NOVEL INTEGRATED METHODS OF EVOLVING SPIKING NEURAL NETWORK AND PARTICLE SWARM OPTIMISATION

HAZA NUZLY BIN ABDULL HAMED

A thesis submitted to
Auckland University of Technology
in fulfillment of the requirements for the degree of
Doctor of Philosophy (PhD)

2012

Knowledge Engineering and Discovery Research Institute

CONTENTS

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extend has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Auckland, 2012

Haza Nuzly Bin Abdull Hamed

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
BP	Backpropagation
C	Proportion Factor
CT	Continuous Stochastic Threshold
DQiPSO	Dynamic Quantum-inspired Particle Swarm Optimisation
EA	Evolutionary Algorithm
ECoS	Evolving Connectionist System
EESNN	Extended Evolving Spiking Neural Network
ESNN	Evolving Spiking Neural Network
GA	Genetic Algorithm
GENESIS	GEneral NEural SImulation System
GPS	Global Positioning System
LIF	Leaky Integrate and Fire
LSM	Liquid State Machine
LTD	Long Term Depression
LTP	Long Term Potential
MLP	Multilayer Perceptron
Mod	Modulation Factor
NBC	Naïve Bayesian Classifier
NeuCom	Neuro-computing Decision Support Environment
PESNN	Probabilistic Evolving Spiking Neural Network
POC	Population Rank Order Coding
PSNM	Probabilistic Spiking Neuron Model
PSO	Particle Swarm Optimisation
PSP	Post-synaptic Potential
QEA	Quantum-inspired Evolutionary Algorithm
QiGA	Quantum-inspired Genetic Algorithm
QiPSO	Quantum-inspired Particle Swarm Optimisation

Qubit	Quantum Bit
RESNN	Reservoir-based Evolving Spiking Neural Network
ReSuMe	Remote Supervised Method
ROC	Rank Order Coding
SI	Swarm Intelligence
Sim	Similarity value
SNN	Spiking Neural Network
SR	Stochastic Reset
SRM	Spike Response Model
ST	Step-wise Stochastic Threshold
STDP	Spike Time Dependent Plasticity
SVM	Support Vector Machine
vQEA	Versatile Quantum-inspired Evolutionary Algorithm

ACKNOWLEDGEMENTS

I would like to take this opportunity to acknowledge some of the many people who have inspired, supported, and educated me over the past three years.

First and foremost, I am very grateful to my primary supervisor, *Professor Nikola Kasabov*, who gave me the chance to study at the renowned Knowledge Engineering and Discovery Research Institute (KEDRI). I wish to express my sincere appreciation to him for all his kind guidance and inspiration to make this research possible. His personality, enthusiasm, patience and intellectual spirit made him a great supervisor and invaluable role model for my professional career.

Special thanks also to my co-supervisor, *Professor Siti Mariyam Shamsuddin*, who introduced me to the computational intelligence and machine learning worlds, for which I am truly indebted. Her suggestions and support for my research are very much appreciated. I must also thank my other co-supervisor *Dr. Stefan Schliebs*, for his generous help throughout the duration of this study. He devoted a lot of time to teach me the basic principles of spiking neural networks, reservoir computing and visualisation. I am also very grateful to him for all his jokes which helped me relax during the breaks. His hard work and kindness will always be an inspiration to me.

While conducting this research, I have had a great time working and learning from a lot of people in KEDRI, who provided a vibrant and multicultural research environment. Many thanks to *Joyce D'Mello*, the manager and the soul of KEDRI for all her tremendous support from my first day in KEDRI. I will always remember her frequent encouragement especially at difficult times during my study. I am also very indebted to the associate director of KEDRI, *Dr. Russel Pears*, for allowing me to attend his class on Data Mining and Knowledge Discovery. I am grateful as well to the members of The National Institute of Information and Communication Technology (NICT) in Japan project group for our collaborative work, especially to *Gary Chen*, *Zbynek Michlovsk* and

Nuwan Amila Gunasekara who assisted me with string kernels. Other KEDRI members I would also like to acknowledge are: *Kshitij Dhoble* and *Nuttapod Nuntalid* for being a great officemates; *Dr. Ammar Mohemmed*, *Dr. Raphael Hu*, *Dr. Harya Widiputra* for the many conversations on various technical topics and who's sometimes help and kindness reminded me of my home country; *Muhaini Othman* and *Norhanifah Murli* who are friends from Universiti Tun Hussein Onn Malaysia where I used to work; *Dr. Paul Pang*, *Dr. Peter Hwang*, *Dr. Qun Song*, *Boris Bacic*, *Linda Liang* and *Marin Karaivanov* for many scientific and non-scientific discussions.

A lot of thanks also to the Ministry of Higher Education Malaysia, Universiti Teknologi Malaysia and Auckland University of Technology for the scholarships and financial supports which have been crucial for this research.

Also, I would like to acknowledge my reviewers, whose comments and suggestions I appreciate very much.

I would like to thank the following people who have read and corrected my thesis and other publications during this research: *Dr. Zaidah Mustaffa*, *John Graves*, *Rudy Mahli* and *Diana Kassabova*.

My special regards and sincere blessings to all of those who have supported me in any respect during the completion of the study.

Last but in no way least, I am delighted to thank my *beloved family: Mak-Azizah Jaffar, Abah-Abdul Hamed Abdullah, Ina, Faqihah, Acik, Ajo and Nazri* for their total support throughout my academic journey. Without their love, patience and encouragement, this work would probably never have been completed.

ABSTRACT

This thesis proposes and presents several methods for classification problems. Spatial and spatiotemporal classification problems have been considered in this study. A novel integration between Evolving Spiking Neural Network (ESNN) and Particle Swarm Optimisation (PSO) is proposed for ESNN model optimisation. ESNN, motivated by the principle of Evolving Connectionist System (ECoS), is a relatively new classifier in the neural information processing area. Proper combination of ESNN parameters would influence the ESNN performance. On the other hand, PSO is a bio-inspired optimiser and was developed based on a study of school of fish and flock of birds behaviour. In this framework, all ESNN parameters are optimised by the PSO to achieve optimal parameter combination for the model. A wrapper approach is implemented in the ESNN-PSO frameworks and a few other integrated frameworks that are also proposed in this work. The classifier uses information provided by the particles during learning and generates a fitness value for each solution candidate. Particles interact with each other and update their information based on the global best particle $gbest$ and their own best solution $pbest$. The learning process continues until termination criteria are met.

When dealing with high dimensional problems, only some of the input features are relevant. In this case, selection of features is required. Since standard PSO is not able to handle probability computation, the quantum computation principle is embedded into PSO. This combination is referred to as Quantum-inspired Particle Swarm Optimisation (QiPSO). The integrated ESNN-QiPSO is proposed in this study for simultaneous feature selection and parameter optimisation. This combination provides promising results that may lead to better and faster learning. However, several problems have been identified that led to the development of enhanced QiPSO and ESNN. A hybrid particle and new search and update strategy is proposed for the QiPSO and is presented in the Dynamic QiPSO (DQiPSO) model. Subsequently, an integrated framework of

DQiPSO and ESNN is proposed for efficient feature selection and parameter optimisation. The probabilistic element is also embedded into ESNN as part of its enhancement. In the Probabilistic ESNN (PESNN), the evolving connection is introduced. In the proposed integrated PESNN-DQiPSO, the classifier works together with the optimiser where the connection, feature and parameter components are optimised synchronously for better classification.

Real world problems are often spatiotemporal. Standard ESNN architecture lacks the ability to process both spatial and temporal components in spatiotemporal problems. This study proposes two new ESNN frameworks for spatiotemporal classification utilising the reservoir computing principle. The Extended ESNN (EESNN) is proposed where a simple memory is used to accumulate all spatial and temporal information before passing them to ESNN. In the second approach, more complex Liquid State Machine (LSM) reservoir is incorporated into the ESNN. The reservoir-based ESNN (RESNN) accumulates all information and generates the reservoir responses that can be measured at any simulation time. These responses are encoded into liquid states before sending them to ESNN for classification.

All proposed frameworks have been evaluated on synthetic and real world problems. This study also proposes a spatiotemporal synthetic problem called Rotating Dot. The purpose of introducing this benchmark dataset is to have a spatiotemporal problem with controllable difficulty that can be used for evaluation of the methods. In this problem, the noise can be set at a small value to generate a simple problem or at a high value for more difficult problems. Results obtained with all proposed frameworks are promising and warrant future exploration.

Chapter 1

INTRODUCTION

1.1 BACKGROUND

The human brain is a constant inspiration to scientists in various fields. The highly complex human brain structure comprises billions of interconnected neurons. Information between neurons is passed using short electrical pulses, also known as spikes. Various spikes strength received by a neuron produce an output spike and stimulate other neurons in the system. Motivated to understand how the brain works, researchers have developed brain-inspired mathematical models that simulate the capability of the brain. The most notable model is the Artificial Neural Network (ANN). Many ANN applications have been developed and most applications are for predicting future events based on historical data. Processing power of ANN allows the network to learn and adapt, in addition to making it particularly well suited to tasks such as classification, pattern recognition, memory recall, prediction, optimisation, and noise filtering (Luger, 2004). The primary significance of ANN is the ability of the network to learn from its environment and improve its performance through learning (Haykin, 1998). The well known ANN architecture is the Multilayer Perceptron (MLP) (Rumelhart, Hintont, & Williams, 1986). Better understanding of how the biological brain works has led to the introduction of Spiking Neural Networks (SNN) (Maass, 1997). SNN can be considered a third generation Artificial Neural Network (ANN) whereby neurons send and receive information based on spikes rather than on continuous variables. One of the SNN architecture is Evolving SNN (ESNN). ESNN architecture was first discussed by Wysoski, Benuskova, and Kasabov (2006b) and followed up

on the evolving concept suggested by Kasabov (1998a) through ECoS. ECoS methods allow the structure of the network to evolve as part of the training process. This method is sensitive to the selection of its parameters and the correct choice of parameters allows the network to evolve to the best structure and ensure the best output. Therefore, an optimiser is needed to find the best combination of parameters.

Biologically-inspired techniques used to develop an effective optimiser have received numerous attention nowadays. The most famous techniques for optimisation are the Genetic Algorithms (GA) (Holland, 1975), which are inspired by the evolution processes in biological chromosomes. Another well known optimiser is the PSO (Eberhart & Kennedy, 1995), which is inspired by how the biological swarm of animals works to achieve a desirable objective for the group. Since its introduction, PSO has been widely used to solve many real world problems. Kennedy and Eberhart (1997) also introduced the binary version of PSO. There have been a lot of developments and improvements in this area, those suggested by Khanesar, Teshnehlab, and Shoorehdeli (2007) and Yuan, Nie, Su, Wang, and Yuan (2009). Quantum computation has received more attention due to its dynamic characteristics used to address binary problems that involve probability, e.g. to select or not to select certain components. Sun, Feng, and Xu (2004) introduced the quantum principle into PSO and proposed the QiPSO specifically to tackle such problems.

Apart from efforts to solve the one dimensional spatial or temporal problems, many endeavours are also made to solve spatiotemporal problems. Spatiotemporal problems are unique because both spatial and temporal elements are important for making a decision. Real life spatiotemporal problems include: cloud formation for rain forecasting, traffic movement for route identification based on Global Positioning System (GPS), motion and human gesture recognition, brain signals and their recognition, and many more. These problems prompted the need for an efficient information processing method where both spatial and temporal components can be captured effectively in order to attain better recognition solutions.

