

**Advanced Bayesian Neural Network Classifiers
of Head-movement Directions for Severely
Disabled People**

by

Son Thanh Nguyen

Submitted to the Faculty of Engineering
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
at the University of Technology, Sydney

Sydney, August 2006

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree, except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Candidate

Production Note:
Signature removed prior to publication.

Acknowledgements

First of all, I am very grateful to my supervisor, Professor Hung Tan Nguyen, for giving me the valuable opportunity to work with assistive technologies and for his expert guidance. I have greatly benefited from his continuous support, including his knowledge and spirit.

I greatly appreciate Philip Taylor's assistance during the phase of data collection. I would also like to thank Leslie King for his help in setting up the hardware system.

Many thanks to Pat Skinner for her great help in editing this thesis.

Finally, I would like to dedicate this thesis to my mother, my older brother, my sister-in-law, my nephew, and niece, who always encouraged me during my stay in Sydney.

Son Nguyen

Sydney, August 2006

Contents

Nomenclature	vii
List of Figures	x
List of Tables	xiii
Abbreviations	xv
Abstract	xvii
Chapter 1. Introduction	1
1.1 Problem statement	1
1.2 Objectives of the thesis	4
1.3 Thesis contributions	4
1.4 The structure of the thesis	6
1.5 Publications related to the thesis	8
Chapter 2. Literature Review	9
2.1 Introduction	9
2.2 Neural networks enhancing the adaptability of assistive devices	10
2.3 Standard neural networks	12
2.4 Bayesian neural networks	13
2.5 Developed hands-free wheelchair control methods	15
2.5.1 Speech-based control	15

2.5.2	Chin-based control	17
2.5.3	Gesture-based control	17
2.5.3.1	Face direction	18
2.5.3.2	Head movement	20
2.5.3.3	Eye wink	21
2.5.4	Brain-signal-based control	21
2.6	Assistive wheelchairs	22
2.6.1	Recent and current developments	22
2.6.2	Basic components of assistive wheelchairs	24
2.6.3	Multiple operating modes in assistive wheelchairs	25
2.7	Discussion	27

Chapter 3. Mobility Assistance after Spinal-cord Injury Using Neural

	Networks	29
3.1	Introduction	29
3.2	Spinal-cord injury	30
3.2.1	The type of injury	30
3.2.2	The location of the injury	31
3.2.2.1	Cervical injuries	32
3.2.2.2	Thoracic injuries	32
3.2.2.3	Lumbar and sacral injuries	32
3.3	Functional restoration of upper extremities after spinal cord injury	34
3.4	Hands-free device access	35
3.5	Neural networks	36
3.5.1	Artificial neurons	38
3.5.2	Network architectures	39
3.5.2.1	Feed-forward networks	39
3.5.2.2	Feedback networks	39
3.5.2.3	Multi-layer networks	40
3.5.3	Perceptrons.....	41
3.5.4	Network learning	42
3.5.4.1	The type of training	42
3.5.4.2	Error back-propagation	43

3.5.4.3	Generalisation and regularisation	43
3.5.5	Two-layer perceptron classifiers	45
3.5.5.1	Forward propagation	45
3.5.5.2	Error functions	47
3.5.5.3	Network regularisation	49
3.5.5.4	Error gradient	49
3.5.5.5	The Hessian matrix	52
3.5.5.6	Adjusting the weights and biases.....	55
3.6	Discussion	56
Chapter 4. A Bayesian Neural Network for Detecting Head-movement		
	Commands	58
4.1	Introduction.....	58
4.2	The Bayesian neural network classifier	61
4.2.1	Prior weight distribution	61
4.2.2	Posterior weight distribution	62
4.2.3	Posterior probability of the hyperparameters	63
4.2.4	The Gaussian approximation to the evidence	64
4.2.5	Determination of the hyperparameters	65
4.3	Advanced optimisation training algorithms for Bayesian neural network classifiers	66
4.3.1	Line search	68
	4.3.1.1 Bracketing the minimum	68
	4.3.1.2 Locating the minimum itself	69
4.3.2	Conjugate-gradient training algorithm.....	70
4.3.3	Scaled conjugate-gradient training algorithm	72
4.3.4	Quasi-Newton training algorithm	74
4.3.5	Training time	76
4.4	Training Bayesian neural networks	77
4.4.1	Data acquisition	78
4.4.2	Network specifications	82
4.4.3	Experiment 1	82
4.4.4	Experiment 2.....	87

4.5	Discussion	90
Chapter 5. Optimal Adaptive Bayesian Neural Network Classifiers		91
5.1	Introduction	91
5.2	Evaluating the evidence	95
5.2.1	The Occam factor for the weights and biases	96
5.2.2	The Occam factor for the hyperparameters	97
5.2.3	Combining the terms of evidence	98
5.3	Adaptive Bayesian neural networks	99
5.3.1	Head-movement data	99
5.3.2	Training Bayesian neural networks on high-performance computational clusters	99
5.3.3	The optimal network architecture	101
5.3.4	Head-movement classification	105
5.3.4.1	Experiment 1	105
5.3.4.2	Experiment 2 (Adaptive network training)	107
5.4	Discussion	109
Chapter 6. Alternative Frameworks for Improving the Accuracy of Detecting Head-movement Commands		111
6.1	Introduction.....	111
6.2	Maximum evidence of early stopping of neural networks	112
6.2.1	The evidence for neural networks	113
6.2.2	Prior weight distribution	114
6.2.3	The data-set likelihood	115
6.2.4	Evaluating the evidence	115
6.2.5	Training neural networks for detecting head-movement commands	116
6.2.5.1	The use of data	116
6.2.5.2	Network architecture	117
6.2.5.3	Network training	117
6.3	The Monte Carlo methods for Bayesian neural network training	121
6.3.1	The Monte Carlo methods	121

6.3.2	The Metropolis-Hasting algorithm	122
6.3.3	Training Bayesian neural networks for detecting head- movement commands	126
6.4	Combining the performances of independent Bayesian neural network classifiers	128
6.4.1	Bayes' rule of combination	129
6.4.2	Dempster-Shafer theory of combination	131
6.4.3	Combining two Bayesian neural network classifiers	133
6.4.3.1	Experiment 1 (Bayes' rule of combination)	134
6.4.3.2	Experiment 2 (Dempster-Shafer theory of combination)	136
6.5	Discussion	138
Chapter 7. Conclusions and Future Work		139
Appendix A. Power Wheelchairs		144
A.1	Overview of power wheelchairs	144
A.2	Power wheelchair components	145
A.2.1	Joystick and alternative control	145
A.2.2	Wheelchair battery	147
A.2.3	Wheelchair motors	147
A.2.4	Motor drive	149
A.2.5	Drive train	150
A.2.6	Control module	150
Appendix B. ADXL202EB-232A Evaluation Board		153
B.1	Hardware	153
B.2	Software	154
B.3	Applications	154
Appendix C. GPSB Wheelchair Direction-Control Device		155

Appendix D. National Instrument USB-6008 Multifunction Data	
	Acquisition Module 157
D.1	Hardware 157
D.2	Software 163
D.3	Applications 163
Appendix E. The Advanced Bayesian Neural Network Toolbox for Matlab.. 164	
Appendix F. Implementation of the Bayesian Neural Network based Real- Time Head-Movement Command Detection 244	
Appendix G. Publications related to The Thesis 253	
Bibliography	

List of Figures

2.1	Block diagram of a voice recognition system.....	16
2.2	Face direction computation	19
2.3	The configuration of a head-oriented wheelchair control	20
2.4	Diagram of an eye-wink control interface	21
2.5	Functional diagram of assistive wheelchairs	24
3.1	The synergic controller for restoring elbow extension	35
3.2	Diagram of a typical hands-free user interface with the use of neural networks	36
3.3	An artificial neuron	38
3.4	A feed-forward neural network	39
3.5	A feedback neural network	40
3.6	A perceptron	41
3.7	An illustration of neural network generalisation	44
3.8	A two-layer perceptron classifier	45
4.1	Block diagram of head-movement-based direction control of power wheelchair using a Bayesian neural network	59
4.2	Dual-axis tilt sensor (ADXL202EB-232A)	60
4.3	LabVIEW real-time computer interface	60
4.4	Data-acquisition and direction-control devices	61
4.5	Golden section search	69
4.6	An illustration of the process of parabolic interpolation used to perform line-search minimisation	70

4.7	Windowed samples of user 1 (able-bodied person)	79
4.8	Windowed samples of user 5 (C5-injury-level person)	80
4.9	Windowed samples of user 6 (C4-injury-level person)	81
4.10	Averaged network training time of the three training algorithms	84
4.11	Averaged convergence time of the three training algorithms	84
4.12	Cost-function value versus training cycle in Experiment 1	86
4.13	Cost-function value versus training cycle in Experiment 2	88
5.1	A common topology of PC clusters	100
5.2	Schematic of the clusters in Faculty of Engineering (UTS)	101
5.3	Log evidence versus number of hidden nodes	104
5.4	Test error versus log evidence	104
5.5	Cost-function value versus training cycle in Experiment 1	106
5.6	Cost-function value versus training cycle in Experiment 2	108
6.1	An illustration of the traditional early stopping in neural networks	113
6.2	Data error versus training cycle	119
6.3	Log evidence versus training cycle.....	119
6.4	An illustration of the Monte Carlo methods	124
6.5	The histogram of twenty samples points generated from the zero- mean Gaussian distribution	125
6.6	Data error versus training cycle	127
6.7	Log evidence versus training cycle	127
6.8	Block diagram of a system for detecting commands through the combination of three hands-free input signals.....	129
6.9	A two-classifier recognition system	130
6.10	A two-Bayesian neural network recognition system	133
A.1	A standard power wheelchair (Invacare Roller M1)	144
A.2	Functional block diagram of power wheelchairs	145
A.3	Schematic of the standard joystick with error checking circuitry	146
A.4	PMDC motor for power wheelchairs (Shihlin Electric manufacturer)..	148
A.5	Block diagram of the motor drive	149

A.6	H-bridge configuration provides bidirectional rotation for the wheelchair motor	149
A.7	Velocity control for power wheelchairs	151
B.1	ADXL202EB-232A evaluation board with RS232 connection cable ..	153
C.1	GPSB device	155
C.2	Functional block diagram of wheelchair direction and speed-control using the GPSB device	156
D.1	Key functional components of the NI USB-6008.....	158
D.2	Signal label application diagram of the NI USB-6008	159
E.1	Structure of the Advanced Bayesian Neural Network Classification Toolbox for Matlab	165

List of Tables

2.1	Some recent and current assistive wheelchair systems	23
3.1	Number of people living with spinal-cord injuries in Australia and the United States of America	30
3.2	Spinal-cord injuries from trauma causes in Australia and the United States of America	30
3.3	The spinal-cord map	33
3.4	Milestones in the development of neural networks	37
4.1	Numbers of extracted head-movement samples of the eight wheelchair users	79
4.2	A relative comparison of the three training algorithms	85
4.3	The change of the hyperparameters according to five re-estimation periods in Experiment 1	87
4.4	Confusion matrix of the trained network in Experiment 1	87
4.5	The change of the hyperparameters according to five re-estimation periods in Experiment 2	89
4.6	Confusion matrix of the trained network in Experiment 2	89
4.7	Sensitivity and specificity of the trained networks in the two experiments	89
5.1	Numbers of extracted head-movement samples of the eight users	99
5.2	The change of the hyperparameters according to five re-estimation periods in Experiment 1	107

5.3	Confusion matrix of the trained network in Experiment 1	107
5.4	The change of the hyperparameters according to five re-estimation periods in Experiment 2	109
5.5	Confusion matrix of the trained network in Experiment 2	109
5.6	Sensitivity and specificity of the trained networks in the two experiments	109
6.1	Numbers of extracted head-movement samples of the eight users	117
6.2	Network training time measured over the training runs	120
6.3	Confusion matrix of the best trained network	121
6.4	Sensitivity and specificity of the best trained network	121
6.5	Confusion matrix of the best trained network	128
6.6	Sensitivity and specificity of the best trained network	128
6.7	Confusion matrix of Bayesian neural network classifier 1	135
6.8	Confusion matrix of Bayesian neural network classifier 2	135
6.9	Confusion matrix of the combination of two Bayesian neural network classifiers	135
6.10	Sensitivity and specificity of the classifiers	136
6.11	Confusion matrix of Bayesian neural network classifier 1	137
6.12	Confusion matrix of Bayesian neural network classifier 2	137
6.13	Confusion matrix of the combination of two Bayesian neural network classifiers	137
6.14	Sensitivity and specificity of the classifiers	138
C.1	Specifications of the GPSB device	156
D.1	Analog terminal assignments of the NI USB-6008	160
D.2	Digital terminal assignments of the NI USB-6008	161
D.3	The signals available on the I/O connectors of the NI USB-6008	162
E.1	List of the functions in the toolbox	166
E.2	List of the demonstrations in the toolbox	167

Abbreviations

ANN	Artificial neural network
ADC	Analog to digital converter
BNN	Bayesian neural network
BCI	Brain computer interface
DGF	Direction of greatest freedom
EWCI	Eye-wink control interface
EEG	Electroencephalogram
EMG	Electromyography
FNS	Functional Neuromuscular Stimulation
IDDM	Insulin-Dependent Diabetes Mellitus
MLP	Multi-layer perceptron
PCA	Personal-care assistance
SCI	Spinal-cord injury
SNN	Standard neural network

Abstract

Assistive technologies have been dedicated to providing additional accessibility to individuals who have physical or cognitive difficulties, impairments and disabilities. Various types of assistive technology products are also available on the market today. However, there are still a significant number of disabled people who are unable to use commercial assistive devices due to their high level of injury.

For severely disabled people with quadriplegia resulting from high-level spinal-cord injuries or cerebral palsy, hands-free control methods have become extremely beneficial for them to reducing their dependence level in daily activities. In this control mode, head movement has been shown to be a very effective, natural and comfortable way to access the device. The need to exactly detect intentional head movements of various forms of disabled people has led to the use of neural networks. Recently, Bayesian neural networks have been proposed for developing neural network applications with finite and/or noisy training data.

In assistive technologies, power wheelchairs are a means of providing independent mobility. This thesis explores the useful properties of Bayesian neural networks in developing an optimal head movement-user interface for hands-free power wheelchair control systems. In such systems, a trainable Bayesian neural network is used to detect head movement commands. This kind of user interface can conveniently be used by various disabled users. The thesis also proposes the techniques for developing the adaptive Bayesian neural network for head movement classification, including the determination of the optimal architecture and the most effective on-line training algorithm for the network.

The experimental results obtained in the thesis show that a Bayesian neural network can be used to detect head movements accurately and consistently. After on-line training, the network is able to adapt well to the head movements of new users. The substantial contributions of the thesis can briefly be summarised as follows:

- Standard methods of neural network training usually require intensive search for network parameters or require the use of a validation set separated from the available training data. In contrast, in this thesis all the available training data have been used to train the network for detecting head movements. As the network could be trained on all the data from a group of different individuals, it is able to classify their new head movements with a very high accuracy, of 99.38%.
- The thesis strongly focuses on advanced training algorithms suitable for Bayesian neural network head-movement classifiers. Especially, the quasi-Newton training and scaled conjugate gradient algorithms have been found to be the most effective, as they can result in the shortest training time for the network.
- In general, the determination of the best network architecture is very difficult and is traditionally based on ad hoc methods. However, this thesis utilises the property of the maximum evidence, which is available in Bayesian neural networks, to select the optimal network architecture. Specifically, the networks containing three hidden neurons are appropriate for successful head-movement classification.
- The thesis also provides a novel method of early stopping in neural networks. In this method, the maximum evidence can be seen as a good criterion to terminate the training process before the network overfits the data. In addition, the use of a validation set for monitoring the generalisation error is no longer needed. Moreover, this thesis shows that the combination of independent Bayesian neural networks can significantly improve the head-movement classification accuracy.

Chapter 1

Introduction

1.1 Problem statement

Independent activities in the daily life of a person play a crucial role in his or her physical, cognitive and social development. In Australia, there are more than 10,000 people living with spinal-cord injuries (SCIs). High-level SCIs cause losses in independent mobility or feeling. Frequent causes of SCIs are trauma (car accident, falls, diving) or diseases (polio, spina bifida, Friedreich's Ataxia). In 2002 - 2003, there were 394 new injuries, 92% of which resulted in neurological loss; 62% of the injuries were from traumatic causes, and of those, 52% were caused by motor vehicles, followed by falls (34%), water and other sports (13%). Males aged from 15 - 34 had the highest number of injuries (Cripps 2004).

People with severe SCIs resulting in paraplegia and quadriplegia often need some form of personal-care assistance (PCA) to help them in daily activities such as getting in or out of bed, bathing, dressing, driving, shopping or cleaning. Individuals with C5/C6 injury levels may need synergic devices to enhance elbow extensions (Giuffrida and Crago 2005). Above the C4 injury level, people may require a ventilator to breathe.

Assistive technologies have been dedicated to providing additional accessibility to individuals who have physical or cognitive difficulties, impairments and disabilities. Various types of assistive technology products are also available on the market today. However, there are still a significant number of disabled people who are unable to use commercial assistive devices due to their high level of injury.

For severely disabled people with quadriplegia resulting from high-level SCIs or cerebral palsy, hands-free control methods have become beneficial for them in reducing their dependence level in daily activities. Nowadays, achievements of assistive technologies can allow severely disabled people to activate and control home appliances such as lighting and television by consciously controlled changes in their head movements or brain signals (Craig, Moses et al. 2002). As a result, their quality of life is significantly improved.

Wheelchairs are an important form of mobility for people with disabilities. A person with a low C-level of SCI is able to use a manual wheelchair. However, high C-level injuries usually require the use of power wheelchairs. Some people with SCIs can use braces or crutches to move around, but these mobility methods do not mean that they will never use wheelchairs. In fact, those people still find wheelchairs more useful for longer distance mobility.

For many years, there have only been three wheelchair varieties: power wheelchairs, scooters and manual wheelchairs. The advantages of manual wheelchairs are that they are cheaper, lighter and more agile than power wheelchairs. However, power wheelchairs are useful for disabled individuals who have difficulties in using standard manual wheelchairs or find it impossible to use them, due to their pain, insufficient arm strength or inability to maintain the posture effective for manual wheelchair propulsion. Recent developments have allowed the propulsion of wheelchairs to be shared between the user and electric motors (Cooper, Corfman et al. 2002).

For many individuals with high-level SCIs, head movement has been found to be a natural way of pointing and can be utilised to control the travel direction of power wheelchairs effectively and comfortably. The need to exactly detect intentional head movements of individuals with different injury levels has led to the use of neural networks (Joseph and Nguyen 1998; Taylor and Nguyen 2003; H.T.Nguyen, King et al. 2004; Nguyen, King et al. 2004; S.T.Nguyen, H.T.Nguyen et al. 2004).

During the last decade, neural networks modelled based on the way the human brain processes information have attracted significant attention from many researchers and have been used in various fields such as radar target recognition (Lee, Choi et al. 2003), text-to-speech conversion (Karaali, Corrigan et al. 1998), credit-card fraud detection (Ghosh and Reilly 1994), industrial control systems (Fukuda and Shibata 1992), solar array modelling and maximum power point prediction (Al-Amoudi and Zhang 2000), etc. In medicine, neural networks have become a powerful tool for enhancing current diagnostic techniques (Tourassi, Floyd et al. 1999; Zhang, Liang et al. 2004).

In most applications of neural networks, multi-layer perceptron (MLP) neural networks are widely used. The crucial issue in developing the MLP neural network is generalisation - how well the network can make predictions for new cases that are not in the training data. A network that is not complex enough may ignore the data, leading to “underfitting”, while a network that is too complex may fit the noise, not just training data, leading to “overfitting”. The complexity of the network is concerned with the network architecture and magnitudes of network weights and biases.

Since MLP neural networks trained by standard back-propagation may cause poor generalisation, several frameworks have been proposed to prevent MLP networks from underfitting or overfitting (Reed 1993; Li, Chow et al. 1995; Takechi and Murakami 1995; Fletcher, Katkovnik et al. 1998; Arifovic and Gençay 2001; Shahjahan, Akhand et al. 2003). However, these frameworks require intensive searching for network parameters or do not make maximum use of the available data.

Recently, the Bayesian method has been proposed to enhance the generalisation abilities of MLP neural networks regardless of finite and/or noisy available data (MacKay 1992a; MacKay 1992b). MLP neural networks trained by the Bayesian method are called Bayesian neural networks (BNNs). In contrast with standard back-propagation learning, Bayesian learning considers the probability distribution of network parameters that can give the best generalisation of the trained network. Especially, this type of neural network can be trained on all of the available data.

So far BNNs have been applied to several areas: fault detection of cylinders (Marwala 2001), fat content measurement (Thodberg 1996), protein structure classification (Hunter and States 1992), fermentation control (Vivarelli, Serra et al. 2000), electric load forecast (Tito, Zaverucha et al. 1999). However, the exploration of the usefulness of BNNs in assistive technologies is still limited and promises a considerable amount of valuable outcomes.

1.2 Objectives of the thesis

- Firstly, as a large number of severely disabled people are able to use their head movements to control power wheelchairs easily and effectively, this thesis aims to demonstrate that BNNs can be used to develop an optimal head movement-user interface for hands-free wheelchair control systems. This kind of user interface is able to adapt well to various forms of wheelchair users.
- Secondly, the thesis intends to provide a unified theoretical framework for optimising different aspects of the BNN used to detect head movement.

1.3 Thesis contributions

First of all, the thesis proposes an optimal hands-free control power wheelchair system. The travel direction of the wheelchair system is controlled by the head movements of the user. In such a system, a BNN is responsible for detecting the head commands of the user.

In the past, different parameters of neural networks were usually optimised based on independent frameworks. In contrast, this thesis shows that different issues of neural network optimisation can be treated in a unique theoretical framework of BNNs. In addition, this framework can easily be implemented with various platforms of hardware and software. The substantial contributions of the thesis can be summarised as follows:

- Firstly, the thesis shows that the Bayesian evidence framework allows all available data to be used to train the BNN. This task is extremely important when the collection of relatively large data is expensive or time-consuming.

-
- The determination of the proper hyperparameters of the BNN requires training the network further, several times. Once the network is trained, there is a sequence of successive steps for searching a particular weight vector to minimise a cost function. The use of the traditional gradient descent algorithm with a fixed step size and search direction to search local minima usually causes an excessively large network training time. So the thesis focuses on three advanced optimisation training algorithms for the network. They are the conjugate gradient, scaled conjugate gradient and quasi-Newton algorithms, which are able to automatically adjust the step size and search direction to their optimal values. By analysing the properties of these training algorithms, the thesis shows that the quasi-Newton algorithm has the fastest rate of convergence.
 - The determination of the optimal network architecture is traditionally a time-consuming task, because it requires gathering significant data and comprehensive statistical knowledge to control the uncertainties of different candidature architectures. Fortunately, the Bayesian evidence framework itself involves a powerful tool for neural network model comparison. In particular, Bayesian model comparison, which embodies *Occam's razor*, can be used to pick up the best network architecture and the maximum *evidence* of the network is a trade-off between the best performance of the network on new data and the natural complexity of the network. As a result, the optimal architecture of the BNN in this research is conveniently determined by evaluating the evidence for various networks with varying numbers of hidden nodes. The architecture resulting in the highest evidence and the lowest test error is chosen for the final use.
 - The experimental results obtained in the thesis show that the determination of the optimal network architecture and the use of the most effective quasi-Newton training algorithm result in a promising framework for developing

BNNs that can be trained on-line. After training, these networks are able to adapt well to the head movements of new individuals.

- In this research, a modification of the Bayesian evidence framework is made to perform early stopping in neural networks. Specifically, the maximum evidence of the network model can be used as a good criterion to stop the network training before the overfitting occurs. Unlike traditional early stopping, a separated validation set is no longer required in this method. Finally, the thesis shows that the combination of independent BNNs can significantly improve the accuracy in detecting the head-movement commands.

1.4 The structure of the thesis

The thesis consists of seven chapters, a bibliography and six appendices. The remaining chapters of the thesis are organised as follows:

- *Chapter 2* presents the important role of neural networks in enhancing the adaptability of assistive devices. Commonly used techniques for neural network optimisation are also discussed. The enormous potential of BNNs is then highlighted. Finally, this chapter discusses hands-free input devices for wheelchair control and assistive wheelchairs developed.
- *Chapter 3* firstly provides some information about SCIs and neural network applications in mobility restoration after SCIs. The chapter also highlights some important features of neural networks. Finally, the chapter describes the principal properties of two-layer perceptron classifiers that can be used to detect head movements successfully.
- *Chapter 4* begins with a description of the head-movement-based wheelchair control system being developed in the University of Technology, Sydney (UTS). The heart of the system is a trainable BNN classifier responsible for detecting head-direction commands, and the main properties of the BNN classifier are then presented. Next, the chapter makes a detailed investigation

of three non-linear parameter optimisation algorithms: conjugate gradient, scaled conjugate gradient and quasi-Newton. These algorithms can be used to train the BNN for detecting head movements. A comparison of these algorithms is also performed based on the properties of the algorithms and experimental results to select the one that is the most appropriate for on-line network training tasks.

- *Chapter 5* describes the procedure of determining the optimal architecture for the BNN in this research. The procedure is based on evaluating the evidence for different network architectures with varying numbers of hidden nodes. The best network architecture is the one with the highest evidence. Finally, the chapter presents experiments on developing the adaptive BNN for detecting the head movements of new persons.
- *Chapter 6* proposes a novel method of early stopping in neural networks. This method is based on evaluating the evidence for the network every training cycle. In contrast with traditional early stopping, this method does not require a validation set. The chapter also describes an alternative implementation of BNNs based on the Monte Carlo sampling technique. Finally, the chapter describes two methods of combining independent BNNs, based on the Bayes' rule and Dempster-Shafer theory, in order to improve the accuracy in detecting head-movement commands.
- *Chapter 7* presents the overall conclusions for this research. The chapter also discusses the potentiality of BNNs in developing other applications in biomedical engineering. Finally, a framework for developing a shared wheelchair control system capable of predicting the user's plans in real-time is proposed.
- Finally, seven appendices are followed by the bibliography.

1.5 Publications related to the thesis

There are three fully refereed international conference papers of the Institute of Electrical and Electronics Engineers (IEEE) related to the thesis (See Appendix G).

They are as follows:

- Son Thanh Nguyen, Hung Tan Nguyen and Philip Taylor (2004). “Hands-Free Control of Power Wheelchairs Using Bayesian Neural Networks”. *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, Singapore, 1 - 3 December 2004, pp. 745 - 759 [ISBN 0-7803-8643-2] (CD-ROM).
- Son Thanh Nguyen, Hung Tan Nguyen and Philip Taylor (2006). “Bayesian Neural Network Classification of Head Movement Direction Using Various Advanced Optimisation Training Algorithms”. *Proceedings of the First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, Pisa, Italy, February 20 - 22, 2006, Paper 45, pp.1 - 6 [ISBN 1-4244-0040-6] (CD-ROM).
- Son Thanh Nguyen, Hung Tan Nguyen and Philip Taylor (2006). “Improved Head Direction Command Classification Using an Optimised Bayesian Neural Network”. *Proceedings of the 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, New York, USA, August 30 - September 3, 2006, pp. 5679 - 5682 [ISBN 14244-0033-3] (CD-ROM).

Chapter 2

Literature Review

2.1 Introduction

For many individuals with spinal cord injuries (SCIs), the use of a power wheelchair can restore the independent mobility they need to increase social participation. In order to control the power wheelchair properly with a standard manual joystick, the user must have at least some level of hand movement.

Recently, there have been several researches on the functional restoration of upper extremities after SCIs (Au and Kirsch 2000; Perreault, Crago et al. 2001; Giuffrida and Crago 2005). These researches are very promising for individuals who are able to retain voluntary control over some of the muscles of the shoulder, arm or elbow extension. However, high cervical-level SCIs usually result in complete paralysis of important muscles, including the pectoralis major, latissimus dorsi, and serratus anterior.

Neural networks are useful for real-time pattern recognition, multi-source information fusion, clustering and mapping data, adaptive control and building up system models without the need for explicit parameter or equation-model specification. In rehabilitation engineering, neural networks have become a powerful tool for functional restoration and prediction after SCIs (Au and Kirsch 2000; Giuffrida and Crago 2005).

The most challenging task in developing a neural network to solve a given problem is how to measure the complexity of the network. Traditional methods for analysing the performances of trained neural networks, such as confidence intervals, are rather primitive. Fortunately, the Bayesian school can provide a theoretical framework for

dealing with the network complexity by combining the evidence from the observed data to the prior knowledge about the problem (MacKay 1992a; MacKay 1992b; Thodberg 1996; Penny and Roberts 1999; Wright 1999).

During the past few years, a number of hands-free system access devices have been developed. These devices allow disabled people not only to control power wheelchairs, but also to activate and control aspects of their home environment.

Assistive wheelchairs have been developed to provide safe and easy independent mobility for individuals who lack upper body strength or have cognitive impairments. In such systems, the user is responsible for global planning and coarse control, whilst the “intelligent” wheelchair controller is responsible for exact and safe motion planning. Moreover, some assistive wheelchairs are capable of changing the degree of control shared between the user and the machine during task executions.

2.2 Neural networks enhancing the adaptability of assistive devices

Assistive devices such as power wheelchairs can be seen as human - machine systems. The usefulness of such a system depends highly on how the system can communicate with the user. The ability of the system to effectively respond to the user is usually influenced by two factors. The first is the ability to form an appropriate model of the user. The second is the ability to communicate and receive feedback from the user across a set of information-bearing channels.

The user interface plays a very important role in obtaining an effective and natural human - machine interaction. Thus, many technologies have emerged to realise such an interface. Also, human - machine systems can be analysed and modelled in several ways (Abbas and Kuo 1990; Chan and Childress 1990a; Chan and Childress 1990b; Aigner and McCarragher 2000).

As human - machine interactions involve communication between the human operator and the machine, they can be modelled using the information theory developed by

Shannon (Shannon 1948). The information measure is described in terms of uncertainty, the uncertainty at the output as to the input. When this kind of uncertainty is applied to human - machine systems, it is called “human - machine noise” (Chan and Childress 1990a).

When a number of randomly arriving tasks need to be performed in a fixed mission time, “stochastic models” for multi-component human - machine systems can be used to analyse the system (Abbas and Kuo 1990). For instance, each component can represent hardware or software. Therefore, these models are able to handle all of the items in the system involving hardware, software or both. In particular, they can combine the following performance measures from human, hardware and software components: 1) the availability of a human - hardware - software system at the task request time, 2) the reliability of a human - hardware - software system during the performance time, 3) the human performance variables, and 4) the system design failure.

In human - machine system modelling, the nature of the human operator should be considered. Human factors, including erroneous behaviour, decision-making and performance are a part of *shared control*, and the task of creating a “user-adaptive model” is, therefore, necessary. Ideally, this model should be able to incorporate distinguishing features of the user in terms of significant cognitive and/or perceptual variables. The model should also include some measure of human performance in the learning task. It is clear that this model must be dynamic and updated using feedback from the user - system interaction. Also, it is very useful to know the types and skills of different classes of users (Maren 1991).

Despite there being different techniques of modelling human operator’s performance, with each technique having its own strengths and applications, there are some weaknesses shared by all. Firstly, analyses that go into building human operator models are quite intensive and expensive. The data which the model is based upon is a statistical representation of a human’s performance through many representations of a set of tasks. This means that the model is only correct for the exact tasks in the

statistical base. Secondly, these models are unable to handle uncertainties well. Under conditions where a human operator might make a mistake, the model may not make a mistake or it may make an inappropriate mistake.

Research in neural networks has recently been active as a new means of developing human - machine systems. Artificial neural networks try to mimic the biological brain in the mathematical models. The brain is a large-scale system connecting many neural cells (neurons) and has excellent characteristics: parallel processing of information, learning functions and self-organisation capabilities. The brain can also provide an associative memory and is good for information-processing, such as pattern recognition. Neural networks such as MLP networks will be best for extracting and recognising patterns in real-time. Finally, neural networks excel at being able to work with partially incomplete and/or noisy data.

Neural networks are, therefore, very powerful for creating assistive devices that are able to be used by everyone in respect of their levels of physical and cognitive impairments. A neural network trained on data derived from human subjects performing a set of tasks will facilitate the interpretation of data clusters describing different user types. In other works, neural networks can be used to enhance the adaptability of assistive devices.

2.3 Standard neural networks

The objective of neural network learning is to search a set of weights which gives a mapping that fits the training set well. We also hope that, after training, the network can generalise well to new examples. Standard neural network (SNN) learning involves the use of one of the gradient descent methods to minimise errors between actual network outputs and expected outputs. Modifications of this technique also include the use of the “momentum” term.

Modelling with flexible models such as neural networks requires carefully controlling the model’s complexity and generalisation ability. For example, without any technique to control the network complexity, as the number of hidden nodes of the network is

increased, the network will fit the noise of the data, leading to an overfitting. There are a number of methods to avoid overfitting. For example, conventional model selection methods and regularisation can be applied.

Conventional model selection for neural networks requires choosing the number of hidden nodes of the network and the connections thereof. The usual approach to model selection is the *pruning technique* (Reed 1993), in which we start a large model and remove connections of hidden nodes during training (Sankar and Mammone 1991; Takechi and Murakami 1995; Shahjahan, Akhand et al. 2003). This approach, although straightforward, is rather inefficient, since many networks may have to be trained before an acceptable one is found.

Regularisation is performed by adding a penalty term that is proportional to the sum of weighted squares to the data error function. This penalty term is sometimes called *weight decay*. The determination of the proper amount of weight decay is generally difficult.

A popular method of determining the proper weight decay is *cross-validation* (Sarle 1995). In this method, the available data is divided into p subsets of (approximately) equal size. The network is trained p times, each time leaving out one of the subsets from training and using the omitted subset to compute whatever error criterion is interested. Despite some approaches having recently been proposed for optimally partitioning the available data set into subsets (Amari, Murata et al. 1997; Larsen and Goutte 1999), this technique may not be practical when only a small data set is available.

2.4 Bayesian neural networks

It is clear that the network training with weight decay is ineffective in terms of standard techniques. It is, therefore, necessary to study objective criteria for setting the proper weight decay and comparing alternative solutions which only depend on training data. Such criteria are especially important in applications where available data are finite.

The Bayesian method automatically and quantitatively embodies Occam's razor without the introduction of ad hoc penalty terms. Complex hypotheses can automatically be penalised under Bayes' rule. The practical Bayesian framework for feed-forward perceptron network learning (MacKay 1992a; MacKay 1992b) can be used for filling the following gaps in the field of neural networks:

- *Objective criteria for comparing alternative solutions:* given a single architecture, there may be many local minima of the cost function. If there is a large disparity in the cost function between the minima, then it is plausible to choose the solution with the smallest value of the cost function. But where the difference is not as great, it is desirable to be able to assign an objective preference to the alternatives.

It is also desirable to be able to assign preferences to neural network solutions using different numbers of hidden nodes. Here, there is an Occam's razor problem: the more free parameters the model has, the smaller the data error it can obtain. So we cannot simply choose the architecture with the smallest data error. That would lead us to an over-complex network which generalises poorly. The use of weight decay does not fully alleviate this problem. Networks with too many hidden nodes still generalise less well.

- *Objective criteria for setting weight decay parameters:* small values of weight decay parameters may cause the network weights to be large, fitting the data noise. This leads to a small value of the data error, so we cannot base our choice of weight decay parameters only on the data error. However, the Bayesian solution can be implemented on-line, that is, it is not necessary to undertake multiple learning runs with different values of weight decay parameters in order to find the best.

-
- *Objective criteria for choosing between a neural network solution and a solution using a different learning model:* for example, we can compare a solution using an MLP network with a solution using a radial basis function neural network or Dempster-Shafer neural network.

BNNs can be applied to regression and classification problems. As the use of BNNs is advantageous to complex applications, there is a need to explore the properties of this type of neural network in assistive technology applications.

2.5 Developed hands-free wheelchair control methods

The determination of the disability level of a person is needed in order to choose which system access method is suitable for him or her. The difficulty and lack of manual dexterity of many individuals in using conventional input devices, such as a hand-operated joystick, keyboard or mouse have led to the development of various hands-free system access devices using speech, chin, eye wink, face direction, head movement and brain signal. Due to the complex and unpredicted operating environments of power wheelchairs, this section discusses the hands-free wheelchair control methods developed recently.

2.5.1 Speech-based control

Speech recognition has been a topic of much interest in research for a long time. There are a number of techniques of speech recognition. For example, speech recognition can be implemented using neural networks (Lim, Woo et al. 2000). A block diagram of a typical speech recognition system is shown in Figure 2.2. The system is able to be trained to recognise the voice of a person speaking out utterances into the microphone. The speech signal is digitalised using the analog-to-digital converter (ADC). Signal processing is then carried out to extract pattern templates.

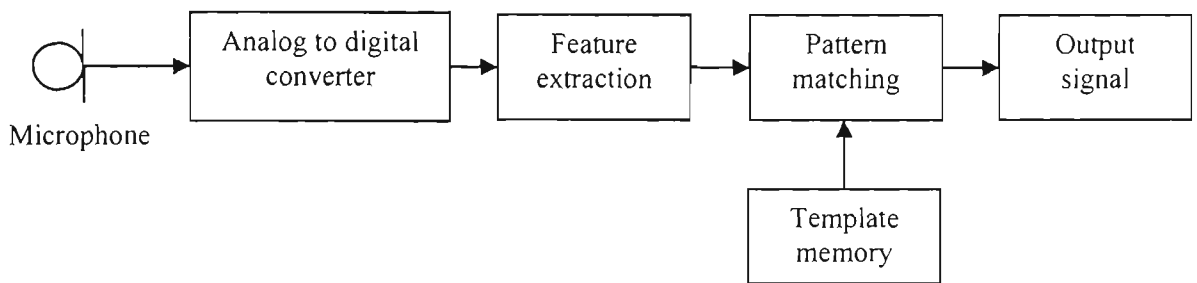


Figure 2.1: Block diagram of a voice recognition system (Venayagamoorthy, Moonasar et al. 1998).

The system recognises the speech by comparing patterns extracted from recorded utterances with respective patterns stored in the computer memory. When matches occur, the recorded utterances are identified. Two important operations for this type of speech recognition are feature extraction, whereby distinct patterns are obtained from recorded utterances and pattern-matching, whereby extracted patterns are compared with those stored in the computer memory.

Speech recognition systems can be classified into two categories: speaker-dependent and speaker-independent systems. Speaker-independent systems respond to a word regardless of who speaks, and thus they can respond to a large variety of speech patterns and inflections. This type of speech recognition system is necessary for applications where many different people use the same system. However, for wheelchair control, speaker-dependent systems are more appropriate.

Despite commercially available speech systems being fairly mature, making their applications both feasible and cost-effective, and recent optimisation techniques making speech-based control more robust (Rockland and Reisman 1998; Roberts, Pruehsner et al. 1999; Simpson and Levine 2002), speech-based wheelchair control is usually ineffective when the wheelchair is operated in cluttered environments. In such areas, the quality of the speech is significantly affected by noise and channel distortions. The noise phenomenon not only affects the acoustics of the speech signal, but also distorts it at the source. This problem is extremely difficult to overcome, especially when there is no prior knowledge of the noise or distortions.

2.5.2 Chin-based control

Chin-operated joysticks enable people with a high level of SCIs to operate a complete range of assistive devices. Two types of chin-operated joysticks have been developed: the position-sensing joystick and the force-sensing joystick.

- A chin-operated position-sensing joystick usually requires neck flexion, extension and rotation of its users. Due to the stick displacement of the joystick, it is very difficult for the user to operate it for a long time. For many individuals, their tremors, spasm, weakness and inadequate active range of motion are some of the reasons that make it difficult or impossible for them to drive power wheelchairs with chin-position-sensing joysticks.
- Chin-operated force-sensing joysticks incorporated with dedicated controllers have been developed to overcome the disadvantages of position-sensing joysticks (Jones, Cooper et al. 1998; Guo, Cooper et al. 2002). These chin joysticks are able to provide a proportional-control alternative for their users. Compared to position sensing-joysticks, force-sensing joysticks do not require the power of users to move their head quickly and accurately. However, the use of force-sensing joysticks has a limitation. Due to the noise from the power supply, the drift of amplifier and strain gauges, there is a “dead zone” in which the user cannot drive the chair. In order to improve the sensitivity of the joysticks, they must be modified or the drift of the amplifier and strain gauges must be eliminated.

2.5.3 Gesture-based control

There are a variety of static and dynamic signs made by a person that are referred to as “gestures” and can be utilised to convey intentions. Those gestures include face direction, head movement and eye-winking.

2.5.3.1 Face direction

A simple way to track the current intention of a person is based on the direction in which he or she is looking. The robotic wheelchair was developed by Kuno and his colleagues (Kuno, Nakanishi et al. 1999) to be controlled by the user's face direction. The use of face direction for wheelchair control can make a "user-friendly" interface. However, the wheelchair system needs to distinguish wheelchair-control behaviours from others. In this system, the user has to move his or her face slowly and steadily if he or she wants the wheelchair to move according to his or her face direction. Thus, the system may ignore quick head movement and may only respond to slow head movement.

In order to capture the wheelchair user's intention, a video camera was set up in front of the user to observe his or her face. The system uses the visual information to compute the user face's direction. Firstly, the skin colour region (Figure 2.2b) is extracted from the original image (Figure 2.2a). Next, the face region is chosen (Figure 2.2c). Finally, the face features, including eyes, eyebrows, nose and mouth are extracted, as shown in Figure 2.2d.

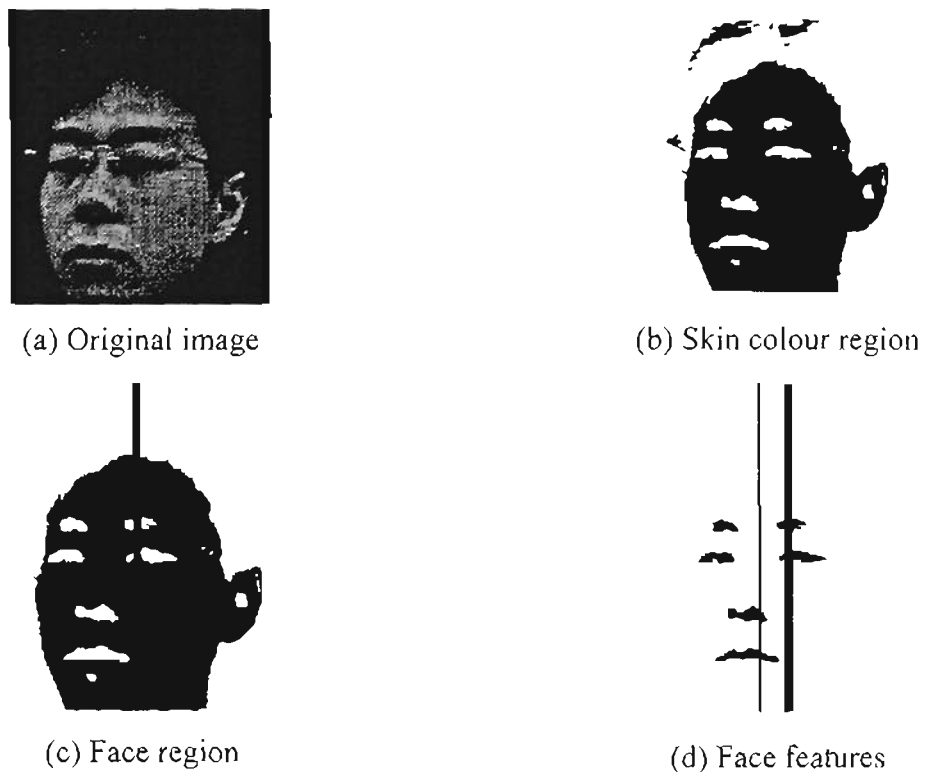


Figure 2.2: Face direction computation (Kuno, Nakanishi et al. 1999).

A comparison between the centroid of the face region and all of the face features is then performed. The vertical line passing the centroid of the face region is drawn in a thick line (Figure 2.2c), while the vertical line passing the centroid of the face features is drawn in a thin line (Figure 2.2d). If the thick vertical line lies to the right or left of the thin vertical line, the face can be considered as turning right or left. The output of the process is the horizontal distance between the two vertical lines, which roughly indicates the face direction.

The system is also able to compute the face direction with a number of captured image frames per second. A filter is applied to face direction data to separate wheelchair-control behaviours from others. The filter used is a simple smoothing filter by averaging the values over a certain number of frames. If this number is large, the system will not be affected by quick and unintentional head movement. However, this makes the user feel uneasy, as the response of the system is slow. Therefore, the number of frames can

be changed according to different levels of quick head movement. The performance of the system significantly depends on the quality of captured visual information.

2.5.3.2 Head movement

For many severely disabled SCI people, head movement is a natural way of pointing that can be used to control the direction of power wheelchairs. Figure 2.3 shows the configuration of a head-oriented wheelchair (Chen, Chen et al. 2002). The system consists of three modules: tilt sensor, signal-processing and main control modules. The head motion of the user is measured using a tilt sensor mounted on a cap worn by the user. An ADC and a low-cost microcontroller are used to detect designer-specified thresholds of head motion. The wheelchair user can also puff his or her cheek to trigger the touch switch to perform an emergency stop using power breaks.

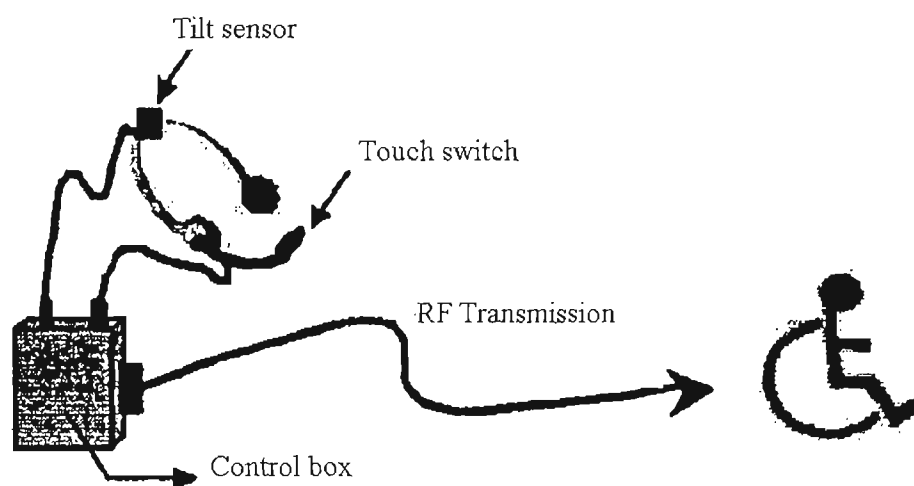


Figure 2.3: The configuration of a head-oriented wheelchair control (Chen, Chen et al. 2002).

The tilt sensor module includes two commercial magneto-resistive tilt sensors; each measures tilt angle in one direction. The detectable range of the tilt sensors is suitable for a measure of any angle within $\pm 45^\circ$. The neutral output voltage of the sensor after calibration is +2.5V. The standard output voltage varying from 0V to 5V is proportional to the detected range between -45° and $+45^\circ$.

The main advantage of this wheelchair control configuration is that it is easy to be implemented. However, due to the use of a fixed head-command detection model for all users, it is unable to adapt to variations in head movement.

2.5.3.3 Eye wink

The eye wink control interface (EWCI) is a hands-free input device that a person with severe motor impairment can use to control power wheelchairs. Eye winks are timed closures of one or both eyelids.

Figure 2.4 shows the diagram of an EWCI in which eye winks are registered electronically via two pairs of infra-red photo-sensors (Crisman, Loomis et al. 1991). The sensors are attached to the ear pieces of a normal pair of eyeglass frames. The sensors continually pass the state (up or down) of the respective lids to a dedicated computer. Commands are defined as a combination of the status of the eyelids and the length of time that status was held, for example, both down-short, left down-long, etc.

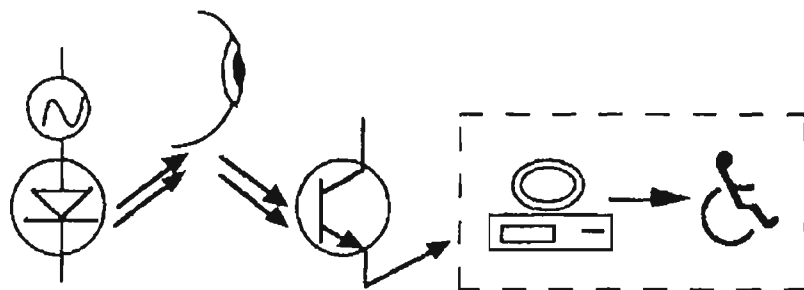


Figure 2.4: Diagram of an eye-wink control interface (Crisman, Loomis et al. 1991).

This type of wheelchair control, however, results in some difficulties for its users. Firstly, many people are only able to wink with one eye. This problem limits the number of commands defined. Secondly, timed closures of eyelids restrict the user's visibility and the fast response of the system in case of emergency.

2.5.4 Brain-signal-based control

Recently, brain computer interface (BCI) has increasingly been developed to allow communication between humans and machines using brain signals. Due to complex and

time-varying properties of brain signals, BCI researches are a multidisciplinary field integrating researchers from neuroscience, psychology, engineering, computer science and rehabilitation.

Over the last three decades, several BCI researches have been reported. For example, a miniature indoor mobile robot or a power wheelchair can be controlled by human electroencephalogram (EEG) (Millan, Renkens et al. 2004; Tanaka, Matsunaga et al. 2005). Since the intentions of a person can be directly conveyed via a BCI, this technology will bring many benefits for severely disabled people in the near future.

2.6 Assistive wheelchairs

2.6.1 Recent and current developments

Assistive power wheelchairs have been developed to allow severely disabled people to quickly, safely and independently operate their own wheelchairs. The majority of assistive wheelchairs represent outgrowths of mobile robot researches.

Several assistive wheelchairs have been developed to operate in a manner very similar to autonomous robots, in which the user gives the wheelchair a final destination and supervises as the chair plans and executes a path to the target location (Yoder, Baumgartner et al. 1996; Pires, Honorio et al. 1997; Lankenau and Rofer 2001). In order to reach a destination, these wheelchairs typically require either a complete map of the area through which they navigate or some sort of modifications to their environment (for example, tape tracks placed on the floor or markers placed on the walls). These wheelchairs are, therefore, unable to compensate for unplanned obstacles or travel in unknown areas. Assistive wheelchairs in this category are most appropriate for users who lack the ability to plan and/or execute a path to a destination and spend the majority of their time within the same controlled environment.

Other assistive wheelchairs are able to provide task-specific navigation assistance in the form of several distinct operating modes, each of which distributes control differently between the chair and the operator (Bourhis, Moumen et al. 1993; Kuno, Nakanishi et

al. 1999; Simpson and Levine 1999). The main features of some recent and current assistive wheelchairs are summarised in Table 2.1.

Table 2.1: Some current and recent assistive wheelchair systems.

System	Sensor	Description
RobChair (Pires, Honorio et al. 1997)	Sonar, infrared, bump	<ul style="list-style-type: none"> - Being based on TinMan wheelchair. - Being capable of local obstacle avoidance assistance.
Rolland (Lankenau, Meyer et al. 1998)	Vision, sonar, dead reckoning, infrared, bump	<ul style="list-style-type: none"> - Being capable of learning an environment while navigating, then planning paths through the learned environment. - Being capable of learning obstacle avoidance behaviours by training.
MAid (Prassler, Scholz et al. 1998)	Sonar, infrared, laser range finder, dead reckoning	<ul style="list-style-type: none"> - Being capable of task-specific behaviours, such as entering a restroom, in semi-autonomous mode. - Being capable of navigating to a goal supplied by the user in fully autonomous mode.
NavChair (Simpson and Levine 1999)	Dead reckoning, sonar	Being provided with multiple operating modes and shared-control navigation with obstacle avoidance routines.
Wheelesely (Yanco 2000)	Vision, infrared, sonar	<ul style="list-style-type: none"> - Being based on TinMan wheelchair. - Being capable of vision-based navigations.
VAHM (Bourhis, Moumen et al. 1993)	Sonar, infrared, dead reckoning	Being capable of performing autonomous navigations based on an internal map and semi-autonomous navigations.

2.6.2 Basic components of assistive wheelchairs

Assistive power wheelchairs are often based on standard power wheelchairs with a supplemental module, as shown in Figure 2.5. The supplemental module consists of three units: 1) a computer containing “intelligent” control software, 2) an array of laser or ultrasonic transducers mounted around the chair, and 3) an interface module to provide necessary circuits for the system.

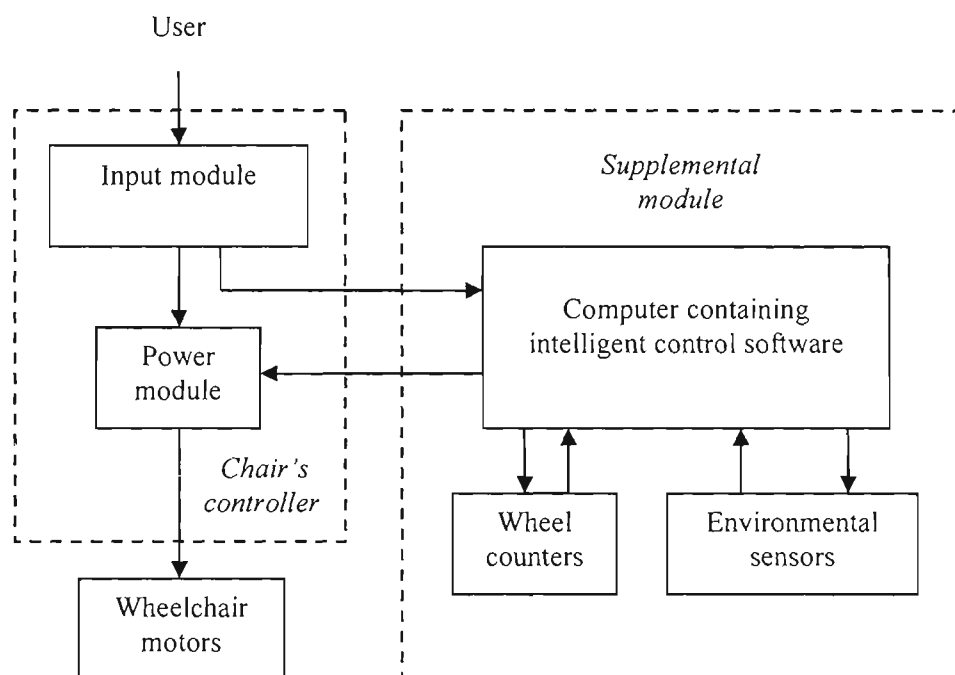


Figure 2.5: Functional diagram of assistive wheelchairs.

The wheelchair controller is divided into two components: 1) the input module, which receives commands from the user and converts them into signals representing desired directions and 2) the power module, which converts output signals of the input module into control signals for driving the left and right wheel motors. During operation, the supplemental module interrupts the connection between the input module and the power module. The input signal representing the user’s desired direction and the readings from environmental sensors, reflecting the wheelchair’s immediate environment, are used to determine control signals sent to the power module so that safe and smooth navigation can be obtained.

2.6.3 Multiple operating modes in assistive wheelchairs

When developing a assistive wheelchair, it is necessary to equip it with several operating modes. This task provides a full range of desired functionality when the chair is operated in different environments and situations.

- *Operator-control mode*: in this mode, the wheelchair monitors commands issued by its operator. If there is no obstacle close to the chair, the “intelligent” controller does not switch to another mode. If there is a dangerous or unexpected obstacle near the chair, the intelligent controller performs a transition into stop-in-time or obstacle avoidance mode.
- *Stop-in-time mode*: the intelligent controller overrules every command given by the user and sets the target speed to zero.
- *General obstacle avoidance mode*: in this mode, the wheelchair can move quickly and smoothly in crowded environments while still maintaining safe distances from obstacles.
- *Door passage mode*: This mode is activated to allow the chair to move between two closely spaced obstacles, such as the posts of a doorway. Door passage is in general the most difficult navigation task an assistive wheelchair needs to perform. It is possible for the chair to fail to successfully pass through a doorway on a given attempt. Typically, this is due to the chair approaching the door at an angle rather than from directly in front of the door. When a failure occurs, the operator is required to back up and approach the door again, hopefully from a better direction. Alternatively, a suitable obstacle avoidance routine is activated to push the chair away from both door posts and toward the centre of the door.
- *Automatic wall-following mode*: when the wheelchair is operated in this mode, the intelligent controller modifies the user’s commands to make the chair

follow the direction of a wall to the left or right of the chair while still maintaining a safe distance from the wall.

The appearance of distinct operating modes creates the need of mode selection procedures. Many approaches require the user to perform mode selection manually by means of his/her interface. Whether a menu-based interface is provided or not, manual selection often remains difficult and tiring for severely disabled users if their perception is limited.

Some approaches introduce automatic mode selections in which the most appropriate mode is automatically activated based on the combination of immediate situation and global location or the direction in which the user points at. Developed automatic-mode selection mechanisms for assistive wheelchairs can be differentiated according to two main approaches.

The first approach performs an automatic change of modes based only on information about the surrounding environment, and does not take the user signals into account. If the user's plans differ from hypotheses from sensorial information, the controller may enter undesired modes. In the second approach, rules to select modes are based on the direction in which the user points. For example, if the user points at a door, then the door passage mode is automatically activated without asking the user for permission. However, if the user cannot realise the mode transition, the "mode confusion" may occur. Although the mode confusion can be solved by interpreting the user's command before the manoeuvre begins (Lankenau, Meyer et al. 1998; Lankenau 2001), this approach still has some disadvantages.

Firstly, due to the use of a fixed user model, it cannot adapt to the change in the user's driving characteristics. Secondly, in order to ensure the safety of the whole system, the wheelchair controller may refuse some of the user's wishes. For example, if an obstacle such as a table appears in front of the chair, the intelligent controller may understand that it is necessary to activate the general obstacle mode while the user really wants to

dock at the table. This problem motivates several frameworks of continuously estimating the user's intention in real-time (Demeester, Nuttin et al. 2003; Vanhooydonck, Demeester et al. 2003; Demeester, Nuttin et al. 2003a). In literature, this framework is also known as "plan recognition". After knowing where the user wants to go, the controller can assist the user with fine motion manoeuvres.

Another problem concerning the fine motion planning also plagues assistive wheelchairs that have originated from robotic applications. While fast manoeuvres are often required for mobile robots to reach a destination, in wheelchair control the driver usually expects smooth and comfortable driving behaviour. This depends not only on the user's driving skills, but also on the behaviour of the fine motion planner in specific driving tasks. The wheelchair should behave safely and deterministically. So if an obstacle is encountered, the best behaviour is slowing down or stopping the chair while waiting for a new command from the user, rather than autonomously avoiding the obstacle.

From the above discussions, we can see that an ideal assistive wheelchair should not work as an autonomous robot with a person sitting on it and should be able to predict the user's intention before assisting him or her to obtain safe and comfortable navigations.

2.7 Discussion

Nowadays, neural networks are a powerful and flexible tool to enhance the adaptability of assistive devices. However, applications of neural networks are easily ineffective if the network training is only carried out using the standard back-propagation which tries to minimise data errors. Since the available data used to train the network are biased or finite, standard back-propagation tends to make network parameters converge into biased solutions. Hence the trained network performs poorly on new data.

