedings of the 1st International Conference on Water Resource Engineering*, pp. 1641–1646. ASCE, New York, USA, 2004.
- [131] N. Poff, S. Tokar and P. Johnson. Stream hydrological and ecological responses to climate change assessed with an artificial neural network. *Limnology and Oceanography*, 41(5), pp. 857–863, 1996.
- [132] M. Tawfik, A. Ibrahim and H. Fahmy. Hysteresis sensitive neural network for modeling rating curves. *Journal of Computing in Civil Engineering*, 11(3), pp. 206–211, 1997.
- [133] K. Hsu, S. Sorooshian, H. Gupta, X. Gao and B. Imam. Hydrologic Modelling and Analysis Using a Self-Organizing Linear Output Network. In *The Proceedings of the International Environmental Modelling and Software Society*, volume 2, pp. 172–177. 2002.
- [134] D. Furundzic. Application example of neural networks for time series analysis: Rainfall-runoff modeling. *Signal Processing*, 64, pp. 383–396, 1998.
- [135] L. See, M. Dougherty and S. Openshaw. Some Initial Experiments with Neural Network Models of Flood Forecasting on the River Ouse. In *Geocomputation*, volume 2, pp. 15–22. August 26–29, 1997.
- [136] L. See and S. Openshaw. Applying Soft Computing Approaches to River Level Forecasting. *Hydrological Sciences*, 44(5), pp. 763–778, 1999.
- [137] L. See and S. Openshaw. Using Soft Computing Techniques to Enhance Flood Forecasting on the River Ouse. In V. Babovic and L. Larsen (editors), *Hydroinformatics*, volume 3, pp. 819–824. 1999.

- [138] B. Zhang and R. Govindaraju. Prediction of watershed runoff using Bayesian concepts and modular neural networks. *Water Resources Research*, 36(3), pp. 753–762, 2000.
- [139] H. Maier and G. Dandy. The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. *Environmental Modelling and Software*, 13, pp. 193–209, 1998.
- [140] R. Abrahart, L. See and P. Kneale. Applying saliency analysis to neural network rainfall-runoff modelling. In *Geocomputation*, volume 3. September 17–19, 1998.
- [141] R. Abrahart. Single-Model-Bootstrap Applied to Neural Network Rainfall-Runoff Forecasting. In *Geocomputation*, volume 6. September 24–26, 2001.
- [142] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Monographs on statistics and applied probability. Chapman and Hall, London, UK, 1993.
- [143] W. Hsieh and B. Tang. Applying Neural Network Models to Prediction and Data Analysis in Meteorology and Oceanography. *Bulletin of the American Meteorological Society*, 79(9), pp. 1855–1870, 1998.
- [144] H. Mase, M. Sakamoto and T. Sakai. Neural Network for Stability Analysis of Rubble-Mound Breakwaters. *Journal of Waterway, Port, Coastal and Ocean Engineering*, 121(6), pp. 294–299, 1995.
- [145] T.-L. Lee and D. Jeng. Application of artificial neural networks in tide-forecasting. *Ocean Engineering*, 29, pp. 1003–1022, 2002.
- [146] C.-P. Tsai and T.-L. Lee. Back-propagation Neural Network in Tide-Level Forecasting. *Journal of Waterway, Port, Coastal and Ocean Engineering*, 125(4), pp. 195–202, 1999.
- [147] T.-L. Lee. Back-propagation neural network for long-term tidal predictions. *Ocean Engineering*, 31, pp. 225–238, 2004.
- [148] M. Deo and C. S. Naidu. Real time wave forecasting using neural networks. *Ocean Engineering*, 26, pp. 191–203, 1999.
- [149] M. Deo and N. K. Kumar. Interpolation of wave heights. *Ocean Engineering*, 27, pp. 907–919, 2000.