1.2 MOTIVATION

SNN model has been proved to be computationally better in simulating information processing in human brain than sigmoid and analog neural networks (Maass, 1996; Schrauwen & Campenhout, 2006). Despite recent research and development in the area of SNN, there is a significant gap in finding the most effective methods for parameter optimisation and feature selection tasks. ESNN has been shown promising in terms of data processing but difficult to find the optimal parameter value. Similarly to other neural network models, the correct combination of parameters influence the performance of the network. On the other hand, using a higher number of features does not necessarily translate into higher accuracy. In some cases, having fewer significant features could reduce processing time and still produce satisfactory results. This research addresses this challenge with the development of a more effective optimisation strategy based on PSO architecture with embedded principle of quantum computation. In addition, PSO for ESNN optimisation and the integrative environment is a novel work.

Because of the complexity of the ESNN network, there are chances that the network can be optimised in order to have better results. Probabilistic computation is one of the elements that can be adapted to the network. This new element is expected to give some versatility to the network since probabilistic computation allows some components to be selected based on the requirements and conditions at a particular time.

Although there are various studies involving solutions for spatiotemporal problems, developing such solutions using ESNN is a new research question. Spatiotemporal problems are more complex than normal spatial problems (mainly classification and clustering) or temporal problems (mainly prediction). In spatiotemporal problems, both elements need to be taken seriously into consideration during the learning process. This is because both spatial and temporal components provide the information required to obtain more accurate results. Thus, capturing both the spatial and temporal components is the primary challenge in this research.

1.3 RESEARCH OBJECTIVES

This research aims to improve spatial and spatiotemporal classification problems by proposing an integrated framework structure. For spatial problems, this thesis proposes an integrated framework that includes both a classifier and an optimiser. The optimiser simultaneously optimises all crucial ESNN components needed for classification. For spatiotemporal problems, an additional module and a reservoir is proposed to capture both spatial and temporal information components. Although plenty of computational intelligence methods have been developed to solve spatiotemporal problems, it is worth exploring new methods to solve these problems more effectively. In addition, using PSO for ESNN optimisation is novel. This study will explore the possibility of using PSO as a model optimisation for ESNN. ESNN structure is completely different from the more commonly used MLP. However, the standard PSO is inadequate for solving problems that require probability computation such as feature selection tasks. Therefore, probability computation in PSO will be also studied in this research.

Based on the above considerations, this research addresses these problems by developing a new optimisation strategy and utilising quantum computation principles. On the other hand, the ESNN (Wysoski et al., 2006b) will be modified and several novel integrated frameworks will be proposed. This research has the following objectives:

1. Propose novel methods, including:
    - The development of a novel integrated framework for simultaneous feature selection and model optimisation. This will allow the optimal usage of the optimiser potential when solving a given problem and to improve the classification results;
    - The development of an improved optimiser for ESNN;
    - The development of a new architecture of ESNN and a novel framework for solving spatial and spatiotemporal problems.
2. Conduct experimental analysis with comparison to the existing well known methods and evaluate their performance, and

3. Share research outcomes with the relevant research community through publications and presentations.

There are three main directions of this study as shown in Figure 1.1. The study starts with furthering the understanding of ESNN as the core of this research. As described in Wysoski et al. (2006b) and Schliebs, Defoin-Platel, and Kasabov (2009a), parameter optimisation is crucial for ensuring that ESNN can produce good results. Therefore, the first task in this study is to develop a new optimiser for parameter optimisation and feature selection for ESNN. The second task of this research aims to explore if ESNN can be enhanced further since no alterations to ESNN have been made since its introduction in 2006 by Wysoski et al. (2006b). This may lead to a better understanding of this classifier. The final task is to solve spatiotemporal problems. This task requires ESNN to be modified to suit the spatiotemporal data processing which currently ESNN is not able to handle.

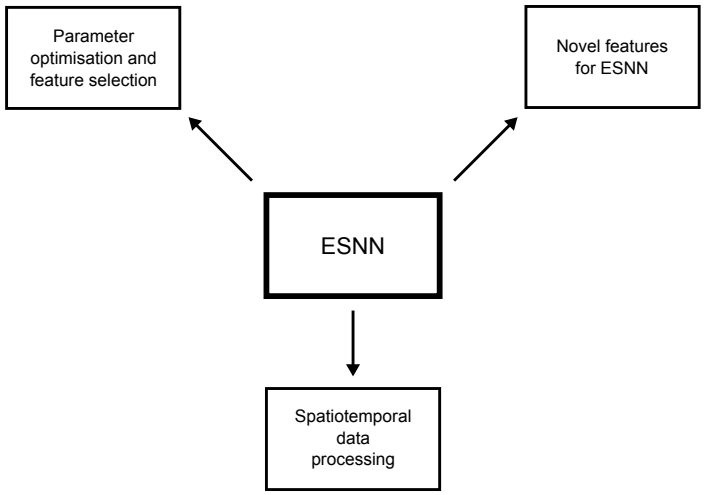

Figure 1.1: Three directions of the research related to the ESNN core: parameter optimisation and feature selection, novel ESNN features and modified ESNN for spatiotemporal data processing.

1.4 SPECIFIC RESEARCH QUESTIONS

In line with the research objectives, the specific research questions for this study are as follows:

1. How to integrate and optimise ESNN with standard PSO and QiPSO? This research is novel in a way that there has been no attempt to integrate ESNN and PSO or QiPSO. The interesting part of this question is to find out how both optimisers work with the given problems, the parameter optimisation and feature selection. In order to feed the input to the network for feature selection, the main question is how to represent the input features. This is crucial because the right mechanism will allow the feature to be selected or removed during the learning process.

2. Can the learning of ESNN be improved by introducing probabilistic elements as suggested by Kasabov (2010)? By incorporating probabilistic elements to the ESNN, what component needs to be optimised and how to optimise it?

3. How to extend the current ESNN in order to solve spatiotemporal pattern recognition problems? Does the input need to be encoded differently? Should additional components be added to the framework to solve these types of problem effectively?

In summary, this research aims to develop an effective technique for model optimisation and to achieve better classification results for spatial and spatiotemporal problems. Some major computational principles will be implemented to achieve the research objectives. These principles include:

1. ECoS,

2. Quantum computation,

3. Reservoir computing, and

4. Knowledge discovery.

1.5 THESIS STRUCTURE

The thesis is organised in ten chapters and is briefly discussed below:

CHAPTER 2 This chapter introduces biological neurons which have inspired the development of SNN. Three main components of SNN, namely the encoding methods, neuron models and learning are also reviewed. ESNN, their principles and their applications are also explained in this chapter.

CHAPTER 3 This chapter reviews the current research in QiPSO. EA and PSO are also explained.

CHAPTER 4 Based on the reviews of ESNN and PSO, this chapter proposes the first integrated structure of ESNN-PSO whereby the PSO acts as an optimiser of ESNN parameters. In line with the research objectives, this chapter explains the proposed integrated ESNN-QiPSO framework where the quantum principle has been used to optimise both parameters and input features. Analysis of the strengths and weaknesses of the proposed methods are also discussed here.

CHAPTER 5 Based on issues identified in the analysis of the proposed method, an enhancement to the QiPSO optimiser is proposed. This chapter also explains the modifications to QiPSO which are crucial for the learning of ESNN. This chapter also proposes a novel integrated framework consisting of enhanced QiPSO - Dynamic QiPSO (DQiPSO) with ESNN and explains why the proposed method performs better than previous methods.

CHAPTER 6 Another objective of this research is to investigate ESNN with probabilistic behaviour. This chapter shows how the new structure of evolving connections in ESNN can affect the performance of the network. An integrated framework is proposed where the connections, input features and parameters can be optimised simultaneously during the learning process.

CHAPTER 7 This chapter proposes an extended version of ESNN for solving spatiotemporal pattern recognition problems. An additional module is added to the ESNN framework in order to capture both the spatial and temporal elements of the problem space.

CHAPTER 8 Another framework for solving spatiotemporal classification is proposed in this chapter. In this approach, the LSM acts as a reservoir. It captures the spatial and temporal information and feeds them to ESNN. A modified encoding method for the reservoir is also discussed in this chapter.

CHAPTER 9 Both methods proposed in Chapter 7 and Chapter 8 for solving spatiotemporal classification are applied to a case study dataset. The LIBRAS dataset is a video sequence containing 15 sign language movements. The proposed methods are used to classify the movements in their respective classes. Results from both methods are discussed in this chapter.

CHAPTER 10 Conclusion and future research directions are discussed in this chapter.

1.6 THESIS CONTRIBUTIONS

A summary of this thesis contributions is visualised in the 3-dimensional representation depicted in Figure 1.2. The three axes represent the increasing complexity of optimisers, classifiers and the datasets respectively. The red text for some the optimisers, classifiers and datasets indicates methods or dataset that have been proposed in this study. The red points indicates the proposed novel integrated methods with the datasets used for testing. Specific chapters where each method is discussed are also shown in the figure.

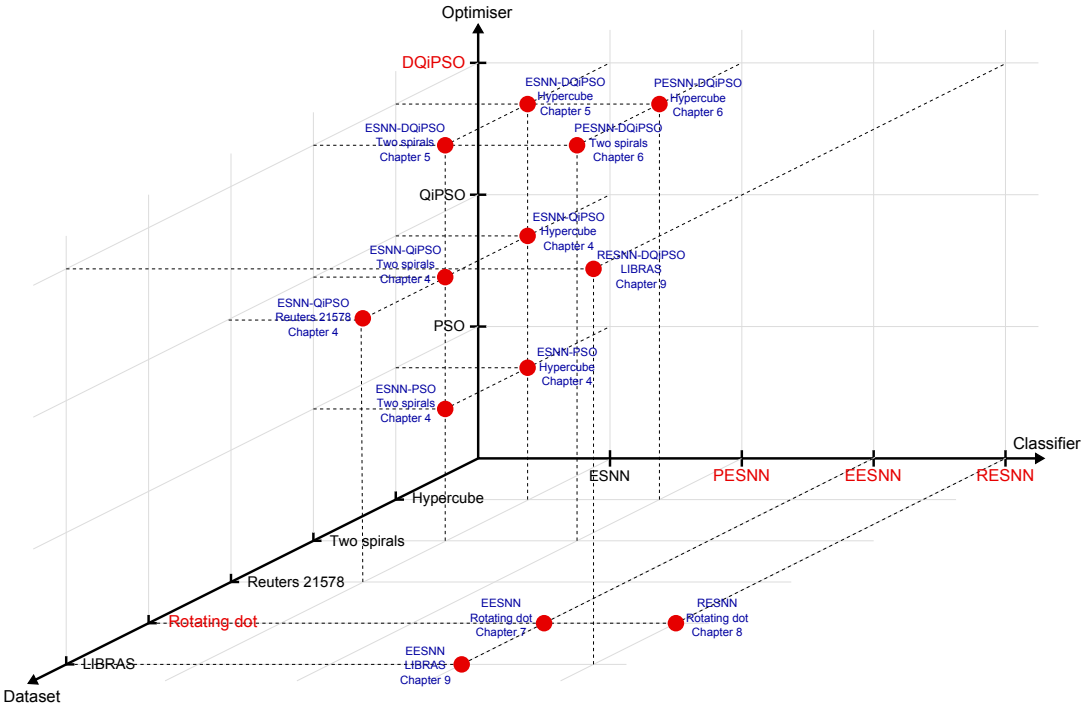

Figure 1.2: A visual summary of contributions that include the proposed new classifiers, optimisers, synthetic dataset and framework that integrate a different combinations of classifiers and optimisers.