Recently, BNNs have been proposed to develop applications of neural networks with finite and/or noisy available data. Especially, a trained BNN itself involves some useful

quantities that can be utilised to measure the performance and the complexity of the network. In addition, the Bayesian model comparison principle can also be applied to comparing different solutions.

Alternative methods of hands-free system access play an important role in maintaining independent activities for severely disabled individuals with quadriplegia, cerebral palsy, cognitive or perceptual impairments, etc.

Head movement is a form of hands-free gesture, which many severely disabled people can comfortably perform and can be utilised to convey intentions effectively and naturally. As BNNs are capable of learning finite and/or noisy data, they should be used to develop devices which are able to detect various forms of head movements.

Finally, the aim of creating autonomous wheelchairs is not suitable for a wide range of wheelchair users, as they still excel at some task executions in comparison with the controller. Ideal assistive wheelchair control systems should therefore be able to maximise the user's driving skills, not replace them.

Chapter 3

Mobility Assistance after Spinal-cord Injury Using Neural Networks

3.1 Introduction

Severe spinal cord injuries (SCIs) result in loss of mobility. Functional electrical stimulation (FES) is a rehabilitative technology that can restore function to paralysed limbs. Recently, neural networks have been exploited in research into functional restoration of upper paralysed limbs after SCI (Au and Kirsch 2000; Riess and Abbas 2000; Giuffrida and Crago 2005). After some degree of hand functions was restored, the disabled person is able to use a standard joystick to control a power wheelchair.

However, complete C4-level SCIs result in the total loss of arm and hand functions and most shoulder motion (Sarver, Smith et al. 1999). In order to control power wheelchairs, quadriplegic individuals must use their residual functions such as head movements or brain signals to generate control commands.

During the last nine years, the development of learnable head-pointing devices that allow persons with high-level SCIs to easily control power wheelchair directions have been the focus in UTS (Joseph and Nguyen 1998; Taylor and Nguyen 2003; Nguyen, King et al. 2004; S.T.Nguyen, H.T.Nguyen et al. 2004; S.T.Nguyen, H.T.Nguyen et al. 2006). In particular, these devices can comfortably be used by any severely disabled user. The heart of the devices is a two-layer perceptron classifier responsible for detecting head movements in real-time.

3.2 Spinal-cord injury

Spinal-cord injury (SCI) is a disturbance of the spinal cord that results in loss of sensation and mobility. The two common types of SCIs are:

- *Trauma*: automobile accidents, falls, diving accidents, etc.
- *Disease*: polio, spina bifida, tumours, Friedreich's ataxia, etc.

SCIs are not the same as back injuries such as ruptured discs, spinal stenosis or pinched nerves. It is possible to break one's neck or back and not sustain a SCI if only the vertebrae are damaged, but the spinal cord remains intact. Tables 3.1 and 3.2 provide some information about SCIs in Australia and the United States of America (Cripps 2004; UAB 2006).

Table 3.1: Number of people living with SCIs in Australia and the United States of America.

	Australia	The United States
People with spinal cord injuries	More than 10,000 (in 2003)	Approximately 250,000 (in 2005)
New spinal cord injuries every year	About 400	About 11,000

Table 3.2: SCIs from trauma causes in Australia and the United States of America.

	Australia (in 2003)	The United States (in 2005)
Vehicle accidents	52%	47.5%
Falls	34%	22.9%
Sports, violence	13%	22.7%
Others	1%	6.8%

3.2.1 The type of injury

The exact effects of a SCI vary according to the type and level of injury, and can be organised into two types:

-
- In a *complete injury*, there is no function below the level of the injury. Voluntary movement is impossible and physical sensation is impossible. Complete injuries are always bilateral (both sides of the body are affected equally).
 - A person with an *incomplete injury* retains some sensation below the level of the injury. Incomplete injuries are variable, and a person with such an injury may be able to move one limb more than another, may be able to feel parts of the body that cannot be moved or may have more functioning on one side of the body than the other.

In addition to a loss of sensation and motor function below the point of injury, individuals with SCIs will often experience other changes.

Bowel and bladder function is associated with the sacral region of the spine, so it is very common to experience dysfunction of the bowel and bladder. Sexual function is also associated with the sacral region, and is also affected very often. At the C1/C2 level, injuries will often result in the loss of many involuntary functions, such as breathing, necessitating mechanical ventilators or diaphragmatic pacemakers.

Other effects of SCIs can include an inability to regulate heart rate, reduced control of body temperature, inability to sweat below the level of injury, and chronic pain. Physical therapy and orthopaedic instruments (for example, wheelchairs, standing frames) are often necessary, depending on the location of the injury. Between about three weeks and twelve years after lesion above T10 (see the spinal-cord map in Table 3.3), autonomic dysreflexia may occur.

3.2.2 The location of the injury

Knowing the exact level of the injury on the spinal cord is important when predicting what parts of the body might be affected by paralysis and loss of function. Typical effects of SCI can be listed by location (refer to the spinal-cord map in Table 3.3).

3.2.2.1 Cervical injuries

Cervical (neck) injuries usually result in full or partial tetraplegia. Depending on the exact location of the injury, a person with a SCI at the cervical level may retain some amount of function as detailed below, but is otherwise completely paralysed.

- *C3 vertebrae and above*: typically loss diaphragm function and requires a ventilator to breathe.
- *C4*: has some use of biceps and shoulders, but weaker than C5.
- *C5*: may retain the use of shoulders and biceps, but not of the wrists or hands.
- *C6*: generally retains some wrist control, but no hand function.
- *C7 and T1*: can usually straighten his or her arms but may still have dexterity problems with the hand and fingers.

3.2.2.2 Thoracic injuries

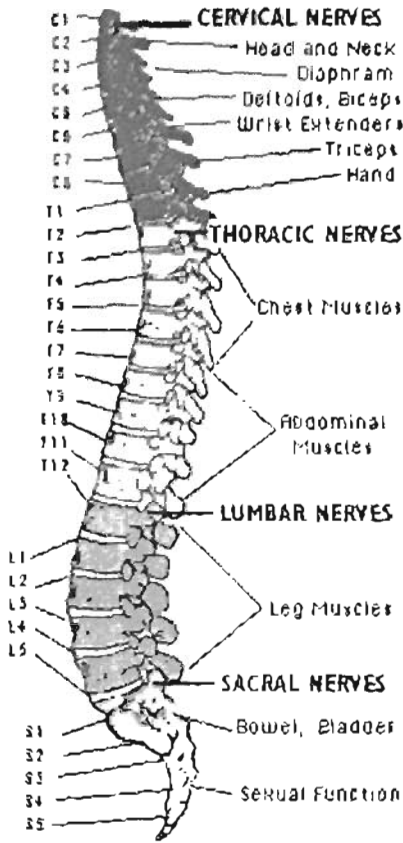
Injuries at the thoracic level and below result in paraplegia. The hands, arms, head, and breathing are usually not affected.

- *T1 to T8*: most often has control of the hands, but lacks control of the abdominal muscles so control of the trunk is difficult or impossible.
- *T9 to T12*: allows good trunk and abdominal muscle control, and sitting balance is very good.

3.2.2.3 Lumbar and sacral injuries

The effects of injuries to the lumbar or sacral region of the spinal cord are decreased control of the legs and hips.

Table 3.3: The spinal-cord map (DIR 2006).

Divisions of spinal segments	Segmental spinal-cord level and function	
	Level	Function
 <p>The diagram shows the human spine from the neck to the pelvis. It is divided into four main regions: Cervical (C1-C8), Thoracic (T1-T12), Lumbar (L1-L5), and Sacral (S1-S5). Key muscles and organs are labeled: Head and Neck, Diaphragm, Deltoids, Biceps, Wrist Extensors, Triceps, Hand, Chest Muscles, Abdominal Muscles, Leg Muscles, Bowel, Bladder, and Sexual Function.</p>	C1 – C6	Neck flexors
	C1 – T1	Neck extensors
	C3, C4, C5	Supply diaphragm (mostly C4)
	C5, C6	<ul style="list-style-type: none"> - Shoulder movement - Raise arm (deltoid) - Flexion of elbow (biceps) - C6 externally rotates the arm (supinates)
	C6, C7, C8	<ul style="list-style-type: none"> - Extend elbow and wrist (triceps and wrist extensors) - Pronates wrist
	C7, C8, T1	Flexes wrist
	C8, T1	Supply small muscles of the hand
	T1 – T6	Intercostals and trunk above the waist
	T7 – L1	Abdominal muscles
	L1, L2, L3, L4	Thigh flexion
	L2, L3, L4	Thigh adduction
	L4, L5, S1	Thigh abduction
	L5, S1, S2	Extension of leg at the hip (gluteus maximus)
	L2, L3, L4	Flexion of leg at the knee (quadriceps femoris)
	L4, L5, S1, S2	Flexion of leg at the knee (hamstring)
	L4, L5, S1	<ul style="list-style-type: none"> - Dorsiflexion of foot (tibialis anterior) - Extension of toes
	L5, S1, S2	<ul style="list-style-type: none"> - Plantar flexion of foot - Flexion of toes

3.3 Functional restoration of upper extremities after spinal-cord injury

Able-bodied individuals use a large number of shoulder and elbow muscles to position the hand throughout the available workspace and to stabilise it at desired fixed positions. However, individuals with cervical SCIs can only move their hands through an extremely limited workspace, restricting their ability to perform the normal activities of daily living.

There have been a number of studies on functional neuromuscular stimulation (FNS) to restore function to paralysed limbs (Riess and Abbas 2000; Kirsch, Parikh et al. 2001; Hincapie, Blana et al. 2004; Giuffrida and Crago 2005). In FNS systems, sequences of current pulses excite the motor neurons, which in turn activate muscles and cause movement. By changing the pulse width, pulse amplitude, or the pulse frequency, the level of the muscle stimulation can be altered to perform a specific task.

In fitting an FNS system to an individual, a large set of parameters must be modified to account for each person's specific injury, anatomy, and muscle condition. The determination of the stimulation pattern is currently performed by trained therapists or engineers who manually alter stimulation levels sent to the muscles. Recently, neural networks, in either feed-forward or feedback configurations, have been used to automatically generate and shape stimulation patterns to paralysed shoulder and elbow muscles (Au and Kirsch 2000; Riess and Abbas 2000).

Figure 3.1 shows a synergic controller employing the remaining voluntary elbow flexor and shoulder electromyography (EMG) to control elbow extension with functional electrical stimulation (FES) (Giuffrida and Crago 2005). Of this, the remaining voluntarily controlled upper extremity muscles were used to train a neural network to control stimulation of the paralysed triceps. Surface EMG was collected from SCI subjects while they produced isometric endpoint force vectors of varying magnitude and direction, using triceps stimulation levels predicted by a biomechanical model. Such

controllers can provide a large range of endpoint force vectors, the ability to grade and maintain forces, the ability to complete a functional overhead reach task.

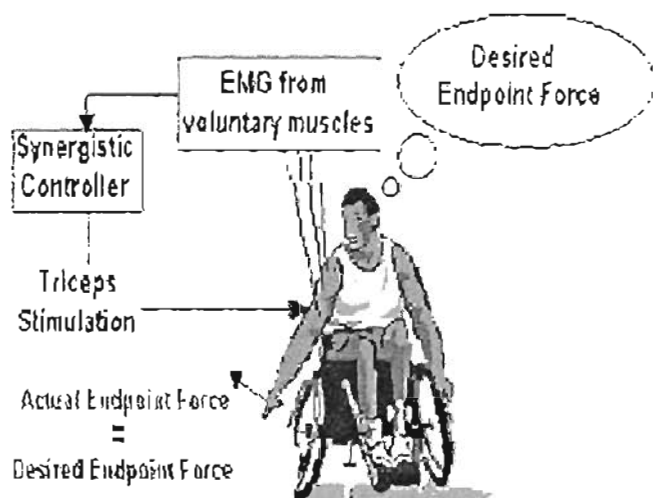


Figure 3.1: The synergistic controller for restoring elbow extension: the synergistic controller is able to let an SCI subject move his arm or apply an endpoint force simply by attempting to do so (Giuffrida and Crago 2005).

3.4 Hands-free device access

Hands-free user interface is extremely useful for individuals with quadriplegia resulting from high-level SCIs to access any device. Many projects have emerged to realise the importance of this kind of user interface (Crisman, Loomis et al. 1991; Joseph and Nguyen 1998; Kuno, Nakanishi et al. 1999; Chen, Chen et al. 2002; Craig and H.T.Nguyen 2005; Tanaka, Matsunaga et al. 2005).

As previously discussed in Chapter 2, head movements and EEG are hands-free signals that a quadriplegic person can use to convey intentions directly and comfortably. In order to develop devices that are able to interpret these hands-free signals of different subjects, neural networks capable of learning different forms of the signal have been used in several recent researches (Joseph and Nguyen 1998; Taylor and Nguyen 2003; Nguyen, King et al. 2004; S.T.Nguyen, H.T.Nguyen et al. 2004; Craig and H.T.Nguyen 2005; S.T.Nguyen, H.T.Nguyen et al. 2006; S.T.Nguyen, H.T.Nguyen et al. 2006).

The diagram of a typical hands-free user interface with the use of neural networks is illustrated in Figure 3.2. A tilt sensor or a set of electrodes installed on a cap worn by the user operate to sense head movement or brain signal (EEG). A data-acquisition device is responsible for converting measured analog signals into digital signals suitable for computer-processing. A computer works as a central controller containing a programme for pattern extraction and neural network-based pattern-matching. After patterns were recognised, the computer sends control signals to the peripheral device via the data-acquisition device.

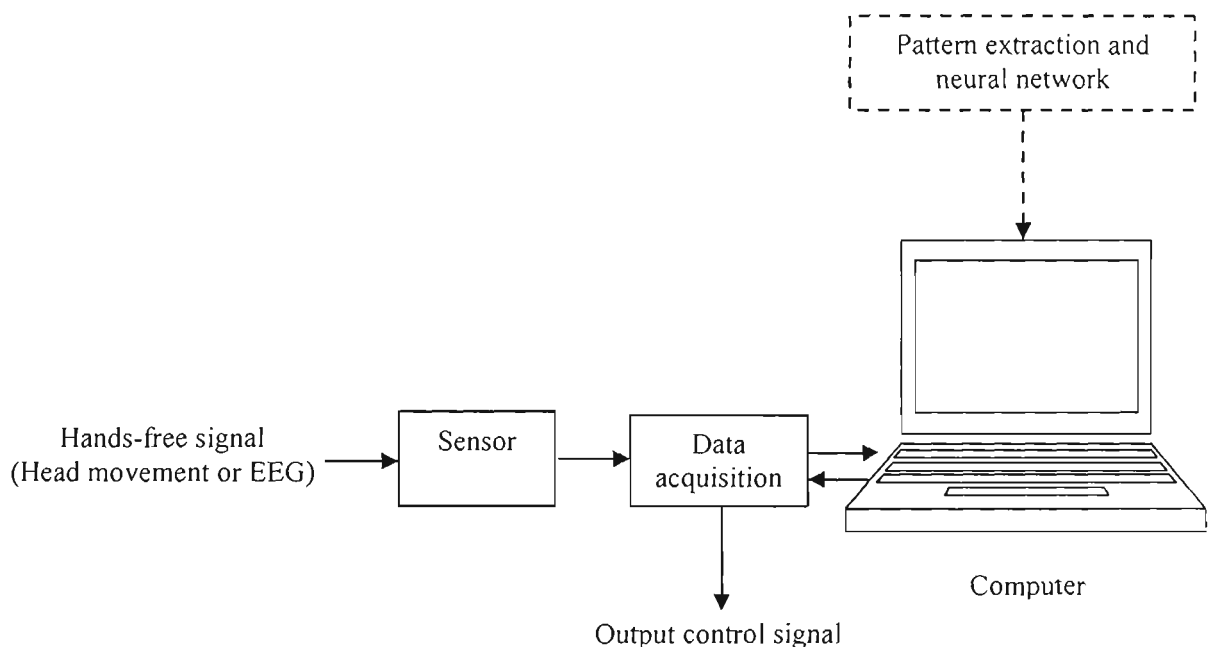


Figure 3.2: Diagram of a typical hands-free user interface with the use of neural networks.

3.5 Neural networks

The field of neural networks was established more than sixty years ago. The historical development of neural networks can be divided into memorable milestones shown in Table 3.4.

Table 3.4: Milestones in the development of neural networks.

Milestones	Events
1943	McCulloch and Pitts developed models of neural networks based on their understanding of neurology.
1958	Rosenblatt stirred considerable interest and activity in the field when he designed and developed the <i>Perceptron</i> . The perceptron had three layers, with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit.
1960	Widrow and Hoff developed the <i>Adaline</i> (Adaptive Linear Element). The ADALINE was an analog electronic device made from simple components. The method used for learning was different from that of the perceptron: it employed the <i>Least Mean Squares</i> (Bolmsjo, Neveryd et al.) learning rate.
1967	Amari was involved with theoretical developments; he published a paper that established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification. Fukushima developed a step-wise trained multi-layer neural network for the interpretation of handwritten characters.
1972	Klopf developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called <i>heterostasis</i> .
1974	Werbos developed and used the back-propagation learning method.
1988	Grossberg founded a school of thought which explores resonating algorithms. He and colleagues developed the <i>ART</i> (Adaptive Resonance Theory) networks based on biologically plausible models.

3.5.1 Artificial neurons

Computational neurobiologists have constructed very complicated computer models of neurons in order to run detailed simulations of activities in the brain. As engineering scientists, we are more interested in the general properties of neural networks, independent of how they are actually implemented in the brain.

In artificial neurons, a basic computational element is often called a “node” or “unit”, as shown in Figure 3.3. The node receives input from some other nodes, or perhaps from an external source.

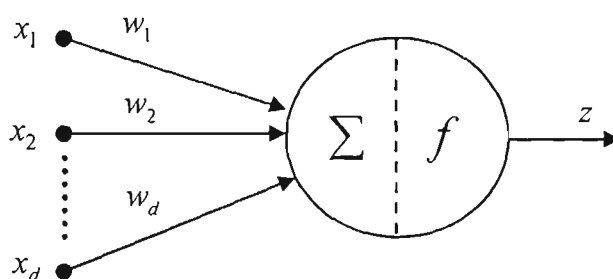


Figure 3.3: An artificial neuron.

Each input x_i ($i = 1, \dots, d$) has an associated “weight” w_i , which can be modified so as to model synaptic learning. The node computes the function f of the weighted sum of its inputs:

$$z = f\left(\sum_{i=1}^d w_i x_i\right) \quad (3.1)$$

The output of the node, in turn, can serve as an input to other nodes. The weighted sum $\sum_{i=1}^d w_i x_i$ is called the activation to the node and is denoted by a . Finally, $f(\cdot)$ is called the activation function of the node.

3.5.2 Network architecture

3.5.2.1 Feed-forward networks

A feed-forward network, as shown in Figure 3.4, allows signals to travel one way only from input nodes to output nodes. There is no feedback (loops), that is, the output of any layer does not affect that in the same layer. This network architecture is extensively used in pattern recognition and is also referred to as the bottom-up or top-down architecture.

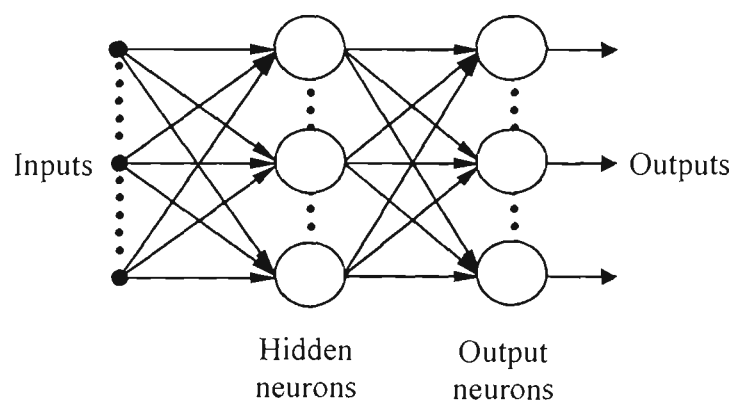


Figure 3.4: A feed-forward neural network.

3.5.2.2 Feedback networks

A feedback network, as shown in Figure 3.5, can have signals travelling in both directions by introducing loops in the network. Feedback networks can become extremely complicated. Feedback networks are dynamic, that is, their “state” change continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as recurrent networks.

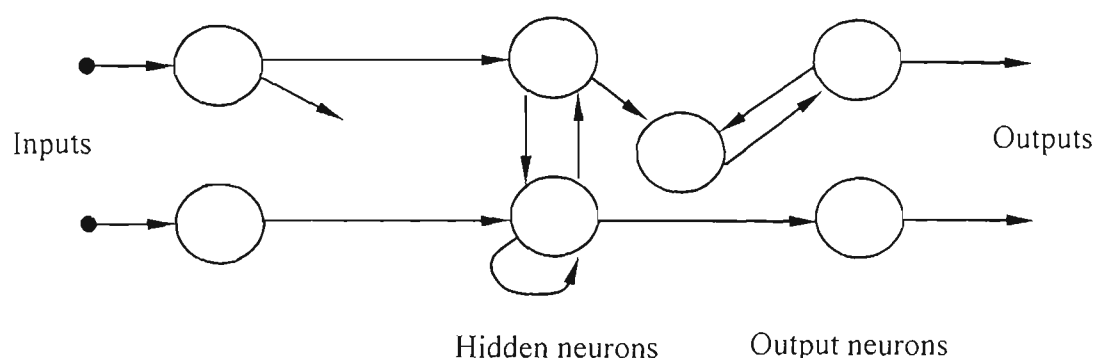


Figure 3.5: A feedback neural network.

3.5.2.3 Multi-layer networks

We also distinguish between single-layer and multi-layer architectures. The single-layer organisation, in which all neurons are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organisations. In multi-layer networks, neurons are often numbered by layer, instead of following a global numbering.

The commonest type of multi-layer networks consists of three groups or layers: a layer of input neurons is connected to a layer of hidden neurons, which is connected to a layer of output neurons, as previously depicted in Figure 3.4.

- *Input neurons:* the activity of the input neurons represents the raw information that is fed into the network.
- *Hidden neurons:* the activity of each hidden neuron is determined by the activities of the input neurons and the weights on the connections between the input and hidden nodes.
- *Output neurons:* the behaviour of the output neurons depends on the activity of the hidden neurons and the weights between the hidden and output nodes.

Neural networks with a single hidden layer are usually called “two-layer neural networks”, as only the hidden and output layers are considered to be named.

3.5.3 Perceptrons

The most influential work on neural networks in the 1960's comes under the heading of "perceptrons", a term coined by Frank Rosenblatt (1958). Figure 3.6 shows a perceptron consisting of a processing element with synaptic input connections and a single output.

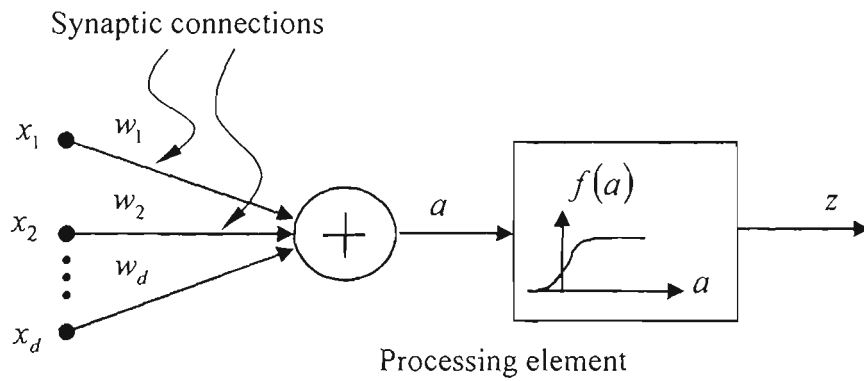


Figure 3.6: A perceptron.

The output of the perceptron is given by:

$$z = f\left(\sum_{i=1}^d w_i x_i\right) \quad (3.2)$$

The activation function $f(\cdot)$ can be one of the following:

- *Linear function:*

$$f(a) = a \quad (3.3)$$

- *Threshold logic unit:*

$$f(a) = \text{sgn}(a) = \begin{cases} +1, & a > 0 \\ -1, & a < 0 \end{cases} \quad (3.4)$$

- *Logistic function:*

$$f(a) = \frac{1}{1 + \exp(-a)} \quad (3.5)$$

- *Hyperbolic tangent function:*

$$f(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \quad (3.6)$$

The soft-limiting activation functions in (3.5) and (3.6) are often called “sigmoidal” characteristics, as opposed to the hard-limiting activation function in (3.4).

3.5.4 Network learning

3.5.4.1 The type of learning

There are two different types of learning: *supervised learning* and *unsupervised learning*.

- In supervised learning, the desired outputs called “targets” are provided by the trainer. The difference between the actual network outputs and the targets is called the “data error” and is used to adjust the network parameters. For this learning mode, a set of input and target pairs called the “training set” is required, and the network parameters are adjusted until the actual network outputs match the targets.
- In unsupervised learning, the targets are not known. Since no information is available regarding the correctness of the network outputs, learning must be based on observations of responses to inputs that we have some knowledge about. In this mode of learning, the network must discover by itself any possible patterns, regularities and separating properties. While discovering these, the network updates its parameters using a “self-organising” process.

3.5.4.2 Error back-propagation

Network learning is performed by minimising the data error function with regard to the weights in the network. This process requires calculating the error derivative with regard to the weights. The technique of error back-propagation is widely used to calculate the error derivatives. Error back-propagation is one form of supervised learning forms.

Most error back-propagation algorithms involve an iterative procedure with adjustments to the weights being made in a sequence of steps. At each such step we can distinguish between two distinct stages:

- *The first stage:* the derivatives of the error function with regard to the weights must be evaluated. At this stage, errors are propagated backwards through the network and the term “back-propagation” is specifically used to describe the process of evaluating the derivatives.
- *The second stage:* the derivatives are then used to compute the adjustments to be made to the weights. The simplest such technique involves gradient descent.

Thus, it is important to recognise that the two stages are distinct from each other. The first stage, namely the propagation of errors backwards through the network in order to evaluate derivatives, can be applied to many kinds of networks and error functions.

3.5.4.3 Generalisation and regularisation

The learning process of feed-forward networks aims at the minimisation of the data error, given a training set. We expect that after the proper selection of the learning samples and appropriate learning, the trained network can be used well on other data not in the training set. In other words, the trained network can generalise well to new examples, as shown in Figure 3.7. The generalisation of the network is mostly influenced by four factors: 1) the number of samples used in the training phase, 2) the

complexity of the problem under consideration, 3) the magnitudes of network weights and biases, and 4) the network architecture.

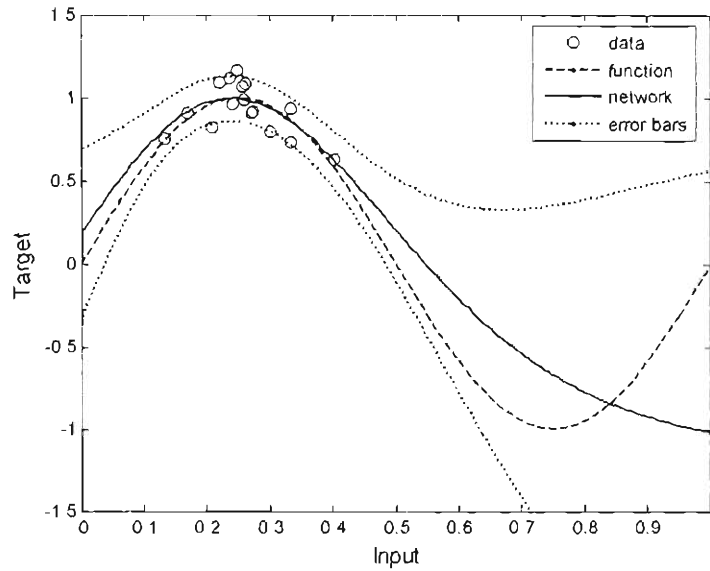


Figure 3.7: An illustration of neural network generalisation: The trained network output generates the smooth solid curve instead of fitting noisy data points (circles) of the true function (dashed curve).

Neural network regularisation is referred to as the reduction of complexity of the network. A common method of network regularisation is weight decay, in which a weight penalty term proportional to the sum of weighted squares is added to the data error function E_D to give

$$S = E_D + \xi \frac{1}{2} \|w\|^2 \quad (3.7)$$

where S is called the cost function and $\xi \frac{1}{2} \|w\|^2$ is known as the weight decay term. ξ is a non-negative parameter and is called the *hyperparameter* needed to be determined properly. There are two common methods used to determine the hyperparameter: cross-validation and Bayesian learning.

- *Cross-validation*: in the simplest form of this method, the amount of weight decay is chosen to optimise performance on a validation set separated from the cases used to estimate the network parameters. However, this method requires intensive search for weight decay. In p -way cross validation, the training set is partitioned into p subsets, each of which is used as the validation set for a network trained on the other $p-1$ subsets. The total error on all these validation sets is used to select an appropriate weight decay.
- *Bayesian learning*: standard network learning is undertaken by maximising the likelihood (equivalent to minimising the data error). In contrast, the result of Bayesian network learning is a posterior distribution of network weights that results in the best network generalisation. This method allows the value of the hyperparameter to be selected automatically without the use of a validation set.

3.5.5 Two-layer perceptron classifiers

3.5.5.1 Forward propagation

The two-layer perceptron shown in Figure 3.8 is commonly used for pattern recognition. The network takes a vector of real inputs, x_i , and from them computes one or more values of activation of the hidden layer.

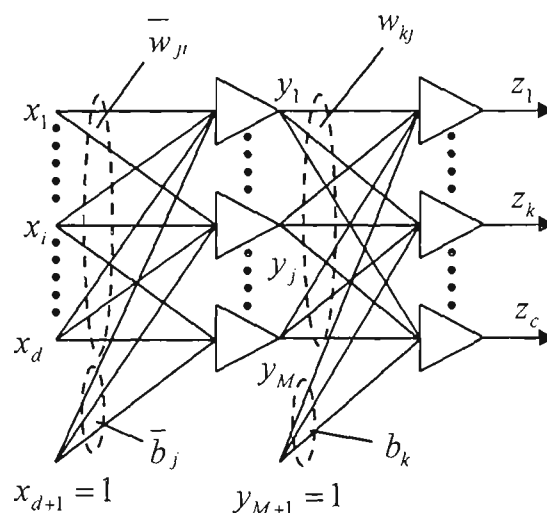


Figure 3.8: A two-layer perceptron classifier.

The value of the j th activation of the hidden layer is computed as follows:

$$\bar{a}_j = \sum_{i=1}^d \bar{w}_{ji} x_i + \bar{b}_j \quad j = 1, \dots, M \quad (3.8)$$

Here \bar{w}_{ji} is the weight on the connection from the i th input node to the j th hidden node, \bar{b}_j is the bias of the j th hidden node, and M is the number of hidden nodes.

The variables \bar{a}_j are then transformed by the non-linear activation functions of the hidden layer. Here we restrict our attention to ‘tanh’ activation functions. The outputs of the hidden nodes are then given by:

$$y_j = \tanh(\bar{a}_j) \quad j = 1, \dots, M \quad (3.9)$$

The outputs y_j of the hidden nodes have the following property:

$$\frac{dy_j}{d\bar{a}_j} = 1 - y_j^2 \quad (3.10)$$

The values y_j are then transformed by the output-layer of weights and biases to give the activations for the output layer as follows:

$$a_k = \sum_{j=1}^M w_{kj} y_j + b_k \quad k = 1, \dots, c \quad (3.11)$$

where w_{kj} is the weight on the connection from the j th hidden node to the k th output node, b_k is the bias of the k th output node, and c is the number of the output classes.

Finally, the values a_k are passed through the output-layer activation function to give the output values of the network z_k . Here we only consider the activation function for classification problems. In c -class ($c > 2$) classification problems, the target variables are discrete class labels indicating one of c possible classes. The “softmax” (generalised logistic) model can be used to define the conditional probabilities of the various classes of a network with c output nodes, as follows:

$$z_k = \frac{\exp(a_k)}{\sum_{k'=1}^c \exp(a_{k'})} \quad (3.12)$$

A significant benefit from using the softmax outputs on classification problems is that the network outputs can be treated directly as estimates of class-conditional probabilities.

3.5.5.2 Error functions

For classification, it is often advantageous to optimise the network outputs to represent the posterior probabilities of each class (Bishop 1995). Let us first consider a classification problem with two output classes C_1 and C_2 . We use a single output z representing the posterior probability $P(C_1 | x)$, where x is the input vector. So the posterior probability of class C_2 is $P(C_2 | x) = 1 - P(C_1 | x)$. The target coding scheme is $t = 1$ if the input vector x belongs to class C_1 and $t = 0$ if it belongs to class C_2 . Therefore, the general form of the posterior probability is

$$p(t | x) = z^t (1 - z)^{1-t} \quad (3.13)$$

Taking the negative logarithm of (3.13) and summing over N sample patterns yields an “cross-entropy” error function as follows

$$E_D = -\sum_{n=1}^N \left\{ t^{(n)} \ln z^{(n)} + (1-t^{(n)}) \ln(1-z^{(n)}) \right\} \quad (3.14)$$

Because the target variables are independent, their joint distribution is given by

$$p(t|x) = \prod_{k=1}^c p(t_k|x) \quad (3.15)$$

From equations (3.14) and (3.15), we obtain the cross-entropy error function for the entire targets as:

$$E_D = -\sum_{n=1}^N \sum_{k=1}^c \left\{ t_k^{(n)} \ln z_k^{(n)} + (1-t_k^{(n)}) \ln(1-z_k^{(n)}) \right\} \quad (3.16)$$

In c -class ($c > 2$) classification problems, the network has a single output variable z_k ($k = 1, \dots, c$) for each class and the target data has a 1-of- c coding, so that $z_k^{(n)} > 0.5$ if the n th pattern belongs to class C_k . The probability of observing the particular target, given an input vector $x^{(n)}$, is $P(C_k | x^{(n)})$. The conditional probability for this pattern can be written as:

$$p(t^{(n)} | x^{(n)}) = \prod_{k=1}^c (z_k^{(n)})^{t_k^{(n)}} \quad (3.17)$$

Similarly, by taking the negative logarithm of (3.17) and summing over N patterns, we obtain an “entropy” error function as:

$$E_D = -\sum_{n=1}^N \sum_{k=1}^c t_k^{(n)} \ln z_k^{(n)} \quad (3.18)$$

3.5.5.3 Network regularisation

For classification perceptrons, a regularisation procedure can be performed by adding the weight decay term to the data error function, as follows:

$$S(w) = E_D + \sum_{g=1}^G \xi_g E_{w_g} \quad (3.19)$$

$$E_{w_g} = \frac{1}{2} \|w_g\|^2 \quad g = 1, \dots, G \quad (3.20)$$

Here $S(w)$ is called the cost function, G is the number of groups of weights and biases in the network. The second term on the right-hand side of equation (3.19) is referred to as the weight decay term. ξ_g is the hyperparameter for the distribution of weights and biases in group g . E_{w_g} and w_g are the error and the vector of weights and biases in group g , respectively.

We can control the degree of regularisation by adjusting the hyperparameters. This technique can prevent any weights from becoming too large, because large weights may result in poor generalisation for new test cases.

3.5.5.4 Error gradient

Back-propagation is a general technique for evaluating derivatives of functions defined in terms of multi-layer perceptron networks. The most common application involves the evaluation of derivatives of an error function with regard to weights and biases in the network. It is based on successive applications of the chain rule of partial derivatives and leads to an algorithm in which “error” signals (error derivatives) are propagated backward through the network, starting from the output nodes.

The first step in evaluating the error derivatives is to perform a forward propagation in order to compute the output values $z_k^{(n)}$ for each pattern n ($n = 1, \dots, N$) in the training

data set. For each pattern n , the partial derivatives of the data error function with respect to $a_k^{(n)}$ are

$$\frac{\partial E_D}{\partial a_k^{(n)}} = z_k^{(n)} - t_k^{(n)} \quad k = 1, \dots, c \quad (3.21)$$

Thus, in order to evaluate the derivatives of E_D with regard to the weight and bias parameters, we first calculate the quantities $\delta_k^{(n)}$ given by:

$$\delta_k^{(n)} = z_k^{(n)} - t_k^{(n)} \quad (3.22)$$

where $\delta_k^{(n)}$ represent “errors” for the output nodes on the n th pattern. The derivatives of E_D with respect to the output-layer weights are given by:

$$\frac{\partial E_D}{\partial w_{kj}} = \sum_{n=1}^N \delta_k^{(n)} y_j^{(n)} \quad (3.23)$$

while the derivatives for the biases of the output nodes are given by:

$$\frac{\partial E_D}{\partial b_k} = \sum_{n=1}^N \delta_k^{(n)} \quad (3.24)$$

In order to find the corresponding derivatives for the hidden-layer parameters, the “error” $\delta_k^{(n)}$ must first be back-propagated through the output-layer weights to obtain “error” signals for the hidden units. The back-propagation equations take the form:

$$\bar{\delta}_j^{(n)} = \frac{\partial y_j}{\partial a_j^{(n)}} \sum_{k=1}^c w_{kj} \delta_k^{(n)} = \left(1 - (y_j^{(n)})^2\right) \sum_{k=1}^c w_{kj} \delta_k^{(n)} \quad (3.25)$$

The derivatives with regard to the hidden-layer weights are then given by:

$$\frac{\partial E_D}{\partial w_{ji}} = \sum_{n=1}^N \bar{\delta}_j^{(n)} x_i^{(n)} \quad (3.26)$$

while the derivatives for the biases of the hidden nodes are given by:

$$\frac{\partial E_D}{\partial b_j} = \sum_{n=1}^N \bar{\delta}_j^{(n)} \quad (3.27)$$

As the cost function $S(w)$ is the sum of the data error function and the weight decay term, the derivatives of the cost function are equal to the sum of the derivatives of E_D and the derivatives of the weight decay.

The derivatives of the weight decay term with regard to the weights and biases are computed as:

$$\frac{\partial}{\partial w_{ji}} \left(\sum_{g=1}^G \xi_g E_{W_g} \right) = \xi_1 \bar{w}_{ji} \quad (3.28)$$

$$\frac{\partial}{\partial b_j} \left(\sum_{g=1}^G \xi_g E_{W_g} \right) = \xi_2 \bar{b}_j \quad (3.29)$$

$$\frac{\partial}{\partial w_{kj}} \left(\sum_{g=1}^G \xi_g E_{W_g} \right) = \xi_3 w_{kj} \quad (3.30)$$

$$\frac{\partial}{\partial b_k} \left(\sum_{g=1}^G \xi_g E_{W_g} \right) = \xi_4 b_k \quad (3.31)$$

where four hyperparameters ξ_1 , ξ_2 , ξ_3 and ξ_4 correspond to four distributions of the weight and bias groups in the network: 1) the weights on the connection from the input node to the hidden nodes, 2) the biases of the hidden nodes, 3) the weights on the connection from the hidden nodes to the output nodes and 4) the biases of the output nodes.

Finally, the derivative components of the cost function with regard to the weights and biases are grouped into a single vector convenient for minimising the cost function in the network training phase.

3.5.5.5 The Hessian matrix

In some cases such as Bayesian learning, we also have to evaluate the Hessian matrix (second-order derivatives) for the data error function E_D at a particular weight vector w_0 :

$$H = \nabla \nabla E_D(w_0) \quad (3.32)$$

However, it is more convenient to indirectly evaluate the Hessian matrix H by calculating the product of the Hessian matrix and an arbitrary vector v , and then returning the full Hessian matrix from that product.

The Hessian matrix H appears in the expansion of the error gradient evaluated at w_0 as follows:

$$\nabla E_D(w_0 + \Delta w_0) = \nabla E_D(w_0) + H \Delta w_0 + O(\|\Delta w_0\|^2) \quad (3.33)$$

where ∇E_D is the gradient of E_D and $O(\|\Delta w_0\|^2)$ is order $\|\Delta w_0\|^2$ operations. If we choose $\Delta w_0 = r v$, where r is a small value, then equation (3.33) becomes:

$$Hv = \frac{\nabla E_D(w_0 + rv) - \nabla E_D(w_0)}{r} + O(r) \quad (3.34)$$

If r is small enough, $O(r)$ is insignificant and Hv is given by:

$$Hv = \frac{\nabla E_D(w_0 + rv) - \nabla E_D(w_0)}{r} \quad (3.35)$$

Møller (Møller 1993) chooses the value of r relating to vector v as follows:

$$r = \frac{r_0}{\|v\|} \quad (3.36)$$

where r_0 is a very small number. Equation (3.35) is used to approximate the product of the Hessian matrix and the search direction in the scaled conjugate gradient training algorithm.

Fortunately, there is a way to exactly compute Hv , rather than just approximating it. Pearlmutter (Pearlmutter 1994) developed the $\mathfrak{R}\{\cdot\}$ operator technique to directly calculate Hv . Firstly, the limit of equation (3.35) is taken as follows:

$$Hv = \lim_{r \rightarrow 0} \left[\frac{\nabla E_D(w_0 + rv) - \nabla E_D(w_0)}{r} + O(r) \right] = \frac{\partial}{\partial r} \nabla E_D(w_0 + rv) \Big|_{r=0} \quad (3.37)$$

Then a differential operator is defined as follows:

$$\mathfrak{R}\{f(w_0)\} = \frac{\partial}{\partial r} f(w_0 + rv) \Big|_{r=0} \quad (3.38)$$

We note that

$$\Re\{\nabla E_D(w_0)\} = Hv \quad (3.39)$$

$$\Re\{w_0\} = v \quad (3.40)$$

We now consider how to apply the $\Re\{\cdot\}$ operator technique for neural networks. Firstly, we have

$$\Re\{\bar{a}_j\} = \sum_{i=1}^d v_{ji} x_i \quad (3.41)$$

where v_{ji} is the element of v corresponding to w_{ji} .

$$\Re\{y_j\} = g'(\bar{a}_j) \Re\{\bar{a}_j\} \quad (3.42)$$

$$\Re\{a_k\} = \sum_{j=1}^M v_{kj} y_j + \sum_{j=1}^M w_{kj} \Re\{y_j\} \quad (3.43)$$

For softmax outputs, $\Re\{z_k\}$ is computed as:

$$\Re\{z_k\} = z_k \Re\{a_k\} - z_k \sum_{k'=1}^c z_{k'} \Re\{a_{k'}\} \quad (3.44)$$

Using (3.22) and (3.25) and the operator $\Re\{\cdot\}$, we obtain

$$\Re\{\delta_k\} = \Re\{z_k\} \quad (3.45)$$

$$\Re\{\bar{\delta}_j\} = g''(\bar{a}_j) \Re\{\bar{a}_j\} \sum_{k=1}^c w_{kj} \delta_k + g'(\bar{a}_j) \sum_{k=1}^c v_{kj} \delta_k + g'(\bar{a}_j) \sum_{k=1}^c w_{kj} \Re\{\delta_k\} \quad (3.46)$$

Finally, from (3.23), (3.24), (3.26) and (3.27) and the operator $\Re\{\cdot\}$, we have

$$\Re\left\{\frac{\partial E_D}{\partial w_{kj}}\right\} = \sum_{n=1}^N \left(\Re\{\delta_k^{(n)}\} y_j^{(n)} + \delta_k^{(n)} \Re\{y_j^{(n)}\} \right) \quad (3.47)$$

$$\Re\left\{\frac{\partial E_D}{\partial b_k}\right\} = \sum_{n=1}^N \Re\{\delta_k^{(n)}\} \quad (3.48)$$

$$\Re\left\{\frac{\partial E_D}{\partial w_{j'}}\right\} = \sum_{n=1}^N \Re\{\delta_j^{(n)}\} x_j^{(n)} \quad (3.49)$$

$$\Re\left\{\frac{\partial E_D}{\partial b_j}\right\} = \sum_{n=1}^N \Re\{\delta_j^{(n)}\} \quad (3.50)$$

Finally, all the components of the product Hv are grouped into a single vector. The full Hessian matrix H is then returned using the W -by- W identity matrix, where W is the number of weights and biases. Each column in the identity matrix can be seen as an arbitrary vector v . The product of the Hessian matrix and each column in the identity matrix returns a corresponding column of the full Hessian matrix. Therefore, in order to return all the columns of the full Hessian matrix, there is a need for W operations of Hv .

3.5.5.6 Adjusting the weights and biases

The problem of learning in neural networks has been formulated in terms of the minimisation of the cost function $S(w)$, which is a function of the adaptive parameters (weights and biases) in the network. Also, we can conveniently group the network weights and biases together into a single W -dimensional weight vector, denoted by w , with components $w_1 \dots w_W$.

For more general networks with more than one layer of adaptive weights, the cost function will typically be a highly non-linear function of the weights, and there may be many minima satisfying the following condition:

$$\nabla S(w) = 0 \quad (3.51)$$

The minimum for which the value of the cost function is smallest is called the *global minimum*, while other minima are called *local minima*. In practice, it is impossible to find closed-form solutions for the minima. Instead, we consider algorithms that involve a search through the weight space with a succession of steps of the form:

$$w_{m+1} = w_m + \alpha_m d_m \quad (3.52)$$

where m labels the iteration step. Different advanced training algorithms differ in the choice of *search direction* d_m and the method used to determine the *step size* α_m .

3.6 Discussion

High-level SCIs result in complete paralysis. Therefore, the remaining functions such as gestures from the neck upward or brain signals should be utilised to indicate intentions or to generate control commands. On the other hand, neural networks capable of learning by examples can allow us to develop hands-free pointing devices that can comfortably be used by various forms of disabled users.

Two-layer perceptron neural networks are able to approximate any function and are commonly used in many applications of pattern recognition. This type of neural network can therefore be used to successfully detect hands-free signals, including head movements and EEG.

The performance of a trained neural network depends on how well it can generalise to new data. Regularisation is traditionally used to improve the generalisation of neural networks. Weight decay is one form of regularisation, in which the magnitudes of

weights and biases in the network are constrained or penalised by introducing a term which is proportional to the sum of weighted squares and is added to the data error function. The determination of the proper weight decay term is, in general, very demanding.

The minimisation of the data error needs to compute the error gradient. Error back-propagation is popularly used for this task. In some circumstances, there is also a need to evaluate the Hessian matrix of the error function. Instead of evaluating the Hessian matrix directly, the $\Re\{\cdot\}$ operator technique can be used to calculate the product of the Hessian matrix and an arbitrary vector. The full Hessian matrix is then returned by multiplying that product by the identity matrix.

Chapter 4

A Bayesian Neural Network for Detecting Head-movement Commands

4.1 Introduction

In recent years, several head-pointing devices have been developed to allow disabled people to control power wheelchairs. However, in contrast with other head pointing devices, we have been focused on developing perceptron classifiers that are capable of effectively detecting head-direction commands from various forms of wheelchair users (Joseph and Nguyen 1998; Taylor and Nguyen 2003; Nguyen, King et al. 2004). Especially, Bayesian neural networks (BNNs) have recently been used to improve the head-movement classification accuracy. In 2004, it was shown that a trained BNN is able to classify the head movements of severely disabled persons accurately and consistently (S.T.Nguyen, H.T.Nguyen et al. 2004; S.T.Nguyen, H.T.Nguyen et al. 2006a; S.T.Nguyen, H.T.Nguyen et al. 2006b).

Figure 4.1 illustrates the block diagram for controlling the travel direction of power wheelchairs using head movements. The wheelchair platform is based on a commercial power wheelchair (Roller M1). The movement of the user's head is detected by analysing data from the output of a small tilt sensor, shown in Figure 4.2, installed in a cap worn by the user. The tilt sensor (see Appendix B) consists of a dual-axis accelerometer and a low-cost PIC microcontroller. The output of the sensor is connected to the serial port of a laptop computer via an RS232 communication protocol.

The head movement data is collected with a sampling period of 100 ms. The laptop computer, working as a central controller and containing a trainable BNN classifier, is responsible for classifying input signals into output classes corresponding to forward,

backward, left and right movements in real-time.

The start of the head movement for controlling the travel direction of the chair is determined to be the point where the deviation from the neutral position reaches 25% of the maximum value on the relevant axis. The classification of the movement is determined as the first classification made by the BNN after the start of the movement. The input to the BNN comprises a window of 20 samples from each axis. The success of the system heavily depends on the network training.

The computer interface as shown in Figure 4.3 consists of the feedback to the user: real-time graphical displays of the accelerometer data allow the user to track the deviation of his/her head from the neutral position. Boolean outputs from the classifier and the numerical values of the BNN inform the user about how the classifier is interpreting his/her head movements.

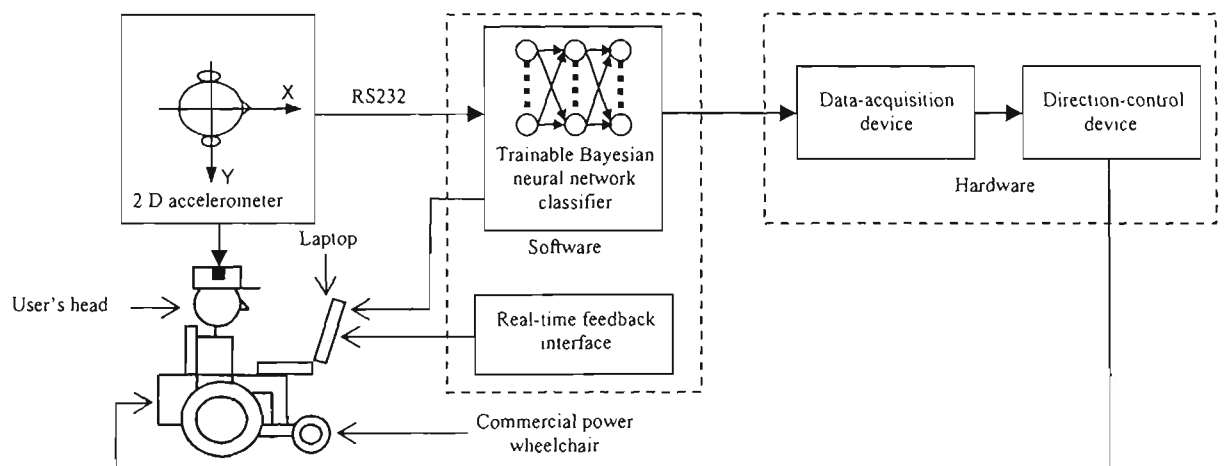


Figure 4.1: Block diagram of head-movement-based direction control of power wheelchair using a Bayesian neural network.

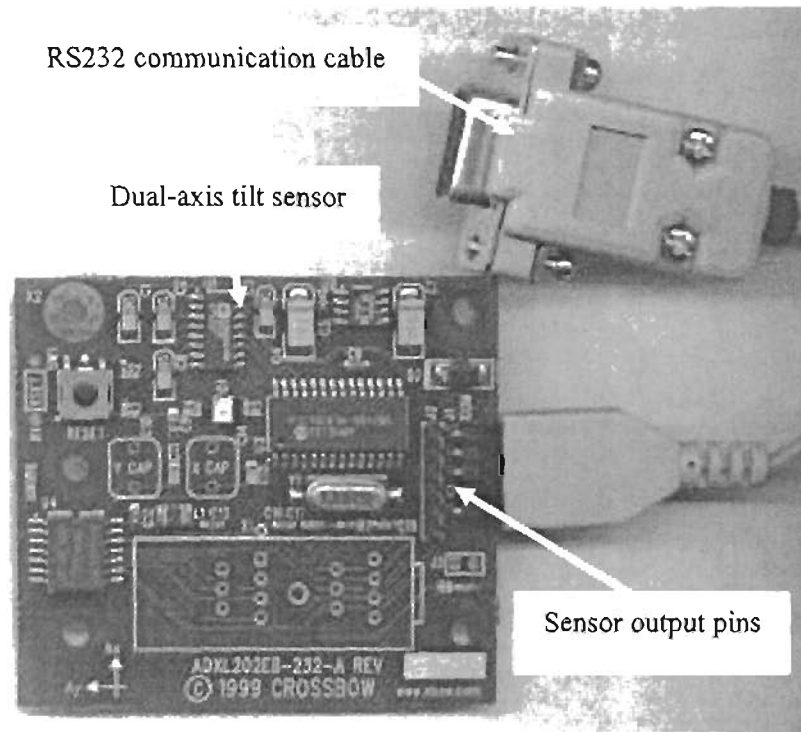


Figure 4.2: Dual-axis tilt sensor (ADXL202EB-232A).

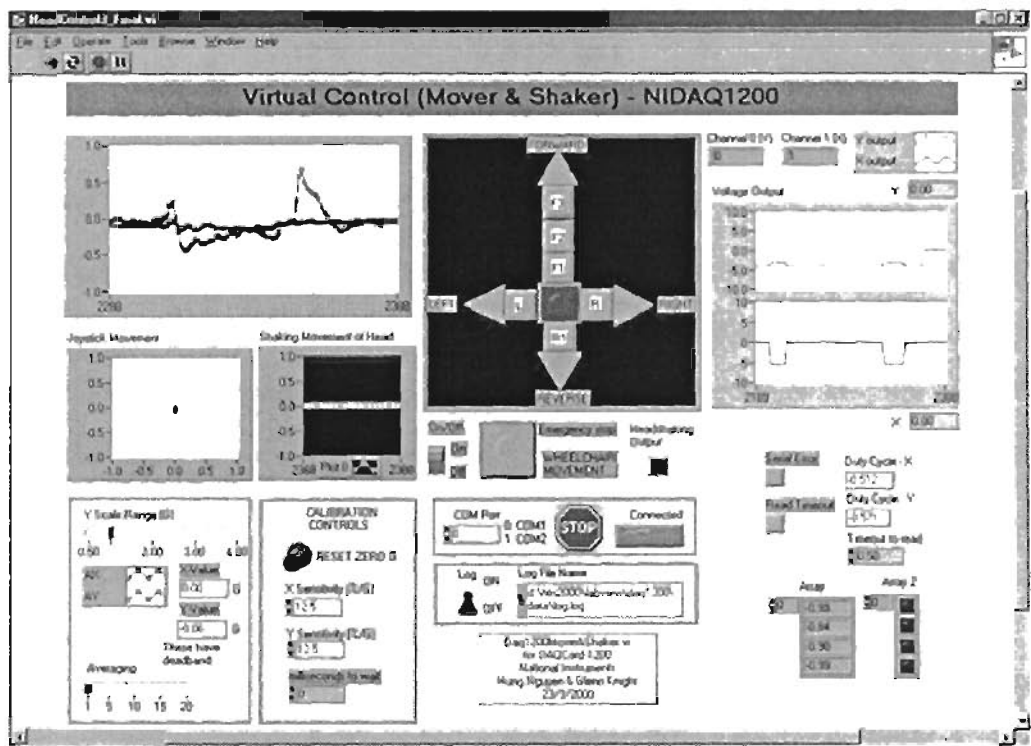


Figure 4.3: LabVIEW real-time computer interface.

The Boolean outputs of the BNN classifier are converted into corresponding values at the analog output of the data-acquisition device, which is a National Instruments USB 6008 multifunction data-acquisition box (see Appendix D). The output of the data-

acquisition module is then fed to the input of the direction-control device (see Appendix C) to control the travel direction of the wheelchair.

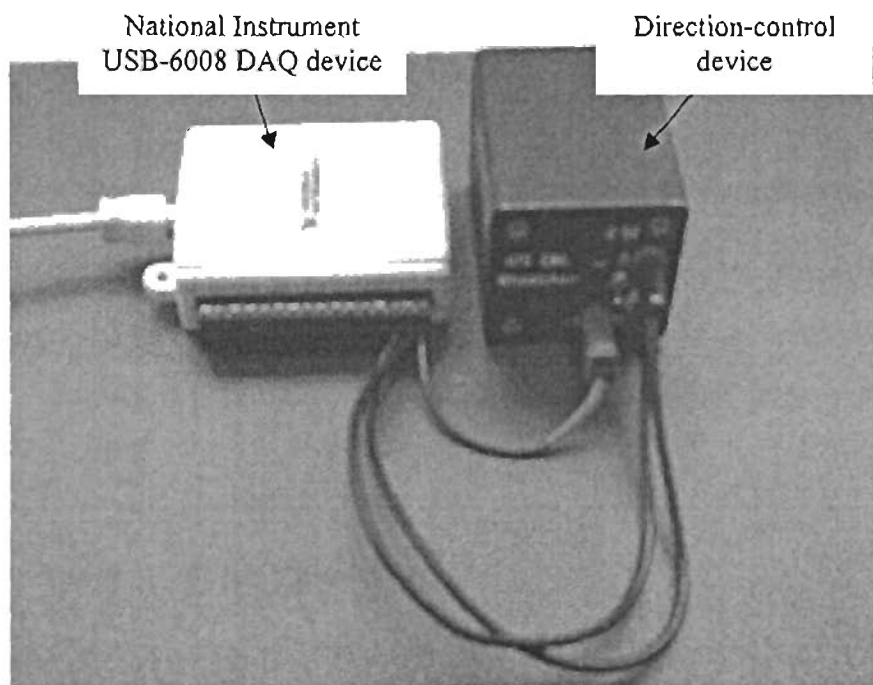


Figure 4.4: Data-acquisition and direction-control devices.

4.2 The Bayesian neural network classifier

Maximum likelihood network learning techniques attempt to find a single vector of weights to minimise the data error. In contrast, the Bayesian approach considers a probability distribution function over the weight space, representing the relative degree of belief in different values for the weight vector. This function is initially set to a prior distribution. Once the data has been observed, it can be converted to a posterior distribution using Bayes' theorem. The posterior distribution can be used to evaluate the predictions of the network for new values of the input variables.

4.2.1 Prior weight distribution

In the neural network, the weights and biases can be divided into G groups. Let W_g denote the number of weights and biases in group g and w_g denotes the vector of weights and biases in group g .

For convenience, the prior distribution of the weight vector in group g is assumed to be a zero-mean Gaussian (see Appendix A):

$$p(w_g | \xi_g) = \sqrt{\frac{\xi_g}{2\pi}} \exp\left(-\frac{1}{2} \xi_g w_g^2\right) \quad (4.1)$$

where ξ_g is the hyperparameter for constraining weights and biases in group g .

The prior weight distribution in (4.1) is defined based on the assumption that positive and negative weights are equally frequent and the weights have a finite variance (MacKay 1992).

The prior distribution of all the weights w is

$$p(w | \psi) = \frac{1}{Z_w(\psi)} \exp\left(-\sum_{g=1}^G \xi_g E_{w_g}\right) \quad (4.2)$$

where ψ is the vector of hyperparameters.

$$\psi = [\xi_1 \dots \xi_G]^T \quad (4.3)$$

$Z_w(\psi)$ is a normalisation constant given by (see Appendix A)

$$Z_w(\psi) = \prod_{g=1}^G \left(\frac{2\pi}{\xi_g}\right)^{w_g/2} \quad (4.4)$$

4.2.2 Posterior weight distribution

We would like to adjust the weights to their most probable (MP) values given the training data D . We can also compute the posterior weight distribution using Bayes' theorem in the form:

$$p(w|D,\psi) = \frac{p(D|w,\psi)p(w|\psi)}{p(D|\psi)} \quad (4.5)$$

Equation (4.5) is expressed in words as follows:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} \quad (4.6)$$

For convenience, the posterior distribution of weights and biases is also assumed to be a Gaussian distribution with the form:

$$p(w|D,\psi) = \frac{\exp(-S(w))}{Z_S(\psi)} \quad (4.7)$$

where $Z_S(\psi)$ is called a Gaussian integral given by:

$$Z_S(\psi) = \int \exp(-S(w)) dw \quad (4.8)$$

In the Bayesian framework, the most probable weight vector w_{MP} should maximise the posterior weight distribution. This task is equivalent to minimising the cost function $S(w)$.