- [150] C.-P. Tsai, C. Lin and J.-N. Shen. Neural network for wave forecasting among multi-stations. *Ocean Engineering*, 29, pp. 1683–1695, 2002.
- [151] W. Huang and C. Murray. Application of an Artificial Neural Network to Predict Tidal Currents in an Inlet. Technical Report CHETN-IV-58, US Army Corps of Engineers, 2003. URL cirp.wes.army.mil/cirp/cetns/chetn-iv58.pdf.
- [152] O. Makarynskyy. Improving wave predictions with artificial neural networks. *Ocean Engineering*, 31, pp. 709–724, 2004.
- [153] F. Tangang, W. Hsieh and B. Tang. Forecasting regional sea surface temperatures in the tropical Pacific by neural network models, with wind stress and sea level pressure as predictors. *Journal of Geophysical Research*, 103(C4), p. 75117522, 1998.
- [154] M. Perrone. *Improving regression estimation: Averaging methods for variance reduction with extensions, to general convex measure optimization*. Ph.D. thesis, Brown University, Rhode Island, 1993.
- [155] G. Valentini and F. Masulli. Ensembles of learning machines. In M. Marinaro and R. Tagliaferri (editors), *Neural Nets WIRN Vietri-02, Series Lecture Notes in Computer Sciences*. Springer-Verlag, Heidelberg, Germany, 2002. URL citeseer.ist.psu.edu/valentini02ensembles.html.
- [156] U. Naftaly, N. Intrator and D. Horn. Optimal ensemble averaging of neural networks. *Network*, 8, pp. 283–296, 1997.
- [157] M. Deo, A. Jha, A. Chaphekar and K. Ravikant. Neural networks for wave forecasting. *Ocean Engineering*, 28, pp. 889–898, 2001.
- [158] M. Deo and S. Jagdale. Prediction of breaking waves with neural networks. *Ocean Engineering*, 30, pp. 1163–1178, 2003.
- [159] A. El-Shazly. The Efficiency of Neural Networks to Model and Predict Monthly Mean Sea Level from Short Spans Applied to Alexandria Tide Gauge. In *FIG Working Week 2005: From Pharaohs to Geoinformatics*. April 16–21, 2005.
- [160] D. Specht. A general Regression Neural Network. *IEEE Transactions on Neural Networks*, 2(6), pp. 568–576, 1991.

- [161] J. Medina, J. Gonzales-Escriva, J. Garrido and J. de Rouck. Overtopping analysis using neural networks. In N. Allsop (editor), *ICCE 2002*, volume 28. Thomas Telford, 2003.
- [162] H. Verhaeghe, J. van der Meer, G.-J. Steendam, P. Besley, L. Franco and M. van Gent. Wave Overtopping Database As the Starting Point for a Neural Network Prediction Method. In J. Melby (editor), *Coastal Structures 2003*, pp. 418–430. ASCE, August 26–30, 2003.
- [163] H. Verhaeghe, J. van der Meer and G.-J. Steendam. Report: database on wave overtopping at coastal structures. Technical report, CLASH, Ghent University, Belgium, 2003.
- [164] B. Pozueta, M. van Gent and H. van den Boogaard. Neural network modelling of wave overtopping at coastal structures. In J. M. Smith (editor), *The Proceedings of the 29th International Conference on Coastal Engineering 2004*, volume 4. Lisbon, Portugal, September 19–24, 2004.
- [165] R. Andrews, J. Diederich and A. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), pp. 373–389, 1998.
- [166] G. Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*. Ph.D. thesis, University of Wisconsin-Madison, 1991.
- [167] Y. Dibike, A. Minns and M. Abbott. Applications of artificial neural networks to the generation of wave equations from hydraulic data. *Journal of Hydraulic Research*, 37(1), pp. 81–97, 1999.
- [168] C. Dawson and R. Wilby. Hydrological modelling using artificial neural networks. *Progress in Physical Geography*, 25(1), pp. 80–108, 2001.
- [169] M. Kearns. A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split. In *Advances in Neural Information Processing Systems*, volume 8, pp. 524–532. MIT Press, Cambridge, MA, USA, 1996.
- [170] J. de Rouck. Crest Level Assessment of Coastal structures by Full Scale Monitoring, Neural Network Prediction and Hazard Analysis on Permissible Wave Overtopping. Available at <http://www.clash-eu.org>.