1.7 PUBLICATIONS

During the three and half years of study, the following eight international blind peer reviewed publications have been produced that include a book chapter, journal and conference proceedings.

1. Book chapter

   - **Hamed, H.N.A.**, Kasabov, N., & Shamsuddin, S.M. (2011). Quantum-inspired Particle Swarm Optimisation for Feature Selection and Parameter Optimisation in Evolving Spiking Neural Networks for Classification Tasks. In Eisuke Kita (Ed.), Evolutionary Algorithm, ISBN 978-953-307-171-8. INTECH Publications.

2. Journal papers

   - **Hamed, H.N.A.**, Kasabov, N., & Shamsuddin, S.M. (2011). Dynamic Quantum-inspired Particle Swarm Optimisation as Feature and Parameter Optimiser for Evolving Spiking Neural Networks. International Journal of Modeling and Optimisation (In Print).
   - Kasabov, N. & **Hamed, H.N.A.** (2011). Quantum-inspired Particle Swarm Optimisation for Integrated Feature and Parameter Optimisation of Evolving Spiking Neural Networks. International Journal of Artificial Intelligence, 7(11),114-124.
   - **Hamed, H.N.A.**, Kasabov, N., & Shamsuddin, S.M. (2010). Probabilistic Evolving Spiking Neural Network Optimisation Using Dynamic Quantum-inspired Particle Swarm Optimisation. Australian Journal of Intelligent Information Processing Systems, 11(1), 23-28.

3. Papers in conference proceedings

   - Schliebs, S., **Hamed, H.N.A.**, & Kasabov, N. (2011). Reservoir-based Evolving Spiking Neural Network for Spatio-temporal Pattern Recognition. Proceedings of the 18th International Conference on Neural Information Processing, Shanghai, China, 160-168, Part II, LNCS 7063.
   - **Hamed, H.N.A.**, Kasabov, N., Shamsuddin, S.M., Widiputra, H., & Dhoble, K. (2011). An Extended Evolving Spiking Neural Network Model for Spatio-Temporal Pattern Classification. Proceedings of the 2011 International Joint Conference on Neural Networks, San Jose, CA, 2653-2656, IEEE Press.
   - **Hamed, H.N.A.**, Kasabov, N., Michlovsk, Z., & Shamsuddin, S.M. (2009). String Pattern Recognition Using Evolving Spiking Neural Networks and Quantum-inspired Particle Swarm Optimisation. Proceedings of the 16th International Conference on Neural Information Processing, Bangkok, Thailand, 611-619, Part II, LNCS 5864.
   - **Hamed, H.N.A.**, Kasabov, N., & Shamsuddin, S.M. (2009). Integrated Feature Selection and Parameter Optimisation for Evolving

Spiking Neural Networks using Quantum-inspired Particle Swarm Optimisation. Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition, Malacca, Malaysia, 695-698, IEEE Press.

1.8 CHAPTER SUMMARY

This chapter introduces and explains the background, motivation, objectives and research questions formulated for this research. This research will study three main areas related to its ESNN core: the optimiser, enhanced ESNN features and also spatiotemporal data processing. Thesis structure, thesis contributions and publications during the course of this study are also outlined in the last three sections.

The next chapter reviews the main components in this research - the ESNN which is derived from the SNN architecture.

Chapter 2

REVIEW OF EVOLVING SPIKING NEURAL NETWORKS

This chapter reviews the foundation of Evolving Spiking Neural Networks (ESNN). The main discussion in this chapter is about the fundamental components of ESNN which itself is derived from the standard SNN. This discussion includes the data encoding, neuron models and learning algorithms. Finally, the structure of ESNN is discussed.

2.1 INTRODUCTION TO SPIKING NEURAL NETWORKS

The human brain has always been an inspiration for the neural network research. Although current neural network models do not exactly model the complex brain structure, some of its principles have been taken into consideration when developing such artificial systems. Comprehensive explanation of the structure, function, chemistry and physiology of human brain neurons can be found in (Kandel, 2000). Many ANN models have been developed and applied for learning from data and for generalisation to new data (Arbib, 1995). Their applications include: classification, time series prediction, associative storage and retrieval of information, robot and process control, medical and business decision support, and many others (Arbib, 1995; Kasabov, 1996). Most of these ANN use simple and deterministic models of artificial neurons, such as the McCulloch and Pitts model (Mcculloch & Pitts, 1943) introduced in 1943. They also use rate coded information representation, where average activity of a neuron or an ANN is represented as a scalar value. Despite the large structural diversity of existing ANN, the limited functionality of the neu-

rons and connections between them has constrained the scope of applications of ANN and has limited their efficiency when modelling large scale, noisy, dynamic and stochastic processes (Kasabov, 2010). According to Kasabov (2010), new knowledge on information processing in biological neurons have explained several additional parameters also have to be considered for a neuron to spike in addition to the input signals, such as gene and protein expression (Kasabov, 2007; Kojima & Katsumata, 2009), the physical properties of connections (Huguenard, 2000), the probabilities of spikes being received at the synapses and the emitted neuro-transmitters or open ion channels (Ikegaya, Matsumoto, Chiou, Yuste, & Aaron, 2008; Abbott & Nelson, 2000). Many of these properties have been mathematically modelled and used to study biological neurons such as in Gerstner and Kistler (2002a); Izhikevich (2004, 2006); Izhikevich and Edelman (2008), but have not been properly utilised with more efficient ANN for solving complex AI problems. The third generation neural networks, the SNN models are made up of artificial neurons that use trains of spikes to represent and process pulse coded information (Gerstner, Ritz, & Hemmen, 1993; Gerstner & Hemmen, 1994; Gerstner, 2001; Maass, 1997, 1999; Gerstner & Kistler, 2002a; Kasabov, 2008). In biological neural networks, neurons are connected at synapses and electrical signals (spikes) pass information from one neuron to another. SNN are biologically plausible and offer some means for representing time, frequency, phase and other features of the information being processed.

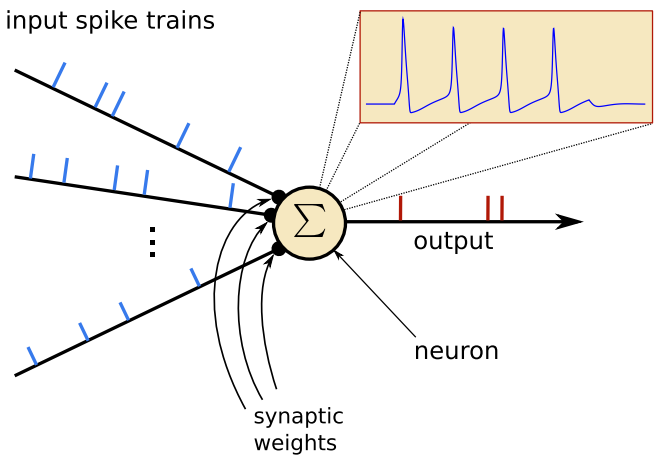

Figure 2.1: A schematic diagram of SNN neuron model.

To illustrate the SNN model, a simplified schematic diagram is presented in Figure 2.1. A neuron receives electrical spike stimulation through connections from a number of pre-synaptic neurons. Every synapse has its own synaptic weight. Once the accumulated Post-synaptic Potential (PSP) is larger than a specific threshold, an output spike is emitted and propagated via the axon to connected post-synaptic neurons. Figure 2.2 illustrates the process of PSP computation in a neural model. As each input spike is received, it will stimulate the PSP until the PSP reaches a predefined threshold and the neuron emits an output spike. Then the potential is reset for next spike computation.

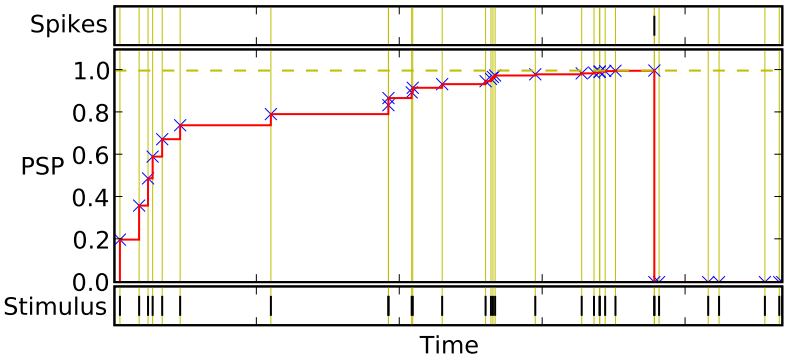

Figure 2.2: Illustration of spiking neuron model. When enough input spikes are received and the potential reaches the threshold, an output spike is emitted and the potential is reset.

There are three main components that have to be considered in the SNN architecture. Since all the information is propagated through spikes, an encoding method is necessary. The original form of data must be encoded into spike trains to be recognised by the network. Model of a spiking neuron is the second component in the SNN architecture. Several different neuron models have been introduced in the recent past. From the most complex model that simulates the biological neuron as closely as possible to the simple and effective model. Every model has its own level of complexity and behaviour that may affect the computation of output spike, thus making it a crucial component of the system. The third component is the learning algorithm of the

network. Since the information in the network is represented as spikes, a different learning mechanism is required for SNN. All three SNN components will be reviewed in the next three sections.

2.2 DATA ENCODING

Since data in SNN needs to be represented in spikes, it must be encoded in the spike trains. In this section, the first component of SNN, the encoding method for the SNN is discussed. There are two main neural encoding theories, the pulse codes and the rate codes, where each generates a different spike characteristic for the network.

2.2.1 *Rate codes*

In this approach, the mean firing rate of a neuron is assumed to hold most of the information. However, there are two interpretation of the mean firing rate. The first interpretation usually considers the ratio of the average number of spikes n_{sp} observed over a specific time interval T as shown in Equition 2.1.

$$v = \frac{n_{sp}}{T} \tag{2.1}$$

The study of data encoding began as early as 1926 when Adrian (1926) successfully applied the rate code approach in sensory and motor neural systems. But this mean firing rate concept has been criticised for the slow transmission of information between neurons because each neuron has to wait for new spike activity for a certain period of time T (Rieke, Warland, Steveninck, & Bialek, 1999). The second understanding of mean firing rate is defined as the average spike activity over a population of neurons. In this concept, a population of pre-synaptic neurons produce a certain spike activity A that is then sent to the post-synaptic neuron. Figure 2.3 illustrates the rate codes calculation from Equation 2.2.