4.2.3 Posterior probability of the hyperparameters

So far the most probable weight vector, w_{MP} , given some values of the hyperparameters, have not yet been determined. Now, we have to determine the hyperparameters given the model and data. Again, we can express the posterior distribution of the hyperparameters using Bayes' theorem as follows:

$$p(\psi|D) = \frac{p(D|\psi)p(\psi)}{p(D)} \quad (4.9)$$

In equation (4.9), the denominator $p(D)$ is independent of the hyperparameters. $p(\psi)$ is the prior distribution of the hyperparameters and is simply assumed to be an uniform distribution. Thus $p(\psi)$ will be ignored subsequently, because to infer the values of the hyperparameters, we only search the values of the hyperparameters to maximise $p(D|\psi)$, which is called the *evidence* in (4.5). On the other hand, the evidence can be exactly computed by taking the following integral:

$$p(D|\psi) = \int p(D|w, \psi) p(w|\psi) dw \quad (4.10)$$

where $p(D|w, \psi)$ is the probability of the data, traditionally called the dataset likelihood and has the form:

$$p(D|w, \psi) = \exp(-E_D) \quad (4.11)$$

By putting (4.11) and (4.2) into (4.10), we have

$$p(D|\psi) = \int \exp(-E_D) \frac{1}{Z_w(\psi)} \exp\left(-\sum_{g=1}^G \xi_g E_{w_g}\right) dw \quad (4.12)$$

$$= \frac{Z_S(\psi)}{Z_w(\psi)} \quad (4.13)$$

4.2.4 The Gaussian approximation to the evidence

In order to compute $Z_S(\psi)$, the cost function $S(w)$ is approximated around the most probable weight w_{MP} by the quadratic form:

$$S(w) = S(w_{MP}) + \frac{1}{2} (w - w_{MP})^T A (w - w_{MP}) \quad (4.14)$$

where A is the Hessian matrix of the cost function $S(w)$ evaluated at w_{MP} and is given by:

$$A = H + \sum_{g=1}^G \xi_g I_g \quad (4.15)$$

In (4.15), $H = \nabla \nabla E_D(w_{MP})$, the Hessian matrix of the data error function E_D evaluated at w_{MP} , and $I_g = \nabla \nabla E_{w_g}$ is a diagonal matrix having ones along the diagonal that picks off weights and biases in group g . So $Z_S(\psi)$ in (4.13) is a Gaussian integral that can be approximated to the form:

$$Z_S(\psi) \approx \exp(-S(w_{MP})) (2\pi)^{W/2} (\det A)^{-1/2} \quad (4.16)$$

By substituting (4.16) and (4.4) into (4.13), we obtain

$$p(D|\psi) = \exp(-S(w_{MP})) (\det A)^{-1/2} \left(\prod_{g=1}^G (\xi_g)^{W_g/2} \right) \quad (4.17)$$

Taking the logarithm of (4.17) gives

$$\ln p(D|\psi) = -S(w_{MP}) - \frac{1}{2} \ln(\det A) + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g \quad (4.18)$$

Equation (4.18) is the basis for the determination of the hyperparameters.

4.2.5 Determination of the hyperparameters

The most probable hyperparameters are determined by taking the derivative of (4.18) with regard to ξ_g ($g = 1, \dots, G$).

$$\frac{\partial}{\partial \xi_g} \ln p(D|\psi) = -E_{W_g}^{MP} + \frac{W_g}{2\xi_g} - \frac{1}{2} \text{tr}(A^{-1}I_g) \quad (4.19)$$

By setting (4.19) to zero, we obtain the following relationship:

$$2E_{W_g}^{MP} \xi_g = W_g - \xi_g \text{tr}(A^{-1}I_g) \quad (4.20)$$

The right-hand side of equation (4.20) is equal to a value γ_g defined as the number of *well-determined* parameters for the weights and biases in group g . Finally, ξ_g is given by

$$\xi_g = \frac{\gamma_g}{2E_{W_g}^{MP}} \quad (4.21)$$

4.3 Advanced optimisation training algorithms for Bayesian neural network classifiers

The BNN training requires minimising the cost function $S(w)$, which is a non-linear function of the weights. For many years, the problem of minimising continuous, differentiable functions of multi-dimensional variables has been widely studied, and various approaches are directly applicable to neural network training.

The network training consists of a sequence of iterative steps. At the m th step, we make a step size α_m in the search direction d_m to adjust or update the weights as follows:

$$w_{m+1} = w_m + \alpha_m d_m \quad (4.22)$$

In the gradient descent algorithm, the step size α_m is fixed for every step and is usually called the learning rate η . The algorithm also makes the simplest choice for d_m by setting it to $-g_m$ (negative gradient).

$$w_{m+1} = w_m - \eta g_m \quad (4.23)$$

One of the limitations of the gradient descent technique is the need to choose a suitable value for the learning rate η . By setting the value of η “too” large, the function value may increase. By setting η sufficiently small, we can always ensure that the function value decreases steadily. However, it is possible that if very small reductions in the function values are taken, then any limiting sequence may not even be a local minimum. In order to overcome this problem, a procedure called *line search* is used to exactly find a local minimum of the cost function, given the search direction (Section 4.3.1).

Unfortunately, even with the use of line search, the algorithm still exhibits oscillatory behaviour when it encounters steep valleys and progresses too slowly. Also, the negative gradient search direction $-g_m$ does not point directly towards the minimum. The algorithm then takes many steps to reach the minimum and is a very inefficient procedure.

One very simple technique for reducing iterative steps is to use a *momentum* term μ ($0 < \mu \leq 1$) as follows:

$$w_{m+1} = w_m - \eta g_m + \mu(w_m - w_{m-1}) \quad (4.24)$$

The use of the momentum can lead to a significant improvement in the performance of gradient descent. However, the inclusion of this term also introduces a second parameter whose value needs to be chosen.

It is, therefore, necessary to consider algorithms that are capable of choosing the step size and search direction properly. This section presents three advanced optimisation training algorithms for BNN classifiers. They are conjugate gradient, scaled conjugate gradient and quasi-Newton algorithms, which belong to a class of automated non-linear parameter optimisation techniques. In contrast with the gradient descent algorithm,

which depends on parameters specified by the user, these algorithms are able to automatically adjust the step size and search direction to obtain the fast convergence for network training.

4.3.1 Line search

The line search is required for the conjugate gradient algorithm (Section 4.3.2) and quasi-Newton algorithm (Section 4.3.4). The line search is a process for searching the step size α_{\min} to minimise the cost function $S(w)$ given the search direction d_m . A line search consists of two stages:

- *Bracketing the minimum*: this stage aims to find an interval that includes a triple $a < b < c$ such that $S(a) > S(b)$ and $S(c) > S(b)$. Since the cost function is continuous, this always ensures that there is a local minimum somewhere in the interval (a, b) .
- *Locating the minimum itself*: since the cost function is smooth and continuous, the minimum can simply be obtained by a process of parabolic interpolation. This involves fitting a quadratic polynomial to the cost function through three successive points $a < b < c$ and then moving to the minimum of the parabola.

4.3.1.1 Bracketing the minimum

For a given bracketing triple $\{a, b, c\}$, we wish to find a new bracket as narrow as possible, shown in Figure 4.5. This task requires choosing a new trial point $x \in (a, c)$ and verifying this point. The position of the new trial point x can be determined using the *golden section search*:

$$\frac{c-x}{c-a} = 1 - \frac{1}{\phi} \quad (4.25)$$

where ϕ is called the *golden ratio*.

$$\phi = \frac{1}{2}(1 + \sqrt{5}) = 1.6180339887 \tag{4.26}$$

If $x > b$ then the new bracketing triple is $\{a, b, x\}$ if $S(x) > S(b)$ and is $\{b, x, c\}$ if $S(x) < S(b)$. A similar choice is made if $x < b$. This algorithm is very robust, since no assumption is made about the function being minimised, other than continuity.

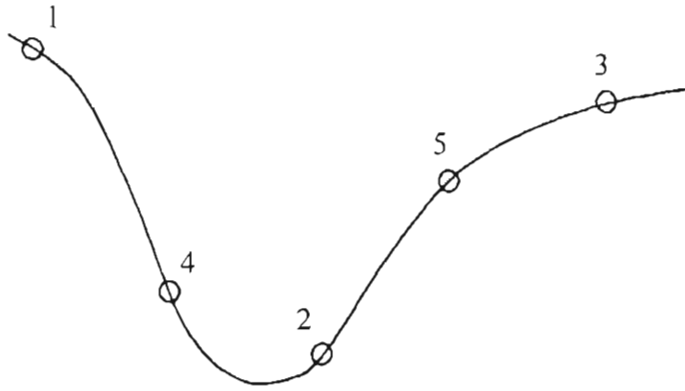


Figure 4.5: Golden section search: initial bracket (1, 2, 3) becomes (4, 2, 3), (4, 2, 5), etc.

4.3.1.2 Locating the minimum itself

As shown in Figure 4.6, the minimum point of the cost function inside the interval (a, c) can be approximated as a point d that is the minimum of a parabola fitted through three points $(a, S(a))$, $(b, S(b))$ and $(c, S(c))$ as:

$$d = b - \frac{1}{2} \frac{(b-a)^2 [S(b) - S(c)] - (b-c)^2 [S(b) - S(a)]}{(b-a)[S(b) - S(c)] - (b-c)[S(b) - S(a)]} \tag{4.27}$$

However, equation (4.27) does not guarantee that d always lies inside the interval (a, c) . Also, if b is already at or near the minimum of the interpolating parabola, then the new point d is close to b . This causes slow convergence if b is not close to the local minimum. In order to avoid this problem, this approach must be combined with the golden section search for finding the exact local minimum.

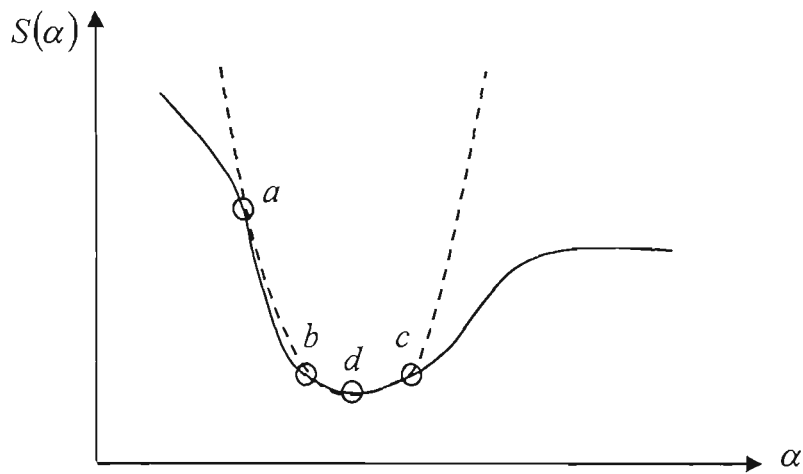


Figure 4.6: An illustration of the process of parabolic interpolation used to perform line search minimisation: A parabola (shown dotted) is fitted to the three points a , b , c . The minimum of the parabola, at d , gives an approximation to the minimum of $S(\alpha)$.

4.3.2 Conjugate-gradient training algorithm

In the conjugate-gradient training algorithm, the new search direction is chosen so that (Charalambous 1992; Bishop 1995):

$$d_m^T A_m d_{m-1} = 0 \quad (4.28)$$

d_m is the search direction at step m and d_{m-1} is the search direction at step $m-1$. A_m is the Hessian matrix of the cost function evaluated at w_m . We say that d_m is *conjugate* to d_{m-1} . If the cost function has the quadratic form, the step size d_m can be written in the form:

$$\alpha_m = \frac{g_m^T d_m}{d_m^T A_m d_m} \quad (4.29)$$

According to equation (4.29), the determination of α_m requires the evaluation of the Hessian matrix A_m . However, α_m can be found by performing a “very accurate” line search. The new search direction is then given by:

$$d_{m+1} = -g_{m+1} + \beta_m d_m \quad (4.30)$$

where

$$\beta_m = \frac{(g_{m+1} - g_m)^T g_{m+1}}{g_m^T g_m} \quad (4.31)$$

Equation (4.31) is called the *Polak-Ribiere* expression.

Finally, the key steps of the algorithm can be summarised as follows:

1. Choosing an initial weight vector w_1 .
2. Evaluating the gradient vector g_1 and setting the initial search direction $d_1 = -g_1$.
3. At step m , minimising $S(w_m + \alpha d_m)$ with regard to α to give $w_{m+1} = w_m + \alpha_{\min} d_m$.
4. Testing to see whether the stopping criterion is satisfied.
5. Evaluating the new gradient vector g_{m+1} .
6. Evaluating the new search direction using equations (4.30) and (4.31).

4.3.3 Scaled conjugate-gradient training algorithm

We have seen that the use of line search allows the step size in the conjugate gradient algorithm to be chosen without having to evaluate the Hessian matrix. However, the line search itself introduces some problems. In particular, every line minimisation involves several error function evaluations, each of which is computationally expensive. Also, the line search itself necessarily involves some parameters whose values determine the termination for each line search. The overall performance of the algorithm can be sensitive to the value of these parameters. An insufficiently accurate line search causes an incorrectly determined value of α_m , while an excessively accurate line search can represent a good deal of wasted computation.

Møller (Møller 1993) introduced the scaled conjugate gradient algorithm as a way to avoid the line-search procedure of the conventional conjugate gradient algorithm. In particular, in order to make the denominator of (4.29) always positive to avoid the increase in the value of the cost function, a multiple of the identity matrix is added to the Hessian matrix in the denominator of (4.29) to obtain

$$\alpha_m = \frac{\mathbf{g}_m^T \mathbf{d}_m}{\mathbf{d}_m^T \mathbf{A}_m \mathbf{d}_m + \lambda_m \|\mathbf{d}_m\|^2} \quad (4.32)$$

where λ_m is a non-negative *scale parameter* for adjusting the step size α_m . The denominator in (4.32) can be written as:

$$\delta_m = \mathbf{d}_m^T \mathbf{A}_m \mathbf{d}_m + \lambda_m \|\mathbf{d}_m\|^2 \quad (4.33)$$

If $\delta_m < 0$, we can increase λ_m to make $\delta_m > 0$. Let the raised value of λ_m be $\bar{\lambda}_m$. Then the corresponding raised value of δ_m is given by:

$$\bar{\delta}_m = \delta_m + (\bar{\lambda}_m - \lambda_m) \|\mathbf{d}_m\|^2 \quad (4.34)$$

In order to make $\bar{\delta}_m > 0$, Møller (1993) chooses

$$\bar{\lambda}_m = 2 \left(\lambda_m - \frac{\delta_m}{\|d_m\|^2} \right) \quad (4.35)$$

Substituting (4.35) into (4.34) gives

$$\bar{\delta}_m = -\delta_m + \lambda_m \|d_m\|^2 \quad (4.36)$$

$\bar{\delta}_m$ is now positive. This value is used as the denominator in (4.32) to compute the step size α_m . In order to find λ_{m+1} , a comparison parameter is firstly defined as:

$$\Delta_m = 2 \left[\frac{S(w_m) - S(w_m + \alpha_m d_m)}{\alpha_m d_m^T d_m} \right] \quad (4.37)$$

Then the value of λ_{m+1} can be adjusted using the following prescriptions:

$$\text{If } \Delta_m > 0.75 \text{ then } \lambda_{m+1} = \frac{\lambda_m}{2} \quad (4.38)$$

$$\text{If } \Delta_m < 0.25 \text{ then } \lambda_{m+1} = 4\lambda_m \quad (4.39)$$

In equation (4.32), $d_m^T A_m d_m$ can be approximated as (Pearlmutter 1994):

$$d_m^T A_m d_m \approx d_m^T \left[\frac{\nabla S(w_m + \sigma d_m) - \nabla S(w_m)}{\sigma} \right] \quad (4.40)$$

where $\sigma = \frac{\sigma_0}{\|d_m\|}$ and σ_0 is chosen to be a very small value.

By substituting (4.40) into (4.32), we have

$$\alpha_m = \frac{g_m^T d_m}{d_m^T \left[\frac{\nabla S(w_m + \sigma d_m) - \nabla S(w_m)}{\sigma} \right] + \lambda_m \|d_m\|^2} \quad (4.41)$$

Equation (4.41) is used to scale the step size. The new search direction is also determined using the same procedure in the conjugate gradient algorithm.

Finally, the algorithm is summarised as follows:

1. Choosing the initial weight vector w_1 .
2. Evaluating the gradient vector g_1 , setting the initial search direction $d_1 = -g_1$ and the initial scale parameter $\lambda_1 = 1$.
3. At step m , adjusting the step size using (4.41) to give $w_{m+1} = w_m + \alpha_m d_m$.
4. Testing to see whether the stopping criterion is satisfied.
5. Evaluating the new gradient vector g_{m+1} .
6. Evaluating the new search direction using equations (4.30) and (4.31).
7. Evaluating the new scale parameter using (4.37), (4.38) and (4.39).

4.3.4 Quasi-Newton training algorithm

In the Newton method, the network weights can be updated as:

$$w_{m+1} = w_m - A_m^{-1} g_m \quad (4.42)$$

The vector $-A_m^{-1}g_m$ is called the *Newton direction* or the *Newton step*. This method is attractive in principle, but there are some practical problems if applying it directly. Firstly, the evaluation of the Hessian matrix can be very demanding computationally. If the cost function is not quadratic, the Hessian matrix may not be positive definite and will lead to an increase in the cost function value.

From equation (4.42), we can form the relationship between the weight vectors at steps m and $m+1$ as:

$$w_{m+1} = w_m - \alpha_m F_m g_m \quad (4.43)$$

If $\alpha_m = 1$ and $F_m = A_m^{-1}$, we have the Newton method, while if $F_m = I$, we have gradient descent with learning rate α_m .

We can also choose F_m to be an approximation to the Hessian matrix. It is important that F_m must be positive definite so that for small α_m we obtain a descent method. In practice, the value of α_m is found by a line search. Equation (4.43) is known as the quasi-Newton condition. The most successful method to compute F_m is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) formula:

$$F_{m+1} = F_m + \frac{pp^T}{p^T v} - \frac{(F_m v)v^T F_m}{v^T F_m v} + (v^T F_m v)uu^T \quad (4.44)$$

where p , v and u are defined as:

$$p = w_{m+1} - w_m \quad (4.45)$$

$$v = g_{m+1} - g_m \quad (4.46)$$

$$u = \frac{p}{p^T v} - \frac{F_m v}{v^T F_m v} \quad (4.47)$$

Compared to the conjugate gradient algorithm, the line search with great accuracy is no longer required, since the line search does not form a critical factor in the algorithm. For the conjugate gradient algorithm, the line search needs to be performed accurately to ensure that the search direction is set correctly.

Finally, the main steps of the algorithm are summarised as follows:

1. Choosing the initial weight vector w_1 .
2. Evaluating the gradient vector g_1 , setting the initial search direction $d_1 = -g_1$ and $F_1 = I$, which is a W by W identity matrix, where W is the number of weights in the network.
3. At step m , minimising $S(w_m + \alpha d_m)$ with regard to α to give $w_{m+1} = w_m - \alpha_{\min} F_m g_m$.
4. Testing to see whether the stopping criterion is satisfied.
5. Evaluating the new gradient vector g_{m+1} .
6. Evaluating the new matrix F_{m+1} using (4.44), (4.45), (4.46), and (4.47).

4.3.5 Training time

The network training time T_{tr} can be measured as:

$$T_{tr} = \sum_{q=1}^Q (C_q + R_q) \quad (4.48)$$

where the index q labels the hyperparameter re-estimation period. C_q is the local minimum convergence time of the cost function, R_q is the time for calculating the new values of the hyperparameters, and Q is the number of hyperparameter re-estimation periods.

On the other hand, C_q is given by:

$$C_q = \sum_{m=1}^{M_q} T_q^{(m)} \quad (4.49)$$

where $T_q^{(m)}$ is the duration of cycle m and M_q is the number of training steps in that cycle.

For a given training data set, C_q depends on the algorithm selected to train the network and R_q significantly depends on the time of evaluating the Hessian matrix of the cost function at the most probable weight vector w_{MP} . Since R_q does not depend on the training algorithm, the network training time can be expressed as:

$$T_{tr} \equiv \sum_{q=1}^Q \sum_{m=1}^{M_q} T_q^{(m)} \quad (4.50)$$

4.4 Training Bayesian neural networks

The development of a BNN for detecting head movement consists of the following steps:

1. Collecting head-movement data from a number of wheelchair users.

2. Specifying the network architecture, including number of input nodes, number of hidden nodes and numbers of output nodes.
3. Choosing a suitable training algorithm.
4. Training the network with different schemes.
5. Testing the trained network on the new data.

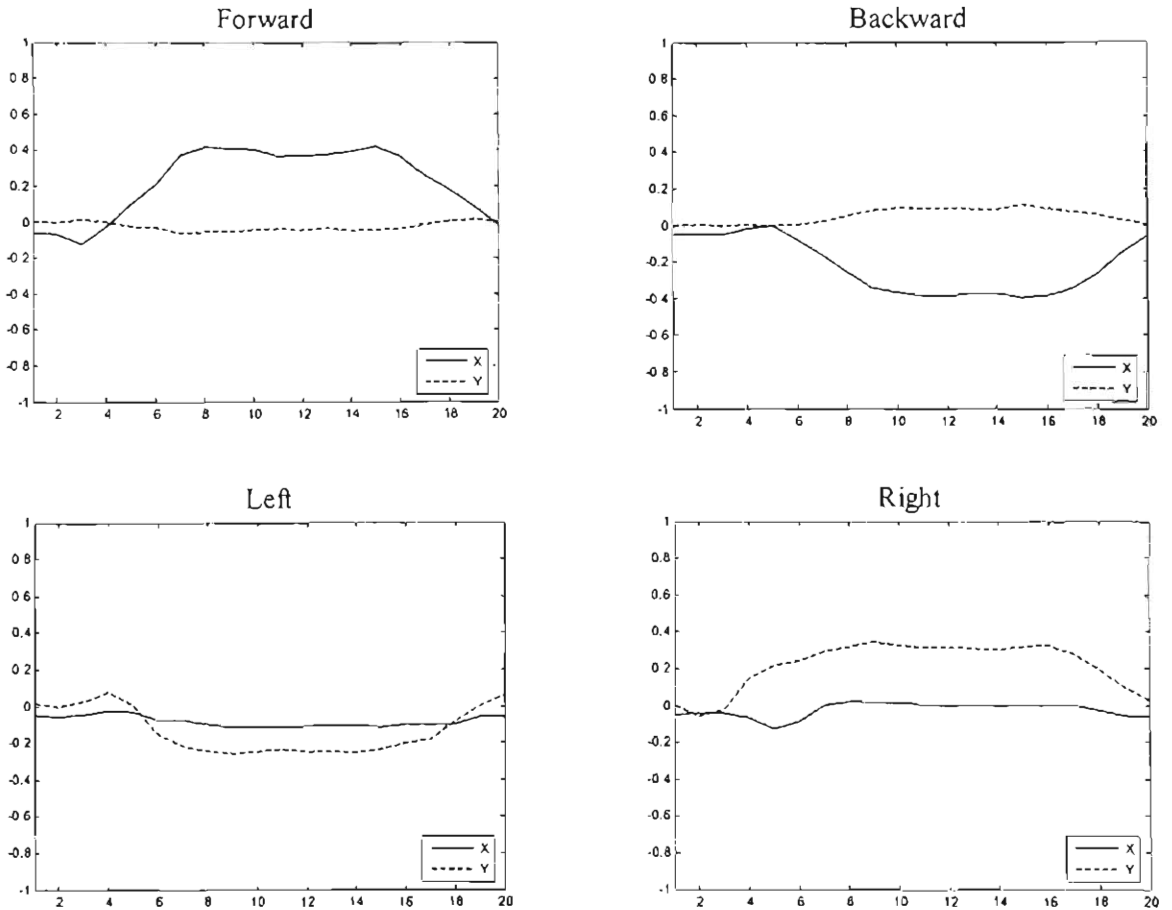
4.4.1 Data acquisition

The head-movement data was collected from eight users, aged between 19 and 56, with approval from the UTS Human Research Ethics Committee. Four had high-level spinal cord injuries (C4 and C5) and were unable to use a standard joystick to control a power wheelchair. The remaining four did not have conditions affecting their head movements. The numbers of extracted movement samples of those users are shown in Table 4.1.

Data for each person were collected in two periods of ten minutes, with the user being prompted to give a specified movement every six seconds. The head movement data were collected with a sampling rate of 100 ms. The start of the head movement for controlling the travel direction of the wheelchair is determined to be the point where the deviation from the neutral position reaches 25% of the maximum value on the relevant axis. Each specified movement was chosen randomly from the following movements: forward, backward, left and right. Each head movement pattern was extracted from a window containing 20 samples from each axis, as shown in Figures 4.7, 4.8 and 4.9.

Table 4.1: Numbers of extracted head-movement samples of the eight wheelchair users.

User	Forward	Backward	Left	Right	Injury level
1	20	20	20	20	-
2	20	20	20	20	-
3	20	20	20	20	-
4	20	20	20	20	-
5	20	20	20	20	C5
6	20	20	20	20	C4
7	20	20	20	20	C4
8	20	20	20	20	C5

**Figure 4.7:** Windowed samples of user 1 (able-bodied person).

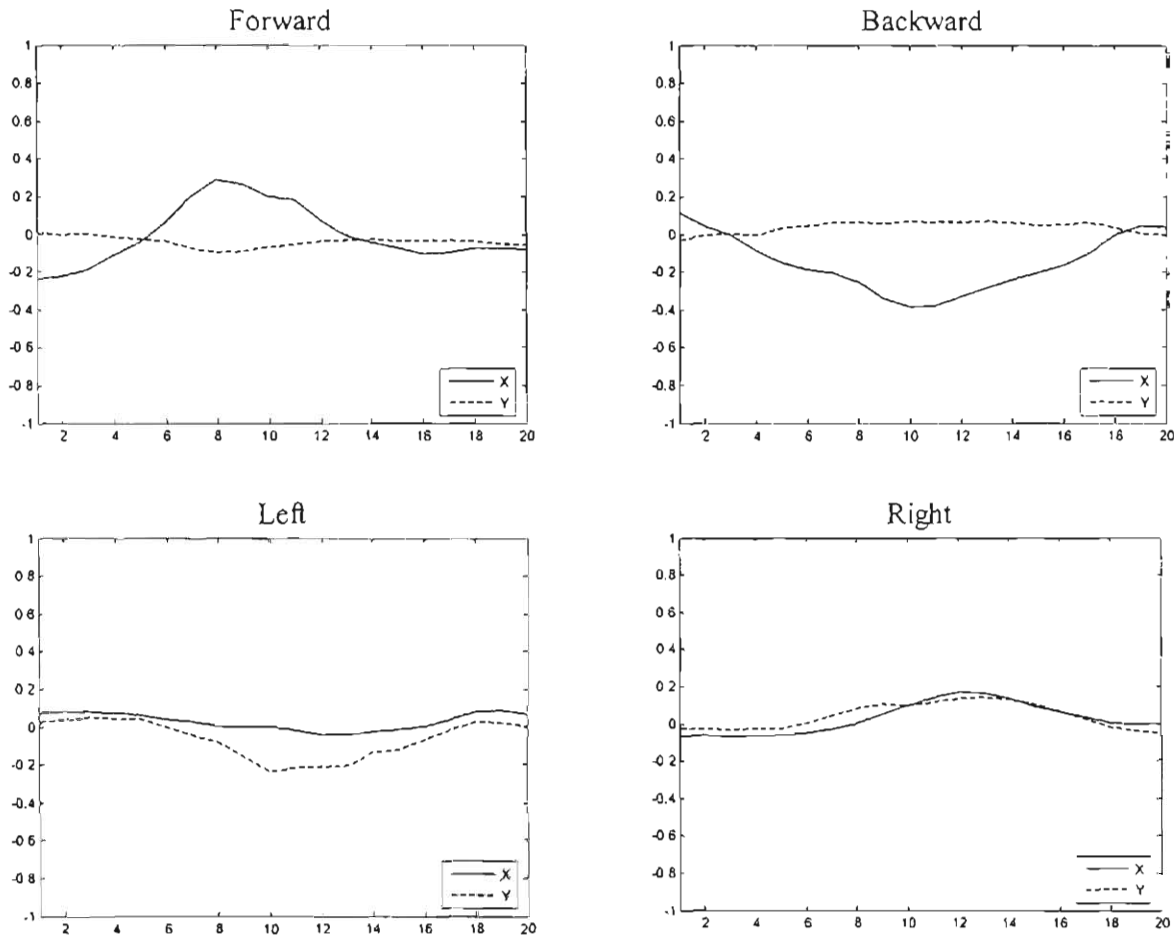


Figure 4.8: Windowed samples of user 5 (C5-injury-level person).

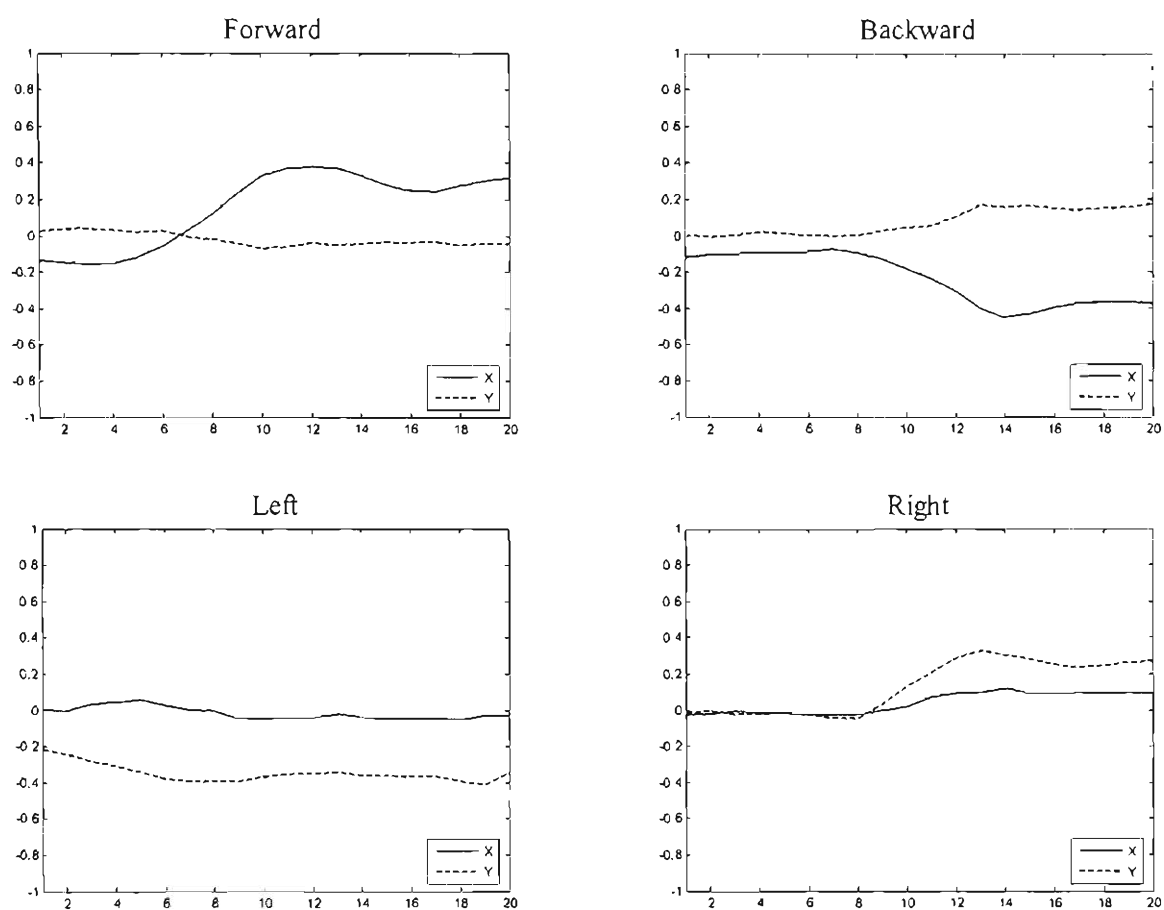


Figure 4.9: Windowed samples of user 6 (C4-injury-level person).

4.4.2 Network specifications

The BNN used to classify head movement has the following specifications:

- 41 inputs, corresponding to 20 samples from x axis, 20 samples from y axis and one augmented input with a constant value of 1;
- three hidden neurons;
- four outputs, each corresponding to one of the classes: forward, backward, left and right movements;
- four hyperparameters ξ_1 , ξ_2 , ξ_3 and ξ_4 for constraining the weights on the connection from the input nodes to the hidden nodes, the bias of the hidden nodes, the weights on the connection from the hidden nodes to the output nodes, and the bias of the output nodes.

4.4.3 Experiment 1

The head-movement data were collected from the eight users, four had high-level spinal cord injuries (C4 and C5) and were unable to use a traditional joystick to control a power wheelchair, with a sampling period of 100 ms. The start of the head movement is the point where the deviation from the neutral position reaches 25% of the maximum value on the relevant axis. All of the samples were randomly divided into two subsets having the same size. The first subset was used for network training and the second subset was used to test the network after training. The training procedure was then implemented as follows:

1. The weights and biases were initialised by random sections from a standard Gaussian distribution and the initial values for the hyperparameters were chosen to be small.

2. The network was trained to minimise the total error function $S(w)$ using the one of the three training algorithms. For each algorithm, ten networks were trained and the averaged training time was then computed. Figure 4.10 shows that the quasi-Newton algorithm has the shortest training time, because it can converge to local minima very quickly every re-estimation period, as shown in Figure 4.11. A relative comparison of the three training algorithms was also conducted, as shown in Table 4.2. Especially, the absolute training time of the most effective quasi-Newton algorithm (less than 20 seconds) is very suitable for the on-line network training.
3. When the network training had reached a local minimum, the values of the hyperparameters were re-estimated as:

$$\gamma_g^{new} = W_g - \xi_g^{old} \text{tr}(A^{-1} I_g)$$

$$\xi_g^{new} = \frac{\gamma_g^{new}}{2E_{W_g}}$$

Table 4.3 shows the change of these parameters according to five re-estimation periods when the network was trained using the quasi-Newton algorithm.

4. Steps 2 and 3 were repeated until convergence was achieved (the total error term was smaller than a pre-determined value and did not change significantly in subsequent iterations).

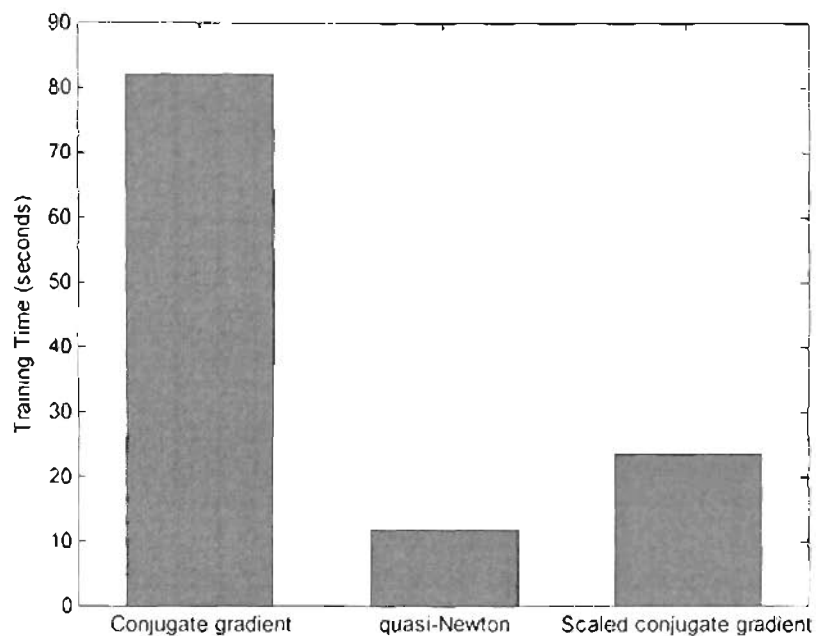


Figure 4.10: Averaged network training time of the three training algorithms.

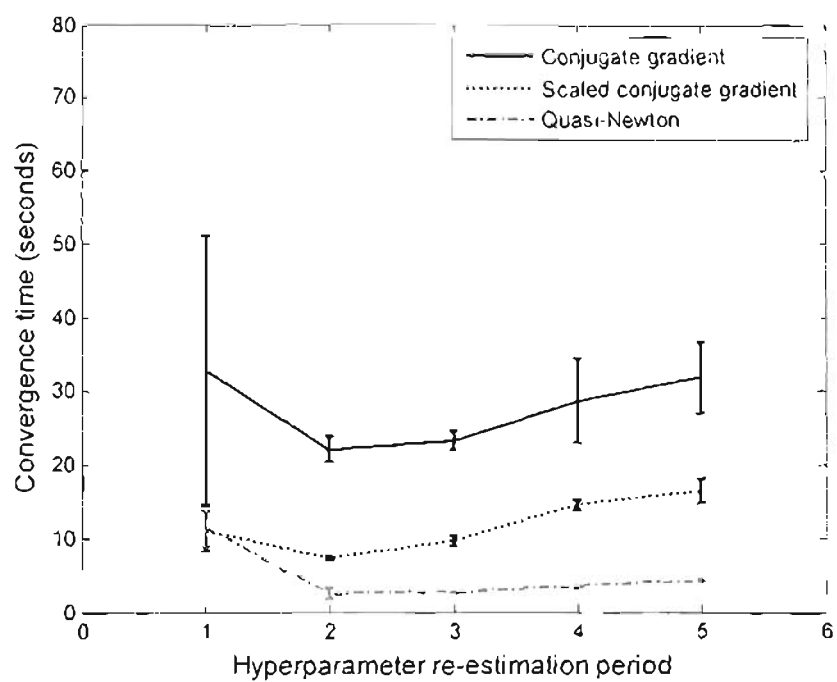
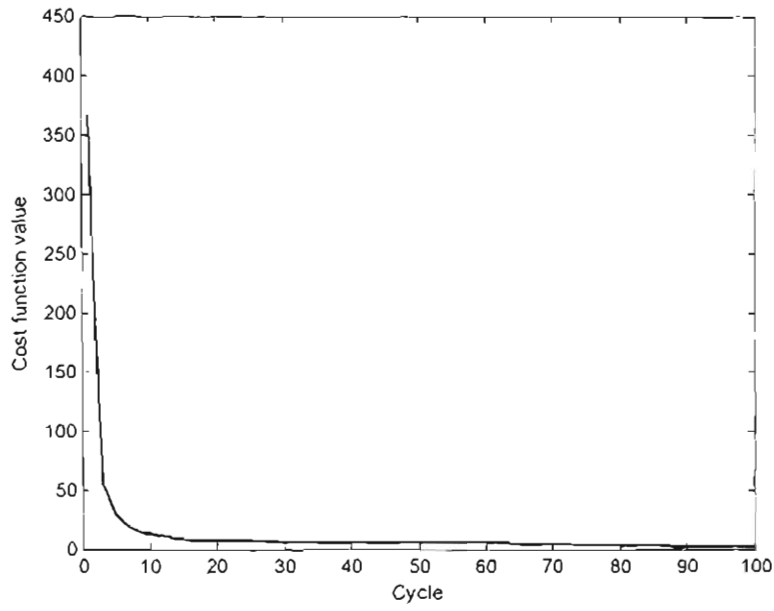


Figure 4.11: Averaged convergence time of the three training algorithms.

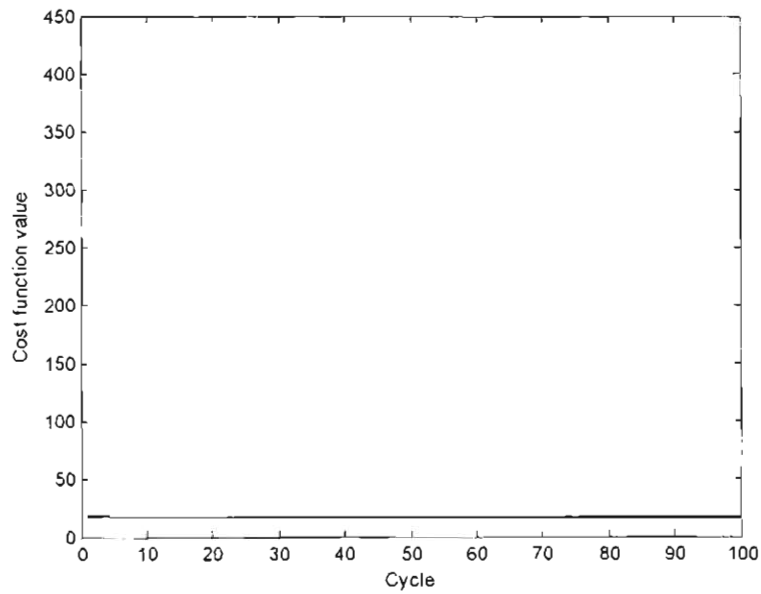
Table 4.2: A relative comparison of the three training algorithms.

Algorithms	T_q	M_q	Properties
Conjugate gradient	Large	Medium	<ul style="list-style-type: none"> - Very accurate line search is required. - Search direction is adequate enough.
Scaled conjugate gradient	Small	Large	<ul style="list-style-type: none"> - No line search is required. - Search direction is not adequate enough.
Quasi-Newton	Medium	Small	<ul style="list-style-type: none"> - Line search is required, but is not very accurate. - Search direction is adequate enough.

As the quasi-Newton training algorithm is the most effective, it was used to train the BNN. The performance of the BNN after training was obtained using the test subset. Figure 4.12 shows cost function value versus training cycle in different hyperparameter re-estimation periods. The change in values of the hyperparameters is shown in Table 4.3. The confusion matrix in Table 4.4 shows that the network can classify the test set with an accuracy of 99.38%.



(a)



(b)

Figure 4.12: Cost-function value versus training cycle in Experiment 1: (a) the first hyperparameter re-estimation period; (b) the fifth hyperparameter re-estimation period.

Table 4.3: The change of the hyperparameters according to five re-estimation periods in Experiment 1.

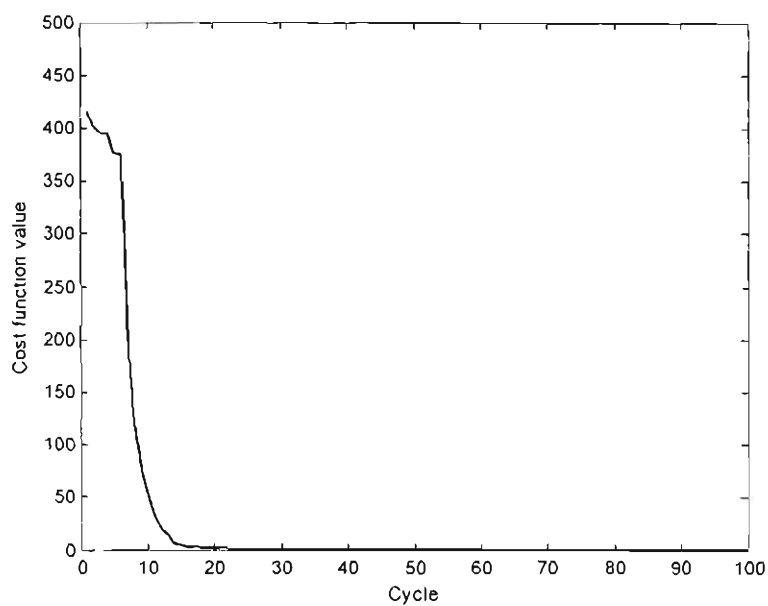
Period	ξ_2	ξ_3	ξ_4	ξ_4
1	0.056929	0.29639	0.026926	0.029799
2	0.20096	2.7378	0.031644	0.081893
3	0.47189	9.9626	0.027007	0.17148
4	0.86856	23.26	0.020751	0.26294
5	1.3856	43.096	0.01589	0.33651

Table 4.4: Confusion matrix of the trained network in Experiment 1.

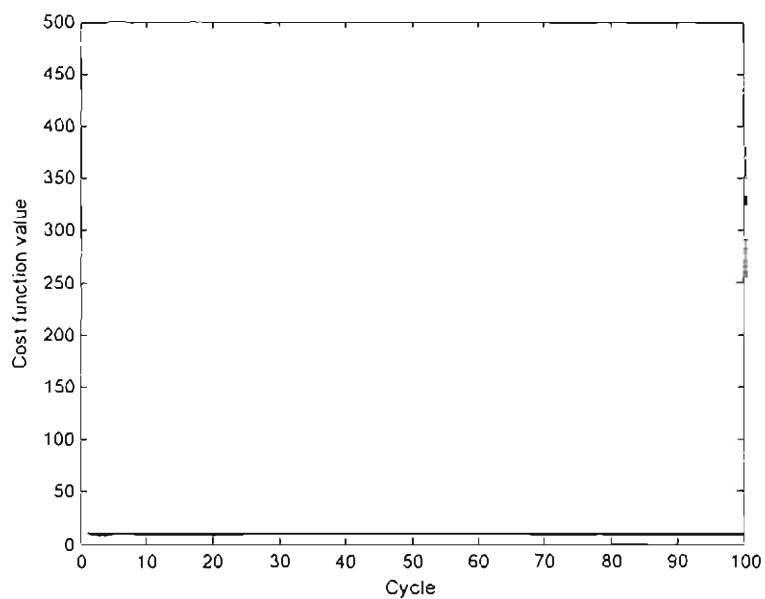
		Predicted Classification				
		Movement	Forward	Backward	Left	Right
Actual Classification	Forward		79	0	0	1
	Backward		0	80	0	0
	Left		0	0	80	0
	Right		0	1	0	79
		Accuracy (%)	99.38			

4.4.4 Experiment 2

In experiment 1, the training set and the test set were taken from all of the eight persons. However, in this experiment, the training data were taken from movement samples of only four users 1, 2, 5 and 6. The similarity of the two experiments is that the size of the test set is the same as the size of the training set. A BNN was obtained using the same procedure as Experiment 1. The quasi-Newton algorithm was also used to train the network. After training, the performance of the network was tested on the data from users 3, 4, 7 and 8. The confusion matrix in Table 4.6 shows that the trained network is able to classify the test data with a success of 95.63%.



(a)



(b)

Figure 4.13: Cost-function value versus training cycle in Experiment 2: (a) the first hyperparameter re-estimation period; (b) the fifth hyperparameter re-estimation period.

Table 4.5: The change of the hyperparameters according to five re-estimation periods in Experiment 2.

Period	ξ_2	ξ_3	ξ_4	ξ_4
1	0.054688	0.96608	0.033244	0.073871
2	0.17272	5.5413	0.049354	0.2645
3	0.378381	18.0739	0.0507679	0.639195
4	0.672648	42.8629	0.0443216	1.21639
5	1.06715	83.5495	0.0358795	1.99973

Table 4.6: Confusion matrix of the trained network in Experiment 2.

		Predicted Classification			
		Forward	Backward	Left	Right
Actual Classification	Movement				
	Forward	75	0	0	5
	Backward	0	76	4	0
	Left	2	0	77	1
Right	0	2	0	78	
Accuracy (%)		95.63			

The classification results of the two experiments are summarised in Table 4.7. It can be seen that very high sensitivity (true positive) and specificity (true negative) have been obtained for both experiments. In particular, the quasi-Newton training algorithm can allow us to develop on-line BNN head movement classifier training.

Table 4.7: Sensitivity and specificity of the trained network in the two experiments.

Experiment	Sensitivity	Specificity
1	0.99	0.99
2	0.96	0.98

4.5 Discussion

Although the sample size is rather limited, the experimental results obtained in this chapter show that the BNN can classify head movement with a very high accuracy as the network training does requires the use of a validation subset separated from the training set. If all eight users (four abled and four disabled persons) were trained, the BNN can classify the test set very accurately (99.38% accuracy). If the training data were taken from only four users (two abled and two disabled persons), the obtained BNN still classifies the test set from the other four users with an excellent accuracy (95.63% accuracy). In other words, experiment 2 shows that the trained network is able to accurately classify head movement of new persons.

Compared to the conjugate and scaled conjugate algorithms, the quasi-Newton algorithm is more effective. However, a potential disadvantage of the quasi-Newton algorithm is that it requires the storage and update of a matrix F of size $W \times W$, where W is the number of weights. If the number of weights is a few thousand, this could lead to the requirement of very significant memory. This problem motivates the development of a new class of the quasi-Newton method that requires less memory (Bortoletti, Di Fiore et al. 2003). However, the size of the BNN in this research is not large, so this problem is insignificant.

Chapter 5

Optimal Adaptive Bayesian Neural Network Classifiers

5.1 Introduction

In supervised learning, the network is trained on a set of input-target pairs $\{x^{(n)}, t^{(n)}\}$, $n=1, \dots, N$. We also assume that there is an underlying (possibly noisy) functional relationship between the outputs and inputs of the network:

$$z = f(x, \varepsilon) \quad (5.1)$$

where ε denotes the noise of network inputs. The aim of the learning process is to approximate this relationship based on the training set. The success of the learned approximation is judged by the ability of the network to approximate the outputs corresponding to inputs on which it was not trained.

The issue in selecting the right network topology is one of the most debated themes in two-layer perceptron modelling. Large networks have more functional flexibility than small networks, so are better able to fit the training data. However, large networks can have higher parameter variance than small networks, resulting in poor generalisation. The network topology is dominated by the number of neurons in the hidden layer (hidden nodes). Therefore, the number of hidden nodes is a crucial factor in the network's ability to generalise.

Recently, several methods for determining the proper number of hidden nodes have been proposed (Takechi and Murakami 1995; Fletcher, Katkovnik et al. 1998;

Ponnappalli, Ho et al. 1999; Arifovic and Gençay 2001; Shahjahan, Akhand et al. 2003; Guimaraes and Ramirez 2004; Son, Lee et al. 2004). However, these techniques can be very expensive in terms of computation time and/or training data.

A well-known method for determining the proper network size and connectivity is the pruning technique. The pruning approach consists of starting with an oversized network model, and then eliminating connections with low magnitude or those which lead to low increases in the training error function (Levin, Leen et al. 1994; Ponnappalli, Ho et al. 1999; Guimaraes and Ramirez 2004).

Nevertheless, the elimination of all weights with low magnitude is not an efficient strategy because we need to consider a sensitivity analysis of the weight removal. The sensitivity analysis may be performed by the Taylor series expansion of $E_D(w)$:

$$\Delta E_D = g^T \Delta w + \frac{1}{2} \Delta w^T H(w) \Delta w \quad (5.2)$$

After training, the weight vector corresponds to a local minimum of the error function. Hence, by neglecting the gradient vector g , we can write ΔE_D in the following form:

$$\Delta E_D(w) \approx \frac{1}{2} \Delta w^T H(w) \Delta w \quad (5.3)$$

According to equation (5.3), we can estimate the variation in the error function according to the disturbance Δw in the weight vector. This disturbance is selected in such a way as to eliminate a set of small weights:

$$\Delta w = - \sum_{i \in I} e_i w_i \quad (5.4)$$

where e_i is the i th column of the $W \times W$ identity matrix and w_i is a small weight from the set I of weights that we may eliminate. On the other hand, the sensitivity of the error function according to the removal of the weight w_i is defined as:

$$\kappa_i = \frac{w_i^2}{2} [H(w)]_{ii} \quad (5.5)$$

where $[H(w)]_{ii}$ is the i th element of the diagonal of $H(w)$. Finally, the pruning algorithm can be summarised as follows:

1. Train the neural network.
2. Compute the Hessian matrix.
3. Find the weight in the set R with the smallest sensitivity, given by (5.5).
4. If $\kappa_i < \zeta$, remove this weight from R , put it in I and go to step 3. Otherwise go to step 5.
5. Assemble the disturbance Δw as in (5.4).
6. Evaluate the error deviation using (5.3).
7. If $\Delta E_D < \zeta$, remove from the neural model the weights in the set I and go to Step 8. Otherwise remove from set I the weight with the biggest sensitivity and go to Step 8.
8. Finally, retrain the pruned neural model.

The value ζ is specified by the user and indicates the acceptable increase in error according to the reduction in the model's complexity.

Despite the fact that the pruning approach is used to remove the neurons having the least effect on the output error, this technique does not take the correlation between the neuron activities into account. In practice, eliminating small weights does not properly account for a weight's effect on the output error. The method also requires re-training the pruned network, which is computationally expensive.

In the Bayesian evidence framework, the principle of *Bayesian model comparison* can be utilised to select the optimal number of hidden nodes in the neural network, given the training data. Suppose that there are a set of neural networks X_i with different numbers of hidden nodes. According to Bayes' theorem, we can write down the posterior probabilities of the network X_i , once the training data set D has been observed, in the form:

$$P(X_i | D) = \frac{p(D | X_i)P(X_i)}{p(D)} \quad (5.6)$$

where $P(X_i)$ is the prior probability of the network X_i and the quantity $p(D | X_i)$ is referred to as the *evidence* for X_i (MacKay 1992a). If we have no reason to assign different priors to the networks, then we can compare the relative probabilities of the networks on the basis of their evidence.

Also, the evidence of the network X_i can be precisely computed as:

$$p(D | X_i) = \int p(D | w, X_i)p(w | X_i)dw \quad (5.7)$$

If the posterior distribution is sharply peaked in the weight space around the most probable weight vector w_{MP} , then the integral (5.7) can be approximated as:

$$p(D | X_i) \approx p(D | w_{MP}, X_i) \left(\frac{\Delta w_{posterior}}{\Delta w_{prior}} \right) \quad (5.8)$$

where Δw_{prior} and $\Delta w_{posterior}$ are the prior and posterior uncertainties in the weights. The ratio $\Delta w_{posterior} / \Delta w_{prior}$ is called the Occam factor (MacKay 1992a).

Equation (5.8) can be expressed in words as follows:

$$Evidence = Best\ fit\ likelihood \times Occam\ factor \quad (5.9)$$

The best-fit likelihood measures how well the network fits the data, while the Occam factor (< 1) penalises the network for having the weight vector w . A network that has a large best-fit likelihood will receive a large contribution to the evidence. However, if the network has many hidden nodes, then the Occam factor will be very small. Therefore, the network with the largest evidence will make the balance between needing a large likelihood to fit the data well and a relatively large Occam factor, so that the model is not too complex.

5.2 Evaluating the evidence

According to Thodberg (Thodberg 1996), the logarithm of evidence for the network X_i , which contains the weight vector w and G hyperparameters ξ_g , at the most probable weight vector w_{MP} , is given by

$$\ln Ev(X_i) = -E_D(w_{MP}) + \ln Occ(w_{MP}) + \sum_{g=1}^G \ln Occ(\xi_g^{MP}) \quad (5.10)$$

where $E_D(w_{MP})$ is the data error evaluated at w_{MP} . $Occ(w_{MP})$ and $Occ(\xi_g^{MP})$ are the Occam factors for the weight vector and the hyperparameters at w_{MP} .

5.2.1 The Occam factor for the weights and biases

The Occam factor for the weights and biases with the Gaussian distributions is given by (MacKay 1992a):

$$Occ(w) = \left[\exp \left(- \sum_{g=1}^G \xi_g^{MP} E_{w_g}^{MP} \right) \right] \left(\prod_{g=1}^G (\xi_g^{MP})^{w_g/2} \right) (\det A)^{-1/2} \quad (5.11)$$

where ξ_g^{MP} is the most probable value of the hyperparameter for the weights and biases in group g evaluated at w_{MP} and $E_{w_g}^{MP}$ is the weight error in group g evaluated at w_{MP} .

We now consider a two-layer perceptron network with “tanh” activation functions in the hidden layer. There are two kinds of symmetries in the network:

- Firstly, by changing the signs of all incoming and outgoing weights of a hidden node, we can obtain an identical mapping from the input nodes to the output nodes. For a network with M hidden nodes, there are 2^M equivalent weight vectors that result in the same mapping from the inputs to the outputs of the network.
- Secondly, interchanging the values of all incoming and outgoing weights of a hidden node with the corresponding values of the weights associated with another hidden node also results in an identical mapping. For a network with M hidden nodes, there are $M!$ of those permutations.

Therefore, a minimum value of the cost function $S(w)$ can be obtained with $2^M M!$ equivalent weight vectors. Hence, $Occ(w)$ in equation (5.11) should be multiplied by $2^M M!$ for a fully connected network:

$$Occ(w) = \left[\exp \left(- \sum_{g=1}^G \xi_g^{MP} E_{w_g}^{MP} \right) \right] \left(\prod_{g=1}^G (\xi_g^{MP})^{w_g/2} \right) (\det A)^{-1/2} (2^M M!) \quad (5.12)$$

5.2.2 The Occam factor for the hyperparameters

For the hyperparameter ξ_g , the logarithm of the evidence for (ξ_g, X_i) is assumed to be a quadratic form in $\ln \xi_g$, which is a natural scale variable (Thodberg 1996).

$$\ln P(D | \xi_g, X_i) = \ln P(D | \xi_g^{MP}, X_i) - \frac{1}{2} \frac{(\ln \xi_g - \ln \xi_g^{MP})^2}{\sigma_{\ln \xi_g}^2} \quad (5.13)$$

where $\sigma_{\ln \xi_g}$ is the variance of the distribution for $\ln \xi_g$, and at $\xi_g = \xi_g^{MP}$ it has the form:

$$\left(\sigma_{\ln \xi_g}\right)^2 = -\xi_g^2 \frac{\partial^2}{\partial^2 \xi_g} \ln p(D | \xi_g, X_i) \quad (5.14)$$

So

$$\begin{aligned} P(D | X_i) &= \int P(D | \xi_g, X_i) P(\ln \xi_g | X_i) d \ln \xi_g \\ &= P(D | \xi_g^{MP}, X_i) \int \exp\left[-\frac{1}{2} \frac{(\ln \xi_g - \ln \xi_g^{MP})^2}{\sigma_{\ln \xi_g}^2}\right] \frac{1}{\ln \Omega} d \ln \xi_g \\ &= P(D | \xi_g^{MP}, X_i) Occ(\xi_g) \end{aligned} \quad (5.15)$$

where

$$Occ(\xi_g) = \frac{\sqrt{2\pi} \sigma_{\ln \xi_g}}{\ln \Omega} \quad (5.16)$$

The parameter Ω can be set to a specific value, for example 10^3 , which indicates a subjective estimate of the hyperparameter. However, in practice, Ω is only a minor factor, as it is the same for every network. Finally, $Occ(\xi_g)$ denotes the Occam factor for the hyperparameter ξ_g .

In equation (5.16), $\sigma_{\ln \xi_g}$ can be approximated as (MacKay 1992b):

$$\sigma_{\ln \xi_g} \approx \sqrt{\frac{2}{\gamma_g}} \quad (5.17)$$

Substituting (5.17) into (5.16) gives:

$$Occ(\xi_g) = \frac{\sqrt{4\pi/\gamma_g}}{\ln \Omega} \quad (5.18)$$

5.2.3 Combining the terms of evidence

We are now ready to combine the terms of evidence. Firstly, by taking the logarithm of (5.12), we have:

$$\ln Occ(w) = -\sum_{g=1}^G \xi_g^{MP} E_{W_g}^{MP} + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g - \frac{1}{2} \ln(\det A) + \ln M! + M \ln 2 \quad (5.19)$$

Similarly, taking the logarithm of (5.18) gives:

$$\ln Occ(\xi_g^{MP}) = \frac{1}{2} \ln \left(\frac{4\pi}{\gamma_g^{MP}} \right) - \ln(\ln \Omega) \quad (5.20)$$

Substituting (5.19) and (5.20) in (5.10) gives:

$$\begin{aligned} \ln Ev(X_i) = & -S(w_{MP}) + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g - \frac{1}{2} \ln(\det A) + \ln M! + M \ln 2 \\ & + \sum_{g=1}^G \left[\frac{1}{2} \ln \left(\frac{4\pi}{\gamma_g^{MP}} \right) - G \ln(\ln \Omega) \right] \end{aligned} \quad (5.21)$$

Equation (5.21) is used to select between different networks. As the Bayesian approach to the model comparison is incorporated with a mechanism for penalising over-complex networks, we might expect that the network with the largest evidence will provide the best results on unseen data.