- [171] J. de Rouck. Second Detailed Interim Report. Technical Report CLA127/296, CLASH, Ghent University, Belgium, February 2004.
- [172] M. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1995.
- [173] B. Widrow and S.D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.
- [174] M. Minsky. Steps Towards Artificial Intelligence. *Proceedings of the Institute of Radio Engineers*, 49, pp. 8–30, 1961.
- [175] M. Hagan and M. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5, pp. 989–993, 1994.
- [176] M. Orr. Introduction to Radial Basis Function Networks. Technical report, Institute for Adaptive and Neural Computation (University of Edinburgh), <http://www.anc.ed.ac.uk/mjo/papers/intro.ps.gz>, April 1996.
- [177] C. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2, pp. 11–22, 1986.
- [178] J. Battjes and H. Groenendijk. Wave height distributions on shallow foreshores. *Coastal Engineering*, 40(3), pp. 161–182, 2000.
- [179] Y. LeCun, L. Bottou, G. Orr and K.-R. Muller. Efficient BackProp. In G. Orr and K. Muller (editors), *Neural networks: tricks of the trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
- [180] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, UK, 1995.
- [181] G. Steendam, J. van der Meer, H. Verhaeghe, P. Besley, L. Franco and M. van Gent. The international database on wave overtopping. In *Proceedings of the 29th International Conference on Coastal Engineering, ASCE*, pp. 4301–4313. World Scientific, Lisbon, Portugal, September 19–24, 2004.
- [182] J. van der Meer, M. van Gent, B. Pozueta, H. Verhaeghe, G.-J. Steendam and J. Medina. Applications of a neural network to predict wave overtopping at coastal structures. In *Coastlines, Structures and Breakwaters: Harmonising Scale and Detail, ICE*. Thomas Telford, London, UK, April 20–22, 2005.

- [183] S. Lawrence, A. Tsoi and A. Back. Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smoothness. In P. Bartlett, A. Burkitt and R. Williamson (editors), *Australian Conference on Neural Networks, ACNN96*, pp. 16–21. Australian National University, 1996.
- [184] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Boston, MA, USA, 1990.
- [185] L. Wessels and E. Barnard. Avoiding false local minima by proper initialisation of connections. *IEEE Transactions on Neural Networks*, 3(6), pp. 899–905, 1992.
- [186] Z. Luo. On the convergence of the LMS algorithm with adaptive learning rate for linear feedforward networks. *Neural Computation*, 3, pp. 226–245, 1991.
- [187] M. Stone. Cross-validated Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society B*, 36(2), pp. 111–147, 1974.
- [188] J. Shao. Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association*, 88(422), pp. 486–494, 1993.
- [189] P. Smyth. Clustering using Monte-Carlo cross-validation. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 126–133. AAAI Press, Menlo Park, CA, USA, August 4, 1996.
- [190] P. Burman. A Comparative Study of Ordinary Cross-Validation, v -Fold Cross-Validation and the Repeated Learning-Testing Methods. *Biometrika*, 76, pp. 503–514, 1989.
- [191] R. Johnson and D. Wichern (editors). *Applied Multivariate Statistical Analysis*. Prentice Hall, Upper Saddle River, NJ, USA, fifth edition, 2002.
- [192] F. Schwenker, H. Kestler and G. Palm. Three learning phases for radial-basis-function networks. *Neural Networks*, 14, pp. 439–458, 2001.
- [193] D. Donoho and I. Johnstone. Projection-Based Approximation and a Duality with Kernel Methods. *The Annals of Statistics*, 17, pp. 58–106, 1989.
- [194] G. Auda and M. Kamel. Modular Neural Networks: a Survey. *International Journal of Neural Systems*, 9, pp. 129–151, 1999.

- [195] T. Hrycej. *Modular learning in neural networks: a modularized approach to classification*. Wiley, New York, USA, 1992.
- [196] S. Johannes, B. Wieringa, M. Matzke and T. Munte. Hierarchical visual stimuli: electrophysical evidence for separating left hemispheric global and local processing mechanisms in humans. *Neuroscience letters*, 210, pp. 111–114, 1996.
- [197] R. Hubner. Hemispheric differences in Global/Local Processing Revealed by Same-Different Judgements. *Visual Cognition*, 5(4), pp. 457–478, 1998.
- [198] J. Piaget. *Meine Theorie der geistigen Intelligenz*. Fischer Taschenbuch Verlag, Frankfurt, Germany, 1983.
- [199] T. Poggio and F. Girosi. Networks for Approximation and Learning. *Proceedings of the IEEE*, 78, pp. 1481–1497, 1990.
- [200] F. Girosi, M. Jones and T. Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7, pp. 219–269, 1995.
- [201] J. Moody. Fast learning in multi resolution hierarchics. In D. Touretzky (editor), *Advances in Neural Information Processing Systems*, volume 1, pp. 29–39. Morgan Kaufmann, Carnegie Mellon University, USA, 1989.
- [202] S. Geman, E. Bienenstock and R. Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4, pp. 1–58, 1992.
- [203] S. Arisariyawong and S. Charoenseang. Dynamic Self-Organised Learning for Optimizing the Complexity Growth of Radial Basis Function Neural Networks. In *IEEE International Conference on Industrial Technology*. December 11–14, 2002.
- [204] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematical Control Signals Systems*, 2, pp. 303–314, 1989.
- [205] J. Park and I. Sandberg. Approximation and Radial-Basis-Function Networks. *Neural Computation*, 5, pp. 304–316, 1993.
- [206] J. Moody and C. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1, pp. 281–294, 1989.