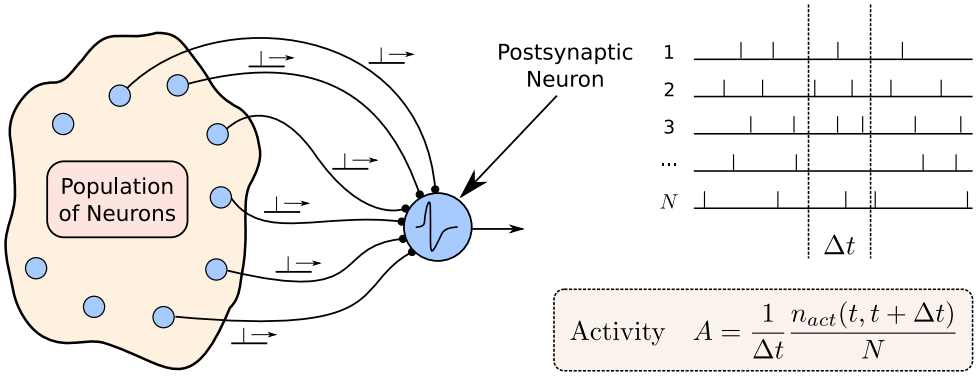

Figure 2.3: Illustration of rate codes generation.

$$A = \frac{1}{\Delta t}\frac{n_{act}(t, t+\Delta t)}{N} \tag{2.2}$$

where $n_{act}(t, t+\Delta t)$ defines the number of active neurons within a short time interval of $[t, t+\Delta t]$ and N is the total number of neurons in the population. This method has been explained in Gerstner (2000).

2.2.2 *Pulse codes*

The second encoding method is called pulse or spike code. In this approach, the exact spike time is considered as information for the network. A study by Lestienne (1995) shows how the temporal component of the spike can be used for learning. Thorpe, Fize, and Marlot (1996) have described this encoding as dependent on the timing of the spikes, where the first spike has a higher weight than a later one. In this work, they argue that since a biological neuron only uses a few milliseconds to process information, only a few spikes are required and emitted. Therefore, the first few spikes with highest information can contribute to the overall learning process. Thorpe et al. (1996) also proposed a specific neuron model which is described in Section 2.3. A neural encoding method has been explained in principle in (Rieke et al., 1999).

There are two well-known practical encoding techniques based on this approach: the Rank Order Coding (ROC) and the Population Rank Order Coding (POC). ROC was proposed by Thorpe and Gautrais (1998) and it encodes in-

formation by using the order of the firing time. For example, for four neurons, where N3 < N1 < N4 < N2, the rank assigned to each neuron is Rank 0 = N2, Rank 1 = N4, Rank 2 = N1 and Rank 3 = N3 as illustrated in Figure 2.4.

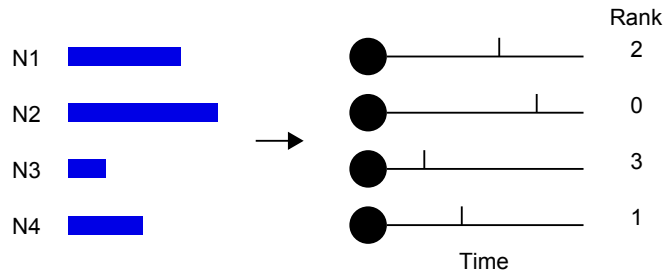

Figure 2.4: ROC Encoding Method.

On the other hand, POC which was studied by Bohte, Kok, and Poutré (2002) distributes a single input value to multiple input neurons denotes as M. Each input neuron holds a firing time as input spikes. The firing time of an input neuron i is calculated using the intersection of Gaussian function as defined in Equation 2.3. The Gaussian centre μ_i is calculated using Equation 2.4 and its width σ is computed using Equation 2.5 with the input variable interval of $[I_{min}, I_{max}]$. $[I_{min}$ and $I_{max}]$ defines the minimum and maximum input values. The parameter β controls the width of each Gaussian receptive field.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.3}$$

$$\mu_i = I_{min} + \frac{2i-3}{2} \cdot \frac{I_{max} - I_{min}}{M-2} \tag{2.4}$$

and width σ:

$$\sigma = \frac{1}{\beta} \cdot \frac{I_{max} - I_{min}}{M-2} \tag{2.5}$$

where $1 \leq \beta \leq 2$. An illustration of this encoding process is shown in Figure 2.5 used in Schliebs, Defoin-Platel, and Kasabov (2009a). For the diagram, $\beta = 2$ was used, the input interval $[I_{min}, I_{max}]$ was set to $[-2.0, 2.0]$ and $M = 5$ input neurons were used. In this example, the input value was defined as 0.70 and five firing times were calculated based on the Gaussian intersections. Both spike encoding methods have been tested in several applications such as visual recognition (Thorpe, Delorme, & Rullen, 2001) and (Wysoski et al., 2006b), audio recognition (Wysoski, Benuskova, & Kasabov, 2007) and speech recognition (Loiselle, Rouat, Pressnitzer, & Thorpe, 2005).

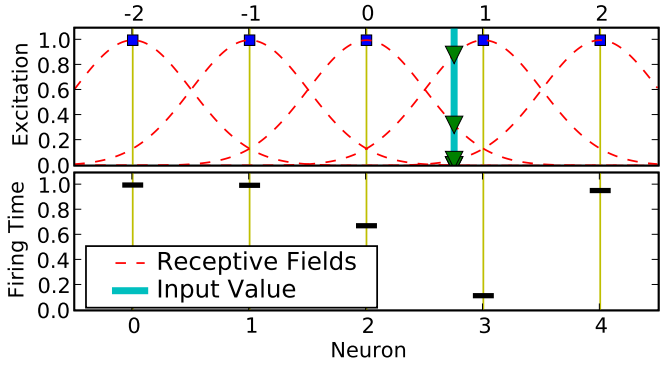

Figure 2.5: Population Encoding Method. Redrawn from (Schliebs, Defoin-Platel, & Kasabov, 2009a).

2.3 NEURON MODELS

Most of the SNN models have been explained by Gerstner and Kistler (2002a). There are several SNN neuron models and the six most commonly used models are discussed in the following six subsections.

2.3.1 *Hodgkin Huxley Model*

The Hodgkin Huxley model has been introduced by Hodgkin and Huxley (1952) in their experiment on the giant axon of a squid. This complex model

simulates the role of ionic mechanisms in calculation and propagation of potentials in the neurons. In the study, they have discovered three ion channels in a neuron, namely Sodium, Potassium and Leakage Channel.

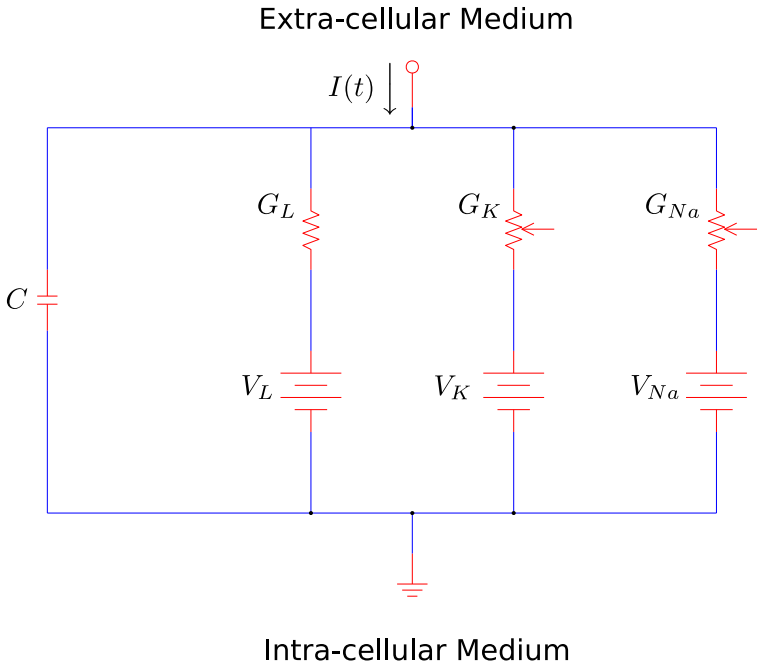

Figure 2.6: Electrical circuit of the Hodgkin - Huxley model according to Hodgkin and Huxley (1952).

The formula used to calculate the membrane potential I_{ion} in the the standard Hodgkin - Huxley model is as follows:

$$\sum I_{ion} = G_{Na} \cdot m^3 \cdot h \cdot (u - V_{Na}) + G_K \cdot n^4 \cdot (u - V_K) + G_L \cdot (u - V_L) \quad (2.6)$$

where G_{Na}, G_K and G_L are Sodium, Potassium and a Leakage Channel respectively, while V_{Na}, V_K and V_L are constants called reverse potentials. If these channels are sometimes blocked, this is expressed by the additional variables m, n and h. These gating variables are described by the following equations:

$$\frac{m}{dt} = \alpha_m(u)(1-m) - \beta_m(u)m \tag{2.7}$$

$$\frac{n}{dt} = \alpha_n(u)(1-n) - \beta_n(u)n \tag{2.8}$$

$$\frac{h}{dt} = \alpha_h(u)(1-h) - \beta_h(u)h \tag{2.9}$$

where m and h control the sodium channel and variable n the potassium channel. α_x and β_x, where $x \in \{m, n, h\}$, are the empirical functions of the capacitor u which control the voltage that needs to be adjusted in order to simulate a specific neuron. Despite this model being the common technique for estimating the parameters of a neuron ionic channel, it also has some disadvantages linked to the approximations required (SaÏghi et al., 2008). Therefore, many improvements have been made to this model as shown in (Guckenheimer & Labouriau, 1993; Fox, 1997; Willms, Baro, Harris-Warrick, & Guckenheimer, 1999).

2.3.2 *Leaky Integrate and Fire Model*

The Leaky Integrate and Fire (LIF) model is a simplified version of the Hodgkin Huxley model, represented by all ion channels with only a single current (Gerstner & Kistler, 2002a). Similarly to the Hodgkin Huxley model, LIF model is also based on the idea of an electrical circuit. The schematic diagram in Figure 2.7 illustrates the model.

In this model, a neuron is represented by an electrical circuit and the current potential is calculated using the appropriate equation. The basic circuit of LIF consists of a capacitor C, parallel with a resistor R, and a current $I(t)$. The current $I(t)$ can be split into two components I_C and I_R as shown in Equation 2.10.

$$I(t) = I_C + I_R \tag{2.10}$$

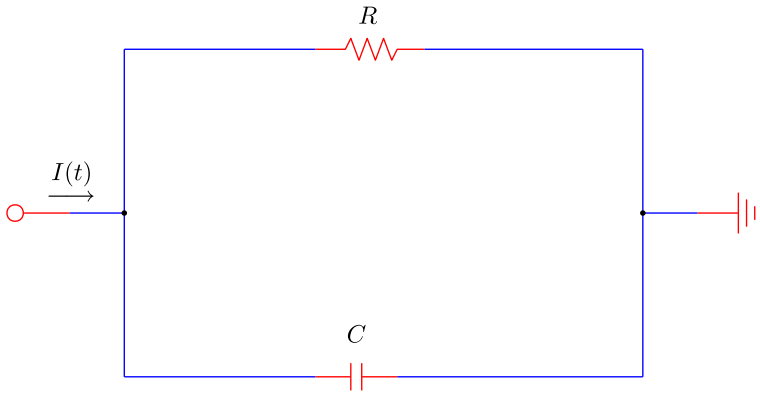

Figure 2.7: Electrical circuit of the LIF from Gerstner and Kistler (2002a).

where I_C charges the capacitor C and I_R passes through the resistor R. Using Ohm's law, capacitance can be calculated as $C = q/u$ where q is the charge, u is the voltage and $I_R = u/R$, the capacitive current $I_c = C \cdot du/dt$, therefore:

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \tag{2.11}$$

A time constant $\tau_m = R \cdot C$ is introduced by the leaky integrator. This yields the standard form of the model:

$$\tau_m \frac{du}{dt} = -u(t) + R\, I(t) \tag{2.12}$$

where u is the membrane potential and τ_m is the membrane time constant of the neuron. The potential leak over the time when no input spikes are received. When the membrane potential reaches the threshold ϑ, the neuron fires and the potential is reset to a new value (resting value). This model can be considered as a suitable model in representing the biological neuron operation and can be applied to large networks due to its low computation load and simplicity.