5.3 Adaptive Bayesian neural networks

5.3.1 Head movement data

The head-movement data previously collected from the eight users were used to train and test the BNNs for detecting the head-movement commands. Four users had high-level SCIs (C4 and C5) and were unable to use standard joysticks to control power wheelchairs. The remaining four users were able-bodied. The extracted movement samples of the eight users are shown in Table 5.1.

Table 5.1: Numbers of extracted head-movement samples of the eight users.

User	Forward	Backward	Left	Right	Injury level
1	20	20	20	20	-
2	20	20	20	20	-
3	20	20	20	20	-
4	20	20	20	20	-
5	20	20	20	20	C5
6	20	20	20	20	C4
7	20	20	20	20	C4
8	20	20	20	20	C5

5.3.2 Training networks on the high-performance computational clusters

As the solution of Bayesian neural network training is sensitive to the initial conditions of weights and biases, it is necessary to perform multiple training runs and then the final solution is selected from the results of different training runs. However, the multiple training runs on the personal computer (PC) usually lead to an excessively large amount of training time as each network training run requires evaluating the Hessian matrix several times. As a result, the high-performance computation is needed in order to speed up the multiple network training runs.

Recently, high-performance (HPC) clusters have been implemented primarily to provide increased computing performance by splitting a computational task across many different nodes in the cluster. Also, the computational power of HPC clusters has been reported in several researches (Lu, Yang et al. 2004; Jiménez-Hornero, Jiménez-Hornero et al. 2006). In addition, a HPC cluster can be built up from cheap PCs, as shown in Figure 5.1.

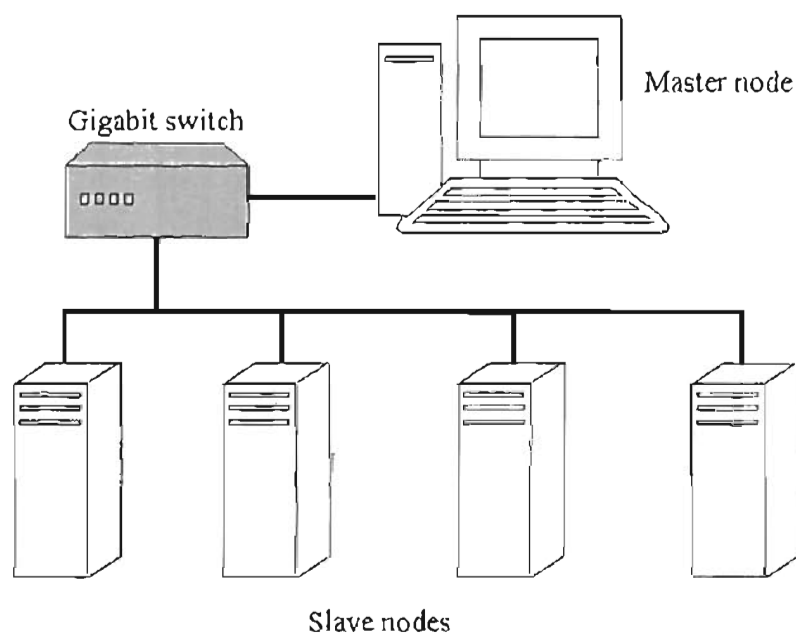


Figure 5.1: A common topology of PC clusters.

Clusters with nodes running Linux as the operating system and free software to perform the parallelism represent one of the most popular HPC implementations. Such clusters run custom programs that have been designed to exploit the parallelism available on the HPC clusters. Message Passing Interface (Vehdari and Lampinen) is a *de facto* standard for communication among the nodes running parallel programs on the clusters (Kepner 2001).

The high-performance computing resource of the Faculty of Engineering (Androutsopoulos, Koutsias et al.) consists of three clusters: 1) the Orion cluster, 2) the Hydrus cluster, and 3) the Titan cluster. All the clusters are connected to a server with 2.8 TB of disk storage via a gigabit network, as shown in Figure 5.2. One person is

allowed to use ten cluster nodes at a time. The Hydrus cluster is the most effective one, as each of its nodes can be up to 70% faster than those in other clusters.

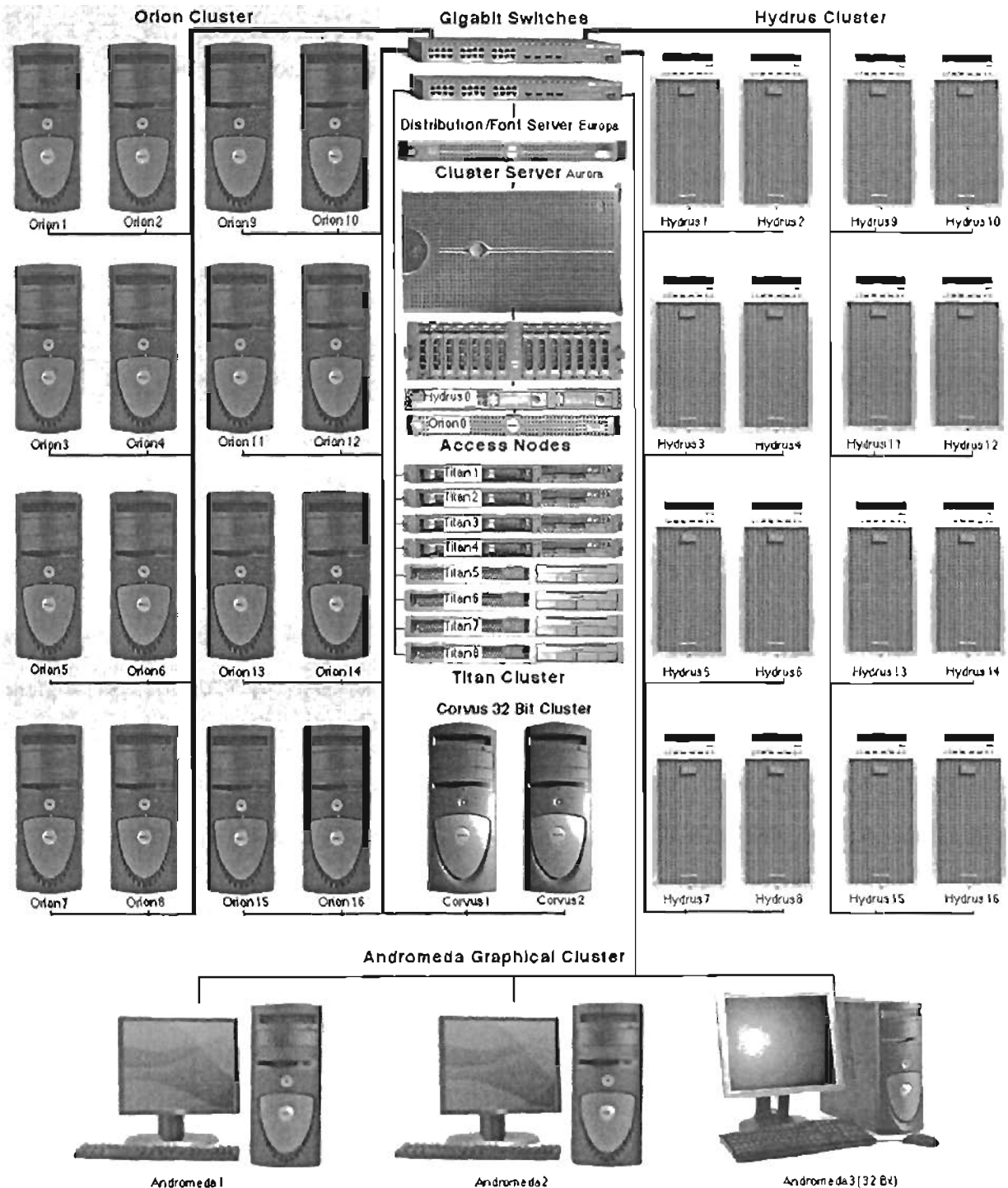


Figure 5.2: Schematic of the clusters in the Faculty of Engineering (UTS).

5.3.2 The optimal network architecture

Extracted samples of user 7 (C4-injury-level person) and user 8 (C5-injury-level person) were randomly divided into two size-equal subsets. The training set was taken from the

samples of users 1, 2, 3, 4, 5 and 6 (480 samples). Different BNNs with varying numbers of hidden nodes were trained in order to select the optimal architecture. These networks have the following specifications:

- 41 inputs, corresponding to 20 samples from the x axis, 20 samples from the y axis and one augmented input with a constant value of 1;
- four outputs, each corresponding to one of the classes: forward, backward, left and right movements;
- four hyperparameters, ξ_1 , ξ_2 , ξ_3 and ξ_4 , to constrain the magnitudes of the weights on the connection from input nodes to hidden nodes, biases of the hidden nodes, weights on the connection from hidden nodes to output nodes, and biases of output nodes.

For a given number of hidden nodes, ten networks with different initial conditions of weights and biases were trained. The training procedure was implemented as follows:

1. The weights and biases in four different groups were initialised by random selections from the standard Gaussian distribution and the initial hyperparameters were chosen to be small values, for example 10^{-4} .
2. The network was trained by minimising the cost function $S(w)$ with the quasi-Newton algorithm.
3. When the network training had reached a local minimum, the values of the hyperparameters were re-estimated as:

$$\gamma_g^{new} = W_g - \xi_g^{old} \text{tr}(A^{-1} I_g)$$

$$\xi_g^{new} = \frac{\gamma_g^{new}}{2E_{W_g}}$$

4. Steps 2 and 3 were repeated until the cost function value was smaller than a pre-determined value and did not change significantly in subsequent re-estimation periods.
5. Finally, the log evidence for the network was evaluated as:

$$\begin{aligned} \ln Ev = & -S(w^{new}) + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g^{new} - \frac{1}{2} \ln(\det A) + \ln M! + M \ln 2 \\ & + \sum_{g=1}^G \left[\frac{1}{2} \ln \left(\frac{4\pi}{\gamma_g^{new}} \right) - G \ln(\ln \Omega) \right] \end{aligned}$$

Once the network training was completed, the performance of the trained network was tested on the test set taken from the first subset of samples from users 7 and 8. As shown in Figure 5.3, the networks with “three hidden nodes” have the highest evidence. These networks also have low test errors (misclassification percentages), as shown in Figure 5.4. This means that for this application, the optimal number of hidden nodes in the network is three. The optimisation of network architecture is very useful for on-line training tasks because incorporating an effective training algorithm such as quasi-Newton or scaled conjugate gradient, the optimal network size can contribute to the least network training time while still maintaining the best generalisation for the trained network.

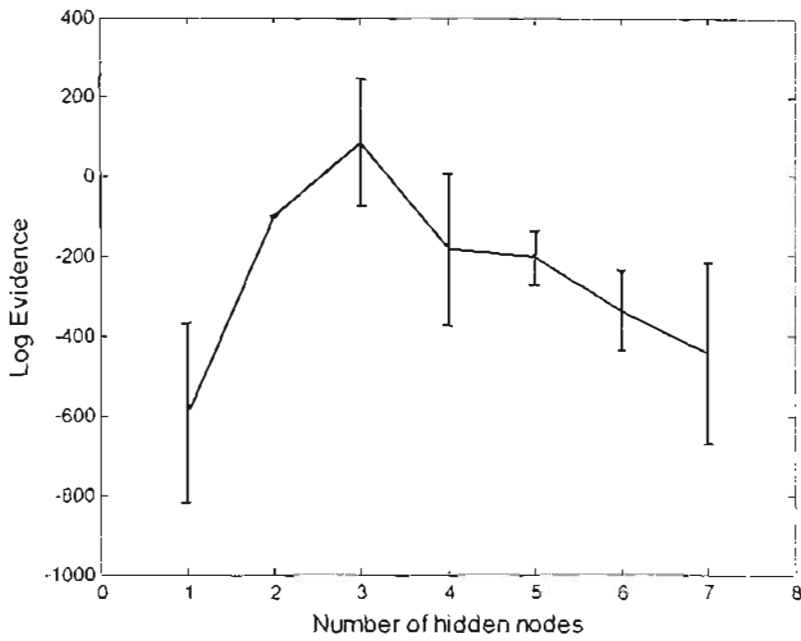


Figure 5.3: Log evidence versus number of hidden nodes: The solid curve shows the evidence averaged over the ten networks. The error bars are at plus and minus standard deviations.

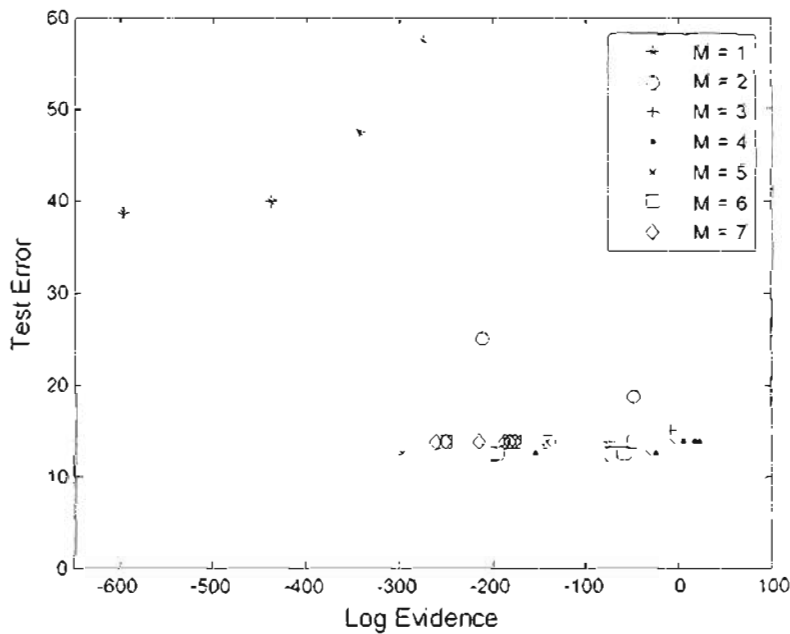
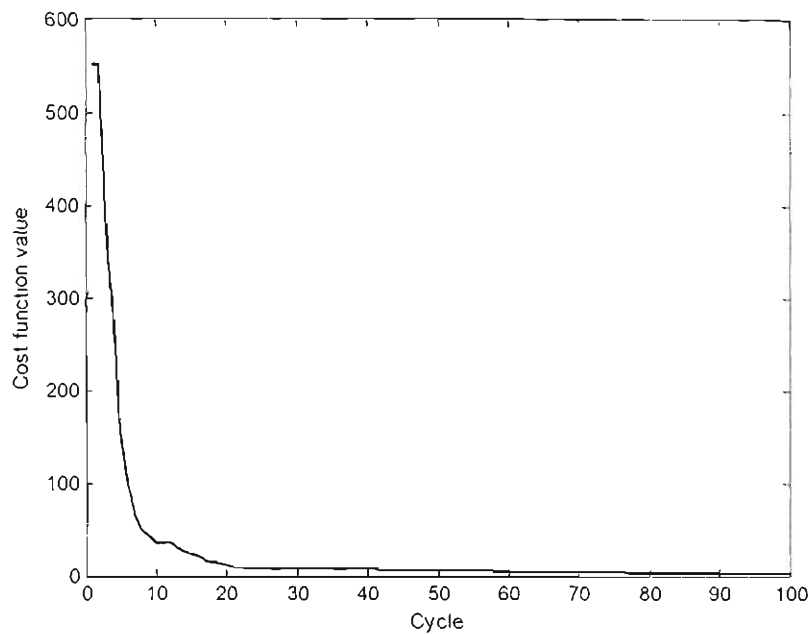


Figure 5.4: Test error versus log evidence. Every symbol appears ten times, once for each of the ten trained networks.

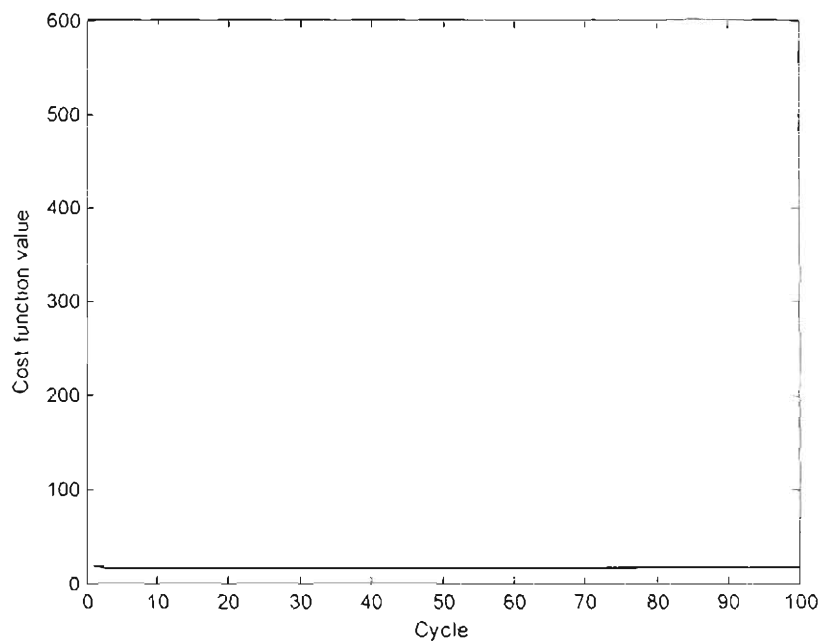
5.3.3 Head movement classification

5.3.3.1 Experiment 1

A BNN classifier with three hidden nodes was trained on the training data taken from users 1, 2, 3, 4, 5 and 6. The network was trained with five periods of hyperparameter re-estimation. As shown in Figure 5.5, the cost function value did not change significantly in the fifth period of hyperparameter re-estimation. Table 5.2 shows the changes in the hyperparameters, according to five re-estimation periods.



(a)



(b)

Figure 5.5: Cost-function value versus training cycle in Experiment 1: (a) the first hyperparameter re-estimation period; (b) the fifth hyperparameter re-estimation period.

Table 5.2: The change of the hyperparameters according to five re-estimation periods in Experiment 1.

Period	ξ_2	ξ_3	ξ_4	ξ_4
1	0.065453	1.0562	0.023448	0.060012
2	0.23603	6.0533	0.027157	0.15485
3	0.540483	16.3908	0.0236425	0.248887
4	0.980702	33.0305	0.0183345	0.324081
5	1.54561	57.5469	0.0138955	0.393726

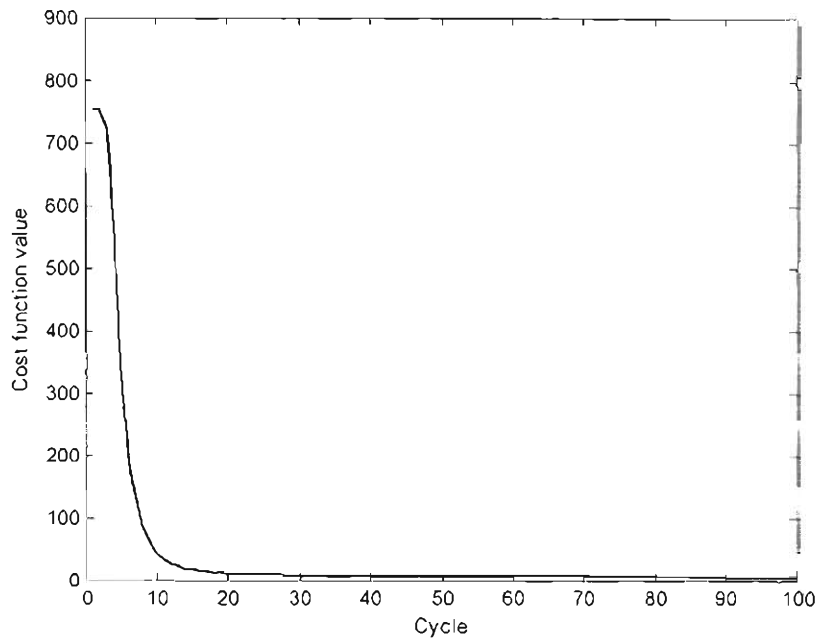
The performance of the trained network was tested on the first subset of samples from users 7 and 8. The confusion matrix in Table 5.3 shows how accurately the trained network can classify all of the movements in the test set. Overall, the trained network can classify the test set with an accuracy of 85%.

Table 5.3: Confusion matrix of the trained network in Experiment 1.

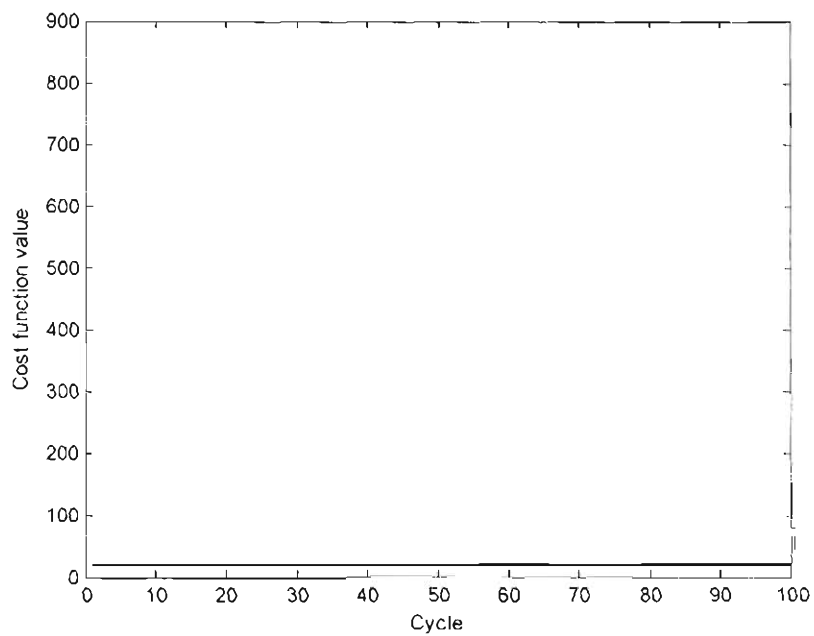
		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	15	0	1	4	
	Backward	0	17	3	0	
	Left	4	0	16	0	
	Right	0	0	0	20	
Accuracy (%)		85				

5.3.3.2 Experiment 2 (Adaptive network training)

More training samples were taken from the second subset of samples from users 7 and 8. The BNN classifier was trained using the same procedure in Experiment 1. The performance of the trained network was tested on the first subset of samples from users 7 and 8. Similarly, a confusion matrix, shown in Table 5.5, was used to evaluate the performance of the trained network. Overall, the network can classify the test set with a success rate of 93.75%.



(a)



(b)

Figure 5.6: Cost-function value versus training cycle in Experiment 2: (a) in the first hyperparameter re-estimation period; (b) in the fifth hyperparameter re-estimation period.

Table 5.4: The change of the hyperparameters according to five re-estimation periods in Experiment 2.

Period	ξ_2	ξ_3	ξ_4	ξ_4
1	0.074005	1.1861	0.027694	00.11958
2	0.27065	7.1365	0.02729	0.21283
3	0.631032	19.0378	0.0223414	0.27603
4	1.15511	37.4862	0.0168927	0.311165
5	1.80667	62.8208	0.0127276	0.331794

Table 5.5: Confusion matrix of the trained network in Experiment 2.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	18	0	1	1	
	Backward	0	19	1	0	
	Left	2	0	18	0	
	Right	0	0	0	20	
Accuracy (%)		93.75				

The classification results of the two experiments are summarised in Table 5.6. It can be seen that very high sensitivity (true positive) and specificity (true negative) have been achieved for Experiment 2. This means that the performance of the trained network has been significantly improved as more samples were included to train the network.

Table 5.6: Sensitivity and specificity of the trained network in the two experiments.

Experiment	Sensitivity	Specificity
1	0.85	0.95
2	0.94	0.98

5.4 Discussion

Although the number of trials is small, the results obtained show that BNNs can be used to classify head movement accurately. For this application, the optimal number of hidden nodes is three. This number of hidden nodes can guarantee the best

generalisation of the trained network. When the BNN was trained on movement samples from the six users, it could classify head movements of two new disabled users with 85% accuracy. However, if the network was trained further with additional movement samples of those two disabled users, it could classify their head movement with a high accuracy of 93.75%. In other words, the network is able to provide an on-line adaptation to head movements of new disabled users.

So far we have expected that the networks with higher evidence will obtain lower generalisation errors. However, this phenomenon can only be observed if the networks are trained on “relatively sufficient” training data. It is therefore recommended to check the test error versus the evidence when applying the evidence framework to determining the optimal number of hidden nodes. A small test set may provide an uncertain estimate of the test error. This problem can be overcome by increasing the size of the test set. Finally, we select the network architecture with the highest evidence. If many network architectures have evidence near or at the maximum, the simplest architecture will be selected.

Chapter 6

Alternative Frameworks for Improving the Accuracy of Detecting Head-movement Commands

6.1 Introduction

The main operation in the Bayesian inference for neural networks is integration over the weight space. In the earlier chapters, this task was performed by using the Laplace approximation, in which a Gaussian distribution was assumed for the posterior weights at the local minima. This approach is often coupled with the use of the evidence procedure to estimate the optimal values of the hyperparameters.

Early stopping is one of the techniques to avoid overfitting. As mentioned in Chapter 5, the maximum evidence for the neural network indicates a trade-off between the ability to fit the training data well and the complexity of the network. The maximum evidence can also be utilised to prevent the network from overfitting. In particular, the stopping point is the one that corresponds to the maximum evidence. Moreover, this approach does not require using an independent validation set, like the traditional method, for monitoring the generalisation error.

An alternative implementation of Bayesian learning of neural networks is to use the Monte Carlo methods (Neal 1996). Especially, this technique can avoid the Gaussian approximation to the posterior distribution of the weights and biases (Rasmussen 1995; Williams 1998; Wright 1999).

Although one of the independent classifiers could yield the best performance, the patterns misclassified by the classifiers would not necessarily overlap. This suggested that different classifier designs potentially offered complementary information about the

pattern to be classified. In order to enhance the accuracy and the reliability of classification systems, two methods for combining the performances of independent BNN classifiers have been proposed: 1) Bayes' rule-based method and the Dempster-Shafer theory-based method (Kittler, Hatef et al. 1998; Denoeux 2000; Al-Ani and Deriche 2002).

Therefore, this chapter describes some alternative frameworks for improving the performances of BNN classifiers. Firstly, a new version of early stopping is proposed and is known as "the maximum evidence-early stopping". Secondly, the chapter describes the Monte Carlo methods for BNN training. In order to enhance the reliability of classification systems, the chapter describes how to combine the solutions of the independent BNN classifiers. Some preliminary results of these frameworks are also presented.

6.2 Maximum evidence for early stopping of neural networks

In the traditional early stopping for neural networks, all the available data are divided into training, validation and testing subsets. The training subset is used to train the network. The error on the validation subset is monitored during the training process. As shown in Figure 6.1, the validation error will normally decrease during the initial phase of network training. However, as the network begins to overfit the data, the error on the validation subset typically starts to rise. After the validation error has increased for a specified number of iterations, the training is stopped, and the weights at the minimum validation error are selected to be the final weights and biases for the entire network design process.

However, the performance of the trained network depends significantly on the way of partitioning the available data into subsets. If the available data are very limited and/or noisy, this method of network training may be ineffective, as the training set with one part left out for making the validation subset may be not sufficient enough to obtain a correct solution.

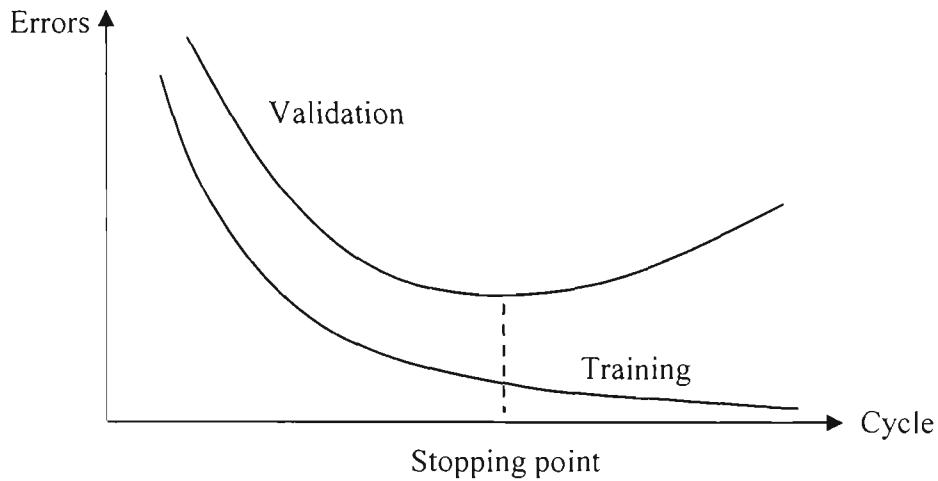


Figure 6.1: An illustration of the traditional early stopping in neural networks.

In statistical inference, the collected data are usually used to select the most probable model from different candidature models. For this purpose, frequentist statistics uses *hypothesis tests*, which are algorithms to state the alternative for (or against) the hypotheses in order to minimise certain risks.

In the Bayesian model comparison, the rules of probability theory are used to select among different hypotheses. These rules automatically encode a preference for simpler and more constrained models. The probability of the data D , given the model X , can be computed by integrating over the unknown parameter value θ in the model:

$$p(D|X) = \int p(D|\theta)p(\theta|X)d\theta \tag{6.1}$$

where $p(D|X)$ is called the evidence for the model X , $p(D|\theta)$ is the dataset likelihood, and $p(\theta|X)$ is the prior distribution of θ .

6.2.1 The evidence for neural networks

In neural networks, hypotheses are different structures and weight vectors. Given a network structure X , the evidence for this network is precisely evaluated as:

$$p(D|X) = \int p(D|w)p(w|X)dw \tag{6.2}$$

where $p(w|X)$ is the prior distribution of weights and biases, and $p(D|w)$ is the dataset likelihood. In the Bayesian evidence framework, the most probable weight vector w_{MP} is chosen so as to maximise the evidence $p(D|X)$. In the early stopping technique, the evidence needs to be evaluated every training cycle and the weight vector that results in the maximum evidence for the network is chosen to be the final weight vector for the network training.

6.2.2 Prior weight distribution

The prior distribution of weights and biases reflects any knowledge about the form of network mapping expected to be found. The prior weights and biases can be initialised by sampling the data from the zero-mean multivariate Gaussian distribution of the form:

$$p(w|X) = \frac{1}{Z_w} \exp\left(-\frac{1}{2} w^T \Sigma^{-1} w\right) \quad (6.3)$$

where Z_w is the normalisation factor given by

$$Z_w = \frac{(2\pi)^{W/2}}{|\Sigma^{-1}|^{1/2}} = \frac{(2\pi)^{W/2}}{\left(\prod_{i=1}^W \tau_i\right)^{1/2}} \quad (6.4)$$

$$|\Sigma^{-1}| = \det(\Sigma^{-1}) \quad (6.5)$$

Σ^{-1} is the covariance matrix and $\tau_1, \tau_2, \dots, \tau_W$ are eigenvalues of Σ^{-1} .

However, $\tau_1, \tau_2, \dots, \tau_W$ can be represented by G parameters, $\xi_1, \xi_2, \dots, \xi_G$, corresponding to G groups of weights and biases in the network. Therefore, equation (6.4) yields:

$$Z_w = \frac{(2\pi)^{W/2}}{|\Sigma^{-1}|^{1/2}} = \frac{(2\pi)^{W/2}}{\prod_{g=1}^G (\xi_g)^{W_g/2}} \quad (6.6)$$

where W_g is the number of weights or biases in group g .

6.2.3 The data-set likelihood

The data-set likelihood measures how likely it is that the trained network fits the training data $D = \{x^{(n)}, t^{(n)}\}$, ($n = 1, \dots, N$) and is given by:

$$p(D|w) = \exp(-E_D(w)) \quad (6.7)$$

We also note that maximising the data-set likelihood $p(D|w)$ is equivalent to minimising the data error $E_D(w)$.

6.2.4 Evaluating the evidence

By substituting (6.3) and (6.7) into (6.2), we have:

$$p(D|X) = \frac{1}{Z_w} \int \exp\left(-E_D(w) - \frac{1}{2} \sum_{g=1}^G \xi_g \|w_g\|^2\right) dw \quad (6.8)$$

Let us define

$$F(w) = \exp\left(-E_D(w) - \frac{1}{2} \sum_{g=1}^G \xi_g \|w_g\|^2\right) \quad (6.9)$$

From equations (6.8) and (6.9), we have:

$$p(D|X) = \frac{1}{Z_w} \int F(w) dw \quad (6.10)$$

Similar to Chapter 4, $\int S(w)dw$ can be approximated around the local minimum by using the Laplace method and has the form:

$$\int S(w)dw = F(w_{\min}) (2\pi)^{W/2} [\det(A)]^{-1/2} \quad (6.11)$$

where w_{\min} is the weight vector at the local minimum and A is the Hessian matrix of $\ln F(w)$ evaluated at w_{\min} .

Finally, the evidence is given by:

$$p(D|X) = \frac{F(w_{\min}) (2\pi)^{W/2} [\det(A)]^{-1/2}}{Z_w} \quad (6.12)$$

By taking the logarithm of $p(D|X)$, we have:

$$\ln p(D|X) = -E_D(w_{\min}) - \frac{1}{2} \sum_{g=1}^G \xi_g \|w_g\|^2 + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g - \frac{1}{2} \ln(\det(A)) \quad (6.13)$$

Equation (6.13) can be recognised as the evidence for the network without considering the full connection property of the network.

6.2.5 Training neural networks for detecting head-movement commands

6.2.5.1 The use of data

The head-movement samples collected from the eight wheelchair users were used to train and test the BNNs, as shown in Table 6.1. The training set was taken from the samples of users 1, 2, 3, 4, 5, and 6, corresponding to 480 patterns. The movement samples from users 7 and 8 were used to assess the performances of the trained networks.

Table 6.1: Numbers of extracted head-movement samples of the eight wheelchair users.

User	Forward	Backward	Left	Right	Injury level
1	20	20	20	20	-
2	20	20	20	20	-
3	20	20	20	20	-
4	20	20	20	20	-
5	20	20	20	20	C5
6	20	20	20	20	C4
7	20	20	20	20	C4
8	20	20	20	20	C5

6.2.5.2 Network architecture

The BNNs used to detect head-movement commands had the following specifications:

- 41 input nodes, corresponding to 20 samples from x axis and 20 samples from y axis, and one augmented input of 1;
- three hidden nodes;
- four output nodes, corresponding to four classes: forward, backward, left and right movements.

6.2.5.3 Network training

The following procedure was used for training the networks:

1. The weights and biases were randomly initialised by using Nguyen and Widrow's method (Nguyen and Widrow 1990), because this method of weight initialisation can improve the learning speed.
2. The data error was minimised using the scaled conjugate-gradient algorithm. Compared to other training algorithms, this algorithm has the shortest duration of each training cycle, since the line minimisation is no longer required. The

evidence for the network was evaluated every cycle, using the following formula:

$$\ln p(D|X) = -E_D(w_{\min}) - \frac{1}{2} \sum_{g=1}^G \xi_g \|w_g\|^2 + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g - \frac{1}{2} \ln(\det(A))$$

3. If the evidence is the maximum at a specific training cycle, the weights and biases at that cycle were selected to be the final weights and biases.

Figures 6.2 and 6.3 show plots of data error and log evidence versus training cycle. We can see that as the number of training iterations increases, the data error decreases and the log evidence increases. However, the evidence reached the maximum value at cycle 23 and then decreased. The weights and biases at training cycle 23 were therefore chosen to be the final weights and biases.

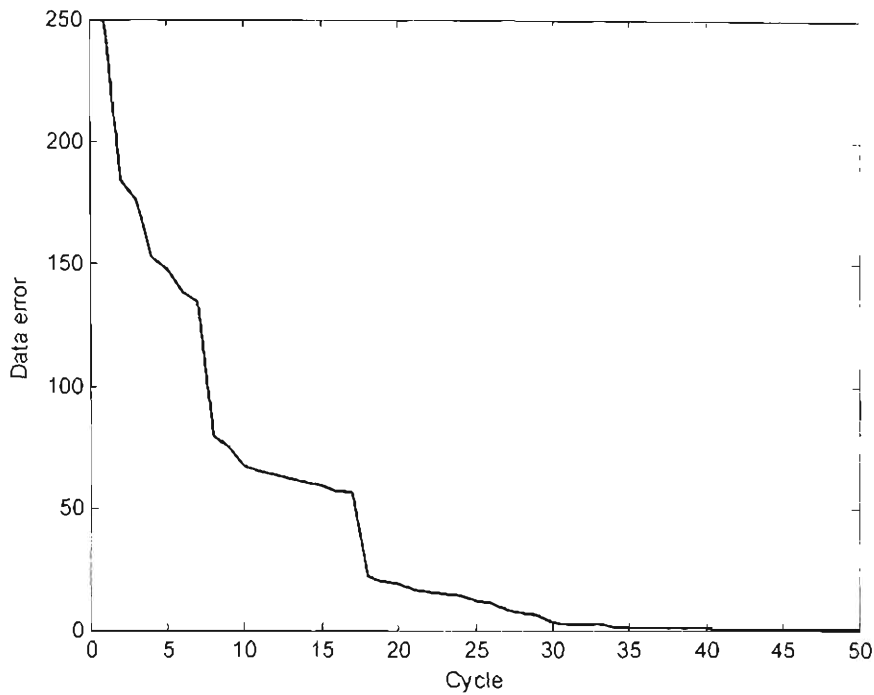


Figure 6.2: Data error versus training cycle.

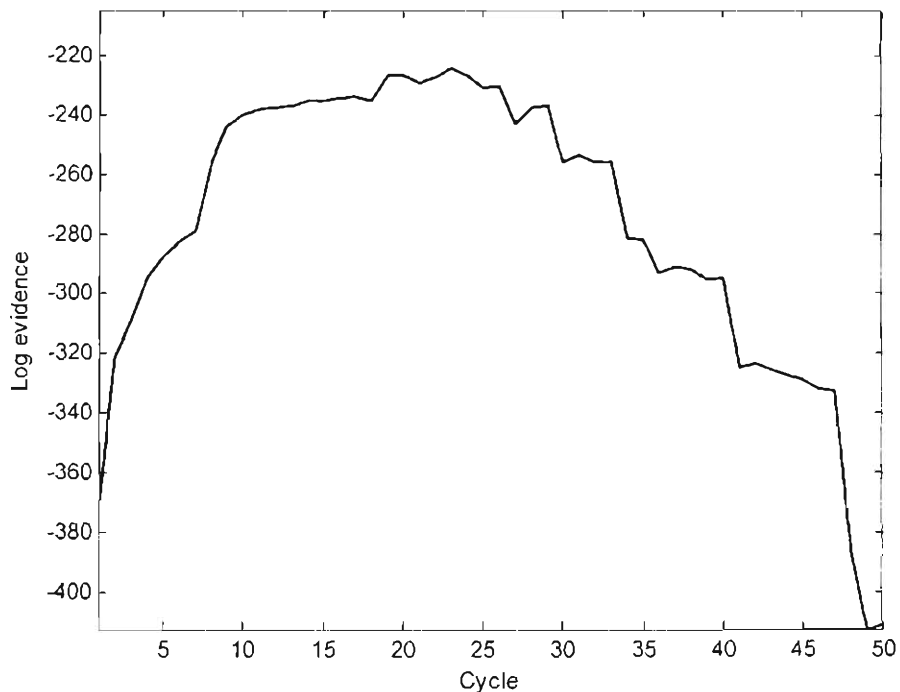


Figure 6.3: Log evidence versus training cycle: the maximum evidence is at training cycle 23.

As the solution of the network training is sensitive to the initial conditions of weights and biases in the network, there is a need to perform multiple training runs in order to find the best solution. However, the network training requires frequent evaluation of the

Hessian matrix, which is computationally expensive. The use of the cluster for training the networks can overcome this problem. Indeed, as shown in Table 6.2, the cluster is able to speed up the network training up to five times, compared to the personal computer (2.66 GHz – Intel Pentium 4 CPU).

Table 6.2: Network training time measured over the ten training runs.

Training run	The personal computer (seconds)	The cluster (seconds)
1	67.637	14.954
2	67.918	14.976
3	68.088	15.234
4	67.687	15.081
5	68.128	14.933
6	67.908	15.022
7	67.768	15.047
8	70.161	15.023
9	72.214	15.252
10	74.397	15.079

In order to find the best solution, twenty training runs were performed on the cluster. The weights and biases that resulted in the highest accuracy of the network on the test set were selected as the final solution for all the training runs. Table 6.3 shows the confusion matrix of the best trained network on the test set, and the trained network can classify the test set with a high success rate of 91.25%. The network also has very high sensitivity and specificity, as shown in Table 6.4.

Table 6.3: Confusion matrix of the best trained network.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	39	0	0	1	
	Backward	0	30	10	0	
	Left	2	0	38	0	
	Right	1	0	0	39	
Accuracy (%)		91.25				

Table 6.4: Sensitivity and specificity of the best trained network.

Sensitivity	Specificity
0.9125	0.9708

6.3 The Monte Carlo methods for Bayesian neural network training

A practical framework of Bayesian training in neural networks uses the Monte Carlo sampling technique (M.Neal 1996). This framework of BNN training can avoid the Gaussian approximation to the posterior distribution of weights (Rasmussen 1996).

6.3.1 The Monte Carlo methods

The Monte Carlo methods aim to solve one or both of the following problems:

- *Problem 1:* to generate samples $\{w^{(r)}\}$ from a desired probability distribution $p(w)$.
- *Problem 2:* to estimate the expectation of the function $\phi(w)$ under the distribution $p(w)$, for example:

$$\langle \Phi(w) \rangle \equiv \int \phi(w)p(w)dw \tag{6.14}$$

We now consider the first problem, known as the sampling problem, because the second problem can be conveniently solved by using the random samples $\{w^{(r)}\}$ to give the estimator:

$$\hat{\Phi} \approx \frac{1}{R} \sum_{r=1}^R \phi(w^{(r)}) \quad (6.15)$$

It is clear that $\{w^{(r)}\}$ are generated from $p(w)$ and the expectation of $\hat{\Phi}$ is Φ . As the number of samples R increases, the variance of $\hat{\Phi}$ will decrease as σ^2 / R , where σ^2 is the variance of ϕ ,

$$\sigma^2 = \int (\phi(w) - \Phi)^2 p(w) dw \quad (6.16)$$

Equation (6.16) is one of the important properties of the Monte Carlo methods.

6.3.2 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is a method of generating the samples from the desired distribution $p(w)$. In this method, the successive random points $w^{(r)}$ are generated by a *random walker* moving through the w space. As the walk becomes longer and longer, the points it connects will closely approximate the desired distribution. The combination of the Monte Carlo integration methods and the Metropolis-Hasting algorithm is sometimes known as a simple form of the Hidden Markov Monte Carlo technique.

The rules of the Metropolis-Hastings algorithm can be summarised as follows:

1. In order to generate a new point $w^{(n+1)}$ from a old point $w^{(n)}$, there is a need to make a trial step r to a trial point $w^{(t)}$. This trial point can be chosen in any convenient manner, for example, it can be generated uniformly at random within a multidimensional cube of small side δ about $w^{(n)}$.

2. The trial step is then “accepted” or “rejected” according to the ratio:

$$r = \frac{P(w^{(t)})}{P(w^{(n)})} \quad (6.17)$$

- If r is greater than one, the trial step is always accepted and $w^{(n+1)} = w^{(t)}$.
- If r is less than one, the trial step is accepted with the probability r and we go to step 3.

3. Generate a random number η uniformly distributed in the interval. If $r > \eta$, the trial step is accepted, otherwise the trial step is rejected and $w^{(n+1)} = w^{(n)}$.

4. Repeat step 1 to step 3 until the number of iterations is reached.

Figure 6.4 shows the random samples generated from the zero-mean Gaussian distribution in the interval $[a = 0, b = 10]$ using the Metropolis-Hastings algorithm. As shown in Figure 6.5, the histogram of 20 sampled points truly reflects the desired Gaussian distribution.

In Figure 6.4, the curve was drawn from the function $f(w) = w^2 - 7w + 3$. According to

the Monte Carlo methods, $\int_a^b f(w)dw$ is computed as:

$$\int_a^b f(w)dw = \frac{S}{R}(Y_1 - Y_2)(b - a) + Y_2(b - a) \quad (6.18)$$

where Y_1 and Y_2 are the upper and lower bounds of the function in the interval. Y_1 is greater than the maximum value of $f(x)$ and Y_2 is less than the minimum value of $f(x)$ in the interval $[a, b]$.

$$Y_1 = f_{\max}(w) + \zeta \quad w \in [a, b] \quad (6.19)$$

$$Y_2 = f_{\min}(w) - \zeta \quad w \in [a, b] \quad (6.20)$$

where ζ is a small-positive value. R is the number of all the samples generated in the rectangle bounded by a, b, Y_1, Y_2 and S is the number of samples below the curve.

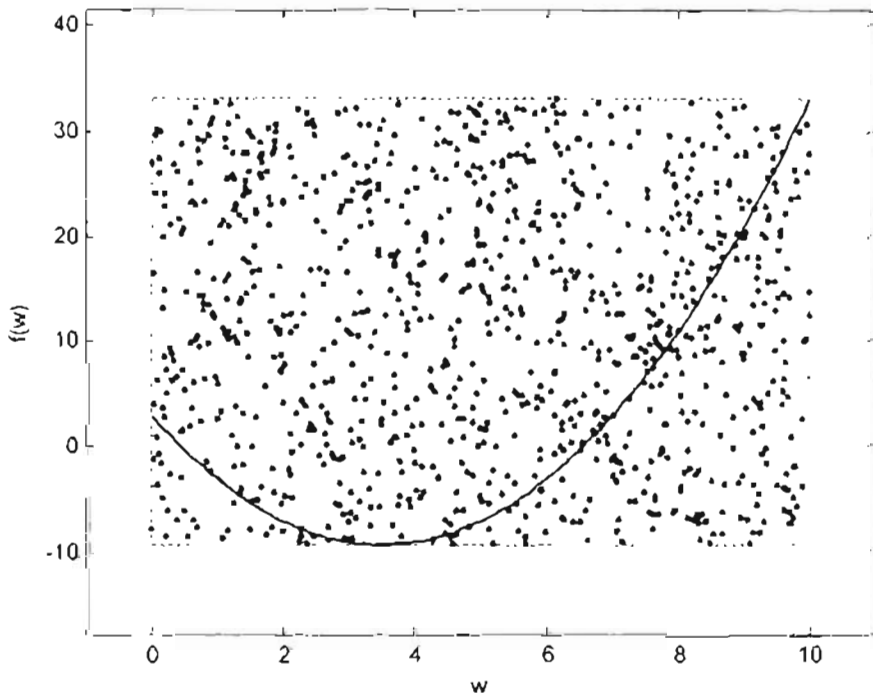


Figure 6.4: An illustration of the Monte Carlo methods: 1,000 samples generated in the rectangle, in which 248 samples below the curve ($f(w) = w^2 - 7w + 3$) are used to evaluate the integral of $f(w)$ in the interval using the numerical Monte Carlo method. The numerical integral is 13.075, while the exact integral is 13.333 and therefore the accuracy of the Monte Carlo methods in this case is 98%.

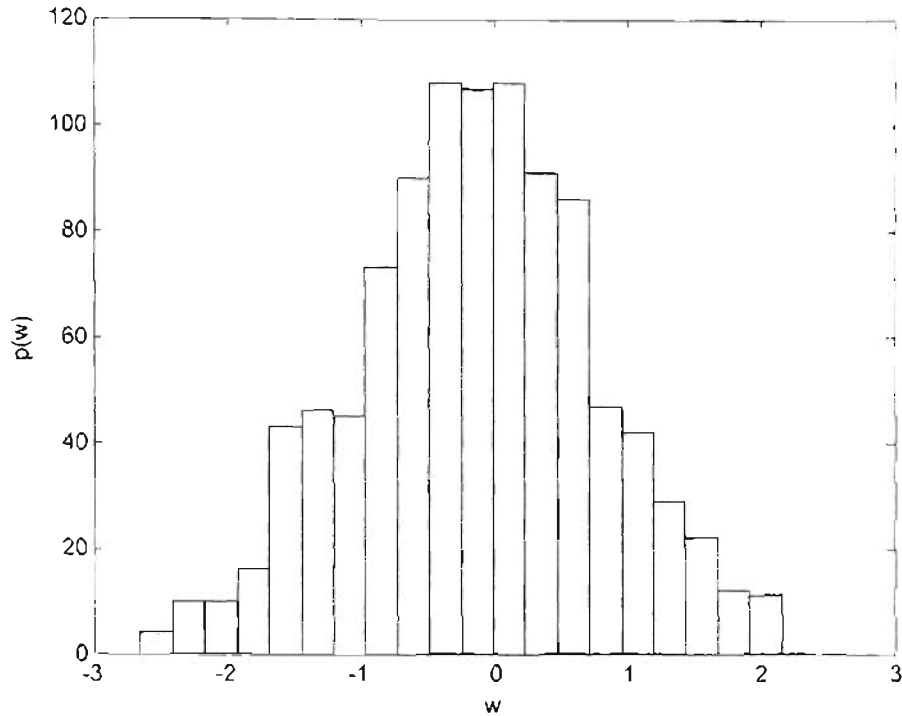


Figure 6.5: The histogram of 20 sampled points generated from the zero-mean Gaussian distribution: The histogram truly reflects the desired Gaussian distribution.

In neural networks, the distribution of the output vector is an integral over the weight space and is given by

$$p(t | x, D) = \int p(t | x, w) p(w | D) dw \tag{6.21}$$

where $p(w | D)$ represents the posterior weight distribution. In the Monte Carlo methods, the integral (6.21) can be approximated by finite sums of the form:

$$p(t | x, D) \approx \frac{1}{R} \sum_{r=1}^R p(t | x, w^{(r)}) \tag{6.22}$$

where $w^{(1)}, w^{(2)}, \dots, w^{(R)}$ represent R weight vectors sampled from the posterior distribution $p(w | D)$.

Recall the evidence for the network

$$p(D|X) = \int p(D|w)p(w|X)dw \quad (6.23)$$

$p(D|X)$ can be computed using the numerical Monte Carlo methods as follows:

$$p(D|X) \approx \hat{P}(D|X) = \frac{1}{R} \sum_{r=1}^R p(D|w^{(r)}) \quad (6.24)$$

where $\{w^{(r)}\}$ are the sampled weight vectors generated from the distribution $p(w|X)$.

6.3.3 Training Bayesian neural networks for detecting head-movement commands

A BNN for detecting head-movement commands was trained using the Monte Carlo methods. The network has three neurons in the hidden layer. The training set was formed from the samples of users 1, 2, 3, 4, 5, and 6. The test set was formed from the samples of users 7 and 8.

The scaled conjugate-gradient algorithm was used to minimise the error function. The evidence for the network was evaluated every training cycle. Figures 6.6 and 6.7 show the change of data error and log evidence according to the training cycle. As the number of training cycles increases, the data error decreases. As shown in Figure 6.7, the maximum evidence can be recognised at training cycle 24.

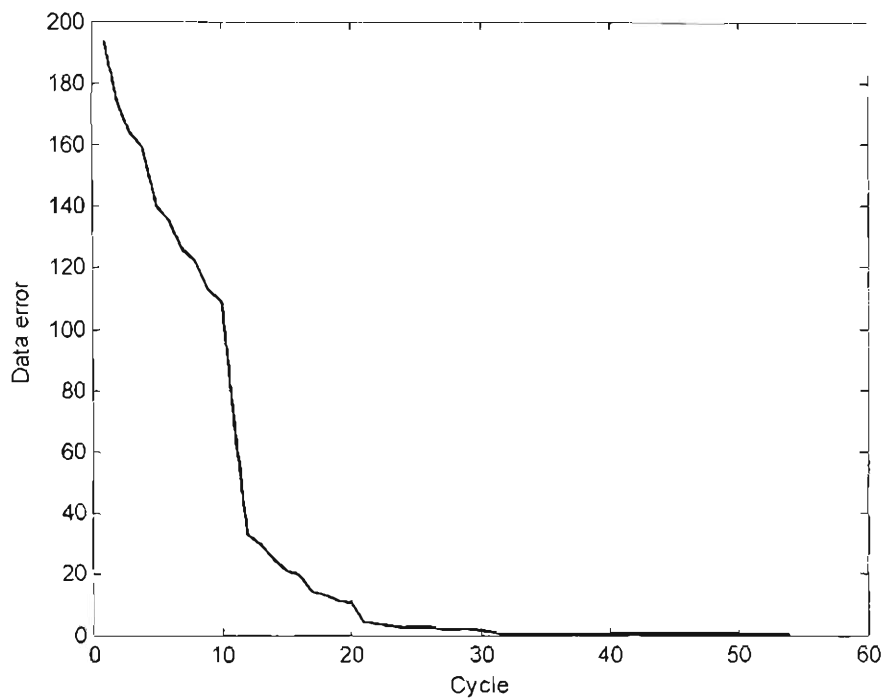


Figure 6.6: Data error versus training cycle.

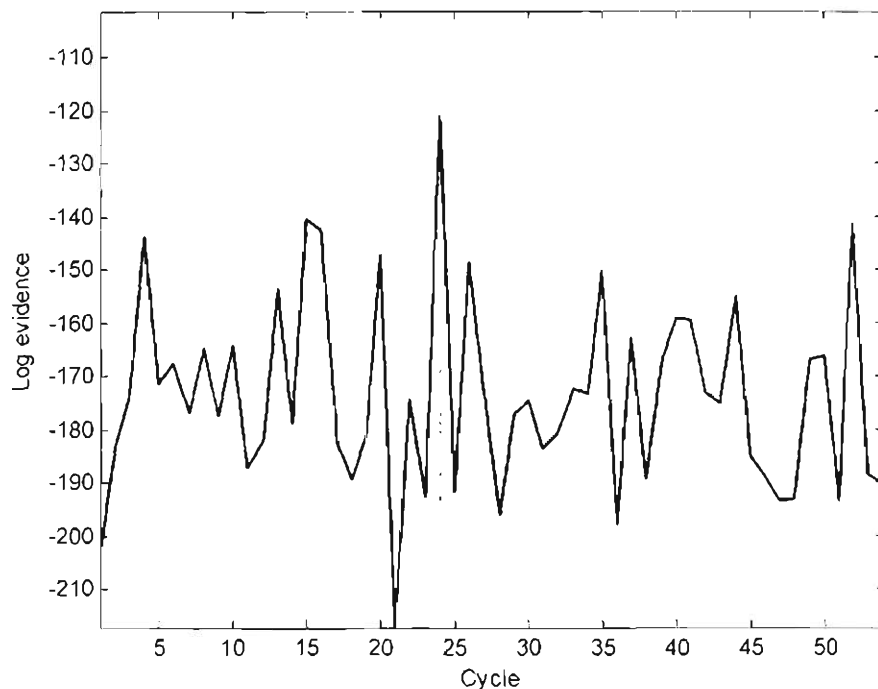


Figure 6.7: Log evidence versus training cycle: the maximum log evidence is at cycle 24.

Similar to Section 6.2.5, twenty training runs with different initial conditions of the weights and biases were conducted on the cluster in order to find the best network parameters. For the Monte Carlo methods, the number of sampled weight vectors

generated from the prior distribution is 500. As the number of sampled weight vectors increases, the duration of each training cycle becomes large.

Table 6.5 shows the confusion matrix of the best trained network. We can see that the network is able to classify the samples in the test set with an accuracy of 90.625%. The network also has very high sensitivity and specificity, as shown in Table 6.6.

Table 6.5: Confusion matrix of the best trained network.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	34	0	0	6	
	Backward	0	33	7	0	
	Left	0	0	40	0	
	Right	0	1	1	38	
		Accuracy (%)	90.625			

Table 6.6: Sensitivity and specificity of the best trained network.

Sensitivity	Specificity
0.9063	0.9688

6.4 Combining the performances of independent Bayesian neural network classifiers

Figure 6.8 illustrates the block diagram of a system for detecting hands-free commands from different channels of signal inputs: head movement, electroencephalogram (EEG) and electromyogram (EMG). The system consists of two levels:

- *Level 1:* is responsible for detecting different hands-free commands using independent BNN classifiers, known as the level for making the evidence.
- *Level 2:* is responsible for combining the decisions from the BNN classifiers to output the global decision, known as the level for judging the evidence.

The advantage of such a multiple hands-free command input recognition system is that in case one channel does not work properly or accurately, the performance of the system is not worse than that of the best channel. The combination of independent classifiers can be achieved by using either the Bayes' rule or the Dempster-Shafer theory.

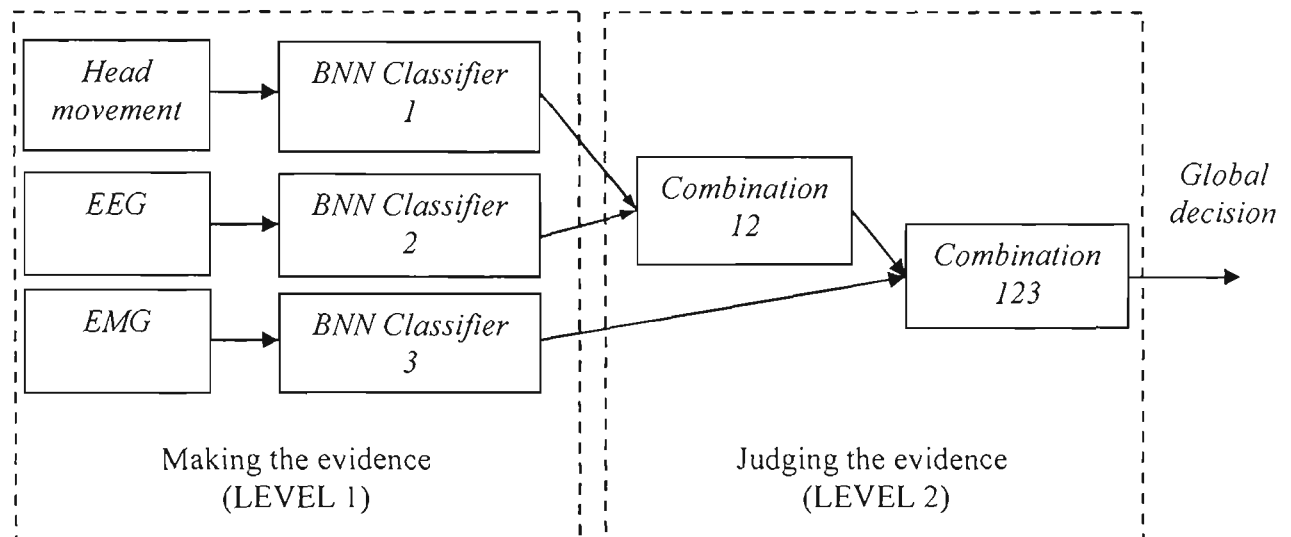


Figure 6.8: Block diagram of a system for detecting commands through the combination of three hands-free input signals: head movement, electroencephalogram (EEG) and electromyogram (EMG).