- [207] G. Hinton, J. McClelland and D. Rumelhart. Distributed Representations. In *Parallel Distributed Processing*, volume 1 of *Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, MA, USA, 1986.
- [208] C. Campbell. Constructive Learning Techniques for Designing Neural Network Systems. In C. Leondes (editor), *Optimization Techniques*, volume 2 of *Neural Network Systems Techniques and Applications*. Academic Press, San Diego, CA, USA, 1998.
- [209] D. Navon. Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, 9, pp. 353–383, 1977.
- [210] S. Christman. Individual Differences in Stroop and Local-Global Processing: A Possible Role of Interhemispheric Interaction. *Brain and Cognition*, 45, pp. 97–118, 2001.
- [211] M. Orr. Matlab Functions for Radial Basis Function Networks. Technical report, Institute for Adaptive and Neural Computation (University of Edinburgh), <http://www.anc.ed.ac.uk/mjo/software/rbf2.zip>, June 1999.
- [212] J. Friedman. Multivariate Adaptive Regression Splines. *Annals of Statistics*, 19, pp. 1–141, 1991.
- [213] D. Harrison and D. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5, pp. 81–102, 1978.
- [214] R. Quinlan. Combining Instance-based and Model-based Learning. In P. Utgoff (editor), *Proceedings of the 10th International Conference on Machine Learning*, pp. 236–243. Morgan Kaufmann, University of Massachusetts, Amherst, USA, 1993.
- [215] P. Ein-Dor and J. Feldmesser. Attributes of the performance of central processing units: a relative performance prediction model. *Communications of the ACM*, 30(4), pp. 308–317, 1987.
- [216] J. Friedman and W. Stuetzle. Projection Pursuit Regression. *Journal of the American Statistical Association*, 76, pp. 817–823, 1981.
- [217] N. Allsop, P. Besley and L. Madurini. Overtopping Performance of Vertical and Composite Breakwaters, seawalls and low reflection alternatives. In *Final MCS Workshop*. Alderney, May 1995.

- [218] J. van der Meer and J. Janssen. Wave run-up and wave overtopping at dikes. In N. Kobayashi and Z. Demirbilek (editors), *Wave Forces on Inclined and Vertical Wall Structures*, pp. 1–27. ASCE, 1995.
- [219] P. Aminti and L. Franco. Wave overtopping on rubble mound breakwaters. In B. Edge (editor), *Proceedings of the 21st International Conference on Coastal Engineering*, pp. 770–781. ASCE, New York, Malaga, Spain, June 21, 1988.
- [220] J. Pedersen and H. Burcharth. Wave forces on crown walls. In B. Edge (editor), *Proceedings of the 23rd International Conference on Coastal Engineering*, pp. 1489–1502. ASCE, New York, Venice, Italy, October 4–9, 1992.
- [221] R. Fletcher. *Practical methods of optimization*. John Wiley, Chichester, UK, second edition, 1987.
- [222] W. Press, B. Flannery, S. Teukolsky and W. Vetterling (editors). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, 1988.