2.3.3 *Spike Response Model*

SRM uses a concept similar to the LIF model and has a simple description of action potential generation in neurons. SRM models the neuron as respond-

ing to an incoming pulse or spike from another neuron by generating its own spike. The state of the neuron is represented by a single variable u. The model uses a number of kernel functions that influence the input spikes, the external stimulation of u and the actual spike and its after-potential. An output spike is generated when the state u reaches a threshold ϑ. The difference between SRM and LIF is that in SRM the threshold ϑ is not necessarily to be fixed. For instance, the threshold ϑ might be increased after the neuron has spiked. The membrane potential $u(t)$ is calculated using Equation 2.13.

$$u(t) = \eta(t - \hat{t}_i) + \int_0^\infty \varepsilon(t - \hat{t}_i, s) I(t - s) ds \tag{2.13}$$

where $\hat{t}$ is the last firing time of the neuron, η and ε are the kernel functions where ε is also called the linear filter of the membrane, $I(t)$ is a stimulating current. The simplified version of the SRM is normally referred to as Simplified SRM where only the last spike of a neuron is used for the calculation of η. This model has been used for analysing the computational power of spiking neurons by Maass (1994) and for modeling collective neuron excitations by Kistler, Seitz, and Hemmen (1998).

2.3.4 *Izhikevich Model*

Izhikevich proposed a model that combines the dynamics of the biologically plausible of the Hodgkin-Huxley model with the computational efficiency of integrate-and-fire models (Izhikevich, 2003). This model works with two variables: v is a variable representing the membrane voltage potential, and u is a variable representing the membrane recovery (activation of potassium currents and inactivation of sodium currents). This model is described by the following formula:

$$\frac{dv}{dt} = 0.04 \cdot v^2 + 5v + 140 - u + I \tag{2.14}$$

where

$$\frac{du}{dt} = a(b \cdot v - u) \tag{2.15}$$

where t represents time, a describes the time scale of u, b describes the sensitivity of u to v. When the voltage v exceeds a threshold value of 30mV, both v and u are reset:

$$\text{if } v \geq 30\text{mV, then } \begin{cases} v & \leftarrow & c \\ u & \leftarrow & u + d \end{cases} \tag{2.16}$$

where c and d describes the after spike reset value of v. Izhikivech claimed that his model can exhibit the firing patterns of all known types of cortical neurons with appropriate choices of parameters a, b, c and d (Izhikevich, 2004).

2.3.5 *Thorpe Model*

Thorpe (1997) proposed a simplified Integrate and Fire spiking neuron model that has been simulated in SpikeNet software (Delorme & Thorpe, 2003). The model inherits the main concept from LIF but simplifies the leaky operation of the computational neuron. The neuron response depends entirely on the arrival time of pre-synaptic input spikes. The earlier input spikes affect the PSP more strongly than later spikes. In this model, each neuron is allowed to fire only once and then is disabled. The model equation is as follows:

$$u_i(t) = \begin{cases} 0 & \text{if fired} \\ \sum w_{ji}\ Mod_i^{order(j)} & \text{else} \end{cases} \tag{2.17}$$

where w_{ji} is the weight of a pre-synaptic neuron j and Mod is a parameter called modulation factor within the interval $[0, 1]$. Function $order(j)$ represents the rank of the spike emitted by neuron j. The $order(j)$ starts at 0 if the neuron spikes first among all pre-synaptic neurons and then increases proportionally to firing time. The potential is reset to $u_i = 0$ after the neuron emits an output

spike. Threshold value is set to $\vartheta = c \cdot u_{max}$ where $0 < c < 1$ and u_{max} is the maximum potential value can be reached by a neuron. Figure 2.8 shows the PSP computation for the Thorpe neural model when a series of input spikes stimulates the neuron through different synapses. When the potential reaches a threshold ϑ, an output spike is emitted and the PSP is reset to 0 for the rest of the simulation.

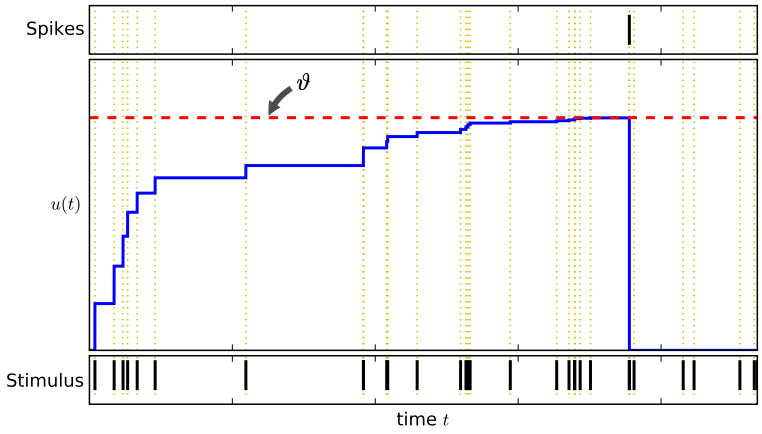

Figure 2.8: PSP calculations in Thorpe's model. Redrawn from (Schliebs, Defoin-Platel, & Kasabov, 2009b).

2.3.6 *Kasabov Model*

Recently Kasabov (2010) introduced the Probabilistic Spiking Neuron Model (pSNM) which is an extension of the LIF model with three additional new probabilistic parameters. It is illustrated in Figure 2.9 parameter $p^c_{j,i}$ represents the probability that a spike from neuron j will reach i, parameter $p^s_{j,i}$ is the probability that synapse (j, i) contributes to potential u_i after it has received a spike from neuron j, and p_i is the probability that neuron i emits an output spike when the total PSP reached the PSP threshold. The PSP calculation is shown in Equation 2.18.

$$u_i(t) = \sum_{p=t_0,\ldots,t} \sum_{j=1,\ldots,m} e_j g(p^c_{j,i}(t-p)) f(p^s_{j,i}(t-p)) w_{j,i}(t) + \eta(t-t_0) \quad (2.18)$$

where $e_j = 1$ if an output spike has been emitted from neuron j, $g(p^c_{j,i}(t))$ collapses into 1 if the spike is propagated and $f(p^s_{j,i}(t)) = 1$ if the synapse contributes to the potential. $\eta(t-t_0)$ representing the decay in PSP. If all probabilities value are set to 1, the model is equivalent to the traditional spike response model.

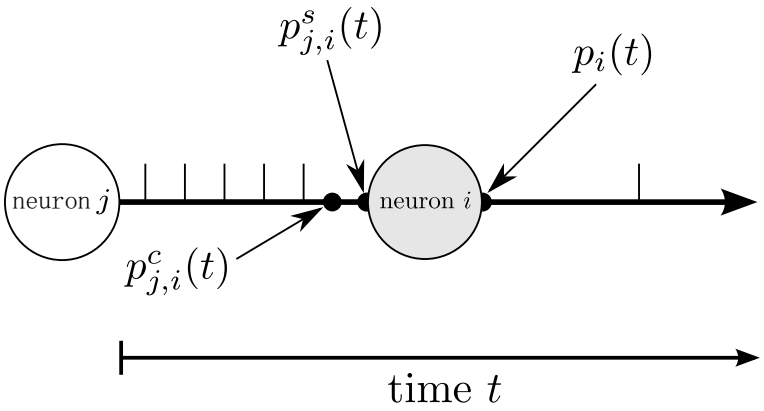

Figure 2.9: Kasabov's Probabilistic Neuron Model.

2.4 LEARNING

This section describes several learning algorithms designed for SNN. Learning in SNN is a complex process since information is represented in time dependent spikes. Most of the SNN use recurrent network topologies where learning is more difficult. Some of the learning algorithms are normally being applied to a specific type of SNN due to its characteristic.

Similarly to learning in traditional neural network, learning in SNN is divided into reinforcement, supervised and unsupervised. Supervised learning is the most commonly used learning algorithm in SNN. Various supervised

learning algorithms have been developed for SNN and have been reviewed by Kasinski and Ponulak (2006).

2.4.1 *SpikeProp*

There have been some attempts to copy the Backpropagation (BP) learning algorithm often used for the MLP. The SNN version of BP algorithm is called Spiking Backpropagation Algorithm or SpikeProp (Bohte, Kok, & Poutré, 2000). In MLP, training is the process to get the optimal set of connection weights while SpikeProp objective is to obtain a set of desired firing times t_j^d of all output neurons j. The fitness function during learning is minimising the error E of the squared difference between training output times t_j^{out} and desired output times t_j^d:

$$E = \frac{1}{2} \sum_j \left(t_j^{out} - t_j^d \right)^2 \tag{2.19}$$

and the error is minimised based on the computation of the weights w_{ij}^k of each synaptic input:

$$\Delta w_{ij}^k = -\eta \frac{dE}{dw_{ij}^k} \tag{2.20}$$

where η represents the learning rate. Several extended versions of SpikeProp have also been proposed such as the dynamic learning parameter (Xin & Embrechts, 2001) and an improvement of the backpropagation rule (Schrauwen & van Campenhout, 2004).

2.4.2 *One-Pass Algorithm*

The One-Pass Algorithm is proposed by Sguier and Mercier (2002). It follows the time-to-first spike learning rule (Thorpe, 1997). In this algorithm, each training sample creates a new output neuron in an output neuron repos-

itory. The trained threshold values and the weight pattern for that particular sample are stored in the repository. However, if the weight pattern of the trained neuron closely resembles a neuron in the repository, it will merge into the most similar neuron. The merging process involves modifying the weight pattern and the threshold of the merged neurons to the average value. Otherwise, it will be added to the repository as a newly trained neuron. The major advantage of this learning algorithm is the ability of the trained network to learn incrementally new samples without retraining the already trained samples (Schliebs, Defoin-Platel, & Kasabov, 2009a). This algorithm has been tested in ESNN in several studies, such as for pattern recognition (Wysoski et al., 2006b; Wysoski, Benuskova, & Kasabov, 2006a), speech recognition (Wysoski et al., 2007), taste identification (Soltic, Wysoski, & Kasabov, 2008), synthetic and ecological problems (Schliebs, Defoin-Platel, & Kasabov, 2009a; Schliebs, Defoin-Platel, Worner, & Kasabov, 2009b, 2009a).

2.4.3 *Spike Time Dependent Plasticity*

Spike Time Dependent Plasticity (STDP) is a form of Hebbian Learning where spike time and transmission time are used in order to calculate the output of a neuron. This unsupervised learning method was inspired from the Hebb's law (Hebb, 1949):.