6.4.1 Bayes' rule of combination

Suppose that two BNN classifiers X_1 and X_2 were trained separately on the independent feature vectors x_1 and x_2 , provided by two sensors S_1 and S_2 , as shown in Figure 6.9. Then, in case of failure of sensor S_1 , the system should be able to detect the fact that classifier X_1 is less reliable, and should consequently decrease its influence on the global decision. If the reliability of classifier X_2 is correctly assessed, then the performance of the whole system should never become worse than the performance of classifier X_2 alone.

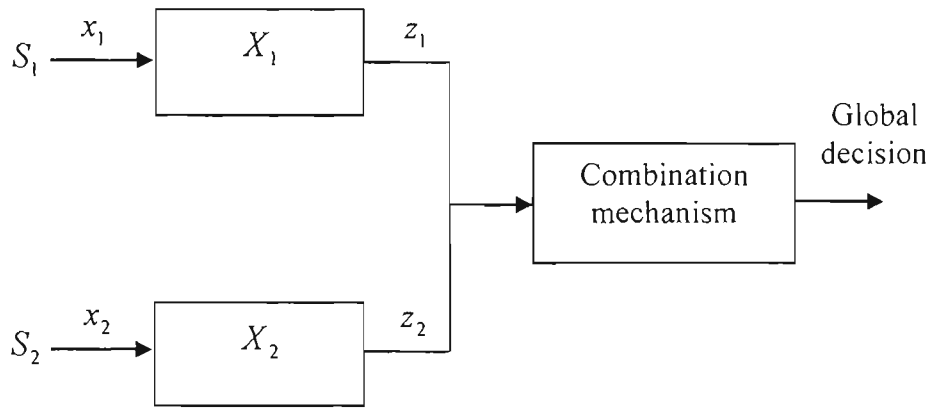


Figure 6.9: A two-classifier recognition system.

As mentioned in Chapter 3, the use of ‘softmax’ output functions can allow us to directly compute the conditional probabilities of the output classes C_k ($k = 1, \dots, c$) $P(C_k | x)$.

$$P(C_k | x) = \frac{\exp(a_k(x))}{\sum_{k=1}^c \exp(a_k(x))} \quad (6.25)$$

where $a_k(x)$ is the activation of the k th output nodes. We also have:

$$\sum_{k=1}^c P(C_k | x) = 1 \quad (6.26)$$

We now assume that the two classifiers provide the estimates for $P(C_k | x_1)$ and $P(C_k | x_2)$, respectively. If x_1 and x_2 are observed simultaneously, then the global decision is made based on $P(C_k | x_1, x_2)$, given by

$$P(C_k | x_1, x_2) = \frac{p(x_1, x_2 | C_k)P(C_k)}{p(x_1, x_2)} \quad (6.27)$$

If x_1 and x_2 are conditionally independent in each class, then

$$p(x_1, x_2 | C_k) = p(x_1 | C_k) p(x_2 | C_k) \quad (6.28)$$

According to the Bayes theorem, we can write

$$p(x_1 | C_k) = \frac{P(C_k | x_1) p(x_1)}{P(C_k)} \quad (6.29)$$

$$p(x_2 | C_k) = \frac{P(C_k | x_2) p(x_2)}{P(C_k)} \quad (6.30)$$

Substituting (6.29) and (6.30) into (6.28) gives:

$$p(x_1, x_2 | C_k) = \frac{P(C_k | x_1) P(C_k | x_2) p(x_1) p(x_2)}{(P(C_k))^2} \quad (6.31)$$

Substituting (6.31) into (6.27) gives:

$$P(C_k | x_1, x_2) = \frac{P(C_k | x_1) P(C_k | x_2)}{P(C_k)} \quad (6.32)$$

Here, the probability of the output class C_k , given x_1 and x_2 , can be obtained by multiplying the posterior probabilities of that class provided by each classifier, dividing by the prior, and renormalising.

6.4.2 Dempster-Shafer theory of combination

During the early 1980s, the Dempster-Shafer (D-S) theory of evidence developed by Arthur P. Dempster and Glenn Shafer has attracted a considerable interest in the artificial intelligence field (System; Henkind and Harrison 1988; Stephanou and Lu 1988; Principe, Gala et al. 1989).

The D-S theory is based on *belief functions* and *plausible reasoning*, which is used to combine separate pieces of information, known as the *evidence*, to calculate the probability of an event (Denoeux 1995; Denoeux 2000).

Let Ω be a finite set of mutually exclusive and exhaustive hypotheses, called the *frame of discernment*. A *basic belief assignment* (BBA) is defined as a function m from 2^Ω to $[0,1]$, verifying:

$$m(\emptyset) = 0 \quad (6.33)$$

$$\sum_{A \subseteq \Omega} m(A) = 1 \quad (6.34)$$

For any $A \subseteq \Omega$, $m(A)$ represents the belief that one is willing to commit exactly to A , given a certain piece of evidence. The subset A of Ω such that $m(A) > 0$ are called the *focal elements* of m . Associated with m are a *belief function* bel and a *plausibility function* pl , defined, respectively, for all $A \subseteq \Omega$ as:

$$bel(A) = \sum_{B \subseteq A} m(B) \quad (6.35)$$

$$pl(A) = \sum_{A \cap B = \emptyset} m(B) \quad (6.36)$$

Two BBAs m_1 and m_2 on Ω , induced by two independent items of evidence, can be combined by the so-called *Dempster's rule of combination* to yield a new BBA $m = m_1 \oplus m_2$, called the orthogonal sum of m_1 and m_2 , and defined as:

$$m(\emptyset) = 0 \quad (6.37)$$

$$m(A) = m(B) \oplus m(C) = \frac{\sum_{B \cap C = A} m_1(B) m_2(C)}{1 - \sum_{B \cap C \neq \emptyset} m_1(B) m_2(C)} \quad (6.38)$$

The computation of m is possible if and only if there are at least two subsets B and C of Ω , with $B \cap C = \emptyset$, such that $m_1(B) \neq 0$ and $m_2(C) \neq 0$. m_1 and m_2 are said to be *combinable*.

6.4.3 Combining two Bayesian neural network classifiers

Figure 6.10 shows a two-BNN classification system, in which each BNN can be trained independently on different training data sets. These BNNs can also have different architectures. The outputs of the networks can be seen as the pieces of information for the combination of the classifiers.

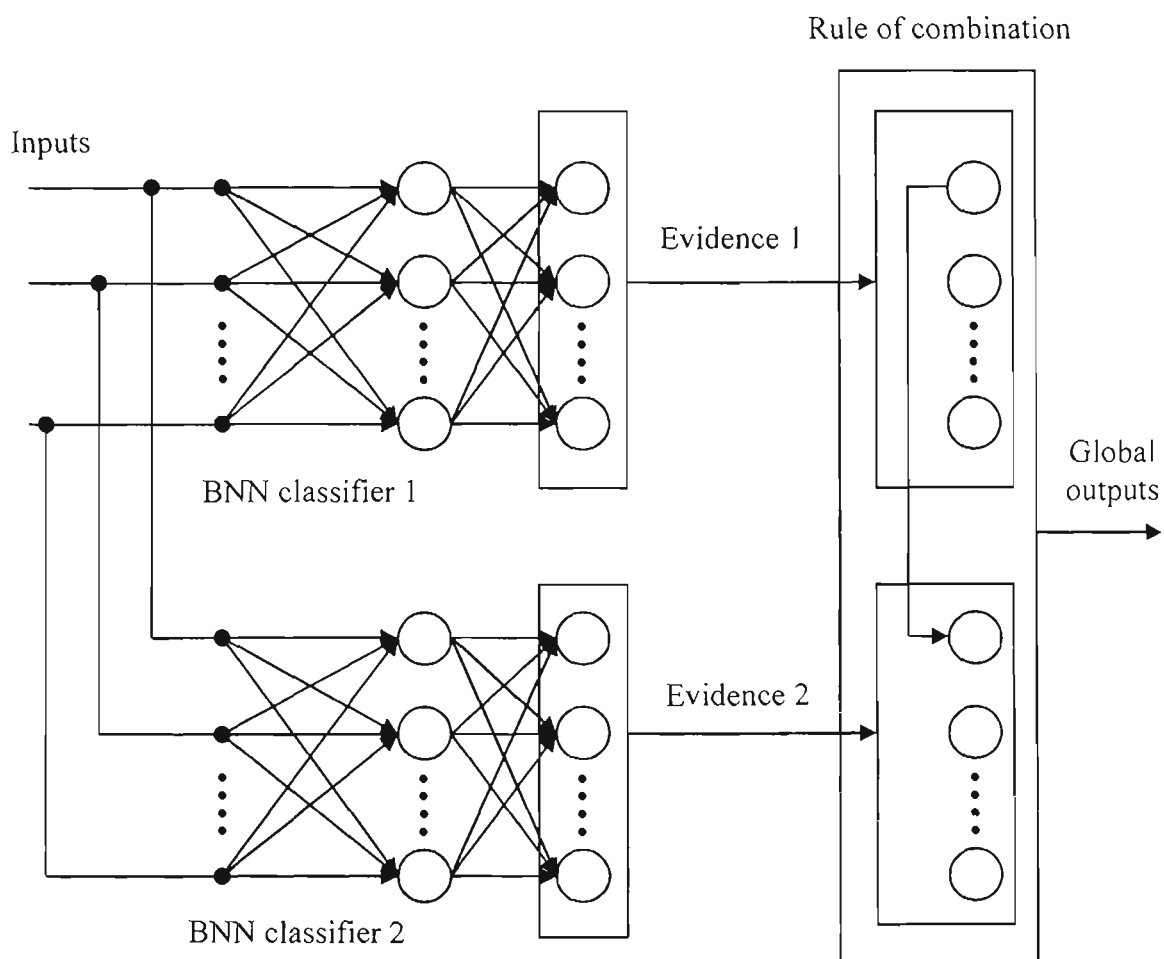


Figure 6.10: A two-BNN classifier recognition system.

6.4.3.1 Experiment 1 (Bayes' rule of combination)

According to Bayes' rule of combination, we have:

$$P_{12}(C_k | x) = \frac{P_1(C_k | x)P_2(C_k | x)}{P(C_k)} \quad k = 1, \dots, c \quad (6.39)$$

where $P_1(C_k | x)$ is the probability of the output class C_k , given the input vector x , for BNN classifier 1. Similarly, $P_2(C_k | x)$ is the probability of the output class C_k , given the input vector x , for BNN classifier 2. $P(C_k)$ is the probability of the output class C_k and $P_{12}(C_k | x)$ is the probability of the output class C_k , given the input vector x , for the combination of the two classifiers.

The training set for BNN classifier 1 was taken from the samples of users 1, 2 (able-bodied persons) and 5 (C5-injury level person). The training set for BNN classifier 2 was taken from the samples of users 3, 4 (able-bodied persons) and 6 (C4-injury level person). The test set for both BNN classifiers was taken from the samples of user 7 (C4-injury level person) and user 8 (C5-injury level person). The quasi-Newton algorithm was used to train the BNN classifiers.

Tables 6.7, 6.8 and 6.9 show the confusion matrices of the trained BNN classifiers on the test set. Obviously, the classification result from the combination of the BNN classifiers is significantly higher than that from the independent BNN classifiers (98.12% accuracy). The combination of the two BNNs also resulted in very high sensitivity and specificity, as shown in Table 6.10.

Table 6.7: Confusion matrix of Bayesian neural network classifier 1.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	32	0	0	8	
	Backward	0	37	3	0	
	Left	1	0	39	0	
	Right	0	2	0	38	
Accuracy (%)		91.25				

Table 6.8: Confusion matrix of Bayesian neural network classifier 2.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	40	0	0	0	
	Backward	0	39	1	0	
	Left	2	0	38	0	
	Right	6	0	2	32	
Accuracy (%)		93.13				

Table 6.9: Confusion matrix of the combination of two BNN classifiers.

		Predicted classification				
		Movement	Forward	Backward	Left	Right
Actual classification	Forward	40	0	0	0	
	Backward	0	40	0	0	
	Left	1	0	39	0	
	Right	0	2	0	38	
Accuracy (%)		98.13				

Table 6.10: Sensitivity and specificity of the classifiers.

	Sensitivity	Specificity
BNN classifier 1	0.9125	0.97083
BNN classifier 2	0.93125	0.97708
Combination of two BNN classifiers	0.98125	0.99375

6.4.3.2 Experiment 2 (Dempster-Shafer theory of combination)

In head-movement classification, there are four independent output classes, corresponding to forward, backward, left and right movements. According to the unnormalised or conjunctive rule of combination \cap , defined for all output classes $C_k \subseteq \Omega$ by (Smets 1990):

$$m = m_1 \cap m_2 \Leftrightarrow m(C_k) = m_1(C_k)m_2(C_k) \tag{6.40}$$

where $m_1(C_k)$ is the belief function of the output class C_k , corresponding to classifier 1 and $m_2(C_k)$ is the belief function of the output class C_k , corresponding to classifier 2.

Similar to Experiment 1, the training set for BNN classifier 1 was formed from the samples of users 1, 2 (able-bodied persons) and 5 (C5-injury level person). The training set for BNN classifier 2 was taken from the samples of users 3, 4 (able-bodied persons) and 6 (C4-injury level person). The test set for both BNN classifiers was formed from the samples of user 7 (C4-injury level person) and user 8 (C5-injury level person). The quasi-Newton algorithm was also used to train the BNN classifiers.

Table 6.11, 6.12 and 6.13 show the confusion matrix of the trained BNN classifiers on the test set. In particular, the combination of two BNN classifiers results in an excellent accuracy of 97.5%. As shown in Table 6.14, the combination of two BNN classifiers also results in the high sensitivity and specificity.

Table 6.11: Confusion matrix of Bayesian neural network classifier 1.

		Predicted classification			
	Movement	Forward	Backward	Left	Right
Actual classification	Forward	30	0	0	10
	Backward	0	36	4	0
	Left	0	0	40	0
	Right	0	2	0	38
	Accuracy (%)	90.00			

Table 6.12: Confusion matrix of Bayesian neural network classifier 2.

		Predicted classification			
	Movement	Forward	Backward	Left	Right
Actual classification	Forward	40	0	0	0
	Backward	0	39	1	0
	Left	1	0	39	0
	Right	3	0	2	35
	Accuracy (%)	95.63			

Table 6.13: Confusion matrix of the combination of two Bayesian neural network classifiers.

		Predicted classification			
	Movement	Forward	Backward	Left	Right
Actual classification	Forward	39	0	0	1
	Backward	0	39	1	0
	Left	0	0	40	0
	Right	0	2	0	38
	Accuracy (%)	97.50			

Table 6.14: Sensitivity and specificity of the classifiers.

	Sensitivity	Specificity
BNN classifier 1	0.9	0.96
BNN classifier 2	0.95	0.98
Combination of two BNN classifiers	0.97	0.99

6.5 Discussion

This chapter has provided three frameworks for improving the accuracy of detecting head-movement commands. The first framework forms a new version of early stopping for BNN classifiers. In this version of the early stopping, all the training data can be used to train the networks.

The second framework is the Monte Carlo method-based implementation of BNN training. In particular, the Gaussian approximation to the posterior distribution of weights and biases is no longer needed. Moreover, the Monte Carlo methods for BNN training allow us to investigate other types of weight distribution.

The third framework is concerned with the two methods of combining the solutions of independent BNN classifiers, based on Bayes' rule and the Dempster-Shafer theory. This framework is promising for the applications of data fusion and shared control. For example, a head movement classifier can be combined with an EEG classifier in order to output alternative hands-free commands for controlling the direction of power wheelchairs.

Finally, the chapter has demonstrated that the exploration of computational clusters results in a significant reduction in the network training time if there are a large number of networks needing to be trained. In this research, the cluster is able to speed up the network training up to five times, compared to the personal computer.

Chapter 7

Conclusions and Future Work

Nowadays, thanks to the progress in assistive technologies, the quality of life of severely disabled people can be significantly improved. Especially, several forms of assistive devices enable disabled people to maintain their daily activities without the help of others.

For a large number of severely disabled people, head movement is one of the hands-free gestures that can be utilised to directly convey intentions. By changing the head direction consciously, a disabled person can generate a number of commands for controlling many assistive devices, such as power wheelchairs. However, the exact detection of head-movement commands of various forms of severely disabled people is still very challenging.

The flexibility of neural networks allowed them to become a powerful modelling and forecasting tool across different research areas in recent years. The need to exactly detect the head commands of many severely disabled people leads to the use of neural networks.

The performance of a neural network depends significantly on how the network has been trained. The traditional method of neural network training is based on the maximisation of the likelihood, which is equivalent to the minimisation of the error between the actual network outputs and the expected network outputs. However, as the training set is insufficient or biased, the minimisation of the error has the potential to make the network overfit the biased data.

In Chapter 3, the regularisation procedure has been used to enhance the generalisation of neural network classifiers for detecting head-movement commands. Specifically, a weight penalty term proportional to the sum of squared weights and biases is added to the data error function in order to prevent the network from overfitting. This penalty term consists of some regularisation parameters or hyperparameters that need to be determined properly.

The determination of the proper hyperparameters requires evaluating the Hessian matrix of the data error function several times. Usually, the time needed to evaluate the Hessian matrix is dominated by the size of the network. Rather than evaluating the Hessian matrix directly, Chapter 3 has described the $\{\mathcal{R}\}$ -operator technique for the fast exact multiplication of the Hessian matrix. The full Hessian matrix is then conveniently returned by using the identity matrix having the same size as the Hessian matrix.

Chapter 4 has described the Bayesian technique in determining the proper values of the hyperparameters. This approach is based on the Gaussian approximation to the posterior distribution of weights and biases at the local minima and does not require using a validation set separated from the available training set. The experimental results obtained show that after different training modes, the networks can at least achieve an accuracy of 95% on the test sets.

Chapter 4 has also provided three modifications of the gradient descent network training algorithm: conjugate gradient, scaled conjugate gradient and quasi-Newton algorithms. These training algorithms can be used to minimise the cost function without manually setting the step size and the search direction.

In the conjugate gradient algorithm, a line search with great accuracy is required in each training cycle to ensure that the subsequent search direction is adequate enough. The line search that is required in the quasi-Newton algorithm does not play a crucial role in the algorithm. Finally, the scaled conjugate gradient algorithm was proposed to avoid the line search.

A detailed comparison of these three algorithms in terms of the convergence rate was also made. The experimental results obtained show that the quasi-Newton and the scaled conjugate gradient algorithms can result in a short training time and are appropriate for the on-line network training.

Chapter 5 has described the method of comparing and ranking various network architectures by evaluating the evidence for the networks. The greatest evidence of the network is a trade-off between the best-fit likelihood and the model complexity. The experimental results have indicated that the network architecture with three neurons in the hidden layer is optimal for this application, as this architecture results in the highest evidence and low test errors for the trained networks. This chapter has also provided a procedure for the adaptive network training. After being trained on the samples from new disabled persons, the network is able to adapt well to the head movements of these people, with an accuracy of 93.75%.

Chapter 6 has demonstrated that the Bayesian evidence framework can also be applied to providing a new version of the early stopping in neural networks. In particular, the network training can be stopped at the cycle with the maximum evidence. However, this approach requires evaluating the Hessian matrix frequently, which is computationally expensive, as the network size is large and many networks need to be trained. Therefore, the chapter has described the utilisation of the cluster in training a large number of neural networks. The experimental results show that the cluster is able to speed up the network training process up to five times, compared to the personal computer. Finally, the chapter has introduced two methods for combining the solutions of independent BNNs in order to improve the accuracy of detecting head-movement commands. These two methods are based on Bayes' rule and the Dempster-Shafer theory for combining the pieces of information.

In the Bayesian evidence framework for neural networks, the central operation is integration over the weight space. For example, the distribution of the network outputs and the network evidence are integrals over the weight space. The Gaussian

approximation to the posterior distribution of weights and biases allows these integrals to be performed analytically. Chapter 6 has described the Monte Carlo sampling technique, known as an alternative method for evaluating multidimensional integrals.

The effectiveness of BNNs can be utilised in order to develop other applications in biomedical engineering, such as developing a Brain Computer Interface (BCI) for wheelchair control (Craig, H.T.Nguyen et al. 2006) and improving the performance of the device for non-invasively monitoring the blood glucose level of Type I-diabetic patients (Ghevondian and H.T.Nguyen 1999; Ghevondian and H.T.Nguyen 2001).

The research into BCI has resulted in considerable achievements over the last decade. Powerful computers, and low-cost equipment for recording neurophysiologic signals are the main reasons that make BCI devices feasible. A BCI is able to give people with severe muscular disabilities the benefit of basic communication capabilities. The dominant neural activity measurement used in current BCI approaches is the EEG. The analysis of EEG has been a major topic in signal-processing and neural network researches. The EEG is a complex, time-varying signal which requires sophisticated analysis techniques after clinically useful information has been extracted from it.

Monitoring the blood glucose levels of Insulin Dependent Diabetes Mellitus (IDDM) is essential for detecting onset of hypoglycaemia and hyperglycaemia. A method using a standard neural network (SNN) has been developed for estimating blood glucose levels non-invasively, using only physiological parameters such as skin impedance and heart rate.

Motivated by an idea that an assistive wheelchair capable of inferring the belief about the states of the user's intention is able to make the decision about how and when it should assist the user, a future wheelchair control system should be able to continuously estimate the user's behaviour.

Microsoft researchers in human - computer interaction have recently focused on the use of Bayesian networks and influence diagrams in embedded applications to make

inferences about the goals of users and to take ideal actions based on probability distributions over these goals (Heckerman 1996). Bayesian agents maintain beliefs about critical variables such as a user's intentions and attention. So the inference of the wheelchair user's intention can be performed by using a Bayesian network, known as a form of graphical model. Before being able to predict the user's intention, the network needs to learn from the earlier user's behaviour with corresponding contexts.

Appendix A

Power Wheelchairs

A.1 Overview of power wheelchairs

Power wheelchairs, as shown in Figure A.1, provide independent mobility for people with both lower and upper extremity impairments. In addition, power wheelchairs are becoming increasingly important as more users transition from manual mobility to powered mobility. The cost of a standard power wheelchair depends on how it has been specialised. While different types of power wheelchairs are now available, all of these configurations use essentially the same software, despite the fact that their system dynamics vary considerably.



Figure A.1: A standard power wheelchair (Invacare Roller M1).

The propulsion system of power wheelchairs typically consists of a pair of electric motors, one for each drive wheel, and a drive train consisting of gears, belts and other mechanical elements, which couples the motor's shaft to the drive wheel shaft. The functional block diagram of power wheelchair components is shown in Figure A.2.

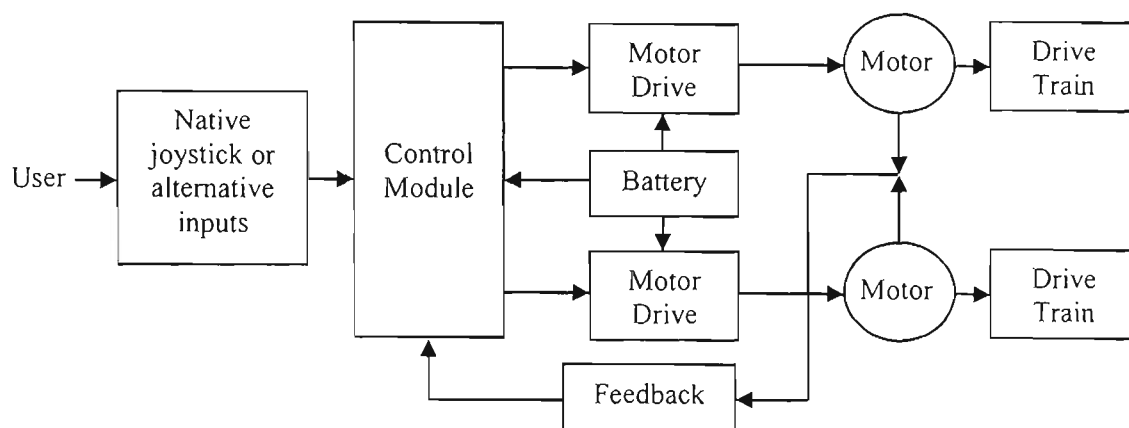


Figure A.2: Functional block diagram of power wheelchairs.

Most power wheelchairs utilise two permanent magnet DC (PMDC) motors with 12-volt batteries providing a 24-volt supply. The principal advantage of a PMDC motor is that it has a linear torque-speed profile. A DC-DC converter drives each motor with a high-frequency, square-wave-pulse train that rapidly turns each motor on and off. Speed and torque generated by each motor are controlled by modulating the pulse width. Electronic switches, such as bi-polar transistors or MOSFETs, connected in an H-bridge configuration, are usually used to switch the voltage polarity to alter the rotation of the PMDC motors.

According to the drive wheel location relative to the system centre of gravity, power wheelchairs can be classified into rear-wheel-drive (RWD), centre-wheel-drive (CWD) and front-wheel-drive (FWD) power wheelchairs. This classification makes it easier to understand and predict how power wheelchairs perform. A RWD chair is the most suitable for situations in which there is limited space behind the driver. If there is limited space in front of and behind the driver, then a CWD chair will be more manoeuvrable. If there is a confined area in front of and more space behind the driver, a FWD chair requires the least space in front to turn through a given arc.

A.2 Power wheelchair components

A.2.1 Joystick and alternative control

In most power wheelchairs, a proportional joystick-control that requires minimal hand movement to control speed and direction is usually used. A schematic of a joystick with error checking circuitry is illustrated in Figure A.3. Light and forward touch progresses

the wheelchair slowly; pushing the joystick further will increase speed. Direction is controlled by moving the joystick towards the left or right, or backwards to drive the wheelchair in reverse. People who do not have a dedicate touch can have a joystick that is not proportional in operation. A slight tremor can be usually accommodated by the electronics of a joystick. However, a more significant tremor may need a more sophisticated control to compensate for it.

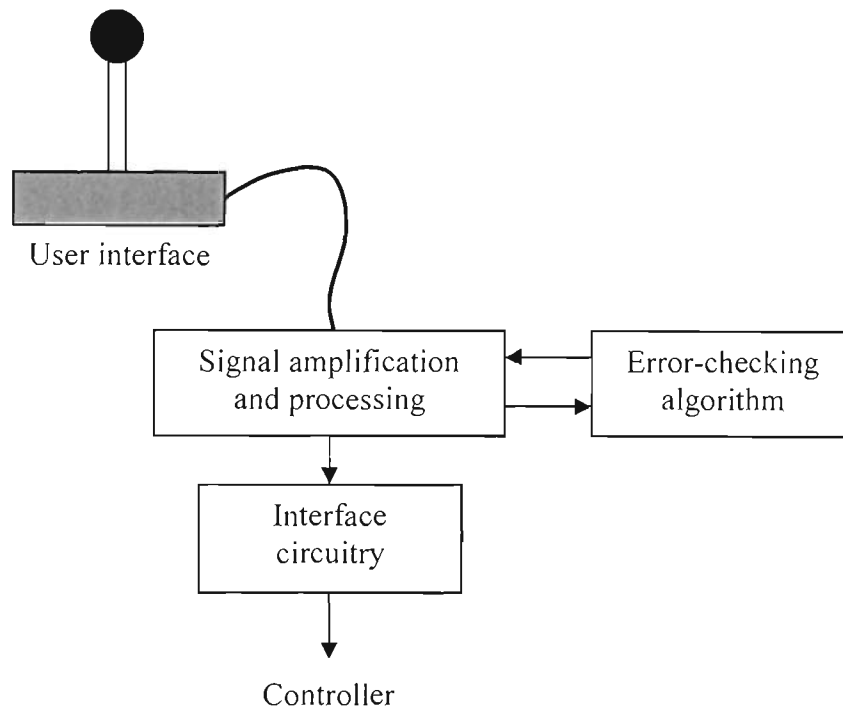


Figure A.3: Schematic of standard joystick with error-checking circuitry.

The joystick is usually mounted on the left or right armrest. Some are permanently fitted at the factory, while others can be changed to the other side quite easily. Many wheelchairs can also have the option of a joystick mounted at the back to enable an assistant to control the chair. This can replace or exist in addition to the armrest joystick. If the wheelchair has dual controls, a switch determines whether the wheelchair user or the assistant is driving the wheelchair.

Many wheelchair manufacturers can supply alternative controls. These may include mounted switches so that other parts of body can be used to control the wheelchair. These alternative switches make it possible for severely disabled people to control their wheelchairs independently. Centrally mounted controls may help to improve the body symmetry and help to promote a stable seating posture.

A.2.2 Wheelchair battery

A power wheelchair battery is different from a car battery because it is used in different ways. A power wheelchair battery is usually charged once a day and must supply a large amount of electricity while it is running. So a power wheelchair battery must be a special type, called *deep-cycle*. Such a battery can be almost completely discharged without damage to the battery.

The wheelchair battery can maintain operation longer if it is kept as close to full charge as practical. It is unlike a small nickel-cadmium battery that needs to be run down occasionally to preserve its full capacity. Most users need to charge the batteries every night to keep a full charge. Modern battery chargers are automatic so there is no problem of overcharging. If a wheelchair is sometimes used, the battery needs to be charged at least once a month and should be charged whenever it is used during the day.

There are two types of power wheelchair batteries: wet and gel. Wet batteries require adding distilled water about every two months. Wet batteries can be damaged permanently if the water level falls below the level of the battery plates. Gel batteries avoid many of the problems of wet batteries, for example, they never need water and can eliminate corrosion problems. However, the main disadvantage of gel batteries is that they have about 10 to 20% less capacity than comparable wet batteries and are more expensive than wet batteries.

A.2.3 Wheelchair motors

Permanent magnet DC (PMDC) motors, as shown in Figure A.4, are frequently the best solution to the motion control, where in compact size, wide operating speed range, ability to adapt to a range of power sources or the safety consideration of low voltage are important. Their ability to produce high torque at low speeds makes them suitable for power wheelchairs.

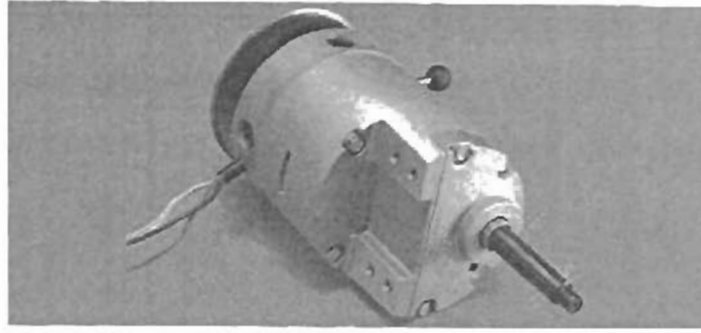


Figure A.4: PMDC motor for power wheelchairs (Shihlin Electric Manufactures).

Inside these motors, permanent magnets bounded to a flux-return ring replace the stator field windings found in shunt motors. The permanent magnets supply the surrounding field flux, eliminating the need for external field current. This design yields a smaller, lighter and energy-efficient motor.

The motor speed is controlled by adjusting the voltage applied to the armature. A feedback device senses the motor speed and sends this information to the controller for keeping the speed at or near the set value. Feedback techniques include voltage tachometers, optical encoders, and electromagnetic pulse generators and back electromotive force monitoring.

A number of magnetic materials are available for permanent magnets. These include iron and rare-earth magnets. Rare-earth magnets support much higher magnetic fields than iron magnets. Motors utilising rare-earth magnets are smaller and lighter than motors with iron magnets. In addition, rare-earth magnet motors are more powerful than motors with iron magnets of a similar size.

PMDC motors use a mechanical commutation scheme to switch the current to the armature winding. Commutator bars connect to the armature windings. A pair of spring loaded-brushes makes mechanical contact with the commutator bars, carrying the current to the armature. Thus, the brushes link the power source to the armature field windings. The armature commutator and the brushes act as a rotary switch for energising the windings.

Unlike a shunt-wound DC motor, a PMDC motor is free of interaction between the permanent magnet field and the armature demagnetising cross-field. The PM motor's

field has a high reluctance (low permeability). This high reluctance yields a constant field, permitting linear operation over the motor's entire speed - torque range. In operation with a constant armature voltage, as speed decreases, available torque increases.

A.2.4 Motor drive

The motor drive can be divided into two parts, as illustrated in Figure A.5. One part is the H-bridge configuration that may consist of power MOSFETs to provide the bidirectional rotation for the motor, and the other part is a signal-conditioning circuit between the microprocessor and the bridge.

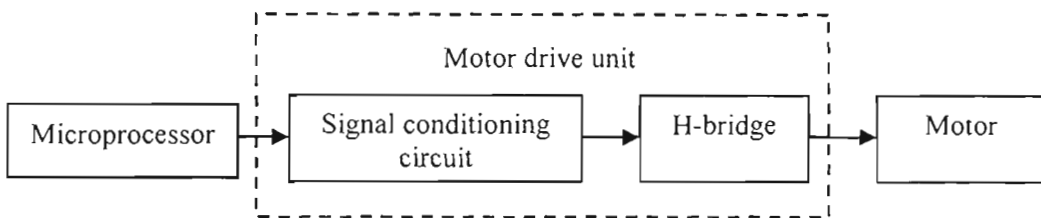


Figure A.5: Block diagram of the motor drive.

Power MOSFETs usually have a diode connected between the source and the drain for commutation. A bidirectional DC-DC converter can be built by combining nFETs and pFETs, as shown in Figure A.6.

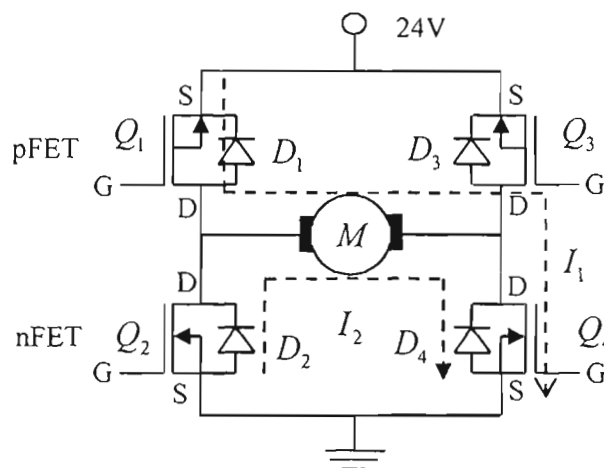


Figure A.6: H-bridge configuration provides the bidirectional rotation for the wheelchair motor.

In order to study the behaviour of the current flow, we consider Q_1 to be driven, Q_2 and Q_3 off and Q_4 on. When Q_1 is turned on, a current I_1 flows through the motor and Q_4 .

When Q_1 is turned off, the load current I_2 is commutated to D_2 . To provide rotation for the motor in the opposite direction, a similar type of operation can be performed on Q_3 with Q_1 and Q_4 off and Q_2 on. A variable drive to Q_1 or Q_3 is developed using a pulse-width modulation (PWM) technique.

A.2.5 Drive train

The drive train is a mechanical system that transfers rotational power from the motor to the drive wheels. There are two types of drive trains: direct transfer and indirect transfer.

- *Direct transfer*: the motor shaft is directly coupled to the drive wheel shaft. Direct drive transfer requires a low-speed, high-torque motor and is mechanically efficient. However, the gear in the direct drive system is prone to breakage and is expensive to repair.
- *Indirect transfer*: the motor is coupled to the drive wheel shaft through a system of gear train and flexible machine elements (belts or chains). The gear train and belt typically serve to reduce the motor speed while proportionally increasing the motor torque. They also act as a 'shock absorber' when the drive wheels are stuck or under heavy load. The cost of maintaining the system is relatively low if adjustments are made on a regular basis.

A.2.6 Control module

In power wheelchairs, the control module is responsible for converting the output signals from the joystick into the control signals for driving the wheelchair motors. The control module has many adjustable parameters.

Many wheelchair control systems have been designed with feedback for sensing whether the motor is responding properly to the joystick position. Such control systems are able to adjust the motor torque so as to maintain near constant speed while the load varies in response to changes in the terrain (incline, bumps) and surface (linoleum, carpeting, concrete, grass and sand). Adjustable controller parameters include high-to-low-speed range selection, maximum high speed, maximum low speed, maximum

acceleration, turning speed, tremor dampening and maximum power delivered under high torque and energy saver.

The most commonly controlled variable in power wheelchairs is speed. During normal operations, the wheelchair drive applies command inputs through a joystick and the operator's perception of the wheelchair's speed and direction. An electronic controller then adjusts the voltage to the DC motors to achieve the desired velocity for each motor. Figure A.7 shows a typical speed control algorithm using a tracking control to follow the speed profile set by the user, regardless of terrain or slope.

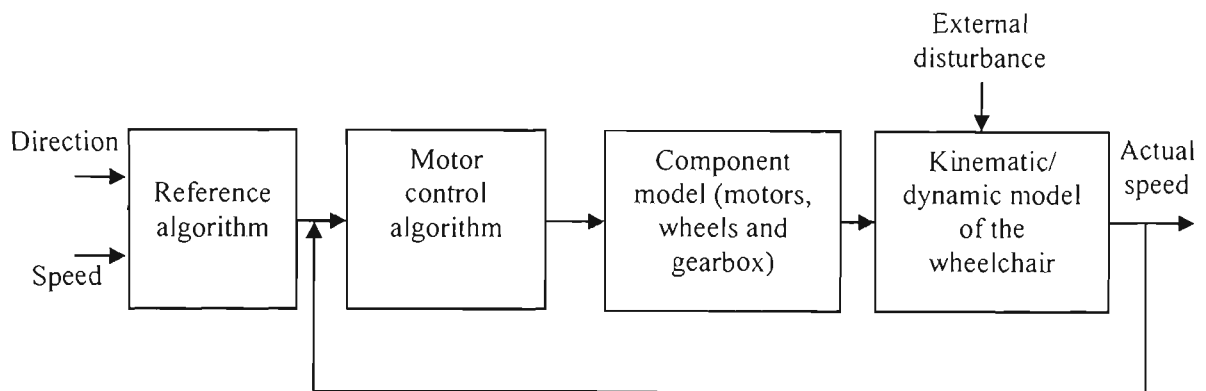


Figure A.7: Velocity control for power wheelchairs: the motor control algorithm uses drive commands (desired direction and speed signals) and feedback signals to develop control signals for the motor-drive electronics.

In order to obtain an optimal algorithm for controlling the speed of a power wheelchair, a model of the wheelchair and its critical components must be developed and used for controller implementation. The wheelchair's motion when driving on sloping surfaces can be simulated using computers to facilitate the design of a wheelchair velocity feedback controller.

Proportional, integral (PI) controllers are traditionally used for power wheelchair control. The control coefficients were selected in advance using known information about the wheelchair and its driver, thereby allowing the wheelchair to adapt to its driver. An optimal adaptive controller, which combines the modified proportional, integral and derivative (PID) control and the variable structure control (VSC), can also be used to improve the performance of the wheelchair system and allows the best

control coefficients to be automatically selected, according to the characteristics of the chair and its driver.

A single microcontroller is commonly used to control the wheelchair and both of its motors. Microcontrollers from the INTEL 8000 or Motorola 6800 family are popularly used. Microcontrollers offer flexible control and accommodation of various input devices (both analog and digital). An internal (resident) program maintains timing and contains the control system algorithms.

Appendix B

ADXL202EB-232A Evaluation Board

B.1 Hardware

The ADXL202EB-232A evaluation board, as shown in Figure B.1, is integrated from an ADXL202 dual axis-accelerometer and a Microchip 16C63 microcontroller. The maximum sampling rate of the board is 275Hz. The board is powered by an extra control signal, RTS, on the RS-232 port.

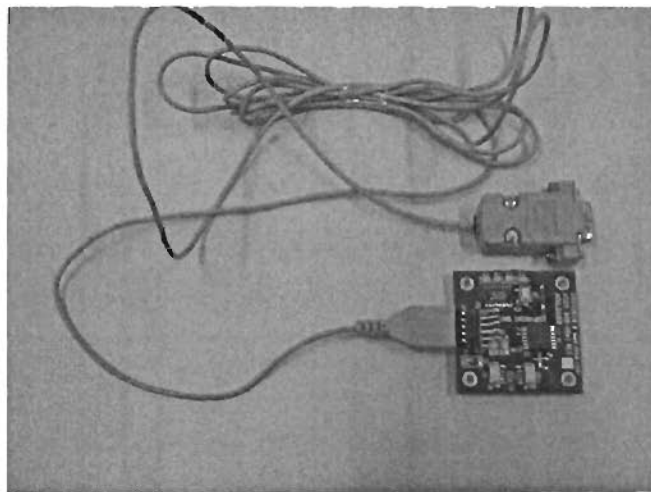


Figure B.1: ADXL202EB-232A evaluation board with RS232 connection cable.

The microcontroller PIC16C63 is configured to read the Pulse Width Modulation (PWM) signals from the ADXL202 accelerometer, and process and convert them to a serial protocol for communication to a PC RS-232 port of PC.

B.2 Software

The *Crossbow* software is provided to read the RS-232 port and display the real-time acceleration data. A data-logging feature is also included, so that data can be stored for the later evaluation by a spreadsheet or other tools.

B.3 Applications

The ADXL202EB-232A evaluation board can be used for the following applications:

- Earthquake detectors
- Alarms and motion detectors
- Battery-powered motion-sensing
- Dual-axis tilt-sensing with faster response than electrolytic, mercury or thermal sensors
- Data-logging
- Instrumentation.

Appendix C

GPSB Wheelchair Direction Control Device

The direction of a power wheelchair can be controlled by using the GPSB device, as shown in Figure C.1. This device is designed to accept the direction and speed control signals fed to it. Table C.1 shows the specifications of the device. Figure C.2 shows the functional diagram for wheelchair direction and speed control using the GPSB device.

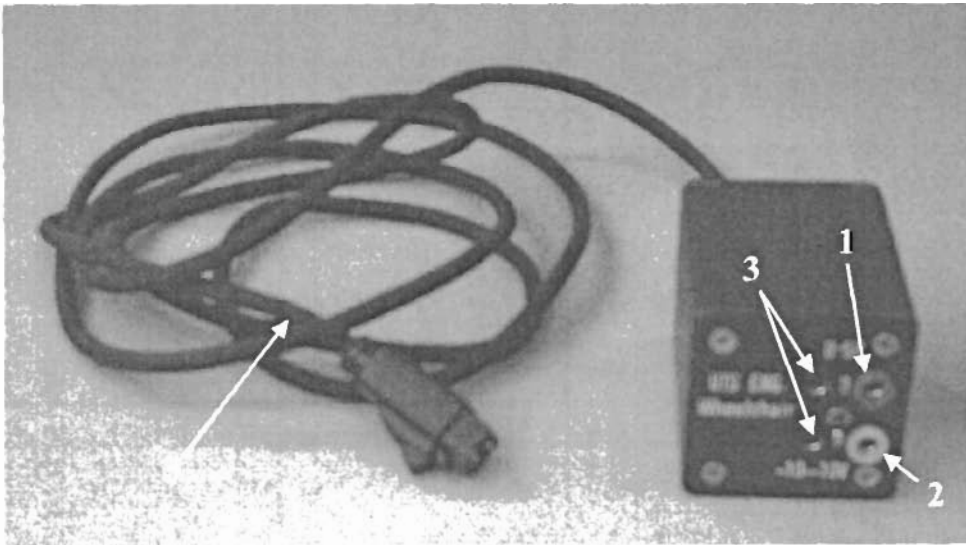


Figure C.1: GPSB device: 1) the direction control input, 2) the speed control pin, 3) ground, and 4) the cable for the connection to the main processing units of the wheelchair.

Table C.1: Specifications of the GPSB device.

Direction control		Speed control	
Input signal	Function	Input signal	Function
2.5 V	Neutral	2.5 V	Neutral
2.5V + 1.7V (4.2V)	Full right turn	2.5V + 1.32V (3.82V)	Full forward speed
2.5V - 1.7V (0.8V)	Full left turn	2.5V - 1.57V (0.93V)	Full reverse speed

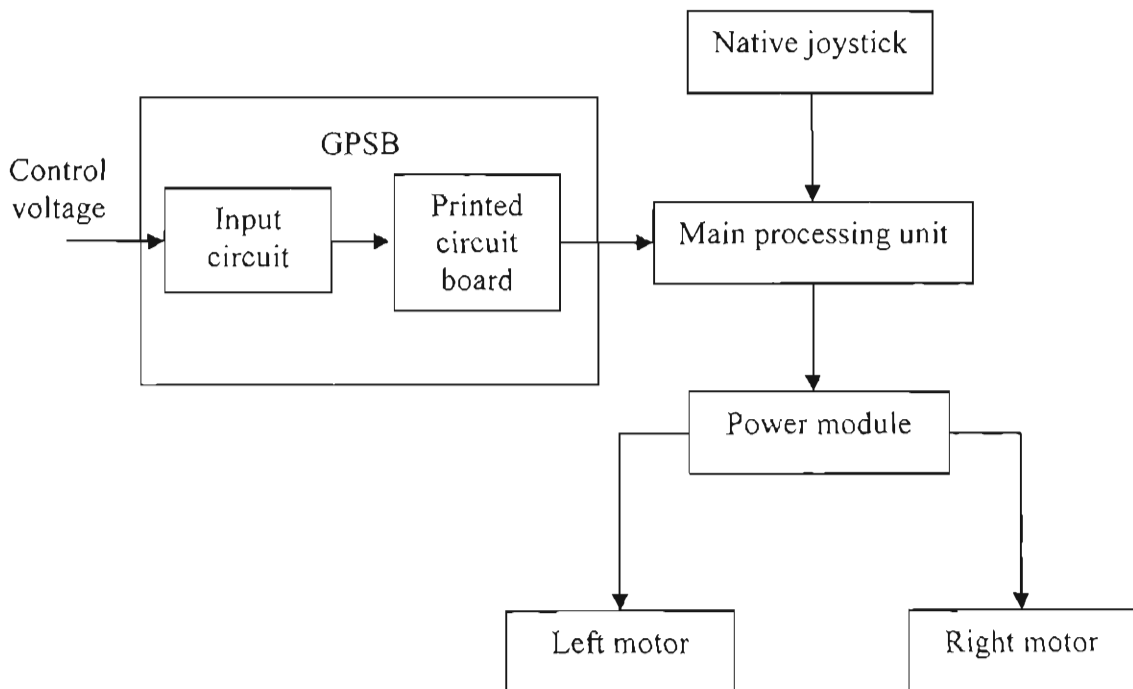


Figure C.2: The functional diagram of wheelchair direction and speed control using the GPSB device.

Appendix D

National Instrument USB-6008 Multifunction Data Acquisition Module

D.1 Hardware

The National Instruments USB-6008 multifunction data-acquisition module with hardware and software can allow us to perform the reliable data-acquisition tasks. With plug-and-play USB connectivity, the device is simple enough for quick measurements, but comprehensive enough for more complex measurement applications. The hardware specifications of the device are:

- 8 true analog inputs with 12-bit input resolution
- 48 kS/s sampling rate
- Built-in, removable connectors for easier and more cost-effective connectivity
- 2 true analog outputs
- 12 digital I/O lines
- 32-bit event counter.

Figure D.1 illustrates the key functional components of the NI USB-6008. The signal label application diagram shown in Figure D.2 is for setting up the hardware. Table D.1 is a list of the analog terminal assignments and Table D.2 a list of the digital terminal assignments. The signals available on the I/O connectors are described in Table D.3.

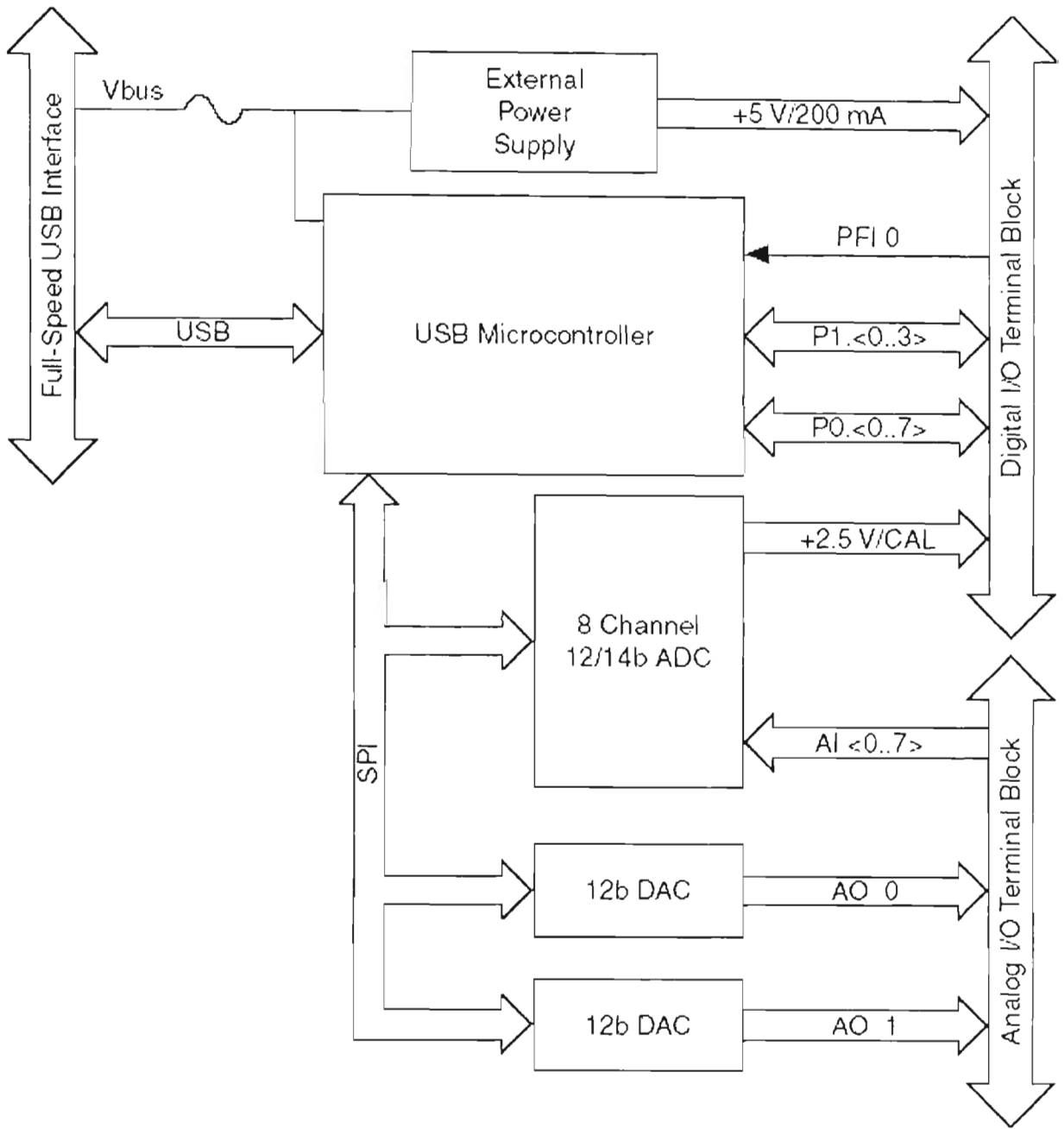


Figure D.1: Key functional components of the NI USB-6008.

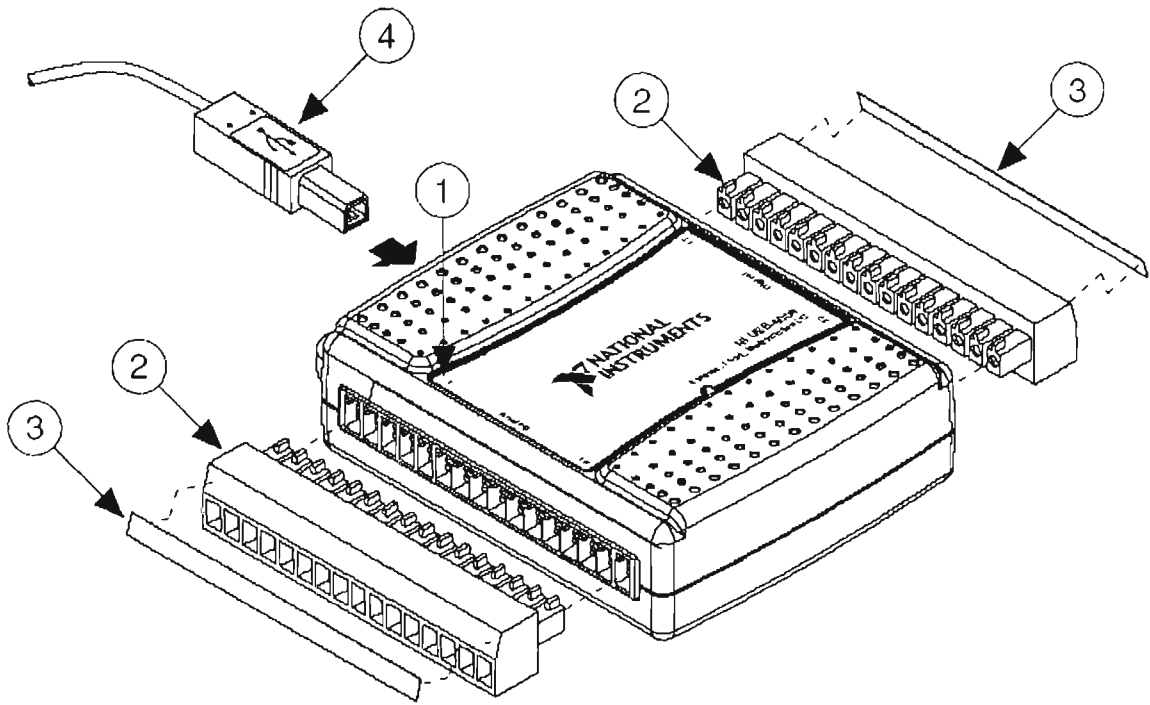


Figure D.2: Signal label application diagram of the NI USB-6008: 1) Overlay label with pin orientation guides, 2) Removable I/O connectors, 3) Signal labels, and 4) USB cable.

Table D.1: Analog terminal assignments of the NI USB-6008.

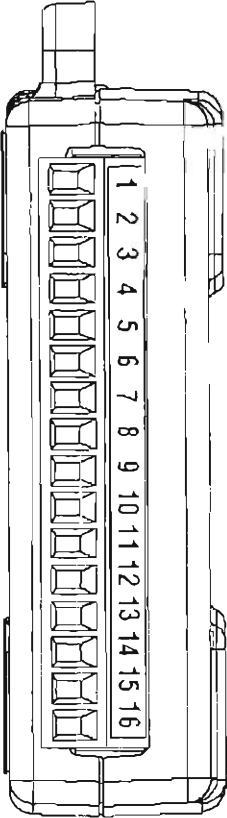
Module	Terminal	Signal, single-ended mode	Signal, differential mode
	1	GND	GND
	2	AI0	AI0+
	3	AI4	AI0-
	4	GND	GND
	5	AI1	AI1+
	6	AI5	AI1-
	7	GND	GND
	8	AI2	AI2+
	9	AI6	AI2-
	10	GND	GND
	11	AI3	AI3+
	12	AI7	AI3-
	13	GND	GND
	14	AO0	AO0
	15	AO1	AO1
	16	GND	GND

Table D.2: Digital terminal assignments of the NI USB-6008.

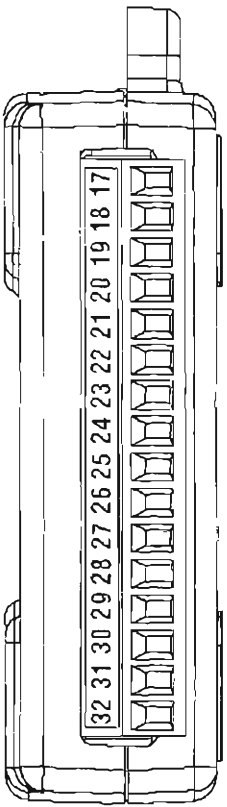
Module	Terminal	Signal
 <p>The diagram shows a side view of the NI USB-6008 module. A vertical strip of 16 terminals is labeled with numbers 17 through 32. Terminal 17 is at the top, and terminal 32 is at the bottom. The labels are oriented vertically along the strip.</p>	17	P0.0
	18	P0.1
	19	P0.2
	20	P0.3
	21	P0.4
	22	P0.5
	23	P0.6
	24	P0.7
	25	P1.0
	26	P1.1
	27	P1.2
	28	P1.3
	29	PF1 0
	30	+2.5V
	31	+5V
	32	GND

Table D.3: The signals available on the I/O connectors of the NI USB-6008.

Signal name	Reference	Direction	Description
GND	—	—	Ground: the reference point for the single-ended AI measurements, bias current return point for differential mode measurements, AO voltages, digital signal at the I/O connector, +5VDC supply, and the +2.5 VDC reference.
AI<0..7>	Varies	Input	Analog input channels 0-7: for single-ended measurements, each signal is an analog input voltage channel. For differential measurements, AI0 and AI4 are the positive and negative inputs of different analog input channel 0. The following signal pairs also form different input channels: <AI1, AI5>, <AI2, AI6>, and <AI3, AI7>.
AO0	GND	Output	Analog channel 0 output: supplying the voltage output of A0 channel 0.
AO1	GND	Output	Analog channel 1 output: supplying the voltage output of A1 channel 1.
P1.<0..3> P0.<0..7>	GND	Input or output	Digital I/O signals: for individually configuring each signal as an input or output.
+2.5V	GND	Output	+2.5V external reference: providing a reference for wrap-back setting.
+5V	GND	Output	+5V power source: providing +5V power up to 200 mA.
PFI 0	GND	Input	PFI 0: this pin is configurable as either a digital trigger or an event counter input.

D.2 Software

The NI USB-6008 hardware is used with the NI-DAQmx high-performance, multithreaded driver software for interactive configuration and data acquisition on Windows operating systems. The software can also be used to develop customised data acquisition applications with National Instruments LabVIEW or C-based development environments.

D.3 Applications

The NI USB-6008 module is ideal for a number of applications where in economy, small size, and simplicity are essential, such as:

- Data logging
- Academic lab use
- Embedded applications.

Appendix E

The Advanced Bayesian Neural Network Classification Toolbox for Matlab

The Advanced Bayesian Neural Network Classification Toolbox for Matlab has been developed by the author to perform the functions and algorithms described in this thesis. The toolbox is independent of the Neural Network Toolbox in Matlab.