Appendix A

Empirical Curve-Fitting

A.1 Besley

A commonly used curve used for estimating overtopping volumes is due to Besley [26]. It was given in section 1.3 and is repeated as equation (A.1) for convenience.

$$q_0 = AT_0 \exp\left(\frac{-BR_0}{T_0}\right) \quad (\text{A.1})$$

where $q_0 = q / (gH_{m0,toe})^{0.5}$ is the dimensionless overtopping discharge, $R_0 = R_c / H_{m0,toe}$ is the dimensionless freeboard, $T_0 = T_{m-1,0toe} (g / H_{m0,toe})^{0.5}$ is the dimensionless mean wave period, q is the mean overtopping discharge rate in $m^3/s/m$, R_c is the crest freeboard, $H_{m0,toe}$ is the significant wave height at the toe of the wall, $T_{m-1,0toe}$ is the mean wave period at the toe of the wall and g is the acceleration due to gravity.

This equation may be applied to smooth impermeable walls with slopes between 1:1 and 1:5 with normal wave approach. Besley follows Allsop *et al.* [217] in using different equations for vertical walls. First he defines a parameter h_* , designed to determine whether waves are mainly impacting or reflecting.

$$h_* = 2\pi \left(\frac{h_0^2}{T_0^2}\right) \quad (\text{A.2})$$

where h_0 is the dimensionless water depth at the toe of the wall. If h_* is greater than 0.3, reflecting waves predominate and the dimensionless discharge is given by equation A.3

$$q_0 = 0.05e^{-2.78R_0} \quad (\text{A.3})$$

If h_* is less than or equal to 0.3, impacting waves predominate and the dimensionless discharge is described by equation A.4.

$$\frac{q_0}{h_*^2} = 0.000137 \left(\frac{R_0}{h_*} \right)^{-3.24} \quad (\text{A.4})$$

Modifications are introduced for angled wave attack and for composite vertical walls. Again the exponential form of the equations is retained.

A.2 Van der Meer and Janssen

Van der Meer and Janssen [218] distinguish the behaviour of 2 different types of wave breaking on sloping structures. They define a 'breaker parameter' ξ by

$$\xi = \frac{\tan(\alpha)}{\sqrt{s}} \quad (\text{A.5})$$

where α is the structure slope and s is the wave steepness, calculated using

$$s = \frac{2\pi H_{m0,toe}}{gT_{m-1,0toe}^2} \quad (\text{A.6})$$

For values of ξ less than 2 plunging waves predominate and overtopping volumes are determined by equation A.7, in which Q_b and R_b are dimensionless quantities related to q_0 and R_0 by equations A.8 and A.9 and the γ values are empirical reduction factors for the berm width, water depth at the toe, friction and angle of wave attack.

$$Q_b = 0.06e^{-4.7R_b} \quad (\text{A.7})$$

$$Q_b = q_0 \sqrt{\frac{s}{\tan(\alpha)}} \quad (\text{A.8})$$

$$R_b = R_0 \frac{\sqrt{s}}{\gamma_b \gamma_h \gamma_f \gamma_\beta \tan(\alpha)} \quad (\text{A.9})$$

For values of ξ greater than 2 surging waves predominate and the relationship of equation A.10 holds.

$$Q_n = 0.2e^{-2.3R_n} \quad (\text{A.10})$$

In this equation Q_n is identical to q_0 and R_n is defined by equation A.11.

$$R_n = \frac{R_0}{\gamma_b \gamma_h \gamma_f \gamma_\beta} \quad (\text{A.11})$$

A.3 Hedges and Reis

As the examples above show, it is possible to make different choices of dimensionless quantities. These quantities may be indicated generically by an asterisk subscript, e.g. R_* , Q_* . Further, some researchers have used a power relationship rather than an exponential relationship between R_* and Q_* , resulting in relationships in the form of equation A.12 [6, 219, 220].

$$Q_* = AR_*^{-B} \quad (\text{A.12})$$

However, all of the approaches described thus far have 2 drawbacks-

- The parameters A and B are determined entirely empirically. No ‘meaning’ may be attached to these parameters, and there is no theoretical justification for their choice, or for the mathematical form taken by the predictive equations.
- Known boundary conditions are not met. Specifically, infinite overtopping rates are predicted by equation A.12 as R_* approaches 0, while zero overtopping levels are not predicted for any finite levels of R_* by equations A.1, A.7, A.10 or A.12.