> When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

The Hebb's law was studied by Bliss and Lomo (1973). Since then, this concept has been further researched and defined as the effectiveness of the synaptic activity in the brain caused by timing of pre- and post-synaptic activity of a neuron. If a pre-synaptic spike arrives at the synapse before the post-synaptic action potential, the synapse is potentiated as normally referred to as long term potential (LTP); if the timing is the other way around, the synapse is depressed and referred as long term depression (LTD) (Markram, Lubke, Frotscher, & Sakmann, 1997; Bi & Poo, 2001). A function $W(t_{pre} - t_{post})$

describes the STDP and is also referred to as the STDP window. The change of synaptic weight depends on the difference between the arrival time t_{pre} of a pre-synaptic spike and the time t_{post} of an action potential emitted by the neuron.

$$W(t_{pre} - t_{post}) = \begin{cases} A_+ \exp(\frac{t_{pre}-t_{post}}{\tau_+}) & \text{if } t_{pre} < t_{post} \\ A_- \exp(-\frac{t_{pre}-t_{post}}{\tau_-}) & \text{if } t_{pre} > t_{post} \end{cases} \tag{2.21}$$

where parameters τ_+ and τ_- define the time interval of the pre- and post-synaptic activity, and A_+ and A_- indicate the maximum fractions of synaptic modification, if $t_{pre} - t_{post}$ is close to zero. More information on STDP can be found in (Bi & Poo, 2001; Gerstner & Kistler, 2002a; Kempter, Gerstner, & van Hemmen, 1999)

2.4.4 *Other Learning Algorithms*

Some other learning algorithms, such as the Reinforcement Learning mechanism, can also be implemented in SNN. The learning process is influenced by the interaction with the environment and is often used in robotic applications (Florian, 2005, 2007; Seung, 2003; Xie & Seung, 2004).

On the other hand, Remote Supervised Method (ReSuMe) that is based on the Hebbian concept is a new supervised learning method for SNN (Ponulak, 2005). The objective of ReSuMe is to generate a desired input-output spike pattern in SNN. For example, to respond to a certain input stimulus, specific target spike trains are generated. Figure 2.10(a) depicts three neurons. Neuron n_i^l acts as a learning neuron that receives spike sequences from a pre-synaptic input neuron $n_k^{in}(i)$ and neuron $n^d(i)$ acts as a teacher for weight w_{ki}. If neuron $n_k^{in}(i)$ releases a spike followed by a spike from teacher neuron $n^d(i)$, then the synaptic weight w_{ki} is increased as shown in Figure 2.10(b). Figure 2.10(c) shows the value of w_{ki} decreases if neuron $n_k^{in}(i)$ spikes before the learning neuron n_i^l is activated. Functions $W^d(s^d)$ and $W^l(s^l)$ as shown in Figure 2.10(b) and Figure 2.10(c) respectively determine the synaptic weight change where s^d represent the difference between the spike times of input neuron $n_k^{in}(i)$ and

teacher neuron $n^d(i)$ and s^l is the difference between the spike times of input neuron $n_k^{in}(i)$ and learning neuron n_i^l.

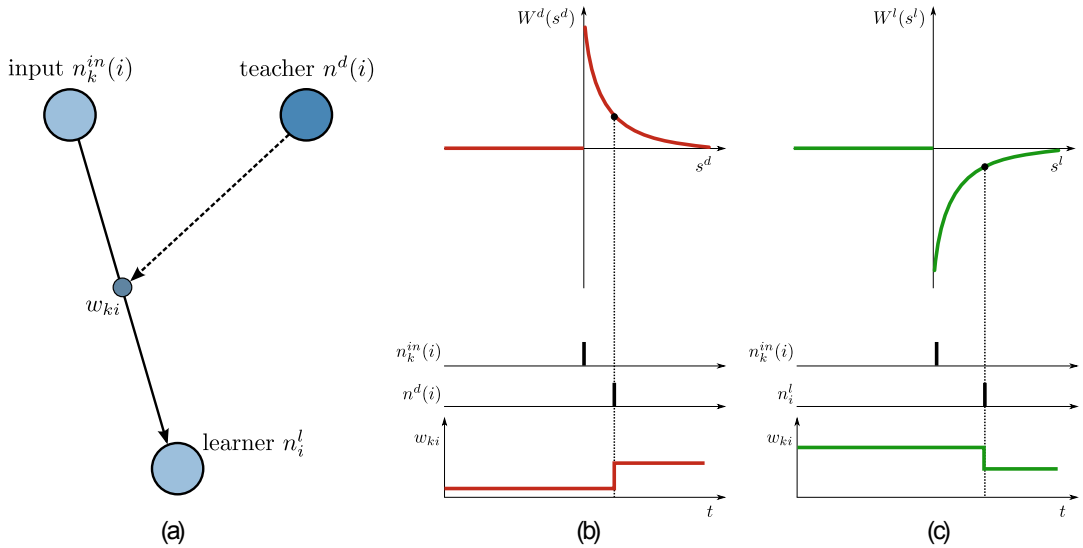

Figure 2.10: ReSuMe learning. Redrawn from (Ponulak, 2005).

2.5 LIQUID STATE MACHINE

LSM introduced by Maass, Natschläger, and Markram (2002) is a concept based on information accumulation. It is a form of reservoir computing and is constructed using recurrent network topology (Verstraeten, Schrauwen, D'Haene, & Stroobandt, 2007). This method is following the concept of dropping an object into a cup of still water. Different objects (input) will produce different waves (output) in the water.

In LSM, the liquid is a component consisting of a large collection of interconnected recurrent neurons that receive input and send output to each others. The synaptic weights, connectivity and neural parameters are predefined and fixed during simulation. The input spike $u(t)$ is propagated into the liquid and this causes the neurons to respond and generate the liquid activity. The liquid snapshots, or normally referred as liquid states $x(t)$, can be recorded at various time points. Finally, a readout function f is applied to convert the liquid

state to a desirable input format for the chosen learning algorithm or classifier. Because of the ability of the network to accumulate both spatial and time information, LSM can be used to solve spatiotemporal problems as reported in (Goodman & Ventura, 2006; Buonomano & Maass, 2009; Schliebs, Nuntalid, & Kasabov, 2010). Figure 2.11 depicts the architecture of the LSM.

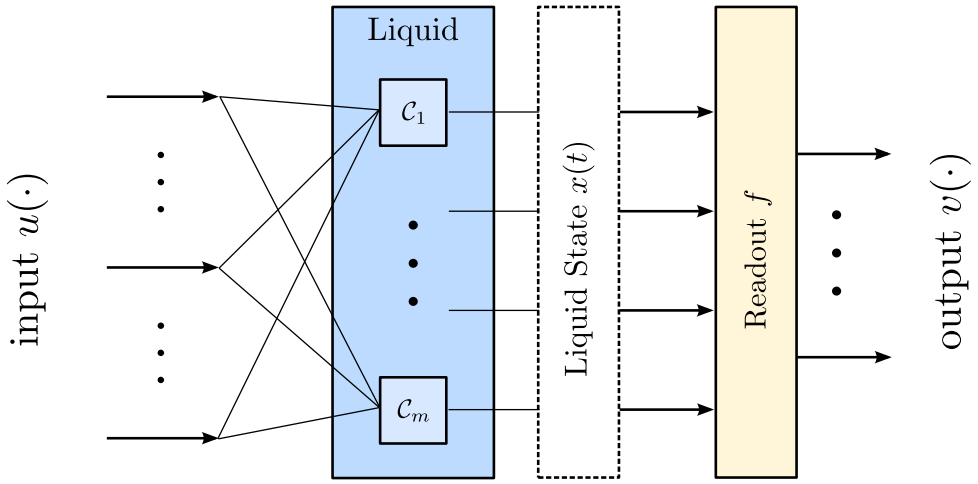

Figure 2.11: The LSM architecture. The input is injected into the reservoir and the liquid state is extracted from the neuron activity inside the reservoir. Then, the readout function is applied to the liquid state and transformed into a required input format for processing - such as classification or clustering.

The LIF neuron model described in Section 2.3 is a well known model for LSM construction. Output spikes from neurons in LSM generate the LSM responses that are crucial for the decision making algorithms. The threshold value plays an important role in the output spike construction. Recent study by Schliebs, Nuntalid, and Kasabov (2010) suggests that a probabilistic neuron may replace or complement the deterministic LIF neuron in the traditional LSM. In this model, threshold value changes over time and affects the output spikes. Three probabilistic neurons have been proposed in their study. The first model is called *step-wise stochastic threshold* (ST). The stochastic threshold for this model is defined as

$$\lim_{t \to t^{(f)}, t > t^{(f)}} \vartheta(t) = \mathcal{N}(\vartheta_0, \sigma_{ST}) \tag{2.22}$$

where $t^{(f)}$ is the firing time of the neuron and σ_{ST} is a parameter of the model that represents the standard deviation of $\vartheta(t)$. After an output neuron fires, a new threshold is sampled according to the ϑ_0-centered Gaussian random variable.

The second model is the *stochastic reset* or *noisy reset* (SR). The reset value u is dynamically modified according to Equation 2.23:

$$\lim_{t \to t^{(f)}, t > t^{(f)}} u(t) = \mathcal{N}(\mu_r, \sigma_{SR}) \tag{2.23}$$

where $\mathcal{N}(\mu, \sigma)$ is a Gaussian distributed random variable, μ is defined as the mean and σ is the standard deviation. Parameter of the model is presented by variable σ_{SR}.

Continuous stochastic threshold (CT) is the third neuron model. This model continuously updates the threshold $\vartheta(t)$ over time following the Ornstein Uhlenbeck principle (Kampen, 2007). Equation 2.24 explains the computation of the new threshold in this model.

$$\tau_\vartheta \frac{d\vartheta}{dt} = \vartheta_0 - \vartheta(t) + \sigma_{CT}\sqrt{2\tau_\vartheta}\xi(t) \tag{2.24}$$

where ξ represents the Gaussian noise, σ_{CT} is the standard deviation of this model and $\vartheta_0 - \vartheta(t)$ is the distance where the threshold drifted to. Details about each model and the experimental results can be found in their paper (Schliebs, Nuntalid, & Kasabov, 2010).

2.6 TOOLS AND APPLICATIONS OF SNN

SNN can be used in two different types of applications. The first type is to use SNN for the understanding the principles of biological neurons and brain functions. The second type of applications are used for solving real world problems. Hodgkin and Huxley (1952) pioneered the work in the area of neuroscience. The understanding and development of the neuron model has been discussed in Section 2.3. Extensive reviews of the nervous system have been

well documented in (Gerstner & Kistler, 2002a) and (Carnevale & Hines, 2006).