The main features of the toolbox are:

1. Advanced training algorithms for two-layer perceptron classifiers:
 - Conjugate gradient
 - Scaled conjugate gradient
 - Quasi-Newton
 - Adaptive learning rate gradient descent
2. Bayesian inference of the regularisation parameters (hyperparameters):
 - The Laplace method
 - Fast evaluation of the Hessian matrix
3. Maximum evidence-early stopping:
 - The Laplace method
 - The Monte Carlo method with the Metropolis-Hastings algorithm
4. Combining the independent BNN classifiers:
 - Bayes' rule-based method
 - Dempster-Shafer theory-based method

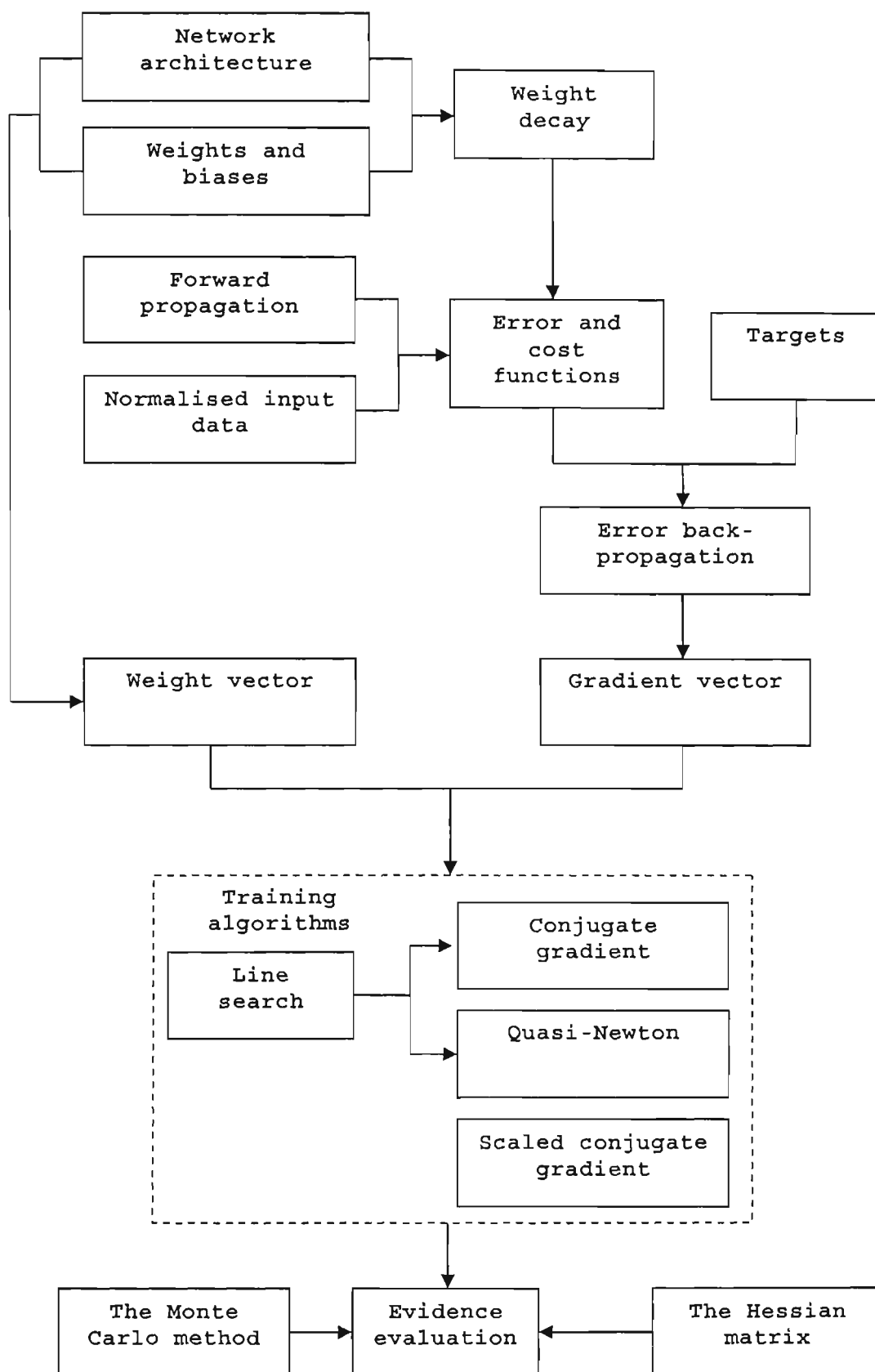


Figure E.1: Structure of the Advanced Bayesian Neural Network Classification Toolbox for Matlab.

Table E.1: List of functions in the toolbox.

No	function	Purpose	Remark
1	netini	Initialise weights and biases using the standard Gaussian	Chapter 4
2	m2v	Convert matrices of weights and biases into a single vector	Chapter 3
3	v2m	Convert the weight vector into matrices of weights and biases	Chapter 3
4	hypers	Assign prior hyperparameters to groups of weights and biases	Chapter 3
5	netfwd	Do forward propagation	Chapter 3
6	netback	Do error-backward propagation	Chapter 3
7	grads	Calculate the gradient	Chapter 3
8	funcs	Calculate the cost, error and weight functions	Chapter 3
9	dhessa	Approximated multiplication of the Hessian matrix	Chapter 3
10	dhesse	Exact multiplication of the Hessian matrix	Chapter 3
11	hessa	Approximated evaluation of the Hessian matrix	Chapter 3
12	hesse	Exact evaluation of the Hessian matrix	Chapter 3
13	linemin	Line minimisation	Chapter 4
14	sgdtrain	Standard gradient descent training algorithm (Fixed learning rate)	Chapter 4
15	gdtrain	Gradient descent training algorithm (Adaptive learning rate)	Chapter 4
16	cgtrain	Conjugate gradient training algorithm	Chapter 4
17	scgtrain	Scaled conjugate gradient training algorithm	Chapter 4
18	qntrain	Quasi-Newton training algorithm	Chapter 4
19	rehypers	Re-estimate the hyperparameters	Chapter 4
20	logevi	Evaluate the log evidence	Chapter 5
21	confm	Calculate the confusion matrix	Chapter 4, 5
22	senspe	Calculate the sensitivity and specificity	Chapter 4, 5
23	bcombin	Combine performances of 2 independent classifiers using Bayes' theorem	Chapter 6
24	dcombin	Combine performances of 2 independent classifiers using Dempster's rule	Chapter 6
25	mvgd	Generate data using Multivariate Gaussian Distributions	Chapter 6
26	mgdinit	Initialise weights and biases using Multivariate Gaussian Distributions	Chapter 6
27	logevlap	Evaluate the log evidence every cycle with the Laplace method	Chapter 6
28	logevmc	Evaluate the log evidence every cycle with the Monte Carlo methods	Chapter 6
29	scglaptrain	Scaled conjugate gradient training and evaluating the evidence every cycle (Laplace method)	Chapter 6
30	scgmctrain	Scaled conjugate gradient training and evaluating the evidence every cycle (Monte Carlo method)	Chapter 6

Table E.2: List of the demonstrations in the toolbox.

No	File	Purpose	Remark
1	demo1	Experiment 1 in Chapter 4	Chapter 4
2	demo2	Experiment 2 in Chapter 4	Chapter 4
3	demo3	Experiment 1 in Chapter 5	Chapter 5
4	demo4	Experiment 2 in Chapter 5	Chapter 5
5	demo5	Experiment 2.1 in Chapter 6	Chapter 6
6	demo6	Experiment 2.2 in Chapter 6	Chapter 6
7	demo7	Experiment 2.3 in Chapter 6	Chapter 6
8	demo8	Change the network architecture and then evaluate the evidence	Chapter 5
9	demo9	Plot the cycle evidence	Chapter 6
10	demo10	Demonstration of maximum evidence early stopping with the Laplace method	Chapter 6
11	demo11	Demonstration of initialising weights and biases using Nguyen and Widrow's method	Chapter 6
12	demo12	Demonstration of the Monte Carlo integration method	Chapter 6
13	demo13	Demonstration of the Metropolis sampling method	Chapter 6
14	demo14	Demonstration of maximum evidence early stopping with the Monte Carlo methods	


```

function net = netini(nin, nhidden, nout);

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Network architecture
net.type = 'MLP Classifier';           % Network type
net.nin = nin;                         % Number of input nodes
net.nhidden = nhidden;                 % Number of hidden nodes
net.nout = nout;                       % Number of output nodes

% Number of all the weights and biases
net.ws = (nin + 1)*nhidden + (nhidden + 1)*nout;

% Initialise weights and biases with Gaussian distributions
net.w1 = randn(nin, nhidden);
net.b1 = randn(1, nhidden);
net.w2 = randn(nhidden, nout);
net.b2 = randn(1, nout);

% Index matrix of weights and biases in groups;
net.index = zeros(net.ws, 4);
m1 = nin*nhidden;
net.index(1:m1, 1) = 1;
m2 = m1 + nhidden;
net.index(m1 + 1: m2, 2) = 1;
m3 = m2 + nhidden*nout;
net.index(m2 + 1: m3, 3) = 1;
m4 = m3 + nout;
net.index(m3 + 1: m4, 4) = 1;

```

```

function w = m2v(net);

% Convert matrix of weights and biases to a single vector

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Convert matrix of weights on connections from input nodes
% to hidden nodes into the first section of weight vector
m = size(net.w1,1);
n = size(net.w1,2);
k = 0;
for i = 1:n
    for j = 1:m
        k = k + 1;
        w1(k) = net.w1(j,i);
    end
end

% Convert vector of biases of hidden nodes into the
% second section of weight vector
m = size(net.b1,1);
n = size(net.b1,2);
k = 0;
for i = 1:n
    for j = 1:m
        k = k + 1;
        b1(k) = net.b1(j,i);
    end
end

% Convert matrix of weights on connections from hidden nodes
% to output nodes into the third section of weight vector
m = size(net.w2,1);
n = size(net.w2,2);
k = 0;
for i = 1:n
    for j = 1:m
        k = k + 1;
        w2(k) = net.w2(j,i);
    end
end

% Convert vector of biases of output nodes into the
% fourth section of weight vector
m = size(net.b2,1);
n = size(net.b2,2);

```

```
k = 0;
for i = 1:n
    for j = 1:m
        k = k + 1;
        b2(k) = net.b2(j,i);
    end
end

% Group sections into a vector
w = [w1 b1 w2 b2];
```

```

function net = v2m(net, w);
% Convert weight vector into matrices of weights and biases

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

nin = net.nin;
nhidden = net.nhidden;
nout = net.nout;

m1 = nhidden*nin;
m2 = m1 + nhidden;
m3 = m2 + nout*nhidden;
m4 = m3 + nout;

% Return matrix of weights on connections from
% input nodes to hidden nodes
for i = 1:nhidden
    k1 = (i - 1)*nin;
    k2 = 0;
    while (k1 < (i*nin))
        k1 = k1 + 1;
        k2 = k2 + 1;
        net.w1(k2,i) = w(k1);
    end
end

% Return vector of biases of hidden nodes
k1 = m1;
k2 = 0;
while k1 < m2
    k1 = k1 + 1;
    k2 = k2 + 1;
    net.b1(k2) = w(k1);
end

% Return matrix of weights on connections from
% hidden nodes to output nodes
for i = 1:nout
    k1 = m2 + (i - 1)*nhidden;
    k2 = 0;
    while k1 < (m2 + i*nhidden)
        k1 = k1 + 1;
        k2 = k2 + 1;
        net.w2(k2,i) = w(k1);
    end
end
end

```

```
% Return vector of biases of output nodes
k1 = m3;
k2 = 0;
while k1 < m4
    k1 = k1 + 1;
    k2 = k2 + 1;
    net.b2(k2) = w(k1);
end
```

```
function net = hypers(net, cw1, cb1, cw2, cb2);
% Hyperparameters for weight and bias groups

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Hyperparameter for weights on connections from input nodes to
% hidden nodes
net.cw1 = cw1;

% Hyperparameter for biases of hidden nodes
net.cb1 = cb1;

% Hyperparameter for weights on connections from hidden nodes to
% output nodes
net.cw2 = cw2;

% Hyperparameter for biases of output nodes
net.cb2 = cb2;

% Vector of hyperparameters
net.c = [cw1 cb1 cw2 cb2];
```

```
function [z, y, a] = netfwd(net, x);

% Forward propagation

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Number of training samples
n = size(x,1);

% Outputs of hidden nodes
y = tanh(x*net.w1 + ones(n, 1)*net.b1);

% Activations of output nodes
a = y*net.w2 + ones(n, 1)*net.b2;

% Outputs of the classifier
temp = exp(a);
z = temp./(sum(temp, 2)*ones(1, net.nout));
```

```
function g = netback(net, x, y, delout);

% Error back-propagation

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Derivatives of data error with respect to output-layer weights
gw2 = y'*delout;

% Derivatives of data error with respect to bises of output nodes
gb2 = sum(delout, 1);

% Error back-propagation
delhid = (delout*net.w2').*(1.0 - y.*y);

% Derivatives of data error with respect to input-layer weights
gw1 = x'*delhid;
gb1 = sum(delhid, 1);

% Group derivative components into a single vector
g = [gw1(:)', gb1, gw2(:)', gb2];
```



```
function [gs, ge] = grads(net, x, t);
% Calculate gradient vectors of cost function and data function

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Forward propagation
[z, y, a] = netfwd(net, x);

% Output errors
delout = z - t;

% Gradient vector of data error function
ge = netback(net, x, y, delout);

% Matrix of hyperparameters in different weight and bias groups
for i = 1:4
    net.hypers(:,i) = net.c(i)*net.index(:,i);
end

% Gradient vector of weight decay
w = m2v(net);
gwd = zeros(1, size(w, 2));
for i = 1:4
    gwd = gwd + (w.*(net.hypers(:,i)))';
end

gs = ge + gwd;
```

```

function dh = dhesse(net, x, t, d);
% Fast multiplication by the Hessian (Exact method)

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Calculate outputs of layer-hidden and output nodes
[z, y, a] = netfwd(net, x);

% g1 = dy/dal, g1 = d(g1)/dal
g1 = 1 - y.*2;
g11 = -2*y.*g1;
dnet = v2m(net, d);
n = length(x);

% The {R}-operator technique
ra1 = x*dnet.w1 + ones(n, 1)*dnet.b1; % R{a1}
ry = g1.*ra1; % R{y};
ra2 = y*dnet.w2 + ry*net.w2 + ones(n, 1)*dnet.b2; % R{a2}
c = size(t, 2); % Number of output classes
rz = z.*ra2 - z.*(sum(z.*ra2, 2)*ones(1, c)); % R{z}

% Output errors
delout = z - t;

delhid = g1.*(delout*net.w2'); % Error signals for the hidden layer
rdelhid = g11.*ra1.*(delout*net.w2') + ...
          g1.*(delout*dnet.w2') + g1.*(rz*net.w2');

rew1 = x'*rdelhid;
reb1 = sum(rdelhid, 1);
rew2 = y'*rz + ry'*delout;
reb2 = sum(rz, 1);

% The product of vector d and the Hessian (dh)
dh = [rew1(:)', reb1, rew2(:)', reb2];

```

```

function [f, fd, fw] = funcs(net, x, t);
% Calculate cost function, data function, weight function

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

[z, y, a] = netfwd(net, x);

% Data error
delta = 1e-100;
z = z + delta; % to avoid log of zero
fd = -sum(sum(t.*log(z))); % entropy function

% Vector of weight and bias errors
w = m2v(net);
fw = 0.5*w.^2*net.index;

% Weight decay
wd = 0;
for i = 1:4
    wd = wd + net.c(i)*fw(i);
end

uplimit = 1.0e+5; % Upper bound on functions
if fd >= uplimit
    fd = uplimit;
end

if fw >= uplimit
    fw = uplimit;
end

% Cost function
f = fd + wd;

```

```
function dh = dhessa(net, x, t, d);
% Fast multiplication by the Hessian (approximation method)

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Calculate gradient at (w)
w = m2v(net);
[gs ge] = grads(net, x, t);

% Calculate sigma = sigma0/||d||
sigma0 = 1.0e-6;
sigma = sigma0/sqrt(d*d');

% Calculate gradient at (w + sigma*d)
wplus = w + sigma*d;
netplus = v2m(net, wplus);
[gs_plus ge_plus] = grads(netplus, x, t);

% The product of vector d and the Hessian (dh)
dh = (ge_plus - ge)/sigma;
```

```
function h = hesse(net, x, t);
% Return the full Hessian (hess) from H*d and identity matrix
% (exact method)

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

I = eye(net.ws);    % The ws-by-ws indentity matrix
h = zeros(net.ws, net.ws);
for i = 1:(net.ws)
    d = I(i,:);
    dh = dhesse(net, x, t, d);
    h(i, :) = h(i, :) + dh;
end
```

```
function h = hessa(net, x, t);
% Return the full Hessian (hess) from H*d and identity matrix
% (approximation method)

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

I = eye(net.ws);    % The ws-by-ws identity matrix
h = zeros(net.ws, net.ws);
for i = 1:(net.ws)
    d = I(i,:);
    dh = dhessa(net, x, t, d);
    h(i, :) = h(i, :) + dh;
end
```

```

function [net, a, b, c, fmin, bmin] = linemin(net, x, t, dir, nsteps);
% Line minimisation

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

phi = 0.5*(1 + sqrt(5)); % Golden ratio
theta = 1 - 1/phi;

w0 = m2v(net);          % Weight vector
net0 = net;

% Initial bracketing triple {a b c}
a = 0;
c = 0.5;
b = 0.5*(c + a);
x1 = a; % Initial trial point

i = 1;

while (i <= nsteps)

    x1 = c - theta*(c - a);

    wx1 = w0 + x1*dir;
    net = v2m(net0, wx1);
    fx1 = funcs(net, x, t);

    wa = w0 + a*dir;
    net = v2m(net0, wa);
    fa = funcs(net, x, t);

    wb = w0 + b*dir;
    net = v2m(net0, wb);
    fb = funcs(net, x, t);

    wc = w0 + c*dir;
    net = v2m(net0, wc);
    fc = funcs(net, x, t);

    % If x > b then the new bracketing triple is {a, b, xn}
    % if fx > fb and is {b, xn, c} if fd < fb
    if (x1 > b)
        if (fx1 > fb)
            c = x1;
        end
    end
end

```

```
        if (fx1 < fb)
            a = b;
            b = x1;
        end
    end
end
% If x < b then the new bracketing triple is {x, b, c}
% if fx > fb and is {a, xn, b} if fx < fb
if (x1 < b)
    if (fx1 > fb)
        a = x1;
    end
    if (fx1 < fb)
        b = x1;
        c = b;
    end
end
end

i = i + 1;

end

bmin = b;
wbmin = w0 + bmin*dir;
net = v2m(net0, wbmin);
fmin = funcs(net, x, t);
```



```

function [net, f, fc, cycle]= cgtrain(net, x, t, train_steps, search_steps);
% Conjugate gradient training algorithm

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
ncycles = 100; % Number of training cycle
fnew = funcs(net, x, t);
gnew = grads(net, x, t); % Initial gradient
d = -gnew;

i = 1;
% Main loop
while (i < train_steps)

    wold = w;
    fold = fnew;
    gold = gnew;

    gg = gold*gold';
    if (gg == 0) % Network training is done
        f = fnew;
        return;
    end;

    if (gnew*d' > 0) % Search direction is uphill in conjugate
        d = -d;
    end

    % Line minimisation
    dline = d/norm(d);
    [net, a, b, c, fmin, alpha] = linemin(net, x, t, dline, search_steps);
    w = w + alpha*dline; % Update weight vector

    net = v2m(net, w);
    f = funcs(net, x, t); % Evaluate cost function

    % Update search direction using Polak - Ribiere formula
    gnew = grads(net, x, t);
    beta = ((gnew - gold)*(gnew)')/gg;
    d = -gnew + (d.*beta);

    % Display cycle error
    cycle(i) = i; % Training cycle
    ss(i) = alpha; % Step size
    fc(i) = f; % Function value

```

```
fprintf(1, ' Cycle: %4d    Function: %11.6f    Step size: %11.6f\n', ...  
        i, fc(i), ss(i));
```

```
i = i + 1;
```

```
end
```

```

function [net, f, fc, cycles]= scgtrain (net, x, t, train_steps, search_steps);
% Scaled conjugate gradient training algorithm

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

sigma0 = 1e-4;
fold = funcn(net, x, t);
f = fold;
gold = grads(net, x, t); % Initial gradient
gnew = gold;
d = -gnew; % Initial search direction
success = 1;
lamda = 1.0; % Initial scale parameter
lamdaold = lamda;
lamin = 1.0e-100; % Lower bound on scale
lamax = 1.0e+100; % Upper bound on scale

w = m2v(net); % Initial weight vector
ncycles = 100; % Number of training cycles
nwts = net.ws; % Number of network parameters
tolerance = 10e-3;

i = 1; % Count number of training cycles
% Main loop
while ((f >= tolerance)&&(i <= train_steps))

    if (success == 1)
        mu = d*gnew';
        if (mu >= 0)
            d = -gnew;
            mu = d*gnew';
        end
        dd = d*d';
        if dd < eps % if d is too small, it is done
            f = fnow;
            return;
        end
        sigma = sigma0/sqrt(dd);
        wplus = w + sigma*d;
        net = v2m(net, wplus);
        gplus = grads(net, x, t);
        theta = (d*(gplus' - gnew'))/sigma;
    end

    % Make delta always positive and evaluate step size
    delta = theta + lamda*dd;

```

```

if (delta <= 0)
    delta = -delta + lamda*dd;
    lamda = 2*(lamda - theta/dd);
end
alpha = -mu/delta;

% Update weights and evaluate cost function
wnew = w + alpha*d;
net = v2m(net, wnew);
fnew = funcs(net, x, t);

% Calculate comparison ratio (compars)
compars = 2*(fnew - fold)/(alpha*mu);
if (compars >= 0)
    success = 1;
    w = wnew;
    f = fnew;
else
    success = 0;
    f = fold;
end

if (success == 1)
    if (max(abs(alpha*d)) < 1.0e-5)...
        &&(max(abs(fnew - fold)) < 1.0e-5)
            f = fnew;
            return;
        else
            % Prepare parameters for the next cycle
            fold = fnew;
            gold = gnew;
            net = v2m(net, w);
            gnew = grads(net, x, t);
            % If the gradient is zero, then training is done
            if (gnew*gnew' == 0)
                f = fnew;
                return;
            end
        end
    end
end

% Update scale parameter based on comparison ratio
if (compars < 0.25)
    lamda = min(4*lamda, lamax);
end
if (compars > 0.75)
    lamda = max(0.5*lamda, lamin);
end

% Update search direction based on Polak-Ribiere recipe
if (success == 1)

```

```
gnew = grads(net, x, t);
beta = ((gnew - gold)*(gnew)')/mu;
d = -gnew + (d.*beta);
end

% Display cycle error on screen
cycles(i) = i;
fc(i) = f;
scale(i) = lamda;
fprintf(1, ' Cycle: %4d      Function: %11.6f      Scaler: %11.6f\n', ...
        i, fc(i), lamda);

i = i + 1;

end
```

```

function [net, f]= qntrain(net, x, t, train_steps, search_steps);
% Quasi-Newton training algorithm

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
fnew = funcs(net, x, t);
gnew = grads(net, x, t); % Initial gradient
frac = 1.0e-2;
nwts = net.ws; % Number of network parameters
F = eye(nwts); % Initial matrix F
p = -gnew;

i = 1;
% Main loop
while (i <= train_steps)

    wold = w;
    fold = fnew;
    gold = gnew;

    w = wold + p;
    net = v2m(net, w);
    p = -gnew;

    if (gnew*p' >= 0)
        p = -p;
    end

    % If Newton step did not reduce function significantly,
    % then perform line search
    if (fnew > (fold + frac*(gnew*p')))
        [net, a, b, c, fmin, alpha] = linemin(net, x, t, p, search_steps);
        w = wold + alpha*p;
    end

    % Prepare parameters for the next cycle
    net = v2m(net, w);
    fnew = funcs(net, x, t);
    f = fnew;
    gnew = grads(net, x, t);

    p = w - wold;
    v = gnew - gold;
    vp = v*p';

```

```
% Use the BFGS method to update F
napla = eps*sum(v.^2)*sum(v.^2);
if (vp*vp < napla)
    Fv = (F*v)';
    vFv = sum(v.*Fv);
    u = p/vp - Fv/vFv;
    F = F + (p'*p)/vp - (Fv'*Fv)/vFv + vFv*(u'*u);
end

p = -(F*gnew)';

% Display cycle error on screen
fc(i) = f;
ss(i) = alpha ;
fprintf(1, ' Cycle: %4d    Function: %11.6f    Step size: %11.6f \n',...
        i, fc(i), ss(i));

i = i + 1;
end
```

```

function [net, f]=gdtrain(net, x, t, train_steps, search_steps);
% Adaptive learning rate gradient descent training algorithm

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
g = grads(net, x, t);
d = -g;

i = 1;

% Main loop
while (i <= train_steps)

    g = grads(net, x, t);
    d = -g;

    % Line search
    [net, a, b, c, fmin, alpha] = linemin(net, x, t, d, search_steps);
    w = w + alpha*d; % Update weight vector

    net = v2m(net, w);
    f = funcs(net, x, t); % Evaluate cost function

    % Print cycle error on screen
    ss(i)= alpha;
    fc(i) = f;
    fprintf(1, ' Cycle: %4d    Function: %11.6f    Step size: %11.6f\n',...
            i, fc(i), ss(i));

    i = i + 1;

end

```



```

function [net, f] = sgdtrain(net, x, t, train_steps, search_steps);
% Standard gradient descent training algorithm

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
g = grads(net, x, t);
d = -g;

i = 1;

% Main loop
while (i <= train_steps)

    g = grads(net, x, t);
    d = -g; % Search direction
    alpha = 0.01; % Learning rate
    w = w + alpha*d; % Update weight vector
    net = v2m(net, w);
    f = funcs(net, x, t); % Evaluate cost function

    % Print cycle error on screen
    ss(i)= alpha;
    fc(i) = f;
    fprintf(1, ' Cycle: %4d   Function: %11.6f   Step size: %11.6f\n',...
            i, fc(i), ss(i));

    i = i + 1;

end

```

```

function [net, A] = rehypers(net, x, t);
% Re-estimate hyperparameters;

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

H = hessa(net, x, t);
index = net.index;

I = eye(net.ws);

Iw1 = I;
for i = 1:net.ws
    for j = 1:net.ws
        temp = index(:,1);
        Iw1(i,i) = Iw1(i,i)*temp(i);
    end
end

Ib1 = I;
for i = 1:net.ws
    for j = 1:net.ws
        temp = index(:,2);
        Ib1(i,i) = Ib1(i,i)*temp(i);
    end
end

Iw2 = I;
for i = 1:net.ws
    for j = 1:net.ws
        temp = index(:,3);
        Iw2(i,i) = Iw2(i,i)*temp(i);
    end
end

Ib2 = I;
for i = 1:net.ws
    for j = 1:net.ws
        temp = index(:,4);
        Ib2(i,i) = Ib2(i,i)*temp(i);
    end
end

cw1 = net.cw1;
cb1 = net.cb1;
cw2 = net.cw2;
cb2 = net.cb2;

```

```
A = H + cw1*Iw1 + cb1*Ib2 + cw2*Iw2 + cb2*Ib2;
theta = (1.0e-10)*eye(net.ws, net.ws);
A = A + theta;
invA = inv(A);

kw1 = sum(index(:,1));
kb1 = sum(index(:,2));
kw2 = sum(index(:,3));
kb2 = sum(index(:,4));

[f, fd, fw] = funcs(net, x, t);

gam_w1 = abs(kw1 - cw1*trace(invA*Iw1));
gam_b1 = abs(kb1 - cb1*trace(invA*Ib1));
gam_w2 = abs(kw2 - cw2*trace(invA*Iw2));
gam_b2 = abs(kb2 - cb2*trace(invA*Ib2));

net.gw1 = gam_w1;
net.gb1 = gam_b1;
net.gw2 = gam_w2;
net.gb2 = gam_b2;

net.cw1 = 0.5*gam_w1/fw(1);
net.cb1 = 0.5*gam_b1/fw(2);
net.cw2 = 0.5*gam_w2/fw(3);
net.cb2 = 0.5*gam_b2/fw(4);

net.gam = [net.gw1 net.gb1 net.gw2 net.gb2];

net.c = [net.cw1 net.cb1 net.cw2 net.cb2];
```

```

function [C, rate] = confm(z, t1);
% Confusion matrix

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Rows of the matrix are true classes
% Columns of the matrix are predicted classes
% t2 are network output classes
% t1 are true classes

[n c] = size(t1);
t2 = zeros(n, c);
zmax = max(z')';

for i = 1:n
    for j = 1:c
        if z(i, j) == zmax(i)
            t2(i,j) = 1;
        end
    end
end

C = zeros(c, c);

for k1 = 1:c
    for k2 = 1:c
        for i = 1:n
            if (t1(i, k1) == 1)&&(t2(i,k2) == 1);
                C(k1, k2) = C(k1, k2) + 1;
            end
        end
    end
end

rate = [trace(C)/sum(sum(C))*100 trace(C)];

```

```

function [sens, spec] = senspe(C);
% Calculate sensitivity and specificity

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

A = 0;
for i = 1:size(C,1)
    for j = 1:size(C,2)
        A = A + C(i,j);
    end
end

for i = 1:size(C,2)
    TP(i) = C(i,i);
end

for i = 1:size(C,2)
    B(i) = sum(C(i,1:size(C,1)))+ sum(C(1:size(C,2),i))-C(i,i);
end

TN = A - B;
k1 = sum(C,2)';
FN = k1 - TP;
k2 = sum(C,1);
FP = k2 - TP;

sens = mean(TP./(TP+FN));
spec = mean(TN./(TN+FP));

```

```

function x = mvgd(N, d, mu, variance);
% Multivariate Gaussian distribution

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

[m, n] = size(mu);

if (n == 1)
    mu = mu';
end

x = randn(N,d);
stdmean = mean(x);

for i = 1:N
    x(i,:) = x(i, :) - stdmean;
end

% Check whether X is a positive definite or not
% If X is positive definite, p is positive interger
% If X is not positive definite, p is zero
[R,p] = chol(variance);

if p > 0
    x = x*sqrtm(variance);
else
    x = x*R;
end

for k = 1:n
    x(:,k) = x(:,k) + mu(k);
end

```

```

function net = reini(nin, nhidden, nout, mu1, sig1, mu2, sig2,...
                    mu3, sig3, mu4, sig4);
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

% Network architecture
net.type = 'MLP Classifier';           % Network type
net.nin = nin;                         % Number of input nodes
net.nhidden = nhidden;                 % Number of hidden nodes
net.nout = nout;                       % Number of output nodes

% Number of all the weights and biases
net.ws = (nin + 1)*nhidden + (nhidden + 1)*nout;

% Initialise weights and biases with multivariate Gaussian distributions
net.w1 = mvgd(nin, nhidden, mu1, sig1);
net.b1 = mvgd(1, nhidden, mu2, sig2);
net.w2 = mvgd(nhidden, nout, mu3, sig3);
net.b2 = mvgd(1, nout, mu4, sig4);

% Index matrix of weights and biases in groups;
net.index = zeros(net.ws, 4);
m1 = nin*nhidden;
net.index(1:m1, 1) = 1;
m2 = m1 + nhidden;
net.index(m1 + 1: m2, 2) = 1;
m3 = m2 + nhidden*nout;
net.index(m2 + 1: m3, 3) = 1;
m4 = m3 + nout;
net.index(m3 + 1: m4, 4) = 1;

```

```

function logbaysevi = logbevi(net, x, t, vw1, vb1, vw2, vb2);
% Evaluate the log Bayesian evidence for a fixed
% network structure(logbevi)
%
% Is used in Experiment 1 in Chapter 6
%
% net: network information
% x : input data
% t: target data
% vw1: variance of hidden-layer weights
% vb1: variance of hidden-layer biases
% vw2: variance of output-layer weights
% vb2: variance of output-layer biases
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w1 = net.w1;
b1 = net.b1;
w2 = net.w2;
b2 = net.b2;
w = m2v(net); % Weight vector

thetal = 1.0e-100; % Is used to avoid log of zero

cw1 = (1/vw1) + thetal;
cb1 = (1/vb1) + thetal;
cw2 = (1/vw2) + thetal;
cb2 = (1/vb2) + thetal;

[f, fd, fw] = funcs(net, x, t); % Functions

term1 = -fd - 0.5*(cw1*sum(sum(w1.*w1))...
          + cb1*sum(b1.*b1)...
          + cw2*sum(sum(w2.*w2))...
          + cb2*sum(b2.*b2));

kw1 = sum(net.index(:,1)); % Number of hidden-layer weights
kb1 = sum(net.index(:,2)); % Number of hidden-layer biases
kw2 = sum(net.index(:,3)); % Number of output-layer weights
kb2 = sum(net.index(:,4)); % Number of output-layer biases

term2 = 0.5*(kw1*log(cw1)...
          + kb1*log(cb1)...
          + kw2*log(cw2)...
          + kb2*log(cb2));

```



```

m1 = kw1;
m2 = m1 + kb1;
m3 = m2 + kw2;
m4 = m3 + kb2;

temp = ones(net.ws,net.ws);
for i = 1:m1
    temp(i,:) = temp(i,)*cw1;
end

for i = (m1 + 1):m2
    temp(i,:) = temp(i,)*cb1;
end

for i = (m2 + 1):m3
    temp(i,:) = temp(i,)*cw2;
end

for i = (m3 + 1):m4
    temp(i,:) = temp(i,)*cb2;
end

I = eye(net.ws,net.ws);
phi = temp.*I;           % "Phi" matrix
h = hessa(net, x, t);    % Evaluating the Hessian matrix
h1 = abs(h + phi);       % H + Phi
term3 = -0.5*log(det(h1) + thet1);

logbaysevi = real(term1 + term2 + term3); % Log Bayesian evidence

clear h1; % To empty the memory

```

```

function [net, f, fc, cycles, logevidence] = cgevi(net, x, t, x1, t1,...
        train_steps, search_steps, vw1, vb1, vw2, vb2);
% Conjugate gradient training algorithm with Bayesian evidence evaluation
% (Laplace method)

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
ncycles = 100; % Number of training cycle
fnew = funcs(net, x, t);
gnew = grads(net, x, t); % Initial gradient
d = -gnew;

i = 1;
% Main loop
while (i <= train_steps)

    wold = w;
    fold = fnew;
    gold = gnew;

    gg = gold*gold';
    if (gg == 0) % Network training is done
        f = fnew;
        return;
    end;

    if (gnew*d' > 0) % Search direction is uphill in conjugate
        d = -d;
    end

    % Line minimisation
    dline = d/norm(d);
    [net, a, b, c, fmin, alpha] = linemin(net, x, t, dline, search_steps);
    w = w + alpha*dline; % Update weight vector

    net = v2m(net, w);
    f = funcs(net, x, t); % Evaluate cost function

    % Update search direction using Polak - Ribiere formula
    gnew = grads(net, x, t);
    beta = ((gnew - gold)*(gnew)')/gg;
    d = -gnew + (d.*beta);

    % Evaluating the log Bayesian evidence (logbevi)
    logevidence(i) = logbevi(net, x, t, vw1, vb1, vw2, vb2);

```

```
% Test the trained network
z = netfwd(net, x1);
[C, rate] = confm(z, t1);
accuracy(i) = rate(1);
C1{i} = C;
rate1{i} = rate;
save ('netf', 'C1', 'rate1');

cycles(i) = i;
fc(i) = f;
fprintf(1, ' Cycle: %4d    Function: %11.5f    Logevi: %11.5f    Rate: %11.5f\n', ...
        i, fc(i), logevidence(i), accuracy(i));

i = i + 1;

end
```

```

function logevidence = logevimc(net, x, t, L);
% Evaluating the evidence with the Metropolis sampling technique
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% Load data
load data_1/data_1.mat
load data_1/data_2.mat;
load data_1/data_3.mat;
load data_1/data_4.mat;
load data_1/data_6.mat;
load data_1/data_7.mat;
load data_1/data_8.mat;
load data_1/data_9.mat;

% Network architecture
nin = 40;           % Number of input nodes
nhidden = 3;       % Number of hidden nodes
nout = 4;          % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% No prior hyperparameters for weight and bias groups
cw1 = 0;
cb1 = 0;
cw2 = 0;
cb2 = 0;
net = hypers(net, cw1, cb1, cw2, cb2);
W = (nin + 1)*nhidden + (nhidden + 1)*nout;

for i = 1:L
    % The Metropolis sampling algorithm
    X = [];
    X0 = 0;
    delta = 1;
    naccept = 0;
    n = 0;
    while (naccept < W)
        n = n + 1;
        Xt = X0 + delta*(2*rand(1,1)-1);
        ratio = exp(0.5*(X0^2 - Xt^2)/2/pi);
        if (ratio > rand)
            X = [X, Xt];
            X0 = Xt;
            naccept = naccept + 1;
        end
    end
end

```

```
        end
    end
    w{i} = X;
    net = v2m(net, X);
    f = funcs(net, x, t);
    ff{i} = exp(-f);
end

Zs = 0;
for i = 1:L
    Zs = (Zs + ff{i})/L;
end
logevidence = log(Zs);
```

```

function [net, f, fc, cycles, logevidence] = cgevimc(net, x, t, xl, tl,...
            train_steps, search_steps, L);
% Conjugate gradient training algorithm with Bayesian evidence evaluation
% Monte Carlo method

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

w = m2v(net); % Initial weight vector
ncycles = 100; % Number of training cycle
fnew = funcs(net, x, t);
gnew = grads(net, x, t); % Initial gradient
d = -gnew;

i = 1;
% Main loop
while (i <= train_steps)

    wold = w;
    fold = fnew;
    gold = gnew;

    gg = gold*gold';
    if (gg == 0) % Network training is done
        f = fnew;
        return;
    end;

    if (gnew*d' > 0) % Search direction is uphill in conjugate
        d = -d;
    end

    % Line minimisation
    dline = d/norm(d);
    [net, a, b, c, fmin, alpha] = linemin(net, x, t, dline, search_steps);
    w = w + alpha*dline; % Update weight vector

    net = v2m(net, w);
    f = funcs(net, x, t); % Evaluate cost function

    % Update search direction using Polak - Ribiere formula
    gnew = grads(net, x, t);
    beta = ((gnew - gold)*(gnew)')/gg;
    d = -gnew + (d.*beta);

    % Evaluating the log Bayesian evidence with Monte Carlo method
    logevidence(i) = logevimc(net, x, t, L);

```

```
% Test the trained network
z = netfwd(net, x1);
[C, rate] = confm(z, t1);
accuracy(i) = rate(1);
C1{i} = C;
rate1{i} = rate;
save ('netf', 'C1', 'rate1');

cycles(i) = i;
fc(i) = f;
fprintf(1, ' Cycle: %4d    Function: %11.5f    Logevi: %11.5f    Rate: %11.5f\n', ...
        i, fc(i), logevidence(i), accuracy(i));

i = i + 1;

end
```

```

function z12 = bcombin(z1, z2);
% Combine the results of two independent classifiers
% using Bayes' theorem (bcombin)
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% z1 is the out vector of classifier 1
% z2 is the out vector of classifier 1
% z1 = P1(Ck|x1)
% z2 = P2(Ck|x2)
%
%
% 
$$P12(Ck | x1, x1) = \frac{P1(Ck|x1)*P1(Ck|x2)}{P(Ck)}$$

%
% If x1 = x2 = x
%
% 
$$P12(Ck | x, x) = \frac{P1(Ck|x)*P1(Ck|x)}{(P(Ck))^2}$$

%
P1 = z1;
P2 = z2;

N = size(z1, 1); % Number of test patterns
c = size(z1, 2); % Number of output classes
PCK = 1/c;
P12 = P1.*P2/PCK/PCK;

% Normalise P12
z12 = zeros(N, c);
for i = 1:N
    for j = 1:c
        if P12(i,j) == max(P12(i,:))
            z12(i,j) = 1;
        else
            z12(i,j) = 0;
        end
    end
end
end

```



```

function z12 = dcombin(z1, z2);
% Combine the results of two independent classifiers
% using Dempster's rule of combination (dcombin)
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% z1 is the out vector of classifier 1
% z2 is the out vector of classifier 1
%
% Dempster's rule of combination:
%
%
%

$$m12(Z) = m1(Z) (+) m2(Z) = \frac{\sum [m1(X) * m2(Y)]}{1 - K}$$

%
% if the intersection of X and Y is non-empty
%
% K = sum[m1(X) * m2(Y)]
% if the intersection of X and Y is empty.

N = size(z1, 1); % Number of test patterns
c = size(z1, 2); % Number of output classes

% Cobine basic assignments from the network outputs:
m1 = z1;
m2 = z2;

% For the non-empty intersection, compute the belief (bel)
m1_m2 = m1.*m2;

% For the empty intersection, compute the plausibility (pl)
% Pl(A) = 1 - Bel(-A)
for n = 1:N
    bel(n) = 0;
    for m = 1:c
        bel(n) = bel(n) + m1_m2(n,m);
    end
end

for n = 1:N
    for m = 1:c
        m12(n,m) = m1_m2(n,m)/bel(n);
    end
end

for n = 1:N
    for m = 1:c
        if m12(n,m) == max(m12(n,:))

```

```
        m12(n,m) = 1;
    else
        m12(n,m) = 0;
    end
end
end
end

z12 = m12;
```

```

% Demonstration 1: Head movement classification
% generic training (demo1);
%
% Experiment 1 in Chapter 4
%
% The network is trained on the first half
% of data of 8 subjects
% The network is tested on the second half
% of data of 8 subjects
% Subject 1, 2, 3 and 4 are able-bodied
% Subject 6 and 9 are C5-injury level people
% Subject 7 and 8 are C4-injury level people

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_1/data_1a.mat
load data_1/data_2a.mat;
load data_1/data_3a.mat;
load data_1/data_4a.mat;
load data_1/data_6a.mat;
load data_1/data_7a.mat;
load data_1/data_8a.mat;
load data_1/data_9a.mat;

load data_1/data_1b.mat
load data_1/data_2b.mat;
load data_1/data_3b.mat;
load data_1/data_4b.mat;
load data_1/data_6b.mat;
load data_1/data_7b.mat;
load data_1/data_8b.mat;
load data_1/data_9b.mat;

% Training data set
x = [data_1a; data_2a; data_3a; data_4a;...
     data_6a; data_7a; data_8a; data_9a];           % Input data

t = [target_1a; target_2a; target_3a; target_4a;...
     target_6a; target_7a; target_8a; target_9a]; % Target data

% Test data set
x1 = [data_1b; data_2b; data_3b; data_4b;...

```

```

        data_6b; data_7b; data_8b; data_9b];           % Input data

t1 = [target_1b; target_2b; target_3b; target_4b;...
      target_6b; target_7b; target_8b; target_9b];   % Target data

% Network architecture
nin = 40;           % Number of input nodes
nhidden = 3;       % Number of hidden nodes
nout = 4;          % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% Prior hyperparameters for weight and bias groups
cw1 = 1.0e-4;
cb1 = 1.0e-4;
cw2 = 1.0e-4;
cb2 = 1.0e-4;
net = hypers(net, cw1, cb1, cw2, cb2);

% Training network
search_steps = 10; % At least 10
train_steps = 200;

nest = 3;          % Number of hyperparameter re-estimations
k = 0;
while k < nest

    % Minimise cost function
    [net, f] = scgtrain (net, x, t, train_steps, search_steps);

    % Re-estimate hyperparameters
    [net, A] = rehypers(net, x, t);

    % Evaluate log evidence
    net = logevi(net, A, x, t);
    disp(' ');

    fprintf(1, 'Re-estimation: %2d \n', k);
    hyperparameters = net.c
    well_determined_parameters = net.gam
    log_evidence = net.logevi
    log_Occam_factor = net.logOcc
    disp(' ');
    disp(' ');

    k = k + 1;

end

z = netfwd(net, x1);

```

```
[C, rate] = confm(z, t1);

fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
disp(' ');
Confusion_matrix = C
[sens, spec] = senspe(C);
Sensitivity_Specificity = [sens spec]
```

```

% Demonstration 2: Head movement classification
% non-generic training (demo2);
%
% Experiment 2 in Chapter 4
%
% The network is trained on data from subject 1, 2, 6 and 7
% The network is tested on data from subject 3, 4, 8 and 9
% Subject 1, 2, 3 and 4 are able-bodied
% Subject 6 and 9 are C5-injury level people
% Subject 7 and 8 are C4-injury level people

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_1/data_1.mat
load data_1/data_2.mat;
load data_1/data_3.mat;
load data_1/data_4.mat;
load data_1/data_6.mat;
load data_1/data_7.mat;
load data_1/data_8.mat;
load data_1/data_9.mat;

% Training data set
x = [data_1; data_2; data_6; data_7]; % Input data
t = [target_1; target_2; target_6; target_7]; % Target data

% Testing data set
x1 = [data_3; data_4; data_8; data_9]; % Input data
t1 = [target_3; target_4; target_8; target_9]; % Target data

% Network architecture
nin = 40; % Number of input nodes
nhidden = 3; % Number of hidden nodes
nout = 4; % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% Prior hyperparameters for weight and bias groups
cw1 = 1.0e-4;
cb1 = 1.0e-4;
cw2 = 1.0e-4;

```

```

cb2 = 1.0e-4;
net = hypers(net, cw1, cb1, cw2, cb2);

% Training network
search_steps = 10; % At least 10
train_steps = 200;

nest = 2;          % Number of hyperparameter re-estimations
k = 0;
while k < nest

    % Minimise cost function
    [net, f] = qntrain (net, x, t, train_steps, search_steps);

    % Re-estimate hyperparameters
    [net, A] = rehypers(net, x, t);

    % Evaluate log evidence
    net = logevi(net, A, x, t);

    disp(' ');
    fprintf(1, 'Re-estimation: %2d \n', k);
    hyperparameters = net.c
    well_determined_parameters = net.gam
    log_evidence = net.logevi
    log_Occam_factor = net.logOcc
    disp(' ');
    disp(' ');
    % pause;

    k = k + 1;

end

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
disp(' ');
Confusion_matrix = C
[sens, spec] = senspe(C);
Sensitivity_Specificity = [sens spec]

```

```

% Demonstration 3: Head movement classification
% near-adaptive training (demo3);

% The network is trained on data from subject 1, 2, 3, 4, 6 and 7
% The network is tested on the first part of data from subject 8 and 9
% Subject 1, 2, 3 and 4 are able-bodied
% Subject 6 and 9 are C5-injury level people
% Subject 7 and 8 are C4-injury level people

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_2/data_1.mat
load data_2/data_2.mat;
load data_2/data_3.mat;
load data_2/data_4.mat;
load data_2/data_6.mat;
load data_2/data_7.mat;
load data_2/data_8_1.mat;
load data_2/data_9_1.mat;

% Training data set
x = [data_1; data_2; data_3; data_4;...
     data_6; data_7]; % Input data

t = [target_1; target_2; target_3; target_4;...
     target_6; target_7]; % Target data

% Network architecture
nin = 40; % Number of input nodes
nhidden = 3; % Number of hidden nodes
nout = 4; % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% Prior hyperparameters for weight and bias groups
cw1 = 1.0e-4;
cb1 = 1.0e-4;
cw2 = 1.0e-4;
cb2 = 1.0e-4;
net = hypers(net, cw1, cb1, cw2, cb2);

```



```

% Training network
search_steps = 10; % At least 10
train_steps = 150;

nest = 5;          % Number of hyperparameter re-estimations
k = 0;
while k < nest

    % Minimise cost function
    [net, f]= scgtrain (net, x, t, train_steps, search_steps);

    % Re-estimate hyperparameters
    [net, A] = rehypers(net, x, t);

    % Evaluate log evidence
    net = logevi(net, A, x, t);

    k = k + 1;

    disp(' ');
    fprintf(1, 'Re-estimation: %2d \n', k);
    hyperparameters = net.c
    well_determined_parameters = net.gam
    log_evidence = net.logevi
    log_Occam_factor = net.logOcc
    disp(' ');
    % disp('Press any key to continue !!! ');
    disp(' ');
    % pause;

    % Test the trained network
    % Testing data set
    x1 = [data_8_1; data_9_1];    % Input data
    t1 = [target_8_1; target_9_1]; % Target data

    z = netfwd(net, x1);
    [C, rate] = confm(z, t1);

    fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
    fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
    disp(' ');
    Confusion_matrix = C
    [sens, spec] = senspe(C);
    Sensitivity_Specificity = [sens spec]

end

```

```

% Demonstration 4: Head movement classification
% fully-adaptive training (demo4);

% The network is trained on data from subject 1, 2, 3, 4, 6, 7
% and the second part of data from subject 8 and 9
% The network is tested on the first part of data from subject 8 and 9
% Subject 1, 2, 3 and 4 are able-bodied
% Subject 6 and 9 are C5-injury level people
% Subject 7 and 8 are C4-injury level people

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_2/data_1.mat
load data_2/data_2.mat;
load data_2/data_3.mat;
load data_2/data_4.mat;
load data_2/data_6.mat;
load data_2/data_7.mat;
load data_2/data_8_1.mat;
load data_2/data_8_2.mat;
load data_2/data_9_1.mat;
load data_2/data_9_2.mat;

% Training data set
x = [data_1; data_2; data_3; data_4;...
     data_6; data_7; data_8_2; data_9_2]; % Input data

t = [target_1; target_2; target_3; target_4;...
     target_6; target_7; target_8_2; target_9_2]; % Target data

% Network architecture
nin = 40; % Number of input nodes
nhidden = 3; % Number of hidden nodes
nout = 4; % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% Prior hyperparameters for weight and bias groups
cw1 = 1.0e-4;
cb1 = 1.0e-4;
cw2 = 1.0e-4;
cb2 = 1.0e-4;

```

```

net = hypers(net, cw1, cb1, cw2, cb2);

% Training network
search_steps = 10; % At least 10
train_steps = 150;

nest = 5;          % Number of hyperparameter re-estimations
k = 0;
while k < nest

    % Minimise cost function
    [net, f]= cgtrain (net, x, t, train_steps, search_steps);

    % Re-estimate hyperparameters
    [net, A] = rehypers(net, x, t);

    % Evaluate log evidence
    net = logevi(net, A, x, t);

    k = k + 1;

    disp(' ');
    fprintf(1, 'Re-estimation: %2d \n', k);
    hyperparameters = net.c
    well_determined_parameters = net.gam
    log_evidence = net.logevi
    log_Occam_factor = net.logOcc
    disp(' ');
    % disp('Press any key to continue !!! ');
    disp(' ');
    % pause;

    % Test the trained network
    % Testing data set
    x1 = [data_8_1; data_9_1];    % Input data
    t1 = [target_8_1; target_9_1]; % Target data

    z = netfwd(net, x1);
    [C, rate] = confm(z, t1);

    fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
    fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
    disp(' ');
    Confusion_matrix = C
    [sens, spec] = senspe(C);
    Sensitivity_Specificity = [sens spec]

end

```

```

% Demonstration 5: Design classifier 1
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_2/data_1.mat
load data_2/data_2.mat;
load data_2/data_3.mat;
load data_2/data_4.mat;
load data_2/data_6.mat;
load data_2/data_7.mat;
load data_2/data_8_1.mat;
load data_2/data_8_2.mat;
load data_2/data_9_1.mat;
load data_2/data_9_2.mat;

% Training data set
x = [data_1; data_2; data_3; data_4;...
     data_6; data_7; data_8_2; data_9_2]; % Input data

t = [target_1; target_2; target_3; target_4;...
     target_6; target_7; target_8_2; target_9_2]; % Target data

% Network architecture
nin = 40; % Number of input nodes
nhidden = 3; % Number of hidden nodes
nout = 4; % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% No hyperparameters for weight and bias groups
cw1 = 0;
cb1 = 0;
cw2 = 0;
cb2 = 0;
net = hypers(net, cw1, cb1, cw2, cb2);

% Initialise weights and biases with specified mean and variance
mu = 0;
mul = mu*ones(1, nhidden);
sigmal = 2*eye(nhidden);
wbl = mvgd(nin + 1, nhidden, mul, sigmal);

```

```

net.w1 = wb1(1: nin, :);
net.b1 = wb1(nin + 1, :);

mu2 = mu*ones(1, nout);
sigma2 = nout*eye(nout);
wb2 = mvgd(nhidden + 1, nout, mu2, sigma2);
net.w2 = wb2(1: nhidden, :);
net.b2 = wb2(nhidden + 1, :);

% Training network
train_steps = 10;
search_steps = 10; % At least 10

% Minimise cost function
[net, f]= cgtrain (net, x, t, train_steps, search_steps);
disp(' ');

% Test the trained network
% Testing data set
x1 = [data_8_1; data_9_1]; % Input data
t1 = [target_8_1; target_9_1]; % Target data

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
disp(' ');
Confusion_matrix = C
[sens, spec] = senspe(C);
Sensitivity_Specificity = [sens spec]

net1 = net;
save('classifier1', 'net1');

```

```

% Demonstration 6: Design classifier 2
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_2/data_1.mat
load data_2/data_2.mat;
load data_2/data_3.mat;
load data_2/data_4.mat;
load data_2/data_6.mat;
load data_2/data_7.mat;
load data_2/data_8_1.mat;
load data_2/data_8_2.mat;
load data_2/data_9_1.mat;
load data_2/data_9_2.mat;

% Training data set
x = [data_1; data_2; data_3; data_4;...
     data_6; data_7; data_8_2; data_9_2];    % Input data

t = [target_1; target_2; target_3; target_4;...
     target_6; target_7; target_8_2; target_9_2];    % Target data

% Network architecture
nin = 40;           % Number of input nodes
nhidden = 3;       % Number of hidden nodes
nout = 4;          % Number of output nodes

% Initialise the network
net = netini(nin, nhidden, nout);

% No hyperparameters for weight and bias groups
cw1 = 0;
cb1 = 0;
cw2 = 0;
cb2 = 0;
net = hypers(net, cw1, cb1, cw2, cb2);

% Initialise weights and biases with specified mean and variance
mu = 0;
mu1 = mu*ones(1, nhidden);
sigma1 = 2*eye(nhidden);

```

```

wb1 = mvgd(nin + 1, nhidden, mu1, sigma1);
net.w1 = wb1(1: nin, :);
net.b1 = wb1(nin + 1, :);

mu2 = mu*ones(1, nout);
sigma2 = nout*eye(nout);
wb2 = mvgd(nhidden + 1, nout, mu2, sigma2);
net.w2 = wb2(1: nhidden, :);
net.b2 = wb2(nhidden + 1, :);

% Training network
train_steps = 10;
search_steps = 10; % At least 10

% Minimise cost function
[net, f] = cgtrain (net, x, t, train_steps, search_steps);
disp(' ');

% Test the trained network
% Testing data set
x1 = [data_8_1; data_9_1]; % Input data
t1 = [target_8_1; target_9_1]; % Target data

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
disp(' ');
Confusion_matrix = C
[sens, spec] = senspe(C);
Sensitivity_Specificity = [sens spec]

net2 = net;
save('classifier2', 'net2');

```

```

% Demonstration 7: Combine two independent classifiers
% using Bayes rule of combination

% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load two trained classifiers
load classifier1.mat
load classifier2.mat;

% Load data
load data_2/data_1.mat
load data_2/data_2.mat;
load data_2/data_3.mat;
load data_2/data_4.mat;
load data_2/data_6.mat;
load data_2/data_7.mat;
load data_2/data_8_1.mat;
load data_2/data_8_2.mat;
load data_2/data_9_1.mat;
load data_2/data_9_2.mat;

% Training data set
x = [data_1; data_2; data_3; data_4;...
     data_6; data_7; data_8_2; data_9_2];    % Input data

t = [target_1; target_2; target_3; target_4;...
     target_6; target_7; target_8_2; target_9_2];    % Target data

% Testing data set
X = [data_8_1; data_9_1];    % Input data
T = [target_8_1; target_9_1];    % Target data

% Test classifier 1
z1 = netfwd(net1, X);
[C1, ratel] = confm(z1, T);
fprintf(1, 'The success rate of classifier 1: %3.2f \n', ratel(1));
fprintf(1, 'Correct patterns of classifier 1: %3.2f \n', ratel(2));
Confusion_matrix1 = C1
[sens1, spec1] = senspe(C1);
Sensitivity1_Specificity1 = [sens1 spec1]
disp(' ');
disp(' ');

```



```
% Test classifier 2
z2 = netfwd(net2, X);
[C2, rate2] = confm(z2, T);
fprintf(1, 'The success rate of classifier 2: %3.2f \n', rate2(1));
fprintf(1, 'Correct patterns of classifier 2: %3.2f \n', rate2(2));
Confusion_matrix2 = C2
[sens2, spec2] = senspe(C2);
Sensitivity2_Specificity2 = [sens2 spec2]
disp(' ');
disp(' ');

% Combine the classifiers using Bayes rule
z12 = bcombin(z1, z2);

% Test the combined classifier
[C12, rate12] = confm(z12, T);
fprintf(1, 'The success rate of combining 2 classifiers : %3.2f \n', rate12(1));
fprintf(1, 'Correct patterns of combining 2 classifiers : %3.2f \n', rate12(2));
Confusion_matrix12 = C12
[sens12, spec12] = senspe(C12);
Sensitivity12_Specificity12 = [sens12 spec12]
disp(' ');
disp(' ');
```

```

% Demonstration 8: Evaluate the evidence for different network architecture
% corresponding to varying numbers of hidden nodes
% The number of hidden nodes can be changed from 1 to 6
%
% The network is trained on data from subject 1, 2, 6 and 7
% The network is tested on data from subject 3, 4, 8 and 9
% Subject 1, 2, 3 and 4 are able-bodied
% Subject 6 and 9 are C5-injury level people
% Subject 7 and 8 are C4-injury level people
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_1/data_1.mat
load data_1/data_2.mat;
load data_1/data_3.mat;
load data_1/data_4.mat;
load data_1/data_6.mat;
load data_1/data_7.mat;
load data_1/data_8.mat;
load data_1/data_9.mat;

% Training data set
x = [data_1; data_2; data_6; data_7; data_8; data_9]; % Input data
t = [target_1; target_2; target_6; target_7; target_8; target_9]; % Target data

% Network architecture
nin = 40; % Number of input nodes
nhidden = input('Number of hidden nodes (1-6): '); % Number of hidden nodes
nout = 4; % Number of output nodes

runs = 10; % Number of training runs

for m = 1:runs

    % Initialise the network
    net = netini(nin, nhidden, nout);

    % Prior hyperparameters for weight and bias groups
    cw1 = 1.0e-4;

```

```

cb1 = 1.0e-4;
cb2 = 1.0e-4;
net = hypers(net, cw1, cb1, cw2, cb2);

% Training network
search_steps = 10; % At least 10
train_steps = 100;

nest = 3;          % Number of hyperparameter re-estimations
k = 0;
start = clock;    % Start measuring training time

while k < nest

    % Minimise the cost function
    [net, f] = scgtrain(net, x, t, train_steps, search_steps);

    % Re-estimate hyperparameters
    [net, A] = rehypers(net, x, t);

    % Evaluate log evidence
    net = logevi(net, A, x, t);

    k = k + 1;

    disp(' ');
    fprintf(1, 'Re-estimation: %2d \n', k);
    hyperparameters = net.c;
    well_determined_parameters = net.gam;

    theta = -1.0e+6;

    log_evidence(m) = net.logevi;
    % If log_evidence is infinite, log_evidence = theta
    if (log_evidence(m) <= theta)
        log_evidence(m) = theta;
    end

    log_Occam_factor(m) = net.logOcc;
    % If log_Occam_factor is infinite, log_Occam_factor = 1.0e-6
    if (log_Occam_factor(m) <= theta)
        log_Occam_factor(m) = theta;
    end

    disp(' ');
    % disp('Press any key to continue !!! ');
    disp(' ');
    % pause;

% Test the trained network