Hedges and Reis attempted to solve these problems by forming a model based upon a two-step process [6]. In the first step, the water surface elevation η is calculated. If this exceeds the crest freeboard R_c , the instantaneous overtopping rate q is then calculated using the weir formula of equation A.13, in which C_d is a discharge coefficient.

$$q = \frac{2}{3} C_d \sqrt{2g} (\eta - R_c)^{3/2} \quad (\text{A.13})$$

Hedges and Reis approximated the wave shape by a saw-tooth and estimated a mean overtopping rate. This may be expressed in the form of equation A.14, in which Q_* is the dimensionless overtopping rate, $q / \sqrt{gR_{max}^3}$. R_* is the dimensionless crest freeboard defined by equation A.15 and R_{max} is the maximum run-up induced by the waves. The latter is defined by equation A.16, in which C is a coefficient that is dependent upon the wave conditions. C may adopt different values, allowing for random wave conditions.

$$Q_* = \begin{cases} A(1 - R_*)^B & \text{if } R_* < 1 \\ 0 & \text{if } R_* \geq 1 \end{cases} \quad (\text{A.14})$$

$$R_* = \frac{R_c}{R_{max}} \quad (\text{A.15})$$

$$R_{max} = CH_{m0,toe} \quad (\text{A.16})$$

The coefficient B may also be related to the shape of the waves. In the case of saw-tooth waves it adopts the value 2.5, but different values may be applicable to alternative wave conditions. The value of A is the expected dimensionless overtopping rate given a crest freeboard of zero.

The Hedges and Reis equation (equation A.14) has a number of advantages. The coefficients A , B and C have some theoretical justification and may be understood in a physical sense. Overtopping volumes are predicted to be zero if the maximum run-up does not exceed the crest freeboard. Finally, Q_* does not tend towards ∞ when R_* is very low, but is constrained to be less than or equal to A .

The results obtained using this equation are similar to those obtained with the Besley equation (equation A.1), except for low overtopping rates, where the Hedges and Reis equation gives lower predictions, which are generally closer to the measured overtopping rates. The approach of Hedges and Reis may be considered an improvement on earlier approaches. However, it still suffers from the drawbacks shared by all curve-fitting approaches, i.e. it is only applicable to a limited range of structures and is still constrained by the form of the predictive equation used.

Appendix B

The Conjugate Gradient Method

The conjugate gradient method [221, 85, 86] aims to provide more efficient gradient descent than the back-propagation algorithm, by ensuring that each weight change is made in a direction that is conjugate to all previous steps, with respect to the approximating function.

In order for the current search direction \mathbf{s} to be conjugate to the previous search direction \mathbf{r} the condition of equation B.1 must be met, in which \mathbf{H} is the Hessian matrix for the weights with respect to the error.

$$\mathbf{r}^T \mathbf{H} \mathbf{s} = 0 \quad (\text{B.1})$$

However, it possible to calculate conjugate search directions without explicitly calculating the Hessian. If the error gradient at the current point is \mathbf{g}_i , a search direction that is conjugate to the previous search direction, \mathbf{s}_i , is that described by equations B.2-B.3. Thus each search direction is a combination of the current steepest gradient and the previous search directions. For this reason, the new search direction is orthogonal to *all* previous gradients and search directions, and gradient descent will not be repeated along previous search directions.

$$\gamma = \frac{(\mathbf{g}_i - \mathbf{g}_{i-1}) \cdot \mathbf{g}_i}{\mathbf{g}_{i-1} \cdot \mathbf{g}_{i-1}} \quad (\text{B.2})$$

$$\mathbf{s}_i = \mathbf{g}_i + \gamma \mathbf{s}_{i-1} \quad (\text{B.3})$$

The distance to travel along the search direction may be calculated using line minimisation. One way to do this would be to use Newton's method to find the minimum error along the search direction, as in equations B.4-B.5.

$$\alpha = -\frac{\mathbf{g} \cdot \mathbf{s}}{\mathbf{s}^T \mathbf{H} \mathbf{s}} \quad (\text{B.4})$$

$$\Delta w = \alpha \mathbf{s} \quad (\text{B.5})$$

However, a method that is computationally less expensive is to approximate the error surface around the minimum by a parabola and then use Brent's method [222]. This successively narrows the bounds of the line search until the error is sufficiently close to a minimum value.

The conjugate gradient method is not a full second-order method, like the Levenberg-Marquardt method, but it uses second-order information to guide both the search direction and the distance to move along the search direction. The number of epochs required to achieve convergence is therefore substantially less than that required by back-propagation, but the memory and computational power requirements are much less than those for the Levenberg-Marquardt algorithm.