SNN has been successfully applied to many real world applications. One of the applications is in robotic and control systems (Roggen, Hofmann, Thoma, & Floreano, 2003; Alnajjar & Murase, 2005; Rocke, McGinley, Morgan, & Maher, 2007; Pearson et al., 2007) and power system problems of system identification (Johnson, Venayagamoorthy, & Mitra, 2009). In economics, SNN is widely used as a tool for prediction, as in yield prediction (Lin & Zhongjian, 2011) and in electricity price prediction (Sharma & Srinivasan, 2010). SNN have also been applied for image processing and detection (Perrinet & Samuelides, 2002; Wu, McGinnity, Maguire, Valderrama-Gonzalez, & Dempster, 2010; Wysoski et al., 2006b) and speech recognition (Verstraeten, Schrauwen, & Stroobandt, 2005; Wysoski et al., 2007). The medical field has also made use of SNN, whereby problems such as breast cancer detection (McGinley et al., 2010) and sound source localisation (Liu, Perez-Gonzalez, Rees, Erwin, & Wermter, 2010) have been tackled. Furthermore, SNN has been tested in large scale problems (Iglesias, Eriksson, Grize, Tomassini, & Villa, 2005; Maguire et al., 2007) and in spatiotemporal problems (Jin, 2004). There have been also some attempts to develop a practical SNN processor (Schoenauer, Atasoy, Mehrtash, & Klar, 2002; Pearson et al., 2005; Khan et al., 2008), and hardware development has been summarised in (Cawley et al., 2011).

Several software tools and libraries are available for SNN simulation. The intention is to understand and simulate the neuron behaviour as in GEneral NEural SImulation System (GENESIS)[1]. GENESIS's provides a general platform for neural system simulation and can analyse the available biologically plausible neuron models. Another tool for understanding the complex biological neuron is NEURON[2]. This tool simulates an individual neuron or a network of neurons. The graphical user interface helps users to easily create and manipulate neuron models with a wide range of complexity. For SNN development, Amygdala[3] is one of the tools available and packaged as C++ library. In this tool, several neuron models are offered to facilitate the development of

1 Available at http://www.genesis-sim.org/GENESIS
2 Available at http://www.neuron.yale.edu/neuron
3 Available at http://sourceforge.net/projects/amygdala

an algorithm. SpikeNNS[4] is an SNN tool that implements SRM neuron model. Simulations of spiking self-organising maps and multi-layer architectures can be done using this software. This simulation tool also provides several learning algorithms for SNN. Neuro-computing Decision Support Environment (NeuCom)[5]software also has some modules with capabilities to simulate SNN. The latest and most comprehensive SNN simulator is Brian[6]. This highly flexible tool was written in Python which enables its integration with other tools. Numerous neuron models and architectures can be rapidly developed and tested. Brian provides more flexibility especially when dealing with non-standard neuron models.

2.7 EVOLVING SPIKING NEURAL NETWORK

The ESNN is a type of neural network that follows the principle of ECoS that were first introduced by Kasabov (1998a). ECoS evolves its structure through incremental learning. In ECoS, new connections and neurons are created in the process of learning and are modified to accommodate any new input data, features or classes. The incremental one-pass learning algorithm is employed due to the evolving characteristics of the network. Numerous variants and applications of ECoS have been developed over a decade long period. These include the fuzzy neural network (Kasabov, 1998b), self-organising maps (Deng & Kasabov, 2000) and dynamically evolving fuzzy systems (Kasabov & Song, 2002). More information on ECoS can be found in (Kasabov, 2007) and the development of ECoS has been reviewed by Watts (2009).

Stimulated by ECoS and SNN, the ESNN architecture was introduced in Wysoski et al. (2006b) whereby SNN evolves its structure through learning. Like SNN, the ESNN architecture consists of a data encoding method that transforms real value data to spike trains, neuron model and learning method. For the encoding methods, ESNN utilises the Rank Order Population Coding. A real-value input is mapped into several pre-synaptic input neurons. Each pre-synaptic input neuron holds a spike, which calculates based on the Gaussian

4 Available at http://cortex.cs.nuim.ie/tools/spikeNNS
5 Available at http://www.theneucom.com
6 Available at http://www.briansimulator.org

Algorithm 1 Training an ESNN

Require: parameter Mod_l, C_l, Sim_l for a class label l

initialise neuron repository $R_l = \{\}$ for class l data
for all samples i belonging to class l **do**
  Encode input samples i into firing time of pre-synaptic neurons j using Equation 2.3
  Calculate the connection weights using Equation 2.25
  Calculate the maximum possible potential according to Equation 2.26
  Calculate threshold ϑ based on Equation 2.27
  if $\min(d(w^{(i)}, w^{(k)})) < Sim_l, \quad w^{(k)} \in R_l$ **then**
    $w^{(k)} \leftarrow$ merge $w^{(i)}$and $w^{(k)}$ according to Equation 2.28
    $\vartheta^{(k)} \leftarrow$ merge $\vartheta^{(i)}$and $\vartheta^{(k)}$ according to Equation 2.29
  else
    Add the new neuron $R_l \leftarrow R_l \cup \{w^{(i)}\}$
  end if
end for

intersection as described in Equation 2.4. Figure 2.5 shows an example of how an input value of 0.70 is encoded into five pre-synaptic neurons.

As for the learning, one-pass learning has been selected and it suits well the purpose of ESNN, which is to have a fast learning algorithm that is suitable not only for offline, but also for online applications. The objective of the learning is to create a repository of output neurons with class labels. For each input sample, one output neuron will be created during learning. However, output neurons in the repository evolve according to their weight vector similarity with other output neurons. The learning process of ESNN is shown in Algorithm 1 as described by Schliebs, Defoin-Platel, Worner, and Kasabov (2009a).

The training starts with initialisation of three ESNN parameters - modulation factor (Mod), proportion factor (C) and similarity value (Sim) in the interval $[0, 1]$. Every sample i that belongs to the same class l is encoded into several pre-synaptic input neurons j. The value of weight w_j is computed according to the Mod_l and $order(j)$. Mod_l is the modulation factor of the Thorpe neural model. The $order(j)$ represents the rank of the spike emitted by neuron j. For example, the first spike will be assigned with 0, second 1... and so on.

$$w_j = (Mod_l)^{order(j)}, \quad \forall\, j \mid j \text{ pre-synaptic neuron of } i \tag{2.25}$$

Based on the weight computed, the maximum possible PSP u_{max} is calculated using Formula 2.26

$$u_{max} = \sum_j w_j (Mod_l)^{order(j)} \tag{2.26}$$

The firing threshold ϑ is calculated as follows:

$$\vartheta = C_l \cdot u_{max} \tag{2.27}$$

where C_l is the proportion factor with a value between $[0, 1]$.

As the training process continues, every sample produces an output neuron. The similarity of output neurons is calculated according to the Euclidean distance between the weight vector of the neurons. The parameter Sim controls the similarity distance. If a certain neuron is considered too similar to others, it will merge with the most similar one. The merging process involves the calculation of the mean of the weight vector (Equation 2.28) as well as the threshold value (Equation 2.29). N represents number of samples previously used to update output neuron k.

$$w_j^{(k)} \leftarrow \frac{w_j^{(i)} + N w_j^{(k)}}{1 + N}, \quad \forall j \mid j \text{ pre-synaptic neuron of } i \tag{2.28}$$

$$\vartheta^{(k)} \leftarrow \frac{\vartheta^{(i)} + N \vartheta^{(k)}}{1 + N} \tag{2.29}$$

Figure 2.12 shows a simplified architecture of ESNN where each input value is encoded into multiple pre-synaptic neurons. This process will transform input values into a high dimensional structure where each pre-synaptic neuron

generates a certain spike at a firing time. The firing time is calculated using the intersection of the Gaussian function with the input value. Based on the firing time, a weight for each connection to the output neuron is generated. In the training process, the output neuron stores the computed weight of all pre-synaptic neurons, a threshold value to determine when the output neuron will spike and the class label to which the input sample belongs. In the testing process, similar to the training process, each testing sample is encoded to spikes by the multiple pre-synaptic neurons. Then, the PSP of the output class neurons is calculated. Once the neuron receives certain amount of spikes and the PSP exceeds the threshold value, it fires an output spike and becomes disabled. The testing sample belongs to the output class defined by the output neuron that fires first among all output neurons. The major advantages of ESNN are its fast learning and ability of the trained network to incrementally learn new samples without retraining (Schliebs, Defoin-Platel, & Kasabov, 2009a)

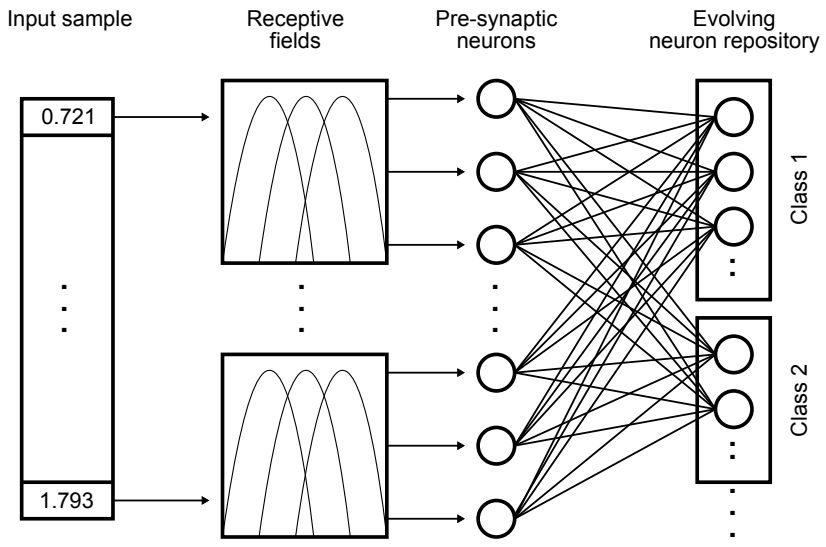

Figure 2.12: A simplified architecture of ESNN.

ESNN is widely applied in classification tasks, such as face recognition (Wysoski et al., 2006b), person authentication based on audiovisual information (Wysoski et al., 2006a), taste recognition (Soltic et al., 2008), ecological problems (Schliebs, Defoin-Platel, Worner, & Kasabov, 2009b) and has achieved better results than traditional methods. ESNN and SNN in general

have a potential for solving complex problems. They were also tested in Maass (1997); Thorpe et al. (2001); Bohte and Kok (2005); Belatreche, Maguire, and McGinnity (2006); Brette et al. (2007).

2.8 SUMMARY

This chapter has discussed the principle elements of SNN that are also the core elements of ESNN. ESNN inherits some SNN's components, particularly data encoding methods and neuron models. However, the unique evolving structure of ESNN makes this classifier able to retrain on new data samples without having to use the entire dataset. Another advantage of this method is the one-pass learning that enables the classifier to learn quickly. However, it can direct the classifier to unsatisfactory results because of the fixed parameter setting during learning. If the parameter is incorrect, it is hard to achieve the optimal result. Because of the optimal parameters are crucial in ESNN, the optimiser is necessary.

In the next chapter, the optimiser for the classifier will be discussed. There are many optimisers available and some with extensive modifications. Choosing an optimiser is equally important to choosing the classifier itself. With the right combination, they produce a stable architecture and more accurate results.

Chapter 3

REVIEW OF QUANTUM-INSPIRED EVOLUTIONARY ALGORITHMS AND SPIKING NEURON METHODS

This chapter discusses the Quantum-inspired PSO (QiPSO). First the basics of PSO are introduced and followed by a discussion of the fundamental elements of quantum computing. Second, the concept of the Quantum-inspired Evolutionary Algorithm (QEA) is explained followed by the explanation of QiPSO that inherits some elements from PSO and QEA.