```

```

% Testing data set
x1 = [data_3; data_4];    % Input data
t1 = [target_3; target_4];    % Target data

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

fprintf(1, 'The percentage of accuracy : %3.2f \n', rate(1));
fprintf(1, 'The number of correct patterns: %3.2f \n', rate(2));
disp(' ');
Confusion_matrix = C;
[sens, spec] = senspe(C);
Sensitivity_Specificity = [sens spec];
disp(' ');
disp(' ');

end

end

stop = clock; % Stop measuring training time

% Returns the training time in seconds
timetrain = etime(stop, start);

% Save the computed evidence and Occam factor for the later analyses
if (nhidden == 1)
    mean_evi_1 = mean(log_evidence);    % Average of log evidence
    std_evi_1 = std(log_evidence);    % Standard deviation of log evidence
    mean_Occ_1 = mean(log_Occam_factor);    % Average of log Occam factor
    std_Occ_1 = std(log_Occam_factor);    % Standard deviation of log Occam factor
    save('logevi_1', 'mean_evi_1', 'std_evi_1', 'mean_Occ_1', 'std_Occ_1');
end

if (nhidden == 2)
    mean_evi_2 = mean(log_evidence);    % Average of log evidence
    std_evi_2 = std(log_evidence);    % Standard deviation of log evidence
    mean_Occ_2 = mean(log_Occam_factor);    % Average of log Occam factor
    std_Occ_2 = std(log_Occam_factor);    % Standard deviation of log Occam factor
    save('logevi_2', 'mean_evi_2', 'std_evi_2', 'mean_Occ_2', 'std_Occ_2');
end

if (nhidden == 3)
    mean_evi_3 = mean(log_evidence);    % Average of log evidence
    std_evi_3 = std(log_evidence);    % Standard deviation of log evidence
    mean_Occ_3 = mean(log_Occam_factor);    % Average of log Occam factor
    std_Occ_3 = std(log_Occam_factor);    % Standard deviation of log Occam factor
    save('logevi_3', 'mean_evi_3', 'std_evi_3', 'mean_Occ_3', 'std_Occ_3');
end

if (nhidden == 4)

```

```
mean_evi_4 = mean(log_evidence);           % Average of log evidence
std_evi_4 = std(log_evidence);            % Standard deviation of log evidence
mean_Occ_4 = mean(log_Occam_factor);      % Average of log Occam factor
std_Occ_4 = std(log_Occam_factor);        % Standard deviation of log Occam factor
save('logevi_4', 'mean_evi_4', 'std_evi_4', 'mean_Occ_4', 'std_Occ_4');
end

if (nhidden == 5)
    mean_evi_5 = mean(log_evidence);       % Average of log evidence
    std_evi_5 = std(log_evidence);        % Standard deviation of log evidence
    mean_Occ_5 = mean(log_Occam_factor);   % Average of log Occam factor
    std_Occ_5 = std(log_Occam_factor);    % Standard deviation of log Occam factor
    save('logevi_5', 'mean_evi_5', 'std_evi_5', 'mean_Occ_5', 'std_Occ_5');
end

if (nhidden == 6)
    mean_evi_6 = mean(log_evidence);       % Average of log evidence
    std_evi_6 = std(log_evidence);        % Standard deviation of log evidence
    mean_Occ_6 = mean(log_Occam_factor);   % Average of log Occam factor
    std_Occ_6 = std(log_Occam_factor);    % Standard deviation of log Occam factor
    save('logevi_6', 'mean_evi_6', 'std_evi_6', 'mean_Occ_6', 'std_Occ_6');
end
```

```

% Demonstration 9:
% Plot of the average and standard deviation of evidence
% of ten trained networks.
% The number of hidden nodes is varied from 1 to 6
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

load logevi_1;      % Load the evidence of one-hidden node networks
load logevi_2;      % Load the evidence of two-hidden node networks
load logevi_3;      % Load the evidence of three-hidden node networks
load logevi_4;      % Load the evidence of four-hidden node networks
load logevi_5;      % Load the evidence of five-hidden node networks
load logevi_6;      % Load the evidence of six-hidden node networks

y1 = [mean_evi_1 mean_evi_2 mean_evi_3 mean_evi_4 mean_evi_5 mean_evi_6];
e1 = [std_evi_1 std_evi_2 std_evi_3 std_evi_4 std_evi_5 std_evi_6];
errorbar(y1, e1, 'linewidth', 2);
xlabel('Number of hidden nodes');
ylabel('Log evidence');
pause;
close;

% y2 = [mean_Occ_1 mean_Occ_2 mean_Occ_3 mean_Occ_4 mean_Occ_5 mean_Occ_6];
% e2 = [std_Occ_1 std_Occ_2 std_Occ_3 std_Occ_4 std_Occ_5 std_Occ_6];
% errorbar(y2, e2, 'linewidth', 2);
% xlabel('Number of hidden nodes');
% ylabel('Log Occam factor');
% pause;
% close;

```

```

% Demonstration 10: Bayesian early stopping (Laplace method)
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

runs = 2;
for m = 1:runs
    fprintf(1, ' Training run: %3d \n', m);

    % Load data
    load data_1/data_1.mat
    load data_1/data_2.mat;
    load data_1/data_3.mat;
    load data_1/data_4.mat;
    load data_1/data_6.mat;
    load data_1/data_7.mat;
    load data_1/data_8.mat;
    load data_1/data_9.mat;

    % Training set
    x = [data_1; data_2; data_8; data_9];
    t = [target_1; target_2; target_8; target_9];

    % Test set
    x1 = [data_3; data_4; data_6; data_7];
    t1 = [target_3; target_4; target_6; target_7];

    % Network architecture
    nin = 40;           % Number of input nodes
    nhidden = 3;       % Number of hidden nodes
    nout = 4;          % Number of output nodes

    % Initialise weights and biases using Nguyen.D and Widrow's method
    bias1 = sqrt(nhidden);
    vw1 = bias1^2;;    % Variance of weights from input nodes to hidden nodes
    vb1 = vw1;        % Variance of biases of hidden nodes
    bias2 = 0.5;
    vw2 = bias2^2;    % Variance of weights from hidden nodes to output nodes
    vb2 = vw2;        % Variance of biases of ouputs nodes
    net = mgdini(nin, nhidden, nout, vw1, vb1, vw2, vb2);

    % No prior hyperparameters for weight and bias groups
    cw1 = 0; cb1 = 0; cw2 = 0; cb2 = 0;

```

```

net = hypers(net, cw1, cb1, cw2, cb2);

train_steps = 100;
search_steps = 20;

% Start measuring the training time
start = clock;

% Minimise the cost function with Bayesian evidence evaluation
[net, f, fc, cycles, logevidence] = cgevi(net, x, t, x1, t1,...
    train_steps, search_steps, vw1, vb1, vw2, vb2);

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

% Stop measuring the training time
finish = clock;

% Training time
traintime(m) = etime(finish, start);

% Find the cycle that corresponds to the maximum evidence
for i = 1:train_steps
    if (logevidence(i) == max(logevidence))
        maxcycle = i;
    end
end

load netf;
Cmax = C1{maxcycle};
ratemax = rate1{maxcycle};
ratemaxevi(m) = ratemax(1);
disp(' ');
zf = netfwd(net, x1);
[Cf, ratef] = confm(zf, t1);
ratefinal(m) = ratef(1);

end

mean_max = mean(ratemaxevi)
mean_final = mean(ratefinal)
mean_time = mean(traintime)

% Plot the cycle error
plot(cycles, fc, '-', 'linewidth', 1);
xlabel('Cycle');
ylabel('Data error');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(fc) fc(maxcycle)];

```



```
plot(k2, kk2, 'r:');
pause;
close;

% Plot the cycle evidence
plot(cycles, logevidence, '- ', 'linewidth', 1);
xlabel('Cycle');
ylabel('Log evidence');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(logevidence) max(logevidence)];
plot(k2, kk2, 'r:');
axis([1 train_steps min(logevidence) logevidence(maxcycle) + 20]);
pause;
close;

Cmax = C1{maxcycle}
ratemax = ratel{maxcycle}
```

```

% Demonstration 10:
% Demonstration of Maximum evidence-early stopping
% with the Laplace method
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

runs = 1;

for m = 1:runs

    fprintf(1, ' Training run: %3d \n', m);

    % Load data
    load data_1/data_1.mat
    load data_1/data_2.mat;
    load data_1/data_3.mat;
    load data_1/data_4.mat;
    load data_1/data_6.mat;
    load data_1/data_7.mat;
    load data_1/data_8.mat;
    load data_1/data_9.mat;

    % Training set
    x = [data_1; data_2; data_3; data_4; data_6; data_7];
    t = [target_1; target_2; target_3; target_4; target_6; target_7];

    % Test set
    x1 = [data_8; data_9];
    t1 = [target_8; target_9];

    % Network architecture
    nin = 40;           % Number of input nodes
    nhidden = 3;       % Number of hidden nodes
    nout = 4;          % Number of output nodes

    % Initialise weights and biases using Nguyen.D and Widrow's method
    bias1 = sqrt(nhidden);
    vw1 = bias1^2;;    % Variance of weights from input nodes to hidden nodes
    vb1 = vw1;         % Variance of biases of hidden nodes
    bias2 = 0.5;
    vw2 = bias2^2;     % Variance of weights from hidden nodes to output nodes
    vb2 = vw2;        % Variance of biases of ouputs nodes

```

```

net = mgdini(nin, nhidden, nout, vw1, vb1, vw2, vb2);

% No prior hyperparameters for weight and bias groups
cw1 = 0; cb1 = 0; cw2 = 0; cb2 = 0;
net = hypers(net, cw1, cb1, cw2, cb2);

train_steps = 50;
search_steps = 20;

% Start measuring the training time
start = clock;

% Minimise the cost function using the scaled conjugate gradient
% algorithm
% and evaluating the Bayesian evidence
% every training cycle using the Laplace method
% [net, f, fc, cycles, logevidence] = cglaptrain(net, x, t, x1, t1,...
%         train_steps, search_steps, vw1, vb1, vw2, vb2);

[net, f, fc, cycles, logevidence] = scglaptrain(net, x, t, x1, t1,...
        train_steps, search_steps, vw1, vb1, vw2, vb2);

% Test the trained net
z = netfwd(net, x1);
[C, rate] = confm(z, t1);

% Stop measuring the training time
finish = clock;

% Training time
traintime(m) = etime(finish, start);

% Find the cycle that corresponds to the maximum evidence
for i = 1:train_steps
    if (logevidence(i) == max(logevidence))
        maxcycle = i;
    end
end

load netf;
Cmax = C1{maxcycle};
ratemax = rate1{maxcycle};
ratemaxevi(m) = ratemax(1);
disp(' ');
zf = netfwd(net, x1);
[Cf, ratef] = confm(zf, t1);
ratefinal(m) = ratef(1);

end

mean_max = mean(ratemaxevi)

```

```
mean_final = mean(ratefinal)
mean_time = mean(traintime)

% Plot the cycle error
plot(cycles, fc, '-', 'linewidth', 2);
xlabel('Cycle');
ylabel('Data error');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(fc) fc(maxcycle)];
plot(k2, kk2, 'r:');
pause;
close;

% Plot the cycle evidence
plot(cycles, logevidence, '-', 'linewidth', 2);
xlabel('Cycle');
ylabel('Log evidence');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(logevidence) max(logevidence)];
plot(k2, kk2, 'r:');
axis([1 train_steps min(logevidence) logevidence(maxcycle) + 20]);
pause;
close;

Cmax = C1{maxcycle}
ratemax = ratel{maxcycle}
```

```

% Demonstration 11: Demonstration of initialising weights and biases
% using Nguyen and Widrow's method
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% Nguyen and Widrow method of weight initialisation:
% -M is the number of hidden nodes
% -din is the fan-in (or in degree) of a neuron
% -The optimal length of  $N^{(1/din)}$  for the randomly initialised weights vectors
% -An optimal bias range  $[-N^{(1/din)}, N^{(1/din)}]$  for neurals in the hidden-layer
% -The weights of the neurons in the output-layer are randomly initialised in
% the interval  $[-0.5, 0.5]$ 

clc;
clear;

nin = 40;
nhidden = 3;
nout = 4;

Nx = nin*nhidden;
Ny = nhidden*nout;

bias1 = sqrt(nhidden);
variancel = bias1^2;
x = mvgd(Nx, 1, 0, variancel);
plot(x, '.');
ylabel('Magnitude');
xlabel('Weights');
axis([0 Nx -6 6]);
pause;
close;

bias2 = 0.5;
variance2 = bias2^2;
y = mvgd(Ny, 1, 0, variance2);
plot(y, '.');
ylabel('Magnitude');
xlabel('Weights');
axis([0 Ny -6 6]);
pause;
close;

```

```

function [integral, S] = mcim(a, b, R);
% Monte Carlo Integration Method (mcim)
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% This program is to numerically compute the integral of
% the following function in the interval [1,3]
% f(x) = x*x + x

resolution = 100;
step = (b - a)/resolution;
x = [a:step:b];
f = funcin(x);

maxf = max(f); % Maximum of the function in [a,b]
minf = min(f); % Minimum of the function in [a,b]

fa = funcin(a); % Function value at a
fb = funcin(b); % Function value at b

theta = 10; % M = f(x) + theta
M = maxf + theta; % M > f(x) in [a,b]
N = minf - theta; % N < f(x) in [a,b]

% The rectangular for generating samples
a1 = [a a];
fa1 = [M N];
hold on;
plot(a1, fa1, ':');

b1 = [b b];
fb1 = [M N];
hold on;
plot(b1, fb1, ':');

m1 = [a b];
fm1 = [M M];
hold on;
plot(m1, fm1, ':');

n1 = [a b];
fn1 = [N N];
hold on;
plot(n1, fn1, ':');

% Plot the function

```

```

plot(x, f, 'r-', 'linewidth', 2);

% Zoom window
xmin = a - 0.1*(b - a);
xmax = b + 0.1*(b - a);
ymin = minf - 0.1*(M - N);
ymax = maxf + 0.1*(M - N);
axis([xmin xmax ymin ymax]);

title('The Monte Carlo integration method');
xlabel('x');
ylabel('y');

xj = a + (b-a)*rand(1,R); % Generate random numbers xj in [a,b]
yj = N + (M - N)*rand(1,R); % Generate random numbers yj in [N,M]
hold on;
plot(xj, yj, 'b.');
```

```

S = 0; % Count the successful samples
for j = 1:R
    fxj(j) = funcin(xj(j));
    if (yj(j) <= fxj(j))
        S = S + 1;
    end
end

integral = (S/R)*(M - N)*(b - a) + N*(b - a);
pause;
close;
```

```
% Demo 12:
% Demonstration of taking the numerical integral of function F(w) (funcin)
% by using the Monte Carlo methods
% Open file "funcin.m" to modify the function you want to take integral
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;
R = 2000;      % Numbers of samples
a = 0;
b = 10;
theta = 1;
[integral, S] = mcim(a, b, R, theta)
```



```

% Demo 13:
% Demonstration of the Metropolis sampling method
% A number of random points X are generated from the desired
% zero-mean Gaussian distribution
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531
%
% The comparison ratio:
%      p(XT)
% r = -----
%      p(X0)
%
% If r > rand, then X = XT
% else X = X0

clc;
clear;

X = [];
X0 = 0;
delta = 1;
naccept = 0;
n = 0;
while (naccept < 1000)
    n = n + 1;
    XT = X0 + delta*(2*rand(1,1)-1);
    ratio = (exp(-0.5*(XT^2 - X0^2)));
    if (ratio > rand)
        X = [X, XT];
        X0 = XT;
        naccept = naccept + 1;
    end
end

hist(X,20);
hold on;
xlabel('w');
ylabel('p(w)');

pause;
close;

```

```

% Demonstration 14:
% Demonstration of maximum evidence early stopping
% with the Monte Carlo methods
%
% Bayesian Neural Network Classification Toolbox for Matlab
% Copyright (c) Son Nguyen 2003-2006
% The University of Technology, Sydney
% The Faculty of Engineering
% E-mail: stnguyen@eng.uts.edu.au
% Ph: 61-2-9514 7531

clc;
clear;

% Load data
load data_1/data_1.mat
load data_1/data_2.mat;
load data_1/data_3.mat;
load data_1/data_4.mat;
load data_1/data_6.mat;
load data_1/data_7.mat;
load data_1/data_8.mat;
load data_1/data_9.mat;

% Training set
x = [data_1; data_2; data_3; data_4; data_6; data_7];
t = [target_1; target_2; target_3; target_4; target_6; target_7];

% Test set
x1 = [data_8; data_9];
t1 = [target_8; target_9];

runs = 5;
for m = 1:runs
    fprintf(1, ' Training run: %3d \n', m);

    % Network architecture
    nin = 40;           % Number of input nodes
    nhidden = 4;       % Number of hidden nodes
    nout = 4;          % Number of output nodes

    % Initialise the network
    net = netini(nin, nhidden, nout);

    % Initialise weights and biases with specified mean and variance
    mu = 0;
    mu1 = mu*ones(1, nhidden);
    sigma1 = 2*eye(nhidden);
    wbl = mvgd(nin + 1, nhidden, mu1, sigma1);
    net.w1 = wbl(1: nin, :);
    net.b1 = wbl(nin + 1, :);

```

```

mu2 = mu*ones(1, nout);
sigma2 = nout*eye(nout);
wb2 = mvgd(nhidden + 1, nout, mu2, sigma2);
net.w2 = wb2(1: nhidden, :);
net.b2 = wb2(nhidden + 1, :);

% No prior hyperparameters for weight and bias groups
cw1 = 0; cb1 = 0; cw2 = 0; cb2 = 0;
net = hypers(net, cw1, cb1, cw2, cb2);

% Training network
train_steps = 50;
search_steps = 20;

% Minimise the error function
R = 400;          % Number of sampled weights vectors
[net, f, fc, cycles, logevidence]= scgmctrain (net, x, t,...
                                             train_steps, R, x1, t1);

z = netfwd(net, x1);
[C, rate] = confm(z, t1);

% Find the cycle that corresponds to the maximum evidence
for i = 1:numel(logevidence)
    if (logevidence(i) == max(logevidence))
        maxcycle = i;
    end
end

load netf;
Cmax = C1{maxcycle};
ratemax = rate1{maxcycle};
ratemaxevi(m) = ratemax(1);
disp(' ');
zf = netfwd(net, x1);
[Cf, ratef] = confm(zf, t1);
ratefinal(m) = ratef(1);

end

mean_max = mean(ratemaxevi)
mean_final = mean(ratefinal)

% Plot the cycle error
plot(cycles, fc, '- ', 'linewidth', 1);
xlabel('Cycle');
ylabel('Data error');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(fc) fc(maxcycle)];

```

```
plot(k2, kk2, 'r:');
pause;
close;

% Plot the cycle evidence
plot(cycles, logevidence, '-','linewidth', 1);
xlabel('Cycle');
ylabel('Log evidence');
hold on;
k2 = [maxcycle maxcycle];
kk2 = [min(logevidence) max(logevidence)];
plot(k2, kk2, 'r:');
axis([1 (numel(logevidence)) min(logevidence) logevidence(maxcycle) + 20]);
pause;
close;

Cmax = C1{maxcycle}
ratemax = rate1{maxcycle}
maxcycle
```

Appendix F

Implementation of the Bayesian Neural Network based Real-Time Head-Movement Command Detection

This appendix describes an implementation of the BNN based real-time head-movement command detection. The implementation consists of the following steps:

1. Collect the head-movement data from a number of subjects, and then extract the training samples.
2. Train the network using the Advanced Bayesian Neural Network Classification Toolbox for Matlab.
3. After training, save the weights and biases of the trained network in text files.
4. In C environment, read the weights and biases from the text files, and write C codes for classifying the input patterns in real-time.
5. Compile the C codes using the GCC compiler.

The C codes include the following files:

1. “readnet.h” : is a header file for reading the weights and biases of the trained BNN.
2. “readx.h” : is a header file for reading the input vectors.
3. “main.c” : is the main program for classifying the input vectors into four output classes, corresponding to forward, backward, left and right movements.

The text files include the storage of the weights and biases of the trained BNN, and the real-time input pattern.

1. "w1.txt" : contains the weights on the connection from input nodes to hidden nodes
2. "b1.txt" : contains the biases of hidden node.
3. "w2.txt" : contains the weights on the connections from hidden nodes to output nodes.
4. "b2.txt" : contains the biases of output nodes.
5. "x.txt" : contains the input vector updated in real-time.

After compiling, the stand-alone program can operate on any Linux operating system based embedded hardware, such as a Personal Digital Assistance (PDA) or a Digital Signal Processor (DSP).

```

//University of Technology, Sydney
//Copyright (c) Son Nguyen (2003 - 2006)
//E-mail: stnguyen@eng.uts.edu.au
//Ph: 61-2-9514 7531
//readnet.h
//Read the weights and biases of the pre-trained net

#include<stdio.h>
#include<stdlib.h>

char c1[150];
char c2[150];
char c3[150];
char c4[150];
float w11[150];
float w22[150];
float b11[15];
float b22[15];
int i;

void readw1(int kk) //Read weights from input to hidden nodes
{
    FILE *fw1; //Declare a FILE pointer
    fw1 = fopen("w1.txt", "r"); //Open a text file to read
    if (fw1 == NULL) //Check if the file exists or not
    {
        printf("Error: the file does not exist.\n");
    }
    else
    {
        printf("\nFile 'w1.txt' has been successfully opened.\n");
        i = 0;
        while(fgets(c1, 200, fw1) != NULL)
        {
            i = i + 1;
            w11[i] = atof(c1); //Convert a string to float
        }
        fclose(fw1); //Close the file
    }
}

void readb1(int kk) //Read biases of hidden nodes
{
    FILE *fb1; //Declare a FILE pointer
    fb1 = fopen("b1.txt", "r"); //Open a text file to read
    if (fb1 == NULL) //Check if the file exists or not
    {
        printf("Error: the file does not exist.\n");
    }
}

```

```

else
{
    printf("\nFile 'b1.txt' has been successfully opened.\n");
    i = 0;
    while(fgets(c3, 200, fb1) != NULL)
    {
        i = i + 1;
        b11[i] = atof(c3);    //Convert a string to float
    }
    fclose(fb1);            //Close the file
}
}

void readw2(int kk)        //Read weights from hidden to output nodes
{
    FILE *fw2;            //Declare a FILE pointer
    fw2 = fopen("w2.txt", "r"); //Open a text file to read
    if (fw2 == NULL)      //Check if the file exists or not
    {
        printf("Error: the file does not exist.\n");
    }
    else
    {
        printf("\nFile 'w2.txt' has been successfully opened.\n");
        i = 0;
        while(fgets(c2, 200, fw2) != NULL)
        {
            i = i + 1;
            w22[i] = atof(c2);    //Convert a string to float
        }
        fclose(fw2);            //Close the file
    }
}

void readb2(int kk)      //Read biases of output nodes
{
    FILE *fb2;            //Declare a FILE pointer
    fb2 = fopen("b2.txt", "r"); //Open a text file to read
    if (fb2 == NULL)      //Check if the file exists or not
    {
        printf("Error: the file does not exist.\n");
    }
    else
    {
        printf("\nFile 'b2.txt' has been successfully opened.\n");
        i = 0;
        while(fgets(c4, 200, fb2) != NULL)
        {
            i = i + 1;

```



```
        b22[i] = atof(c4);    //Convert a string to float
    }
    fclose(fb2);            //Close the file
}
}
```

```

//University of Technology, Sydney
//Copyright (c) Son Nguyen (2003 - 2006)
//E-mail: stnguyen@eng.uts.edu.au
//Ph: 61-2-9514 7531
//readx.h
//Read the input partern x

#include <stdio.h>
#include <stdlib.h>

char c[150];
char filename[20] = "x.txt";
float xin[150];
int i;

void readxin(int kk) //Read weights from input to hidden nodes
{
    FILE *fx; //Declare a FILE pointer
    fx = fopen(filename, "r"); //Open a text file to read
    if (fx == NULL) //Check if the file exists or not
    {
        printf("Error: the file does not exist.\n");
    }
    else
    {
        printf("\nFile has been successfully opened.\n");
        i = 0;
        while(fgets(c, 200, fx) != NULL)
        {
            i = i + 1;
            xin[i] = atof(c); //Convert a string to float
        }
        fclose(fx); //Close the file
    }
}

```

```

//Copyright (c) Son Nguyen (2003 - 2006)
//University of Technology, Sydney
//Key University Research Centre for Health Technologies
//main.c
//The main program
//The code is compiled using GCC in order to run on Linux embedded system
//
//|-----| |-----| |-----|
//|Real-time input vector |--->|Trained BNN|--->|Real-time actual net outputs|
//|-----| |-----| |-----|

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"readnet.h"
#include"readx.h"

//Declare variables
int nin, nhidden, nout;
int i, j, k;
float w1[100][100];
float w2[30][30];
float b1[10];
float b2[10];
float a1[10];
float y[10];
float a2[10];
float z[10];
float temp;
float x[100];

main()
{
    nin = 40;          //Number of input nodes
    nhidden = 3;      //Number of hidden nodes
    nout = 4;         //Number of output nodes

    printf("Real-Time Bayesian Neural Networks\n");
    printf("For Detecting Head-Movement Commands.\n\n");

    readw1(1); //Open the file for reading w1
    for (j=1; j<=nhidden; j++)
        for(i=1; i<=nin; i++)
        {
            w1[j][i] = w11[(j-1)*nin + i];
            printf("w1[%1d][%2d] : %4.4f\n",j, i, w1[j][i]);
        }

    readb1(1); //Open the file for reading b1

```

```

for(j=1; j<=nhhidden; j++)
{
    b1[j] = b11[j];
    printf("b1[%ld] : %4.4f\n",j, b1[j]);
}

readw2(1); //Open the file for reading w2
for (k=1; k<=nout; k++)
    for(j=1; j<=nhhidden; j++)
    {
        w2[k][j] = w22[(k-1)*nhhidden + j];
        printf("w2[%ld][%2d] : %4.4f\n",k, j, w2[k][j]);
    }

readb2(1); //Open the file for reading b2
for(k=1; k<=nout; k++)
{
    b2[k] = b22[k];
    printf("b2[%ld] : %4.4f\n",k, b2[k]);
}

for(;;) //Endless loop
{
    readxin(1); //Read the input vector x
    for(i=1; i<=nin; i++)
    {
        x[i] = xin[i];
        //printf("x[%ld] : %4.4f\n",i, x[i]);
    }

    //Do forward propagation
    //From input to hidden nodes
    for(j=1; j<=nhhidden; j++)
    {
        a1[j] = b1[j];
        for(i=1; i<=nin; i++)
        {
            a1[j] = a1[j] + w1[j][i]*x[i];
        }
        y[j] = tanh(a1[j]);
    }

    //From hidden to output nodes
    temp = 0;
    for(k=1; k<=nout; k++)
    {
        a2[k] = b2[k];
    }
}

```

```
        for(j=1; j<=nhidden; j++)
        {
            a2[k] = a2[k] + w2[k][j]*y[j];
        }
        temp = temp + exp(a2[k]);
    }

    printf("\nThe actual network outputs:\n");
    for(k=1; k<=nout; k++)
    {
        z[k] = exp(a2[k])/temp;        //Softmax function
        printf("\nz[%1d] : %4.4f\n", k, z[k]);
    }

    printf("\n\n");
}
}
```

Appendix G

Publications related to the Thesis



IEEE 2006 International Conference of the Engineering in Medicine and Biology Society

Engineering Revolution In BioMedicine

Pre-Conference Workshops: August 29-30, 2006

Conference: August 30-Sept. 3, 2006

Conference Site and Hotel:

Marriott at Times Square

New York City, New York, USA

Improved Head Direction Command Classification using an Optimised Bayesian Neural Network

Son T. Nguyen¹, Hung T. Nguyen¹, Philip B. Taylor¹, James Middleton²

¹Key University Research Centre for Health Technologies, Faculty of Engineering,
University of Technology, Sydney, NSW, AUSTRALIA

²Rehabilitation Research Centre, The University of Sydney, NSW, AUSTRALIA

Abstract—Assistive technologies have recently emerged to improve the quality of life of severely disabled people by enhancing their independence in daily activities. Since many of those individuals have limited or non-existing control from the neck downward, alternative hands-free input modalities have become very important for these people to access assistive devices. In hands-free control, head movement has been proved to be a very effective user interface as it can provide a comfortable, reliable and natural way to access the device. Recently, neural networks have been shown to be useful not only for real-time pattern recognition but also for creating user-adaptive models. Since multi-layer perceptron neural networks trained using standard back-propagation may cause poor generalisation, the Bayesian technique has been proposed to improve the generalisation and robustness of these networks. This paper describes the use of Bayesian neural networks in developing a hands-free wheelchair control system. The experimental results show that with the optimised architecture, classification Bayesian neural networks can detect head commands of wheelchair users accurately irrespective to their levels of injuries.

Index Terms— Bayesian neural networks, hands-free control, head movement, power wheelchairs.

I. INTRODUCTION

SINCE many of disabled people have limited or non-existing control from the neck downward, alternative hands-free input modalities have become very important for them to access assistive devices. In hands-free control, head movement has been proven to be a very effective user interface as it can provide a comfortable, reliable and natural way to access the device.

The need to design challenging systems that are able to detect head direction commands of various forms of power wheelchair users leads to the use of neural networks [1-3]. In such a system, a trained multi-layer perceptron neural network is responsible for classifying input head movement signals into four output classes corresponding to forward, backward, left and right head movement in real-time. Since multi-layer perceptron neural networks trained using the Bayesian technique (Bayesian neural networks) can have superior generalisation properties than standard multi-layer

perceptron neural networks in case of finite available training data, they have been used to develop effective head movement classifiers in our hands-free power wheelchair control systems [4, 5].

In this paper, we continue to optimise the classification Bayesian neural networks used to classify head direction commands of wheelchair users. The structure of the paper is organised as follows. The Section II provides an overview of a head movement-based wheelchair control system. In Section III, the formulation of classification Bayesian neural networks is briefly described. Section IV presents the selection of an appropriate architecture of classification Bayesian neural networks for this application by comparing the log evidence of different candidature architectures and how a disabled user-adaptive head movement classification can be performed. Section V provides a discussion and conclusion for this research.

II. SYSTEM OVERVIEW

The wheelchair platform is based on a commercial powered wheelchair. The movement of the user's head is detected by analysing data from a dual-axis accelerometer installed in a cap worn by the user. The head movement data was collected with a sampling period of 100 ms. A pre-trained classification Bayesian neural network is used to classify four gestures in real-time, corresponding to commands for forward, backward, left and right.

The start of the head movement to control the travel direction of the wheelchair is determined to be the point where the deviation from the neutral position reached 25% of the maximum value on the relevant axis. The classification of the movement was determined as the first classification made by the Bayesian neural network after the start of the movement. The input to the Bayesian neural network is comprised of a window of 20 samples from each axis.

The computer interface module consists of the feedback to the user: real-time graphical displays of the accelerometer data allow the user to track the deviation of their head from the neutral position. Boolean outputs from the classifier and the numerical values of the Bayesian neural network inform the user of how the classifier is interpreting their head movement. The success of the system heavily depends on

the network training.

III. CLASSIFICATION BAYESIAN NEURAL NETWORKS

Bayesian learning of multi-layer perceptron neural networks is performed by considering Gaussian probability distributions of the weights which can give the best generalisation [6, 7]. In particular, the weights w in network X are adjusted to their most probable values given the training data D . Specifically, the posterior distribution of the weights can be computed using Bayes' rule as follows

$$p(w|D, X) = \frac{p(D|w, X)p(w|X)}{p(D|X)} \quad (1)$$

where $p(D|w, X)$ is the likelihood function, which contains information about the weights from observations and the prior distribution $p(w|X)$ contains information about the weights from background knowledge. The denominator, $p(D|X)$, is known as the evidence for network X .

Given a set of candidate networks X_i having different numbers of hidden nodes, the posterior probability of each network can be expressed as

$$p(X_i|D) = \frac{p(D|X_i)p(X_i)}{p(D)} \quad (2)$$

If the networks are assumed to be equally probable before any data is observed, then $p(X_i)$ is the same for all networks. Since $p(D)$ does not depend on the network, then the probable network is the one with the highest evidence $p(D|X_i)$. Therefore, the evidence can be used to compare and rank different candidature networks.

Regularisation can be used to prevent any weights becoming excessively large, which can lead to poor generalisation. For a multi-layer perceptron neural network classifier with G groups of weights and biases, a weight decay penalty term proportional to the sum of squares of the weights and biases is added to the data error function E_D to obtain the cost function

$$S = E_D + \sum_{g=1}^G \xi_g E_{w_g} \quad (3)$$

$$E_{w_g} = \frac{1}{2} \|w_g\|^2 \quad (g = 1, \dots, G) \quad (4)$$

where S is called the cost function, ξ_g is a non-negative scalar, sometimes known as a *hyperparameter*, ensuring the distribution of weights and biases in group g and w_g is the

vector of weights and biases in group g .

In network training, the hyperparameters are initialised to be arbitrary small values. The cost function is then minimised using an advanced optimisation technique. When the cost function has reached a local minimum, the hyperparameter ξ_g ($g = 1, \dots, G$) must be re-estimated. This task requires computing the Hessian matrix of the cost function:

$$A = H + \sum_{g=1}^G \xi_g I_g \quad (5)$$

where H is the Hessian matrix of E_D and I_g is the identity matrix, which selects weights in the g th group. The number of 'well-determined' weights γ_g in group g is calculated based on the old value of ξ_g as follows [7]

$$\gamma_g = W_g - \xi_g \text{tr}(A^{-1} I_g) \quad (g = 1, \dots, G) \quad (6)$$

The new value of the hyperparameter ξ_g is then re-estimated as [7]

$$\xi_g = \frac{\gamma_g}{2E_{w_g}} \quad (g = 1, \dots, G) \quad (7)$$

The hyperparameters need to be re-estimated several times until the cost function value ceases to change significantly between consecutive re-estimation periods. After the network training is completed, the values of parameters γ_g and ξ_g are then used to compute the log evidence of network X_i having M hidden nodes as follows [8, 9]

$$\begin{aligned} \ln \text{Ev}(X_i) = & -S + \sum_{g=1}^G \frac{W_g}{2} \ln \xi_g - \frac{1}{2} \ln |A| + \ln M! + M \ln 2 \\ & + \sum_{g=1}^G \frac{1}{2} \left(\frac{4\pi}{\gamma_g} \right) - G \ln(\ln \Omega) \end{aligned} \quad (8)$$

where W_g is the number of weights and biases in group g , and Ω is set to be 10^3 [9]. However, Ω is a minor factor because it is the same for all models and therefore does not effect to the relative comparison of log evidence of different network architectures. Equation (8) is used to compare different networks having different numbers of hidden nodes. The best network will be selected with the highest log evidence.

IV. EXPERIMENTS AND RESULTS

A. Data Acquisition

Data was collected from eight adults, aged between 19 and 56, with approval from the UTS Human Research Ethics Committee and informed consent from the volunteers. Of these, four had high-level spinal cord injuries (C4 and C5) and were not able to use a standard joystick to control a wheelchair. The remaining four did not have conditions affecting their head movement. Data for each person was collected in two periods of ten minutes, with the user being prompted to give a specified movement every 6 seconds. Each specified movement was chosen randomly from the following: forward, backward, left and right. The extracted movement samples of those users are shown in Table I.

B. Optimisation of Network Architecture

The recorded movements of user 7 and 8 were randomly divided into two sets. Each set contained 20 samples of each movement. Training data was taken from the recorded movements of users 1, 2, 3, 4, 5 and 6, corresponding to 480 samples. Different Bayesian neural networks with varying numbers of hidden nodes were trained to select the optimal network architecture. These networks have the following specification:

- four hyperparameters ξ_1 , ξ_2 , ξ_3 and ξ_4 to constrain the magnitudes of the weights on the connection from the input nodes to the hidden nodes, the biases of the hidden nodes, the weights on the connection from the hidden nodes to the output nodes, and the biases of the output nodes;
- 41 inputs, corresponding to 20 samples from x axis, 20 samples from y axis and one augmented input with a constant value of 1;
- four outputs, each corresponding to one of the classes: forward, backward, left and right movement.

For a given number of hidden nodes, ten networks with different initial conditions were trained. The training procedure was implemented as follows:

1. The weights and biases in four different groups were initialised by random selections from zero-mean, unit variance Gaussians and the initial hyperparameters were chosen to be small values.
2. The network was trained to minimise the cost function S using the quasi-Newton algorithm [5].
3. When the network training had reached a local minimum, the values of the hyperparameters were re-estimated according to equation (6) and (7).
4. Steps 2 and 3 were repeated until the cost function value was smaller than a pre-determined value and did not change significantly in subsequent re-estimations.

The performances of the trained networks were tested using a test set taken from the first set of movement samples of user 7 and 8. As shown in Fig.1, the networks having three hidden nodes have the highest evidence. This means that three hidden nodes are sufficient to solve the problem. As shown in Fig.2, these networks also have low test errors (misclassification percentages). The optimisation of Bayesian neural network architecture is extremely important for on-line network training systems because incorporating with the quasi-Newton training algorithm it can contribute to the least network training time while still maintaining the best generalisation for the network.

TABLE I
EXTRACTED MOVEMENT SAMPLES

User	Forward	Backward	Left	Right	Injury Level
1	20	20	20	20	-
2	20	20	20	20	-
3	20	20	20	20	-
4	20	20	20	20	-
5	20	20	20	20	C5
6	20	20	20	20	C4
7	20	20	20	20	C4
8	20	20	20	20	C5

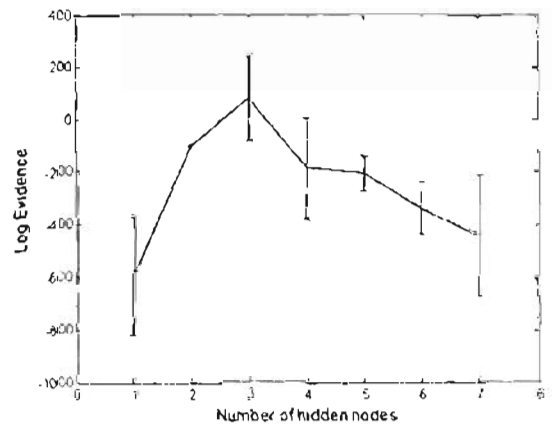


Fig 1. Log evidence versus number of hidden nodes. The solid curve shows the evidence averaged over the ten networks.

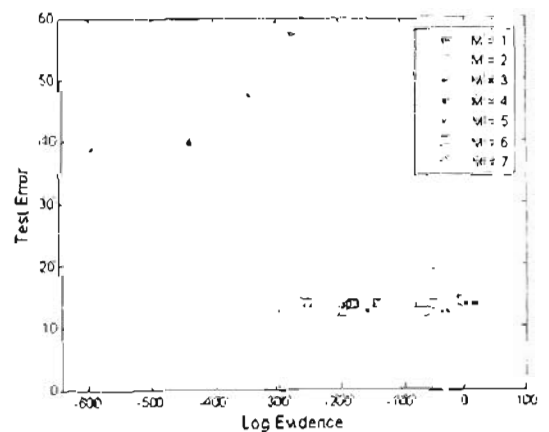


Fig 2 Test error versus log evidence. Every symbol appears ten times, once for each of the ten networks trained.

C. Head Movement Classification

1) Experiment 1:

A Bayesian neural network classifier having three hidden nodes was trained using the training data described in section IV-B. The performance of the trained network was tested using the first set of movement samples of user 7 and 8. The confusion matrix in Table II shows how accurately the trained network classifies each type of movement and the trained network can classify all samples in the test set with an accuracy of 85%.

2) Experiment 2:

More training data was taken from the second set of movement samples of user 7 and 8. The Bayesian neural network was trained using the same procedure in Experiment 1. The performance of the trained network was also tested using the first set of movement samples of user 7 and 8. Similarly, a confusion matrix used to evaluate the performance of the trained network as seen in Table III and the network can classify all samples in the test set with a success rate of 93.75%.

The classification results of Experiment 1 and 2 are summarised in Table IV. Especially, it can be seen that very high sensitivity (true positive) and specificity (true negative) have been achieved for Experiment 2. This means that the performance of the trained network has been significantly improved as more movement samples have been included to train the network.

TABLE II
CONFUSION MATRIX IN EXPERIMENT I

		Predicted Classification			
		Forward	Backward	Left	Right
Actual Classification	Forward	15	0	1	4
	Backward	0	17	3	0
	Left	4	0	16	0
	Right	0	0	0	20
Accuracy (%)		85			

TABLE III
CONFUSION MATRIX IN EXPERIMENT II

		Predicted Classification			
		Forward	Backward	Left	Right
Actual Classification	Forward	18	0	1	1
	Backward	0	19	1	0
	Left	2	0	18	0
	Right	0	0	0	20
Accuracy (%)		93.75			

TABLE IV
SENSITIVITY AND SPECIFICITY OF THE NETWORK

Experiment	Sensitivity	Specificity
1	0.85	0.95
2	0.9375	0.97917

V. DISCUSSION AND CONCLUSION

The results obtained show that Bayesian neural networks can be used to classify head movement accurately. The use of three hidden nodes is an optimal choice for the network architecture as it and the fast training quasi-Newton algorithm can make a significant reduction of on-line network training time. This number of hidden nodes guarantees the best generalisation of the trained network. When the Bayesian neural network was trained using head movement data of the six users, it can classify head movements of two new disabled persons with 85% accuracy. However, if the network was trained further with additional head movement samples of those two disabled persons, it can classify their head movement with a high accuracy of 93.75%. In other words, the network is able to provide an on-line adaptation to the head movement of new disabled wheelchair users.

ACKNOWLEDGMENT

This research was supported partly by the ARC LIEF grant (LE0454081), ARC Discovery grant (DP0666942) and Key University Research Centre for Health Technologies, UTS.

REFERENCES

- [1] T. Joseph and H. Nguyen, "Neural network control of wheelchairs using telemetric head movement," *Proceedings of the 20th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society*, vol. 5, pp. 2731 - 2733, 1998.
- [2] P. B. Taylor and H. T. Nguyen, "Performance of a head-movement interface for wheelchair control," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, pp. 1590 - 1593, 2003.
- [3] H. T. Nguyen, L. M. King, and G. Knight, "Real-time head movement system and embedded Linux implementation for the control of power wheelchairs," *26th Annual International Conference of the Engineering in Medicine and Biology Society*, vol. 2, pp. 4892 - 4895, 2004.
- [4] S. T. Nguyen, H. T. Nguyen, and P. Taylor, "Hands-Free Control of Power Wheelchairs using Bayesian Neural Networks," *Proceeding of IEEE Conference on Cybernetics and Intelligent Systems*, pp. 745 - 749, 2004.
- [5] S. T. Nguyen, H. T. Nguyen, and P. Taylor, "Bayesian Neural Network Classification of Head Movement Direction using Various Advanced Optimisation Training Algorithms," *Proceedings of the first IEEE / RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, Tuscany Italy, 2006*, 2006.
- [6] D. J. C. MacKay, "A practical Bayesian Framework for Backpropagation Networks," *Computation and Neural Systems*, vol. 4, pp. 448-472, 1992b.
- [7] D. MacKay, "The Evidence Framework Applied to Classification Networks," *Neural Computation*, vol. 4, pp. 720-736, 1992.
- [8] W. D. Penny and S. J. Roberts, "Bayesian neural networks for classification: how useful is the evidence framework," *Neural Networks*, vol. 12, pp. 877-892, 1999.
- [9] H. H. Thodberg, "A review of Bayesian neural networks with an application to near infrared spectroscopy," *IEEE Transactions on Neural Networks*, vol. 7, pp. 56 - 72, 1996.



BioRob 2006

The first IEEE / RAS-EMBS International Conference
on Biomedical Robotics and Biomechatronics

Pisa, Tuscany, Italy

February 20-22, 2006

Bayesian Neural Network Classification of Head Movement Direction using Various Advanced Optimisation Training Algorithms*

Son T. Nguyen

*Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
Son.NguyenThanh@uts.edu.au*

Hung T. Nguyen

*Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
Hung.Nguyen@uts.edu.au*

Philip B. Taylor

*Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
Philip.Taylor@uts.edu.au*

Abstract - Head movement is one of the most effective hands-free control modes for powered wheelchairs. It provides the necessary mobility assistance to severely disabled people and can be used to replace the joystick directly. In this paper, we describe the development of Bayesian neural networks for the classification of head movement commands in a hands-free wheelchair control system. Bayesian neural networks allow strong generalisation of head movement classifications during the training phase and do not require a validation data set. Various advanced optimisation training algorithms are explored. Experimental results show that Bayesian neural networks can be developed to classify head movement commands by abled and disabled people accurately with limited training data.

Index Terms - Bayesian neural networks; head-movement classification; powered wheelchair.

I. INTRODUCTION

The use of joysticks as a form of control for severely disabled people is a very demanding task. These people have severe mobility disabilities such as cerebral palsy, tetraplegia, etc. Head movement is one of the most effective hands-free modes for the control of powered wheelchairs as it provides the necessary mobility assistance.

Currently, it remains a major and elusive task to develop an innovative head movement interface for a human-machine system which assures the safety of the operator, remains unobtrusive, is easy to learn and can easily be adapted to a new operator with different mobility impairments. Many people with spinal cord injury who use head controls have very poor control of the musculature supporting the upper body and neck, and consequently control of head position is marginal at the best of times. Other challenges arise from changed position of the body in relation to the wheelchair.

A feed-forward neural network classically trained using back-propagation can be viewed as an effective classifier for head movements of severely disabled people [1], [2], [3]. However, the main disadvantage of standard neural networks is the potential of poor generalisation when facing with limited training data. Recently, Bayesian techniques have been applied to neural networks to improve the accuracy and robustness of neural network classifiers. In our previous research [4], it was shown that Bayesian neural networks

could be used to classify head movement commands consistently with limited training data.

In this paper, we continue to explore the properties of Bayesian neural networks in a hands-free control wheelchair control system using various advanced optimisation training algorithms. In the near future, the optimal Bayesian neural network will provide the ability for the system to be trained on-line for each individual operator, irrespective of his/her disability.

Section II describes the formulation of Bayesian neural network classification. In Section III, we briefly discuss various advanced optimisation algorithms used for the training of Bayesian neural networks. Section IV shows experimental results of head movement classifications in a hands-free wheelchair control system. Section V provides a discussion of these results and directions for the future development of the system.

II. BAYESIAN NEURAL NETWORK CLASSIFICATION

Bayesian neural networks were firstly introduced by MacKay [5], [6], [7]. A Bayesian neural network has the following main benefits compared to a standard neural network:

- Its network training adjusts weight decay parameters automatically to optimal values for the best generalisation. The adjustment is done during training, so the computational intensive search for the weight decay parameters is no longer required.
- Its training converges to different local minima and networks with different numbers of hidden nodes can be compared and ranked.
- As no separate validation set is required, all available data can be used for training.

A. Multi-Layer Perceptron Neural Networks

Multi-layer perceptron (MLP) neural networks are widely used in engineering applications. These networks take in a vector of real inputs, x_i , and from them compute one or more values of the output layer, $z_k(x, w)$. With a one hidden layer network, as shown in Fig 1, the value of the k th output is computed as follows:

* This work was supported by an ARC LIEF grant (LE0454081).

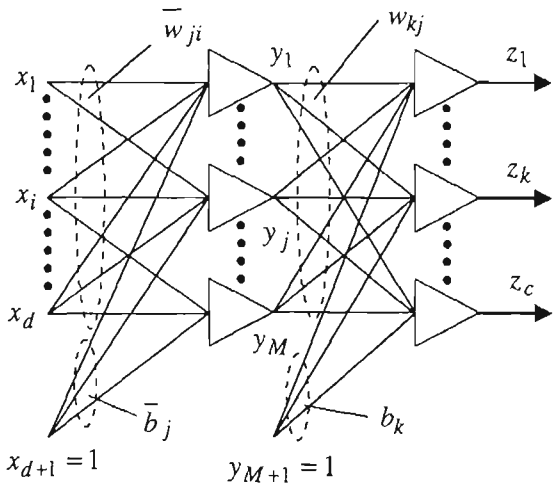


Fig.1 A MLP neural network

$$z_k(x, w) = f_0 \left(b_k + \sum_{j=1}^M w_{kj} \tanh \left(\bar{b}_j + \sum_{i=1}^d \bar{w}_{ji} x_i \right) \right) \quad (1)$$

Here, \bar{w}_{ji} is the weight on the connection from input unit i to hidden unit j ; similarly, w_{kj} is the weight on the connection from hidden unit j to output unit k . \bar{b}_j and b_k are the biases of the hidden and output units; f_0 is the output layer activation function.

B. Regularisation

In neural networks, appropriate regularisation can be used to prevent any weights becoming too large because large weights may give poor generalisation for new cases. Therefore, a weight decay term is added to the data error function E_D to penalise large weights. Specifically, for classification problems, we have:

$$S(w) = E_D + \sum_{g=1}^G \xi_g E_{W_g} \quad (2)$$

where $S(w)$ is the total error function, ξ_g is a non-negative parameter for the distribution of other parameters (weights and biases) and known as a *hyperparameter* and E_{W_g} is the weight error for the g th group of weights and biases, and G is the number of groups of weights and biases in the neural network.

C. Bayesian Inference

The adaptive parameters of neural networks (weights and biases) can be conveniently grouped into a single W -dimensional weight vector w . According to the Bayesian inference, the posterior distribution of the weight vector w of a neural network given a data set D is given by

$$p(w | D, \psi) = \frac{p(D | w, \psi) p(w | \psi)}{p(D | \psi)} \quad (3)$$

where $\psi = \{\xi_1, \dots, \xi_G\}$ and equation (3) is the first level of the inference. $p(w | \psi)$ is the weight prior determined using the theory of prior. According to [7], the prior distribution of the weights is given by

$$p(w | \psi) = \frac{1}{Z_W(\psi)} \exp \left(- \sum_{g=1}^G \xi_g E_{W_g} \right) \quad (4)$$

$$Z_W(\psi) = \prod_{g=1}^G \left(\frac{2\pi}{\xi_g} \right)^{W_g/2} \quad (5)$$

where $Z_W(\psi)$ is the normalisation constant and W_g is the number of weights in the g th group. $p(D | w, \psi)$ is the dataset likelihood and $p(D | \psi)$ is the evidence for ψ or the normalisation factor. If the dataset is identically independently distributed (i.i.d), the dataset likelihood is

$$p(D | w, \psi) = \exp(-E_D) \quad (6)$$

Assuming that the total error function $S(w)$ has a single minimum at the most probable weight vector w_{MP} and $S(w)$ can locally be approximated as a quadratic form obtained by the second-order Taylor series expansion of $S(w)$,

$$S(w) \approx S(w_{MP}) + \frac{1}{2} (w - w_{MP})^T A (w - w_{MP}) \quad (7)$$

The matrix A is the Hessian matrix of the total error function at w_{MP} :

$$A = \nabla \nabla S(w_{MP}) = H + \sum_{g=1}^G \xi_g I_g \quad (8)$$

where $H = \nabla \nabla E_D(w_{MP})$ is the Hessian matrix of the data error function at w_{MP} and I_g is the diagonal matrix having ones along the diagonal that picks off weights in the g th group. After training phase, the posterior distribution of weights can be derived as:

$$p(w | D, \psi) = \frac{1}{Z_S(\psi)} \exp(-S(w)) \quad (9)$$

where $Z_S(\psi)$ is the normalisation constant for the approximating Gaussian, and is therefore given by

$$Z_S(\psi) \approx \exp(-S(w_{MP})) (2\pi)^{W/2} (\det A)^{-1/2} \quad (10)$$

Again, using the Bayes' theorem, we can express the posterior distribution of the hyperparameters as

$$p(\psi | D) = \frac{p(D | \psi) p(\psi)}{p(D)} \equiv p(D | \psi) p(\psi) \quad (11)$$

where $p(\psi)$ is the prior distribution of the hyperparameters and simply we assume that this distribution is uniform. It

will thus be ignored subsequently, because to infer the value of the hyperparameter, we only seek the values of the hyperparameters to maximise $p(D|\psi)$.

Rearranging (3), we have the following form:

$$p(D|\psi) = \frac{p(D|w, \psi)p(w|\psi)}{p(w|D, \psi)} \quad (12)$$

Since all the terms in the right-side of equation (12) are determined from (6), (4) and (9), equation (12) yields

$$p(D|\psi) = \frac{Z_S(\psi)}{Z_W(\psi)} \quad (13)$$

Taking the derivative of $\ln p(D|\psi)$ with respect to ξ_g

$$\frac{\partial}{\partial \xi_g} \ln p(D|\psi) = \frac{W_g}{2\xi_g} - E_{W_g} - \frac{1}{2} \text{tr}(A^{-1})I_g \quad (14)$$

Let this derivative be zero, we can determine ξ_g as follows

$$2\xi_g E_{W_g} = W_g - \xi_g \text{tr}(A^{-1})I_g \quad (15)$$

The right-hand side is equal to a value γ_g defined as

$$\gamma_g = W_g - \xi_g \text{tr}(A^{-1})I_g \quad (16)$$

γ_g is called the number of well-determined parameters in weight group g . Substituting (16) into (15) and rearranging (15), we have

$$\xi_g = \frac{\gamma_g}{2E_{W_g}} \quad (17)$$

The terms ξ_g and γ_g are used with some formulas to compute the logarithm of the evidence in Bayesian model comparison. The optimal model is selected corresponding to the highest logarithm of the evidence. The details of this task can be referred to [8], [9].

III. PARAMETER OPTIMISATION ALGORITHMS FOR BAYESIAN NEURAL NETWORKS

The main problem when training neural networks is that usually suitable values for the learning rate and momentum must be chosen. As this procedure is clearly inefficient, we focus on fast training algorithms which can automatically determine the search direction and step size. In this section, three advanced training algorithms for Bayesian neural network classifiers are developed: conjugate gradient, quasi-Newton and scaled conjugate gradient algorithm.

A. Conjugate Gradient Algorithm

The conjugate gradient algorithm starts by searching in the negative gradient on the first iteration. At the m th step, a line search is performed to find the step size α_m

$$\alpha_m = \frac{g_m^T d_m}{d_m^T A d_m} \quad (18)$$

where g_m and d_m are the gradient and search direction at step m .

The new search direction is then given by

$$d_{m+1} = -g_{m+1} + \beta_m d_m \quad (19)$$

where β_m can be determined using the Polak-Rebire formula as follows

$$\beta_m = \frac{(g_{m+1} - g_m)^T g_{m+1}}{g_{m+1}^T g_m} \quad (20)$$

B. Quasi-Newton Algorithm

Newton's method is an alternative to the conjugate gradient method for fast optimisation. The basic step of this method is based on Newton's formula:

$$w_{m+1} - w_m = -H^{-1}(g_{m+1} - g_m) \quad (21)$$

However, the inverse Hessian matrix $F = H^{-1}$ can be approximated using a class of algorithms called the quasi-Newton method such as the most successful Broyden, Fletcher, Goldfarb and Shanno (BFGS) method

$$F_{m+1} = F_m + \frac{pp^T}{p^T v} - \frac{(F_m v)v^T F_m}{v^T F_m v} + (v^T F_m v) u u^T \quad (22)$$

where p , v and u are defined as follows

$$p = w_{m+1} - w_m; v = g_{m+1} - g_m; u = \frac{p}{p^T v} - \frac{F_m v}{v^T F_m v} \quad (23)$$

C. Scaled Conjugate Gradient Algorithm

In the conjugate gradient algorithm, as the total error function $S(w)$ is not a quadratic form then the Hessian matrix A may not be positive definite and in this case the parameter update formula (18) may increase the function value. This can be overcome by adding a non-negative multiple λ_m of the unit matrix to the Hessian A to obtain $A + \lambda_m I$. The expression (18) becomes

$$\alpha_m = \frac{g_m^T d_m}{d_m^T A d_m + \lambda_m \|d_m\|^2} \quad (24)$$

The denominator of (24) can be written as

$$\delta_m = d_m^T A d_m + \lambda_m \|d_m\|^2 \quad (25)$$

If $\delta_m < 0$, we can increase the value of λ_m in order to make $\delta_m > 0$. Let the raised value of λ_m be $\bar{\lambda}_m$, then the corresponding raised value of δ_m is given by

$$\bar{\delta}_m = \delta_m + (\bar{\lambda}_m - \lambda_m) \|d_m\|^2 \quad (26)$$

In order to $\bar{\delta}_m > 0$, Moller [10] chooses

$$\bar{\lambda}_m = 2 \left(\lambda_m - \frac{\delta_m}{\|d_m\|^2} \right) \quad (27)$$

Substituting (27) into (26) gives

$$\bar{\delta}_m = -\delta_m + \lambda_m \|d_m\|^2 = -d_m^T A d_m \quad (28)$$

This value is positive and used as the denominator in (24) to compute the step-size α_m . In order to find λ_{m+1} , a comparison parameter is firstly defined as

$$\Delta_m = \frac{2[S(w_m) - S(w_m + \alpha_m d_m)]}{\alpha_m d_m^T d_m} \quad (29)$$

Then the value of λ_{m+1} can be adjusted using the following prescriptions:

- If $\Delta_m > 0.75$, then $\lambda_{m+1} = \lambda_m / 2$
- If $0.25 < \Delta_m < 0.75$, then $\lambda_{m+1} = \lambda_m$
- If $\Delta_m < 0.25$, then $\lambda_{m+1} = 4\lambda_m$
- If $\Delta_m < 0$, then $\lambda_{m+1} = 4\lambda_m$ and take no step

IV. HANDS-FREE CONTROL OF POWERED WHEELCHAIRS USING HEAD MOVEMENT

Head movement is one of the most effective hands-free control modes for powered wheelchairs. Joseph [2] reported the use of standard neural networks to classify head movement of disabled users. Taylor [1] reported the performance of head movement interface for wheelchair control accompanied by a standard neural network classifier. Nguyen [3] developed a real-time head-movement system using an embedded neural network Linux implementation. In this paper, we use Bayesian neural networks to classify head movements accurately with limited training data.

TABLE I
EXTRACTED MOVEMENT SAMPLES

User	Forward	Backward	Left	Right	Injury Level
1	20	20	20	20	-
2	20	20	20	20	-
3	20	20	20	20	-
4	20	20	20	20	-
5	20	20	20	20	C5
6	20	20	20	20	C4
7	20	20	20	20	C4
8	20	20	20	20	C5

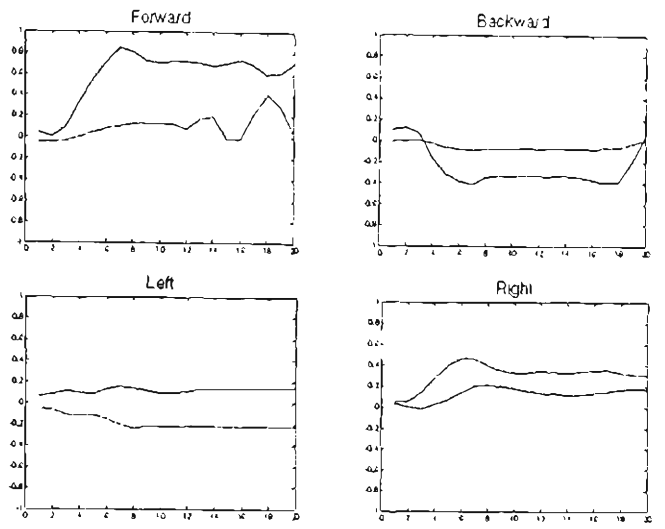


Fig.2 Windowed sample patterns of user 1
(Solid line-x data; dotted line-y data)

In this system, a dual-axis accelerometer is installed in a cap worn by the user to measure head position. A Bayesian neural network classifier is designed to detect four movements: forward, backward, left and right. The success of this system highly depends on the training of the neural network classifier. The training algorithm modifies the classifier's parameters to reduce the difference between the actual and target outputs of the classifier. The computer interface module provides the following important feedback to users. Real-time graphical displays of the accelerometer data allow the user to track the deviation of his/her head from the neutral position. In addition, Boolean outputs from the classifier and the numerical values of the neural network inform the user how the classifier is interpreting their head movements.