Appendix C

Comparisons with alternative methods

In the main body of this work, the results obtained from GL-ANNs have been compared with those from traditional neural networks: MLP and RBF networks. In this appendix, comparisons are made with other methods. The synthetic datasets used in Chapter 8 were used by Cohen and Intrator to test their perceptron radial basis nets (PRBFNs) [107]. The measured datasets in the same chapter were used by Quinlan to test the effectiveness of combined model-based and instance-based approaches [214]. The rest of this appendix briefly describes the approaches used in the earlier papers and compares their results with those obtained from the GL-ANN method.

C.1 Synthetic datasets

Cohen and Intrator's method for creating hybrid networks has been described in section 1.5.8. They cluster the available data and then choose either a sigmoid or RBF neuron to describe the data within each cluster. The splits between training and test data that they used were duplicated in the work reported in Chapter 8, so comparison between GL-ANNs and PRBFN is straightforward. Cohen and Intrator also quote results using Orr's regression tree-RBF method. Table C.1 gives the errors for PRBFN and RT-RBF reported by Cohen and Intrator. GL-ANN results are also repeated for convenience.

The GL-ANNs are seen to give lower errors than the other two methods with all but the 1-D sinewave. However, due to the noise added to the test data this dataset has a theoretical minimum error of 0.01. The results quoted by Cohen and Intrator cannot therefore be taken at face value.

Table C.1: MSEs for the synthetic datasets obtained using PRBFN, RT-RBF and GL-ANN algorithms

	1D sine	2D sine	impedance	Hermite
PRBFN	0.0066	0.00128	0.150	0.00150
RT-RBF	0.0088	0.00228	0.112	0.00152
GL-ANN	0.0120	0.00111	0.098	0.00130
Minimum error	0.01	-	-	-

Table C.2: Relative error, as a percentage, for the measured benchmark datasets, trained using Quinlan’s methods and GL-ANNs

	housing	servo	cpu	auto-mpg
Regression tree	18.6	28.7	17.2	14.7
Tree+instances	16.5	16.5	12.0	16.0
Neural network	13.6	11.4	11.0	12.5
Network+instances	12.9	10.6	11.1	13.4
GL-ANN	12.1	6.3	9.3	12.1

C.2 Measured datasets

Quinlan was interested in combining prediction based on models, such as regression trees and neural networks, with methods based on instances, or prototypes. He observed that the incorporation of prototype information often improves the performance of model-based approaches. This phenomenon is analogous to the improved performance of GL-ANNs over MLPs, since the RBF model is similar to a prototype approach (see section 6.2). The neural networks used were MLPs.

Quinlan used a tenfold cross-validation method and reported his results in terms of the ‘relative error’. This was defined as the mean squared error divided by the variance of the target values, and was expressed as a percentage. In order to allow a fair comparison between methods, a new set of GL-ANNs have been developed, using the same procedure. The results are reported in table C.2, with the results from Quinlan’s paper [214].

The GL-ANNs are seen to outperform the regression trees substantially on all datasets. When compared to the MLPs reported in Quinlan’s paper, the results using GL-ANNs are seen to be a slight improvement in most cases, with a substantial performance increase for the servo dataset. These results are similar to those obtained

using the alternative MCCV method, reported in Chapter 8.

Appendix D

Published Papers

There follow three papers that have been published or are currently in press:

D. Wedge, D. Ingram, D. McLean, C. Mingham and Z. Bandar. On Neural Network Architectures and Wave Overtopping. In *Maritime Engineering*, 158 (MA3), pp.123-133. Thomas Telford, London. September 2005.

D. Wedge, D. Ingram, D. McLean, C. Mingham, and Z. Bandar. A Global-Local Artificial Neural Network with Application to Wave Overtopping Prediction. In W. Duch et al. (editors) *Proceedings of the International Conference on Artificial Neural Networks*, pp.109-114. Springer-Verlag, Berlin Heidelberg. Warsaw, Poland, September 11-15, 2005.

D. Wedge, D. Ingram, D. McLean, C. Mingham, and Z. Bandar. On Global-Local Artificial Neural Network for Function Approximation. In *IEEE Transactions on Neural Networks*. 2006 (in press)