A. Data Acquisition

Head movement data were collected from eight adults aged between 19 and 56, with the approval from the UTS Human Research Ethics Committee. Four users had high-level spinal cord injuries (C4 and C5) and were unable to use a standard joystick. The remaining four did not have conditions affecting their head movement. The extracted movement samples of those users are shown in Table I. The movement of the user's head is detected by analysing data from the accelerometer collected using a sampling period of 100 ms. The input to the Bayesian neural network is comprised of a window of 20 samples from each axis as shown in Fig 2. A pre-trained Bayesian neural network was used to classify the windowed data as corresponding to four types of head movement: forward, backward, left and right.

B. Network Architecture

The Bayesian neural network used to classify head movement has the following architecture:

- 41 inputs, corresponding to 20 samples from x axis, 20 samples from y axis and one augmented input with a constant value of 1
- three hidden neurons
- four outputs, each corresponding to one of the classes: forward, backward, left and right.

There are four hyperparameters ξ_1 , ξ_2 , ξ_3 and ξ_4 corresponding optimal distributions from the weights between the input nodes to the hidden nodes, the bias input node to the hidden nodes, the hidden nodes to the output nodes, and the bias hidden node to the output nodes.

C. Experiment 1

All the available data were divided into two subsets: the first half for training and the second half for testing. The training procedure was then implemented as follows:

1. The weights were initialised randomly and choosing initial values for the hyperparameters.
2. The network was trained to minimise the total error function $S(w)$ using optimal training algorithms. For each algorithm, ten networks were trained and the averaged training time was then computed. Fig.3 shows that quasi-Newton and scaled conjugate gradient algorithms have the fastest convergence.
3. When the network training has reached a local minimum, the values of the hyperparameters were re-estimated

$$\xi_g^{new} = \frac{\gamma_g^{old}}{2E_{W_g}}$$

Table II shows the change of these parameters according to the number of hyperparameter re-estimation periods.

4. Steps 2 and 3 were repeated until convergence was achieved (the total error term was smaller than a pre-determined value and did not change significantly in subsequent iterations).

As the quasi-Newton training algorithm is the most effective in terms of computational time with a similar total error term, it was used to train the Bayesian neural network. The performance of the Bayesian neural network after training was obtained using the test subset. The confusion matrix in Table III shows that the network can classify head movement with an accuracy of 99.375%.

D. Experiment 2

The training data were taken from movement samples of users 1, 2, 5 and 6. A Bayesian neural network was obtained using the same procedure as Experiment 1. The quasi-Newton algorithm was also used to train the network. The performance of this Bayesian neural network was then obtained using the test data from users 3, 4, 7 and 8. The confusion matrix in Table IV shows that this Bayesian neural network can classify head movement with a success of 95.625%.

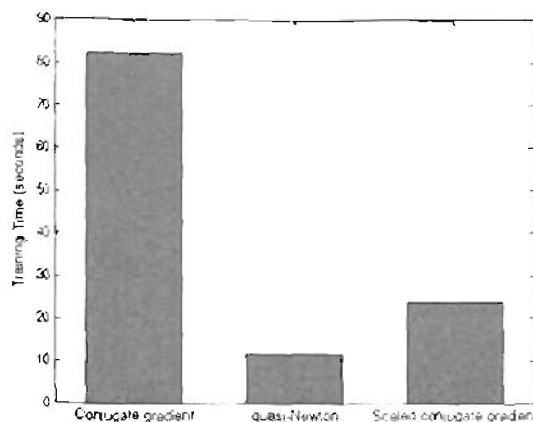


Fig.3 Averaged network training time of different algorithms in Experiment 1

TABLE II
THE CHANGE OF HYPERPARAMETERS ACCORDING TO THE PERIODS OF RE-ESTIMATION IN EXPERIMENT I

Periods	ξ_1	ξ_2	ξ_3	ξ_4
1	0.22914	0.74056	0.033379	0.04562
2	0.85683	10.738	0.028076	0.11036
3	1.7719	48.878	0.019537	0.21202

TABLE III
CONFUSION MATRIX IN EXPERIMENT I

		Predicted Classification				
		Movement	Forward	Backward	Left	Right
Actual Classification	Forward		79	0	0	1
	Backward		0	80	0	0
	Left		0	0	80	0
	Right		0	1	0	79
Accuracy (%)			99.375			

TABLE IV
CONFUSION MATRIX IN EXPERIMENT II

		Predicted Classification				
		Movement	Forward	Backward	Left	Right
Actual Classification	Forward		75	0	0	5
	Backward		0	76	4	0
	Left		2	0	77	1
	Right		0	2	0	78
Accuracy (%)			95.625			

TABLE V
SENSITIVITY, SPECIFICITY, POSITIVE PREDICTIVE VALUE (PPV) AND NEGATIVE PREDICTIVE VALUE (NPV) OF THE BAYESIAN NEURAL NETWORKS

	Sensitivity	Specificity	PPV	NPV
Experiment 1	0.99375	0.99792	0.99379	0.99792
Experiment 2	0.95625	0.98542	0.95689	0.98547

The classification results of the Experiments 1 and 2 are summarised in Table V. It can be seen that very high sensitivity (true positive) and specificity (true negative) have been achieved for both experiments. The effectiveness of the quasi-Newton training algorithm in terms of the overall computational time with similar accuracy will allow us to develop an on-line adaptive Bayesian neural network for each individual disabled operator in the near future.

V. DISCUSSION

The results obtained show that Bayesian neural networks can be used to classify head movement accurately. The classification results are consistent using various forms of training algorithms. If all eight users (4 abled and 4 disabled persons) were trained, the Bayesian neural network classifies the test set very accurately (99.4% accuracy). If the training data were taken from only four users (2 abled and 2 disabled persons), the obtained Bayesian neural network classifies the test set from the other four users with an excellent accuracy (95.6% accuracy).

It is clear that the Bayesian neural network training allows complex models to be developed without the overfitting problems that can occur with standard neural network training. In addition, good generalisation ability of the networks accompanied by the fastest network training algorithm holds promise to the development of an effective on-line adaptive training framework for Bayesian neural network head movement classifiers in the near future.

VI. CONCLUSION

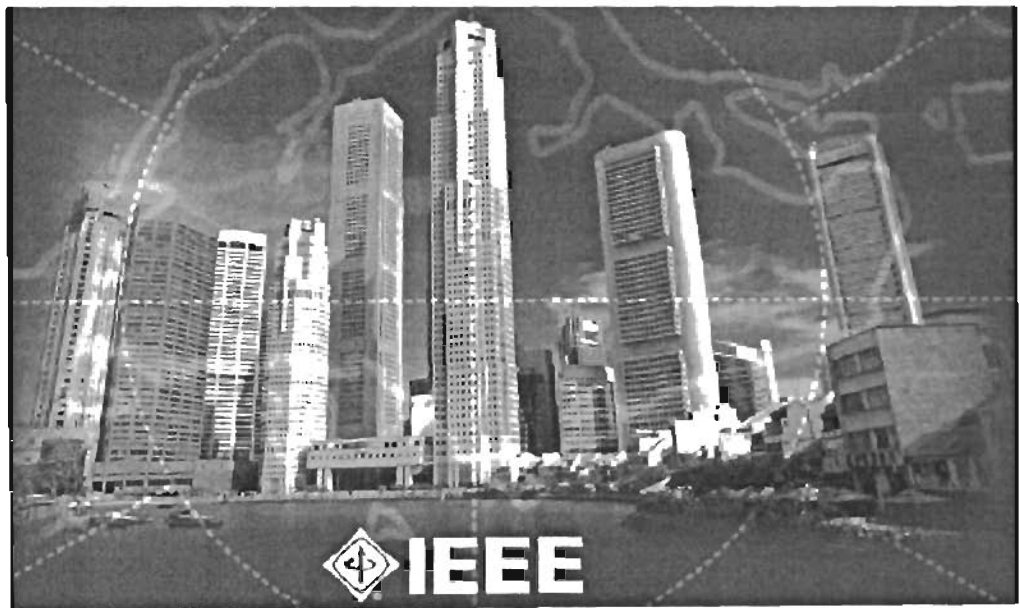
We have developed various Bayesian neural networks for the classification of head movement direction with an excellent overall accuracy. We have also shown that Bayesian neural networks allows complex models to be developed without the requirement for a validation set to overcome the overfitting problem that can occur with standard neural network training.

REFERENCES

- [1] P. B. Taylor and H. T. Nguyen, "Performance of a head-movement interface for wheelchair control," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, pp. 1590 - 1593, 2003.
- [2] T. Joseph and H. T. Nguyen, "Neural network control of wheelchairs using telemetric head movement," *Proceedings of the 20th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society*, vol. 5, pp. 2731 - 2733, 1998.
- [3] H.T. Nguyen, L.M. King and G. Knight, "Real-time head-movement system and embedded Linux implementation for the control of power wheelchair", *Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, San Francisco, USA, 1-5 September 2004, pp. 4892-4895
- [4] S. Nguyen, H. Nguyen, and P. Taylor, "Hands-Free Control of Power Wheelchairs using Bayesian Neural Networks," *Proceeding of IEEE Conference on Cybernetics and Intelligent Systems*, pp. 745 - 749, 2004.
- [5] J. C. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, pp. 415 - 447, 1992.
- [6] J. C. MacKay, "A practical Bayesian Framework for Backpropagation Networks," *Computation and Neural Systems*, vol. 4, pp. 448-472, 1992.
- [7] MacKay, "The Evidence Framework Applied to Classification Networks," *Neural Computation*, vol. 4, pp. 720 -736, 1992.
- [8] H. H. Thodberg, "A review of Bayesian neural networks with an application to near infrared spectroscopy," *IEEE Transactions on Neural Networks*, vol. 7, pp. 56 - 72, 1996.
- [9] M. Bishop, "Neural networks for pattern recognition," *Oxford : Clarendon Press ; New York : Oxford University Press*, 1995.
- [10] M. F. Moller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning," *Neural Networks*, vol. 6, pp. 525 - 533, 1993.

**2004 IEEE Conference on Cybernetics and
Intelligent Systems**

**2004 IEEE Conference on Robotics, Automation
and Mechatronics**



**December 1-3, 2004
Traders Hotel, Singapore**

Hands-Free Control of Power Wheelchairs using Bayesian Neural Network Classification

Son Nguyen

Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
E-mail: Son.NguyenThanh@uts.edu.au

Hung Nguyen

Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
E-mail: Hung.Nguyen@uts.edu.au

Philip Taylor

Faculty of Engineering
University of Technology, Sydney
Broadway NSW 2007 Australia
E-mail: Philip.B.Taylor@uts.edu.au

Abstract—This paper describes the formulation and implementation of Bayesian neural networks for head-movement classification in a hands-free wheelchair navigation system. Bayesian neural network training adjusts the weight decay parameters automatically to their near-optimal values that give the best generalisation. Moreover, no separate validation set is used so all available data can be used for training. Experimental results are presented showing that Bayesian neural network can classify the head movement accurately.

Keywords— Bayesian neural network; head-movement classification; power wheelchairs

I. INTRODUCTION

Power wheelchairs are traditionally used by people with severe mobility disabilities such as cerebral palsy, tetraplegia, etc. However, the operation of these wheelchairs can still be a difficult and demanding task. Head movement is a natural form of pointing and can be used to directly replace the joystick whilst still allowing for similar control.

Through the use of standard feed-forward neural networks and head movement methodology, hands-free trainable control methods for power wheelchairs were developed earlier [1, 2]. In this paper, we explore another advanced neural network which promises to provide an ideal platform for hands-free wheelchair control.

Bayesian neural networks or Bayesian framework for back-propagation networks were firstly introduced by MacKay [3, 4]. They were also used for classification [5]. Bayesian neural networks have the following main benefits compared to standard neural networks:

- Bayesian neural network training adjusts the weight decay parameters automatically to their near-optimal values that give the best generalisation. The adjustment is done during training, so the tedious and computer-expensive search for weight decay parameters is no longer needed.
- Network training converged to different local minima and networks with different numbers of nodes, layers and inputs can be compared and ranked.

- No separate validation set is used. Hence all available data can be used for training.

Bayesian neural network theory is used here in a human-machine interface application, in particular for head-movement classification of hands-free commands in a power wheelchair control system.

II. BAYESIAN NEURAL NETWORKS

A. Multi-Layer Perceptron Networks

Multi-layer perceptron (MLP) neural networks are widely used in engineering applications. These networks take in a vector of real inputs, x_i , and from them compute one or more values of activation of the output layer, $a_k(x, w)$. With a one hidden layer network, as shown in Fig. 1, the activation of the output layer is computed as follows:

$$a_k(x) = b_k + \sum_{j=1}^m w_{kj} \tanh\left(\bar{b}_j + \sum_{i=1}^d \bar{w}_{ji} x_i\right) = b_k + \sum_{j=1}^m w_{kj} y_j \quad (1)$$

Here, \bar{w}_{ji} is the weight on the connection from input unit i to hidden unit j ; similarly, w_{kj} is the weight on the connection from hidden units j to output unit k . The \bar{b}_j and b_k are the biases of the hidden and output units. These weights and biases are the parameters of the neural network.

MLP networks can be used to define probabilistic models for regression and classification tasks by using the network outputs to define the conditional distribution for one or more targets. In c class classification problems, the target variables are discrete class labels indicating one of c possible classes. The *softmax* (generalised logistic) model can be used to define the conditional probabilities of the various classes of a network with c output units as follows:

$$z_k(x) = \frac{\exp(a_k(x))}{\sum_{k=1}^c \exp(a_k(x))} \quad (2)$$

$$S(w) \approx S(w_{MP}) + \frac{1}{2}(w - w_{MP})^T A (w - w_{MP}) \quad (12)$$

There is no first-order term, since it is assumed that w_{MP} is a local minimum of the error function, so $\partial S(w_{MP})/\partial w_i = 0$ for all coordinates. The matrix A is the Hessian matrix of the total error function at w_{MP} :

$$A = \nabla \nabla S(w_{MP}) = \nabla \nabla E_D(w_{MP}) + \alpha I \quad (13)$$

Where $\nabla \nabla E_D(w_{MP})$ is the Hessian matrix of the network function at w_{MP} and I is the unit matrix. After training phase, the posterior distribution of weights can be derived as follows:

$$p(w|D, \alpha) = \frac{1}{Z_S(\alpha)} \exp(-S(w)) \quad (14)$$

Where Z_S is the normalisation constant for the approximating Gaussian, and is therefore given by

$$Z_S(\alpha) = \exp(-S(w_{MP})) (2\pi)^{W/2} (\det A)^{-1/2} \quad (15)$$

Again, using Bayes' theorem, we can express the posterior distribution of the hyperparameter as

$$p(\alpha|D) = \frac{p(D|\alpha)p(\alpha)}{p(D)} \propto p(D|\alpha)p(\alpha) \quad (16)$$

Where $p(\alpha)$ is the prior distribution of the hyperparameter, or *hyperprior*, and simply we assume that this distribution is uniform. It will thus be ignored subsequently, because to infer the value of the hyperparameter, we only seek the value of the hyperparameter to maximise $p(D|\alpha)$.

Rearranging (9), we have the following form:

$$p(D|\alpha) = \frac{p(D|w, \alpha)p(w|\alpha)}{p(w|D, \alpha)} \quad (17)$$

Since all the terms in the right-side of equation (17) are determined from (10), (8) and (14), equation (17) yields

$$p(D|\alpha) = \frac{Z_S(\alpha)}{Z_w(\alpha)} \quad (18)$$

Taking the derivative of $\ln p(D|\alpha)$ with respect to α , we have

$$\frac{d}{d\alpha} \ln p(D|\alpha) = -E_W^{MP} - \frac{1}{2} \frac{d}{d\alpha} \ln(\det A) + \frac{W}{2\alpha} \quad (19)$$

Now we consider how to compute $\frac{d}{d\alpha} \ln(\det A)$. Let $\lambda_1, \dots, \lambda_W$ be the eigenvalues of the data Hessian H . The A has eigenvalues $\lambda_i + \alpha$, and

$$\frac{d}{d\alpha} \ln(\det A) = \sum_{i=1}^W \frac{1}{\lambda_i + \alpha} \quad (20)$$

Substituting (20) into (19) and let the right-side of equation (19) be zero, we can derive the following relationship:

$$2\alpha E_W^{MP} = W - \sum_{i=1}^W \frac{\alpha}{\lambda_i + \alpha} \quad (21)$$

The right-hand side of equation (21) is equal to a value γ defined as

$$\gamma = \sum_{i=1}^W \frac{\lambda_i}{\lambda_i + \alpha} \quad (22)$$

We can view γ is the measure of the number of well-determined parameters [7].

III. HEAD MOVEMENT CLASSIFICATION

In wheelchair navigation systems, head movement is a natural form of pointing. Joseph [2] reported the use of standard neural networks to classify head movement of disabled users. Taylor [1] reported the performance of head movement interface for wheelchair control accompanied by a standard neural network classifier.

The success of these systems is highly dependent on the training of the neural network classifier. Sample patterns (in this case, recordings of head movement), which form the training sequence, are presented to the neural network classifier along with the expected classifier response. The training algorithm modifies the classifier's parameters to reduce the difference between the actual and expected outputs of the classifier, and the iterative training process continues by repeatedly presenting each of a set of sample patterns.

A. Data Acquisition

The head movement data was collected from six adults, aged between 19 and 56, with approval from the UTS Human Research Ethics Committee and informed consent from the

volunteers. Two had high-level spinal cord injuries (C4 and C5) and were unable to use a standard joystick to control a wheelchair. The remaining four did not have conditions affecting their head movements.

The movement of the user's head is detected by analysing data from a two-axis accelerometer, collected with a sampling period of 100 ms. The input to the Bayesian neural network is comprised of a window of 20 samples from each axis [1].

A pre-trained Bayesian neural network was used to classify the windowed data as corresponding to four types of head movement commands: left, right, forwards or backwards. If the window was not classified as belonging to one of the four classes, it was interpreted as being neutral, or equivalently that no command had been given in that window.

Data for each person was collected in two periods of ten minutes, with the user being prompted to give a specified movement every 6 seconds. Each specified movement was chosen randomly from the following: neural, forward, backward, left and right.

B. Bayesian Neural Network Training

The Bayesian neural network used to classify head movement has the following architecture:

- 41 inputs, corresponding to 20 samples from x axis, 20 samples from y axis and one augmented input assigned to be one;
- Three hidden units;
- Four outputs, corresponding to forward, backward, left and right class.

The training procedure was then implemented as follows:

1. Choosing initial values for the hyperparameter α . Initialising the weights in the network.
2. Training the network with a suitable optimisation algorithm such as scaled conjugate gradient to minimise the total error function $S(w)$. The training data was taken from subject 1, 2, 3 (normal people) and 5 (C4 disabled person).
3. When the network training has reached a local minimum, re-estimating the values of the hyperparameters.

$$\alpha^{new} = \frac{\gamma}{2E_w}$$

4. Repeating step 2 and 3 until convergence (the total error function will not change significantly in subsequent iterations).

C. Results

We use Netlab software for Bayesian neural network training [6]. After training, the performance of the Bayesian neural network was tested on the data from Subjects 4 (no disability) and 6 (C5-level disability) separately.

The confusion matrices show that the Bayesian neural network classifier was able to detect head motion of normal and severely disabled people with a success of 99.83% and 86.07%, respectively.

TABLE I. CONFUSION MATRIX FOR USER 4

		Predicted Classification				
		<i>Movement</i>	<i>Forward</i>	<i>Backward</i>	<i>Left</i>	<i>Right</i>
Actual Classification	<i>Forward</i>	166	0	0	1	
	<i>Backward</i>	0	147	0	0	
	<i>Left</i>	0	0	134	0	
	<i>Right</i>	0	0	0	142	
Accuracy (%)		99.8305				

TABLE II. CONFUSION MATRIX FOR USER 6

		Predicted Classification				
		<i>Movement</i>	<i>Forward</i>	<i>Backward</i>	<i>Left</i>	<i>Right</i>
Actual Classification	<i>Forward</i>	57	0	0	0	
	<i>Backward</i>	17	80	0	0	
	<i>Left</i>	14	0	65	0	
	<i>Right</i>	13	0	0	70	
Accuracy (%)		86.0759				

TABLE III. SENSITIVITY, SPECIFICITY, POSITIVE PREDICTIVE VALUE (PPV) AND NEGATIVE PREDICTIVE VALUE (NPV) OF THE BAYESIAN NEURAL NETWORK

	<i>Sensitivity</i>	<i>Specificity</i>	<i>PPV</i>	<i>NPV</i>
<i>Subject 4</i>	0.9985	0.9994	0.9983	0.9994
<i>Subject 6</i>	0.8727	0.9575	0.8911	0.9548

IV. DISCUSSION

Although the number of trials is small, the results obtained show that Bayesian neural networks can be used to classify head movement accurately. The classification results are consistent after different training times. Due to the requirement to determine the eigenvalues of the Hessian matrices to re-estimate the hyperparameter values, Bayesian neural network training is computationally costly.

The accuracy of Bayesian neural network classifiers also depends on the number of hidden units. Fortunately, Bayesian neural network training allows complex neural network models with a large number of hidden units to be used without fear of the "overfitting" that can occur with standard neural network training. Therefore, the determination of the optimal number of hidden units or the minimum number of hidden

units required to still give optimal solutions for this application should be studied further in the future.

Finally, applications of Bayesian neural networks should be applied with more complicated human-machine systems such as the brain wave (EEG)-based user's intention recognition in intelligent wheelchairs.

V. CONCLUSION

We have investigated the performance of Bayesian neural networks for head movement classification in a wheelchair navigation system. In the future, we would like to develop a novel shared control of powered wheelchairs in which the user's signal can be taken into account with environmental information. This framework of shared control of wheelchairs will significantly overcome the current shortages of existing wheelchair systems.

ACKNOWLEDGEMENT

This study was supported by an ARC LIEF grant (LE0454081).

REFERENCES

- [1] P. B. Taylor and H. T. Nguyen, "Performance of a head-movement interface for wheelchair control," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, pp. 1590 - 1593, 2003.
- [2] T. Joseph and H. Nguyen, "Neural network control of wheelchairs using telemetric head movement," *Proceedings of the 20th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society*, vol. 5, pp. 2731 - 2733, 1998.
- [3] J. C. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, pp. 415 - 447, 1992.
- [4] J. C. MacKay, "A practical Bayesian Framework for Backpropagation Networks," *Computation and Neural Systems*, vol. 4, pp. 448-472, 1992.
- [5] MacKay, "The Evidence Framework Applied to Classification Networks," *Neural Computation*, vol. 4, pp. 720 -736, 1992.
- [6] I. T. Nabney, "NETLAB : algorithms for pattern recognitions," *London ; New York : Springer*, 2002.
- [7] M. Bishop, "Neural networks for pattern recognition," *Oxford : Clarendon Press ; New York : Oxford University Press*, 1995.

Bibliography

Abbas, B. S. and W. Kuo (1990). "Stochastic effectiveness models for human-machine systems." *IEEE Transactions on Systems, Man and Cybernetics* 20(4): 826 - 834.

Aigner, P. and B. J. McCarragher (2000). "Modeling and constraining human interactions in shared control utilizing a discrete event framework." *IEEE Transactions on Systems, Man and Cybernetics* 30(3): May 2000.

Al-Amoudi, A. and L. Zhang (2000). "Application of radial basis function networks for solar-array modelling and maximum power-point prediction." *IEE Proceedings-Generation, Transmission and Distribution* 147(5): 310 - 316.

Al-Ani, A. and M. Deriche (2002). "A New Technique for Combining Multiple Classifiers using The Dempster-Shafer Theory of Evidence." *Journal of Artificial Intelligence Research* 17: 333-361.

Amari, S., N. Murata, et al. (1997). "Asymptotic statistical theory of overtraining and cross-validation." *IEEE Transactions on Neural Networks* 8(5): 985 - 996.

Androutsopoulos, I., J. Koutsias, et al. (2000). "An Evaluation of Naive Bayesian Anti-Spam Filtering." *Proceedings of the Workshop on Machine Learning in the New Information Age, 2000*.

Arifovic, J. and R. Gençay (2001). "Using genetic algorithms to select architecture of a feedforward artificial neural network." *Physica A: Statistical Mechanics and its Applications* 289(3-4): 574-594.

Au, A. T. C. and R. F. Kirsch (2000). "EMG-based prediction of shoulder and elbow kinematics in able-bodied and spinal cord injured individuals." *IEEE Transactions on Rehabilitation Engineering* 8(4): 471 - 480.

Bishop, C. M. (1995). "Neural networks for pattern recognition." *Oxford : Clarendon Press ; New York : Oxford University Press*.

Bolmsjo, G., H. Neveryd, et al. (1995). "Robotics in rehabilitation." *IEEE Transactions on Rehabilitation Engineering* 3(1): 77 - 83.

Bortoletti, A., C. Di Fiore, et al. (2003). "A new class of quasi-Newtonian methods for optimal learning in MLP-networks." *IEEE Transactions on Neural Networks* 14(2): 263 - 273.

Bourhis, G., K. Moumen, et al. (1993). "Assisted navigation for a powered wheelchair." *'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on Systems, Man and Cybernetics* 3: 553 - 558.

Chan, R. B. and D. S. Childress (1990). "On a unifying noise-velocity relationship and information transmission in human-machine systems." *IEEE Transactions on Systems, Man and Cybernetics* 20(5): 1125 -1135.

Chan, R. B. and D. S. Childress (1990). "On information transmission in human-machine systems: channel capacity and optimal filtering." *IEEE Transactions on Systems, Man and Cybernetics* 20(5): 1136 -1145.

Charalambous, C. (1992). "Conjugate gradient algorithm for efficient training of artificial neural networks." *Circuits, Devices and Systems, IEE Proceedings G* 139(3): 301 - 310.

Chen, Y.-L., W.-L. Chen, et al. (2002). "A head orientated electric wheelchair for people with disabilities." *Proceedings of the Second Joint, EMBS/BMES Conference, 2002, Engineering in Medicine and Biology, 2002. 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society* 3: 2371 - 2372.

Cooper, R. A., T. A. Corfman, et al. (2002). "Performance assessment of a pushrim-activated power-assisted wheelchair control system." *IEEE Transactions on Control Systems Technology* 10(1): 121 - 126.

Craig, A., P. Moses, et al. (2002). "The effectiveness of a hands-free environmental control system for the profoundly disabled." *Archives of Physical Medicine and Rehabilitation* 83(10): 1455-1458.

Craig, D. A. and H.T.Nguyen (2005). "Wireless Real-Time Head Movement System Using a Personal Digital Assistant (PDA) for Control of a Power Wheelchair." *The 27th Annual International Conference of the Engineering in Medicine and Biology Society, Shanghai, China*: 6235 - 6238.

Craig, D. A., H.T.Nguyen, et al. (2006). "Two channel EEG Thought Pattern Classifier." *Proceedings of the 28th IEEE EMBS Annual International Conference, New York City, USA, Aug 30-Sep 3, 2006*: 1291 - 1294.

Cripps, R. A. (2004). "Spinal Cord Injury, Australia, 2002-03." *Injury Research and Statistics Series, Number 22*.

Crisman, E. E., A. Loomis, et al. (1991). "Using The Eye Wink Control Interface To Control A Powered Wheelchair." *Engineering in Medicine and Biology Society. Proceedings of the Annual International Conference of the IEEE* 13.

Demeester, E., M. Nuttin, et al. (2003). "Assessing the User's Intent Using Bayes' Rule: Application to Wheelchair Control." *ASER 2003, Bardolino, Italy*: 117-124.

Demeester, E., M. Nuttin, et al. (2003a). "Fine Motion Planning for Shared Wheelchair Control: Requirements and Preliminary Experiments." *11th International Conference on Advanced Robotics (ICAR) 2003, Coimbra, Portugal*: 1278-1283.

Denoeux, T. (1995). "A k-nearest neighbor classification rule based on Dempster-Shafer theory." *IEEE Transactions on Systems, Man and Cybernetics* 25(5): 804 - 813.

Denoeux, T. (2000). "A neural network classifier based on Dempster-Shafer theory." *Part A, IEEE Transactions on Systems, Man and Cybernetics* 30(2): 131 - 150.

DIR (2006). "Disability Information and Resources." <http://www.makoa.org/index.htm>.

Fletcher, L., V. Katkovnik, et al. (1998). "Optimizing the number of hidden nodes of a feedforward artificial neural network." *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence* 2: 1608 - 1612.

Fukuda, T. and T. Shibata (1992). "Theory and applications of neural networks for industrial control systems." *IEEE Transactions on Industrial Electronics* 39(6): 472 - 489.

Ghevondian, N. and H.T.Nguyen (1999). "Modelling of blood glucose profiles non-invasively using a neural network algorithm." *Proceedings of the first Joint BMEVEMBS Conferena, Senring Humanity, Advancing Technology Atlanta, GA, USA 2*: 928.

Ghevondian, N. and H.T.Nguyen (2001). "A novel fuzzy neural network estimator for predicting hypoglycaemia in insulin-induced subjects." *Proceeding of the 23 rd Annual EMBS International Conference, October 25-28, Istanbul, Turkey*: 1657 - 1660.

Ghosh, S. and D. L. Reilly (1994). "Credit card fraud detection with a neural-network." *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences* 3: 621 - 630.

Giuffrida, J. P. and P. E. Crago (2005). "Functional restoration of elbow extension after spinal-cord injury using a neural network-based synergistic FES controller." *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 13(2): 147 - 152.

Guimaraes, F. G. and J. A. Ramirez (2004). "A pruning method for neural networks and its application for optimization in electromagnetics." *IEEE Transactions on Magnetics* 40(2): 1160 - 1163.

Guo, S., R. A. Cooper, et al. (2002). "Development of power wheelchair chin-operated force-sensing joystick." *24th Annual Conference on Engineering in Medicine and Biology* 3: 2373 - 2374.

H.T.Nguyen, L. M. King, et al. (2004). "Real-time head movement system and embedded Linux implementation for the control of power wheelchairs." *26th Annual International Conference of the Engineering in Medicine and Biology Society* 2: 4892 - 4895.

Heckerman, D. (1996). "A Tutorial on Learning With Bayesian Networks." *Technical Report MSR-TR-95-06, Microsoft Research, 1995*.

Henkind, S. J. and M. C. Harrison (1988). "An analysis of four uncertainty calculi." *IEEE Transactions on Systems, Man and Cybernetics* 18(5): 700 - 714

Hincapie, J. G., D. Blana, et al. (2004). "Adaptive neural network controller for an upper extremity neuroprosthesis." *Proceeding of the 26th Annual International Conference of the Engineering in Medicine and Biology Society* 6: 4133 - 4136.

Hunter, L. and D. J. States (1992). "Bayesian classification of protein structure." *Expert, IEEE* 7: 67 - 75.

Jiménez-Hornero, J. E., F. J. Jiménez-Hornero, et al. (2006). "A Linux cluster of personal computers for the numerical simulation of natural airflows in greenhouses using a lattice model." *Computers and Electronics in Agriculture* 52(1-2): 79-89.

Jones, D. K., R. A. Cooper, et al. (1998). "Powered wheelchair driving performance using force- and position-sensing joysticks." *Proceedings of the IEEE 24th Annual Northeast Bioengineering Conference*: 130 - 132.

Joseph, T. and H. Nguyen (1998). "Neural network control of wheelchairs using telemetric head movement." *Proceedings of the 20th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society* 5: 2731 - 2733.

Karaali, O., G. Corrigan, et al. (1998). "A high quality text-to-speech system composed of multiple neural networks." *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing* 2: 1237 - 1240.

Kepner, J. (2001). "Parallel Programming with MatlabMPI." *MIT Lincoln Laboratory, Lexington, MA 02420*.

Kirsch, R. F., P. P. Parikh, et al. (2001). "Feasibility of EMG-based control of shoulder muscle FNS via artificial neural network." *Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society* 2: 1293 - 1296.

Kittler, J., M. Hatef, et al. (1998). "On combining classifiers." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(3): 226 - 239.

Kuno, Y., S. Nakanishi, et al. (1999). "Robotic wheelchair with three control modes." *1999. Proceedings, 1999 IEEE International Conference on Robotics and Automation* 4: 2590 - 2595.

Lankenau, A. (2001). "Avoiding mode confusion in service robots." *The 7th International Conference on Rehabilitation Robotics. IOS Press*: 162 - 167.

Lankenau, A., O. Meyer, et al. (1998). "Safety in robotics: the Bremen Autonomous Wheelchair." *AMC '98-Coimbra., 1998 5th International Workshop on Advanced Motion Control*: 524 - 529.

Lankenau, A. and T. Rofer (2001). "A versatile and safe mobility assistant." *Robotics & Automation Magazine, IEEE* 8(1): 29 - 37.

Larsen, J. and C. Goutte (1999). "On optimal data split for generalization estimation and model selection." *Proceedings of the 1999 IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*: 225 - 234.

Lee, J.-H., I.-S. Choi, et al. (2003). "Natural frequency-based neural network approach to radar target recognition." *IEEE Transactions on Signal Processing* 51(12): 3191 - 3197.

Levin, A. U., T. K. Leen, et al. (1994). "Fast Pruning Using Principal Components." *Advances in Neural Information Processing Systems, Morgan Kaufmann Publishers, Inc.* 6: 35 - 42.

Li, J.-Y., T. W. S. Chow, et al. (1995). "The estimation theory and optimization algorithm for the number of hidden units in the higher-order feedforward neural network." *IEEE International Conference on Neural Networks* 3: 1229 - 1233.

Lim, C. P., S. C. Woo, et al. (2000). "Speech recognition using artificial neural networks." *Proceedings of the First International Conference on Web Information Systems Engineering* 1: 419 - 423.

Lu, J., M. Yang, et al. (2004). "High-performance neural network training on a computational cluster." *Proceedings. Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region*: 467 - 472.

MacKay, D. (1992). "The Evidence Framework Applied to Classification Networks." *Neural Computation* 4(5): 720 - 736.

MacKay, D. J. C. (1992a). "Bayesian Interpolation." *Neural Computation* 4(3): 415 - 447.

MacKay, D. J. C. (1992b). "A practical Bayesian Framework for Backpropagation Networks." *Computation and Neural Systems* 4(3): 448-472.

Maren, A. J. (1991). "Neural networks for enhanced human-computer interactions." *IEEE Control Systems Magazine* 11(5): 34 - 36.

Marwala, T. (2001). "Scaled conjugate gradient and Bayesian training of neural networks for fault identification in cylinders." *Computers & Structures* 79(32): 2793-2803.

Millan, J. R., F. Renkens, et al. (2004). "Noninvasive brain-actuated control of a mobile robot by human EEG." *IEEE Transactions on Biomedical Engineering* 51(6): 1026 - 1033.

Møller, M. F. (1993). "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning." *Neural Networks* 6: 525 - 533.

Neal, R. (1996). "Bayesian Learning for Neural Networks." *Springer-Verlag New York, Inc.*

Nguyen, D. and B. Widrow (1990). "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights." *IJCNN International Joint Conference on Neural Networks* 3: 21 - 26.

Nguyen, H. T., L. M. King, et al. (2004). "Real-time head movement system and embedded Linux implementation for the control of power wheelchairs." *26th Annual International Conference of the Engineering in Medicine and Biology Society* 2: 4892 - 4895.

Pearlmutter, B. A. (1994). "Fast Exact Multiplication by the Hessian." *Neural Computation* 6 (1): 147-160.

Penny, W. D. and S. J. Roberts (1999). "Bayesian neural networks for classification: how useful is the evidence framework." *Neural Networks* 12(6): 877--892.

Perreault, E. J., P. E. Crago, et al. (2001). "Postural arm control following cervical spinal cord injury." *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 9(4).

Pires, G., N. Honorio, et al. (1997). "Autonomous wheelchair for disabled people." *Proceedings of the IEEE International Symposium on Industrial Electronics, 1997. ISIE '97* 3: 797 - 801.

Ponnappalli, P. V. S., K. C. Ho, et al. (1999). "A formal selection and pruning algorithm for feedforward artificial neural network optimization." *IEEE Transactions on Neural Networks* 10(4): 964 - 968.

Prassler, E., J. Scholz, et al. (1998). "MAid: mobility assistance for elderly and disabled people." *Proceedings of the 24th Annual Conference of the IEEE, Industrial Electronics Society, 1998. IECON '98* 4: 2493 - 2498.

Principe, J. C., S. K. Gala, et al. (1989). "Sleep staging automaton based on the theory of evidence." *IEEE Transactions on Biomedical Engineering* 36(5): 503 - 509.

Rasmussen, C. E. (1995). "A Practical Monte Carlo Implementation of Bayesian Learning." *Proc. Conf. Advances in Neural Information Processing Systems* 8: 598 - 604.

Rasmussen, C. E. (1996). "Bayesian learning in multi-layer perceptron neural network using Monte Carlo." *Meet. American Statistical Association*: 4 - 8.

Reed, R. (1993). "Pruning algorithms-a survey." *IEEE Transactions on Neural Networks* 4(5): 740 - 747.

Riess, J. and J. J. Abbas (2000). "Adaptive neural network control of cyclic movements using functional neuromuscular stimulation." *IEEE Transactions on Rehabilitation Engineering* 8(1): 42 - 52.

Roberts, A., W. Pruehsner, et al. (1999). "Vocal, motorized, and environmentally controlled chair." *Bioengineering Conference, 1999. Proceedings of the IEEE 25th Annual Northeast*: 33 - 34.

Rockland, R. H. and S. Reisman (1998). "Voice activated wheelchair controller." *Proceedings of the IEEE 24th Annual Northeast Bioengineering Conference*: 128 - 129.

S.T.Nguyen, H.T.Nguyen, et al. (2004). "Hands-Free Control of Power Wheelchairs using Bayesian Neural Networks." *Proceedings of IEEE Conference on Cybernetics and Intelligent Systems, Singapore, 2004*: 745 - 749.

S.T.Nguyen, H.T.Nguyen, et al. (2006a). "Bayesian Neural Network Classification of Head Movement Direction using Various Advanced Optimisation Training Algorithms." *Proceedings of the first IEEE / RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, Tuscany Italy, 2006*.

S.T.Nguyen, H.T.Nguyen, et al. (2006). Improved Head Direction Command Classification using an Optimised Bayesian Neural Network. Appears in Proceeding of IEEE International Conference of the Engineering in Medicine and Biology Society, New York City, New York, USA, August 30-Sept. 3, 2006.

S.T.Nguyen, H.T.Nguyen, et al. (2006b). Improved Head Direction Command Classification using an Optimised Bayesian Neural Network. Appears in Proceeding of IEEE International Conference of the Engineering in Medicine and Biology Society, New York City, New York, USA, August 30-Sept. 3, 2006.

Sankar, A. and R. J. Mammone (1991). "Optimal pruning of neural tree networks for improved generalization." *Seattle International Joint Conference on Neural Networks 2*: 219 - 224.

Sarle, W. S. (1995). "Stopped Training and other Remedies for Overfitting." *In Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*: 352--360.

Sarver, J. J., B. T. Smith, et al. (1999). "A study of shoulder motions as a control source for adolescents with C4 level SCI." *IEEE Transactions on Rehabilitation Engineering* 7(1): 27 - 34.

Shahjahan, M., M. A. H. Akhand, et al. (2003). "A pruning algorithm for training neural network ensembles." *SICE 2003 Annual Conference* 1: 628 - 633.

Shannon, C. E. (1948). "A mathematical theory of communication." *Bell System Technical Journal* 27: 379-423.

Simpson, R. C. and S. P. Levine (1999). "Automatic adaptation in the NavChair Assistive Wheelchair Navigation System." *IEEE Transactions on Rehabilitation Engineering* 7(4): 452 - 463.

Simpson, R. C. and S. P. Levine (2002). "Voice control of a powered wheelchair." *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 10(2): 122 - 125.

Smets, P. (1990). "The combination of evidence in the transferable belief model." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12(5): 447 - 458.

Son, J. S., D. M. Lee, et al. (2004). "A study on genetic algorithm to select architecture of a optimal neural network in the hot rolling process." *Materials Processing Technology*.

Stephanou, H. E. and S.-Y. Lu (1988). "Measuring consensus effectiveness by a generalized entropy criterion." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10(4): 544 - 554

System, U. M. S. <http://www.spinalcord.uab.edu> , Spinal Cord Injury Information Network, 1998-2003 University of Alabama at Birmingham.

Takechi, H. and K. Murakami (1995). "A learning method of multilayer neural networks for pruning hidden units." *Proceedings of the 34th SICE Annual Conference. International Session Papers*: 1531 - 1534.

Tanaka, K., K. Matsunaga, et al. (2005). "Electroencephalogram-Based Control of an Electric Wheelchair." *IEEE Transactions on Robotics* 21(4): 762 - 766.

Taylor, P. B. and H. T. Nguyen (2003). "Performance of a head-movement interface for wheelchair control." *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* 2: 1590 - 1593.

Thodberg, H. H. (1996). "A review of Bayesian neural networks with an application to near infrared spectroscopy." *IEEE Transactions on Neural Networks* 7(1): 56 - 72.

Tito, E. H., G. Zaverucha, et al. (1999). "Bayesian neural networks for electric load forecasting." *Proceedings. ICONIP '99. 6th International Conference on Neural Information Processing* 1: 407 - 411.

Tourassi, G. D., C. E. J. Floyd, et al. (1999). "A constraint satisfaction neural network for medical diagnosis." *International Joint Conference on Neural Networks* 5: 3632 - 3635.

UAB (2006). "Spinal cord injury information network." <http://www.spinalcord.uab.edu/>.

Vanhooydonck, D., E. Demeester, et al. (2003). "Shared Control for Intelligent Wheelchairs: an Implicit Estimation of the User Intention." *Proc. of ASER 2003, Bardolino, Italy*: 176-182.

Vehtari, A. and J. Lampinen (1999). "Bayesian neural networks for industrial applications." *Proceedings of the 1999 IEEE Midnight-Sun Workshop on Soft Computing Methods in Industrial Applications*: 63 - 68.

Venayagamoorthy, Moonasar, et al. (1998). "Voice recognition using neural networks." *Proceedings of the 1998 South African Symposium on Communications and Signal Processing, 1998. COMSIG '98*: 29 - 32.

Vivarelli, F., R. Serra, et al. (2000). "Bayesian neural network for fermentation control." *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000* 6: 279 - 284.

Williams, C. K. I. B., D.; (1998). "Bayesian classification with Gaussian processes." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(12): 1342 - 1351.

Wright, W. A. (1999). "Bayesian approach to neural-network modeling with input uncertainty." *IEEE Transactions on Neural Networks* 10(6): 1261 - 1270.

Yanco, H. A. (2000). "Shared User-Computer Control of a Robotic Wheelchair System." *PhD thesis, Massachusetts Institute of Technology, September 2000*.

Yoder, J.-D., E. T. Baumgartner, et al. (1996). "Initial results in the development of a guidance system for a powered wheelchair." *IEEE Transactions on Rehabilitation Engineering* 4(3): 143 - 151.

Zhang, Q. P., M. Liang, et al. (2004). "Medical diagnostic image fusion based on feature mapping wavelet neural networks." *The Third International Conference on Image and Graphics*: 51 - 54.

Hands-Free Control of Power Wheelchairs using Bayesian Neural Network Classification

Son Nguyen

Faculty of Engineering

University of Technology, Sydney

Broadway NSW 2007 Australia

E-mail: Son.NguyenThanh@uts.edu.au

Hung Nguyen

Faculty of Engineering

University of Technology, Sydney

Broadway NSW 2007 Australia

E-mail: Hung.Nguyen@uts.edu.au

Philip Taylor

Faculty of Engineering

University of Technology, Sydney

Broadway NSW 2007 Australia

E-mail: Philip.B.Taylor@uts.edu.au

Abstract—This paper describes the formulation and implementation of Bayesian neural networks for head-movement classification in a hands-free wheelchair navigation system. Bayesian neural network training adjusts the weight decay parameters automatically to their near-optimal values that give the best generalisation. Moreover, no separate validation set is used so all available data can be used for training. Experimental results are presented showing that Bayesian neural network can classify the head movement accurately.

Keywords— Bayesian neural network; head-movement classification; power wheelchairs

I. INTRODUCTION

Power wheelchairs are traditionally used by people with severe mobility disabilities such as cerebral palsy, tetraplegia, etc. However, the operation of these wheelchairs can still be a difficult and demanding task. Head movement is a natural form of pointing and can be used to directly replace the joystick whilst still allowing for similar control.

Through the use of standard feed-forward neural networks and head movement methodology, hands-free trainable control methods for power wheelchairs were developed earlier [1, 2]. In this paper, we explore another advanced neural network which promises to provide an ideal platform for hands-free wheelchair control.

Bayesian neural networks or Bayesian framework for back-propagation networks were firstly introduced by MacKay [3, 4]. They were also used for classification [5]. Bayesian neural networks have the following main benefits compared to standard neural networks:

- Bayesian neural network training adjusts the weight decay parameters automatically to their near-optimal values that give the best generalisation. The adjustment is done during training, so the tedious and computer-expensive search for weight decay parameters is no longer needed.
- Network training converged to different local minima and networks with different numbers of nodes, layers and inputs can be compared and ranked.

- No separate validation set is used. Hence all available data can be used for training.

Bayesian neural network theory is used here in a human-machine interface application, in particular for head-movement classification of hands-free commands in a power wheelchair control system.

II. BAYESIAN NEURAL NETWORKS

A. Multi-Layer Perceptron Networks

Multi-layer perceptron (MLP) neural networks are widely used in engineering applications. These networks take in a vector of real inputs, x_i , and from them compute one or more values of activation of the output layer, $a_k(x, w)$. With a one hidden layer network, as shown in Fig. 1, the activation of the output layer is computed as follows:

$$a_k(x) = b_k + \sum_{j=1}^m w_{kj} \tanh\left(\bar{b}_j + \sum_{i=1}^d \bar{w}_{ji} x_i\right) = b_k + \sum_{j=1}^m w_{kj} y_j \quad (1)$$

Here, \bar{w}_{ji} is the weight on the connection from input unit i to hidden unit j ; similarly, w_{kj} is the weight on the connection from hidden units j to output unit k . The \bar{b}_j and b_k are the biases of the hidden and output units. These weights and biases are the parameters of the neural network.

MLP networks can be used to define probabilistic models for regression and classification tasks by using the network outputs to define the conditional distribution for one or more targets. In c class classification problems, the target variables are discrete class labels indicating one of c possible classes. The *softmax* (generalised logistic) model can be used to define the conditional probabilities of the various classes of a network with c output units as follows:

$$z_k(x) = \frac{\exp(a_k(x))}{\sum_{k=1}^c \exp(a_k(x))} \quad (2)$$

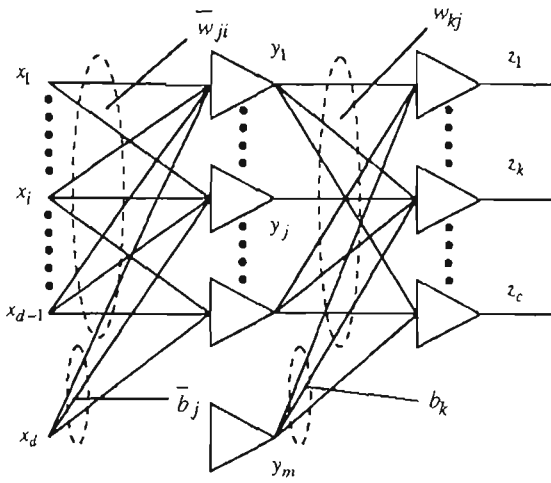


Figure 1. Feed-forward neural network

B. Regularisation

According to [7], we should use the *entropy* error function for c -classes ($c > 2$) classification problem with N sample patterns:

$$E_D = - \sum_{n=1}^N \sum_{k=1}^c t_k^n \ln z_k^n \quad (3)$$

In neural networks, the regularisation can be used to prevent any weights becoming too large because large weights may give poor generalisation for new test cases. Therefore, a weight decay penalty term is added to the error function to penalise large weights. Specifically, for classification problems, we have:

$$S(w) = E_D + \alpha E_w \quad (4)$$

Where $S(w)$ is the total error function, α is a non-negative parameter for the distribution of other parameters (weights and biases) and known as a *hyperparameter* needed to be determined and E_w is the weight error normally originated from the theory of weight priors.

C. Theory of Priors

The requirement for small weights suggests a Gaussian prior distribution with zero mean of the form:

$$p(w|\alpha) = \frac{1}{Z_w(\alpha)} \exp\left(-\frac{\alpha}{2} \|w\|^2\right) \quad (5)$$

$$Z_w(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{w/2} \quad (6)$$

Where, $Z_w(\alpha)$ is the normalisation constant, and W is the number of weights. Ignoring the normalisation constant that does not depend on the weights, this prior is equivalent to a weight error term after taking the negative log.

$$\frac{\alpha}{2} \sum_{i=1}^w w_i^2 = \frac{\alpha}{2} \|w\|^2 = \alpha E_w \quad (7)$$

Substituting (7) into (5), we have

$$p(w|\alpha) = \frac{1}{Z_w(\alpha)} \exp(-\alpha E_w) \quad (8)$$

D. Bayesian Inference

Using Bayesian inference, that the posterior density of the parameters w given a data set D is given by.

$$p(w|D, \alpha) = \frac{p(D|w, \alpha)p(w|\alpha)}{p(D|\alpha)} \quad (9)$$

Equation (9) is the first level of the inference. $p(w|\alpha)$ is the weight prior determined using (8). $p(D|w, \alpha)$ is the dataset likelihood and $p(D|\alpha)$ is the evidence for α or the normalisation factor. This normalisation factor is commonly ignored, since it is irrelevant to the first level of the inference, for example the choice of w . However, it is important to the second level of the inference. If the dataset is identically independently distributed (i.i.d), the dataset likelihood is

$$p(D|w, \alpha) = \exp(-E_D) \quad (10)$$

In this Bayesian framework for neural networks, the optimal weights should maximise the weight posterior distribution. This is equivalent to minimising the total error function.

Based on the assumption of the Laplace approximation [3, 4], we can compute $p(w|D)$ around α_{MP} , the most probable value of the hyperparameter, as

$$p(w|D, \alpha) \approx p(w|D, \alpha_{MP}) \quad (11)$$

To make further progress, we need to perform integrals involving $p(w|\alpha, D)$ and these require further approximations.

Assuming that the total error function $S(w)$ has a single minimum as a function of w at w_{MP} and the total error function, $S(w)$, can be locally approximated as a quadratic form obtained by the second-order Taylor series expansion of $S(w)$.

$$S(w) \approx S(w_{MP}) + \frac{1}{2}(w - w_{MP})^T A (w - w_{MP}) \quad (12)$$

There is no first-order term, since it is assumed that w_{MP} is a local minimum of the error function, so $\partial S(w_{MP})/\partial w_i = 0$ for all coordinates. The matrix A is the Hessian matrix of the total error function at w_{MP} :

$$A = \nabla \nabla S(w_{MP}) = \nabla \nabla E_D(w_{MP}) + \alpha I \quad (13)$$

Where $\nabla \nabla E_D(w_{MP})$ is the Hessian matrix of the network function at w_{MP} and I is the unit matrix. After training phase, the posterior distribution of weights can be derived as follows:

$$p(w|D, \alpha) = \frac{1}{Z_S(\alpha)} \exp(-S(w)) \quad (14)$$

Where Z_S is the normalisation constant for the approximating Gaussian, and is therefore given by

$$Z_S(\alpha) = \exp(-S(w_{MP})) (2\pi)^{W/2} (\det A)^{-1/2} \quad (15)$$

Again, using Bayes' theorem, we can express the posterior distribution of the hyperparameter as

$$p(\alpha|D) = \frac{p(D|\alpha)p(\alpha)}{p(D)} \propto p(D|\alpha)p(\alpha) \quad (16)$$

Where $p(\alpha)$ is the prior distribution of the hyperparameter, or *hyperprior*, and simply we assume that this distribution is uniform. It will thus be ignored subsequently, because to infer the value of the hyperparameter, we only seek the value of the hyperparameter to maximise $p(D|\alpha)$.

Rearranging (9), we have the following form:

$$p(D|\alpha) = \frac{p(D|w, \alpha)p(w|\alpha)}{p(w|D, \alpha)} \quad (17)$$

Since all the terms in the right-side of equation (17) are determined from (10), (8) and (14), equation (17) yields

$$p(D|\alpha) = \frac{Z_S(\alpha)}{Z_w(\alpha)} \quad (18)$$

Taking the derivative of $\ln p(D|\alpha)$ with respect to α , we have

$$\frac{d}{d\alpha} \ln p(D|\alpha) = -E_w^{MP} - \frac{1}{2} \frac{d}{d\alpha} \ln(\det A) + \frac{W}{2\alpha} \quad (19)$$

Now we consider how to compute $\frac{d}{d\alpha} \ln(\det A)$. Let $\lambda_1, \dots, \lambda_W$ be the eigenvalues of the data Hessian H . The A has eigenvalues $\lambda_i + \alpha$, and

$$\frac{d}{d\alpha} \ln(\det A) = \sum_{i=1}^W \frac{1}{\lambda_i + \alpha} \quad (20)$$

Substituting (20) into (19) and let the right-side of equation (19) be zero, we can derive the following relationship:

$$2\alpha E_w^{MP} = W - \sum_{i=1}^W \frac{\alpha}{\lambda_i + \alpha} \quad (21)$$

The right-hand side of equation (21) is equal to a value γ defined as

$$\gamma = \sum_{i=1}^W \frac{\lambda_i}{\lambda_i + \alpha} \quad (22)$$

We can view γ is the measure of the number of well-determined parameters [7].

III. HEAD MOVEMENT CLASSIFICATION

In wheelchair navigation systems, head movement is a natural form of pointing. Joseph [2] reported the use of standard neural networks to classify head movement of disabled users. Taylor [1] reported the performance of head movement interface for wheelchair control accompanied by a standard neural network classifier.

The success of these systems is highly dependent on the training of the neural network classifier. Sample patterns (in this case, recordings of head movement), which form the training sequence, are presented to the neural network classifier along with the expected classifier response. The training algorithm modifies the classifier's parameters to reduce the difference between the actual and expected outputs of the classifier, and the iterative training process continues by repeatedly presenting each of a set of sample patterns.

A. Data Acquisition

The head movement data was collected from six adults, aged between 19 and 56, with approval from the UTS Human Research Ethics Committee and informed consent from the

volunteers. Two had high-level spinal cord injuries (C4 and C5) and were unable to use a standard joystick to control a wheelchair. The remaining four did not have conditions affecting their head movements.

The movement of the user's head is detected by analysing data from a two-axis accelerometer, collected with a sampling period of 100 ms. The input to the Bayesian neural network is comprised of a window of 20 samples from each axis [1].

A pre-trained Bayesian neural network was used to classify the windowed data as corresponding to four types of head movement commands: left, right, forwards or backwards. If the window was not classified as belonging to one of the four classes, it was interpreted as being neutral, or equivalently that no command had been given in that window.

Data for each person was collected in two periods of ten minutes, with the user being prompted to give a specified movement every 6 seconds. Each specified movement was chosen randomly from the following: neutral, forward, backward, left and right.

B. Bayesian Neural Network Training

The Bayesian neural network used to classify head movement has the following architecture:

- 41 inputs, corresponding to 20 samples from x axis, 20 samples from y axis and one augmented input assigned to be one;
- Three hidden units;
- Four outputs, corresponding to forward, backward, left and right class.

The training procedure was then implemented as follows:

1. Choosing initial values for the hyperparameter α . Initialising the weights in the network.
2. Training the network with a suitable optimisation algorithm such as scaled conjugate gradient to minimise the total error function $S(w)$. The training data was taken from subject 1, 2, 3 (normal people) and 5 (C4 disabled person).
3. When the network training has reached a local minimum, re-estimating the values of the hyperparameters.

$$\alpha^{new} = \frac{\gamma}{2E_W}$$

4. Repeating step 2 and 3 until convergence (the total error function will not change significantly in subsequent iterations).

C. Results

We use Netlab software for Bayesian neural network training [6]. After training, the performance of the Bayesian neural network was tested on the data from Subjects 4 (no disability) and 6 (C5-level disability) separately.

The confusion matrices show that the Bayesian neural network classifier was able to detect head motion of normal and severely disabled people with a success of 99.83% and 86.07%, respectively.

TABLE I. CONFUSION MATRIX FOR USER 4

		Predicted Classification			
		<i>Movement</i>	<i>Forward</i>	<i>Backward</i>	<i>Left</i>
Actual Classification	<i>Forward</i>	166	0	0	1
	<i>Backward</i>	0	147	0	0
	<i>Left</i>	0	0	134	0
	<i>Right</i>	0	0	0	142
Accuracy (%)		99.8305			

TABLE II. CONFUSION MATRIX FOR USER 6

		Predicted Classification			
		<i>Movement</i>	<i>Forward</i>	<i>Backward</i>	<i>Left</i>
Actual Classification	<i>Forward</i>	57	0	0	0
	<i>Backward</i>	17	80	0	0
	<i>Left</i>	14	0	65	0
	<i>Right</i>	13	0	0	70
Accuracy (%)		86.0759			

TABLE III. SENSITIVITY, SPECIFICITY, POSITIVE PREDICTIVE VALUE (PPV) AND NEGATIVE PREDICTIVE VALUE (NPV) OF THE BAYESIAN NEURAL NETWORK

	<i>Sensitivity</i>	<i>Specificity</i>	<i>PPV</i>	<i>NPV</i>
<i>Subject 4</i>	0.9985	0.9994	0.9983	0.9994
<i>Subject 6</i>	0.8727	0.9575	0.8911	0.9548

IV. DISCUSSION

Although the number of trials is small, the results obtained show that Bayesian neural networks can be used to classify head movement accurately. The classification results are consistent after different training times. Due to the requirement to determine the eigenvalues of the Hessian matrices to re-estimate the hyperparameter values, Bayesian neural network training is computationally costly.

The accuracy of Bayesian neural network classifiers also depends on the number of hidden units. Fortunately, Bayesian neural network training allows complex neural network models with a large number of hidden units to be used without fear of the "overfitting" that can occur with standard neural network training. Therefore, the determination of the optimal number of hidden units or the minimum number of hidden

units required to still give optimal solutions for this application should be studied further in the future.

Finally, applications of Bayesian neural networks should be applied with more complicated human-machine systems such as the brain wave (EEG)-based user's intention recognition in intelligent wheelchairs.

V. CONCLUSION

We have investigated the performance of Bayesian neural networks for head movement classification in a wheelchair navigation system. In the future, we would like to develop a novel shared control of powered wheelchairs in which the user's signal can be taken into account with environmental information. This framework of shared control of wheelchairs will significantly overcome the current shortages of existing wheelchair systems.

ACKNOWLEDGEMENT

This study was supported by an ARC LIEF grant (LE0454081).

REFERENCES

- [1] P. B. Taylor and H. T. Nguyen, "Performance of a head-movement interface for wheelchair control," *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, pp. 1590 - 1593, 2003.
- [2] T. Joseph and H. Nguyen, "Neural network control of wheelchairs using telemetric head movement," *Proceedings of the 20th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society*, vol. 5, pp. 2731 - 2733, 1998.
- [3] J. C. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, pp. 415 - 447, 1992.
- [4] J. C. MacKay, "A practical Bayesian Framework for Backpropagation Networks," *Computation and Neural Systems*, vol. 4, pp. 448-472, 1992.
- [5] MacKay, "The Evidence Framework Applied to Classification Networks," *Neural Computation*, vol. 4, pp. 720 -736, 1992.
- [6] I. T. Nabney, "NETLAB : algorithms for pattern recognitions," *London ; New York : Springer*, 2002.
- [7] M. Bishop, "Neural networks for pattern recognition," *Oxford : Clarendon Press ; New York : Oxford University Press*, 1995.