3.1 INTRODUCTION TO EVOLUTIONARY ALGORITHMS

An Evolutionary Algorithm (EA) is essentially an algorithm inspired by the principle of natural selection and natural genetics (Goldberg, 1989). EA is a population-based search method that simulates the biological evolutionary process and mechanisms such as selection, recombination, mutation and reproduction in order to solve optimisation problems. In EA, each individual in a population plays a role as a candidate solution for the target problem. Each individual is evaluated by a fitness function that determines its quality. The best individual will be selected as a parent for reproduction of new individuals or solution candidates. Parents reproduce by undergoing operations such as recombination and mutation. Recombination sometimes referred as crossover, is a process where two selected parents exchange chromosome information and this results in one or two new candidate solutions. Mutation is a reproduction process that involves only one parent where information or genetic material is randomly altered to produce new offspring, i.e. candidate solutions. The new candidate will then compete with other candidates to achieve the best fitness

in the next generation (iteration). This process is repeated until a stopping criterion is met such as maximum number of generations is reached or a targeted solution has been found.

Several population-based algorithms that follow the EA concept have been introduced. Amongst all, GA is arguably one of the most commonly used evolutionary techniques and it has been utilised in many applications. The GA concept was first studied in 1960s and became popular after an article was published by Holland (1975). The original idea came from the biological evolution process. GA exploits the idea of the survival of fittest where best solutions are recombined with each other to form new better solutions. The process in GA starts when a population of chromosomes is created, then each individual's fitness is measured. There are two approaches for the reproduction stage. The first approach, called mutation alters the current state of a chromosome to produce a better candidate. The second approach is crossover, where the process selects two chromosomes and the information from both chromosomes is exchanged to create two new chromosomes. The fitness of the new individual is then evaluated. The process is repeated until the stop condition is met.

Another population-based technique that is attracting more attention recently is Swarm Intelligence (SI). SI is defined as any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of the social insect colonies and other animal societies, such as ant colonies, bird flocking and school of fish (Eberhart & Kennedy, 1995; Bonabeau, Dorigo, & Theraulaz, 1999). This population system is made up from a population of candidates interacting with each other in the swarm and leads to a global behaviour. Algorithms in this category are: Particle Swarm Optimisation (Eberhart & Kennedy, 1995), Ant Colony Optimisation (Dorigo, Maniezzo, & Colorni, 1996), Fish School Algorithm (Li, Shao, & Qian, 2002), Bee Colony Optimisation (Karaboga, 2005) and Firefly Algorithm (Xin-She, 2009). However, among all these algorithms, PSO that pioneered the SI, receives the most attention because of its relative simplicity and effectiveness (Bergh & Engelbrecht, 2000).

3.2 PARTICLE SWARM OPTIMISATION

3.2.1 *Principle of PSO*

PSO is one of the algorithms based on the EA concept and was first introduced by Eberhart and Kennedy (1995). PSO is a biologically-inspired technique based around the study of collective behaviour in decentralised and self-organised animal society systems. The systems are typically made up from a population of candidates (particles) interacting with one another within their environment (swarm) to solve a given problem. The particle is initialised by assigning random positions and velocities to particles and potential solutions are then flown through the hyperspace. Unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, each particle has their own fitness value calculated during the optimisation process and the best fitness value achieved so far is stored and normally referred to as personal best or individual best ($pbest$). The overall best fitness value obtained by any particle in the population so far is called global best ($gbest$) and it stores the best solution. The particles learn over time in response to their own experience and the experience of the other particles in their group (Ferguson, 2004). According to Eberhart et al. (2001), each particle keeps track of its best fitness position in hyperspace that has been achieved so far. During each iteration (or epoch), every particle is accelerated towards its own $pbest$ as well as in the direction of the $gbest$ position. The value of $pbest$ and $gbest$ would influence the direction of the particle in the next iteration (Bergh & Engelbrecht, 2000). This is achieved by calculating a new velocity term for each particle based on the distance from its $pbest$, as well as its distance from the $gbest$ position. Figure 3.1 shows the basic PSO procedure.

In order to create a swarm of n particles, at all time points t, each particle n has:

1. A current position X_n
2. A velocity direction V_n
3. A record of its own previous best position $pbest_n$

4. A record of the previous best position of any member in its group $gbest_n$

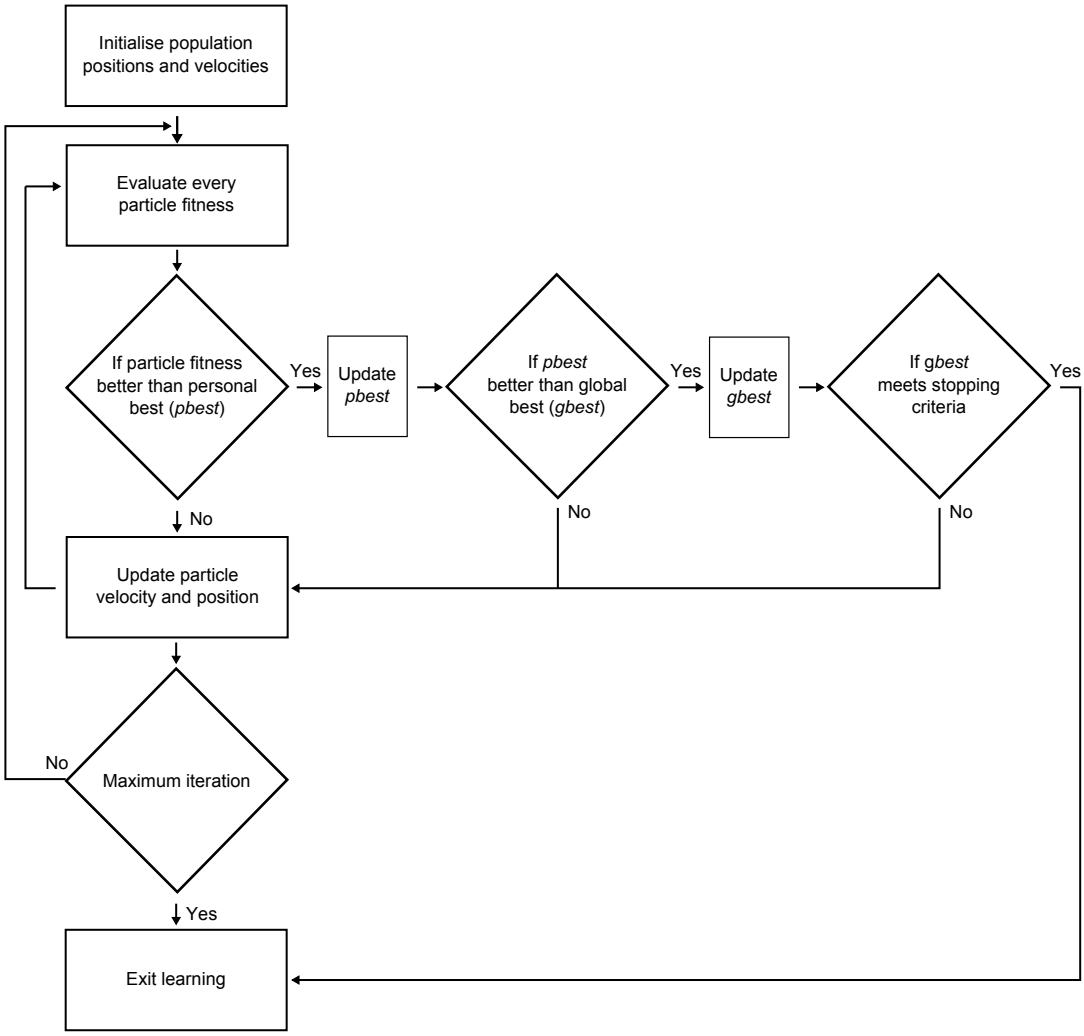

Figure 3.1: PSO flowchart.

Given the current position of each particle, as well as other information, the problem is to determine the change in direction of the particles. As mentioned above, this is done by reference to each particle's own experience and its companions. Its own experience includes the direction V_n and its own $pbest_n$ position. The experience of others is represented by the best previous position of any member in its group $gbest_n$. This suggests that each particle might move to the direction of:

1. The same direction that it comes from V_n
2. The direction of its previous best position $pbest$
3. The direction of the previous best position of any member in its group $gbest$

The next trajectory vector of a particle is calculated using Equation 3.1. This is a formula modified by Shi and Eberhart (1998a) from the original proposed formula where an inertia weight w is added to control the speed of the velocity movement. $rand_1$ and $rand_2$ are two uniform random numbers in interval $[0, 1]$. c_1 and c_2 are constants called the cognitive and social parameters that control the exploration direction between $pbest$ and $gbest$.

$$V_{n,t+1} = w \cdot V_{n,t} + c_1 \cdot rand_1 \cdot (pbest - X_{n,t}) + c_2 \cdot rand_2 \cdot (gbest - X_{n,t}) \quad (3.1)$$

Then, the new position of the particle will simply be:

$$X_{n,t+1} = X_{n,t} + V_{n,t} \quad (3.2)$$

Given the initial values of X_n, V_n, $pbest$ and $gbest$, Equation 3.1 and Equation 3.2 will determine the subsequent path that each particle in the swarm will follow. To avoid particles flying beyond the boundary, the velocities are clamped to a maximum velocity V_{max} (Eberhart, Shi, & Kennedy, 2001). If the sum of accelerations causes the velocity of that dimension to exceed V_{max}, which is a pre-defined parameter, then the velocity is limited to V_{max}.

3.2.2 *A Computation Example*

Figure 3.2 and the following computation demonstrates how a particle (Particle A) moves to the solution $gbest$ in a 2D space problem and is recalculated from an example in Jones (2005). In this example, parameter $w = 1.0$, $c_1 = 0.5$ and $c_2 = 1.0$. Since the value of c_2 is higher than c_1, Particle A will give more weight to the global solution. Assume that Particle A velocity value

calculated in previous iteration is $V_{A,t} = (0, 1)$. Particle A current position at coordinate (x, y) is $(10, 5)$ as shown in Figure 3.2(a), $pbest$ value $(5, 13)$ and $gbest$ at $(15, 13)$. First, the velocity vector must be updated for the current iteration using Equation 3.1.

Calculate the velocity $V_{x,t+1}$ of Particle A:

$$\begin{aligned} V_{x,t+1} &= w \cdot V_{x,t} + c_1 \cdot rand_1 \cdot (pbest - X_x) + c_2 \cdot rand_2 \cdot (gbest - X_x) \\ &= 1.0 \cdot 0.0 + 0.5 \cdot 0.10 \cdot (5.0 - 10.0) + 1.0 \cdot 0.35 \cdot (15.0 - 10.0) \\ &= 1.0 \cdot 0.0 + 0.05 \cdot (-5.0) + 0.35 \cdot (5.0) \\ &= 1.0 \cdot 0.0 + (-0.25) + 1.75 \\ &= 1.5 \end{aligned}$$

Calculate the velocity $V_{y,t+1}$ of Particle A:

$$\begin{aligned} V_{y,t+1} &= w \cdot V_{y,t} + c_1 \cdot rand_1 \cdot (pbest - X_y) + c_2 \cdot rand_2 \cdot (gbest - X_y) \\ &= 1.0 \cdot 1.0 + 0.5 \cdot 0.45 \cdot (13 - 5.0) + 1.0 \cdot 0.20 \cdot (13.0 - 5.0) \\ &= 1.0 \cdot 1.0 + 0.225 \cdot (8.0) + 0.20 \cdot (8.0) \\ &= 1.0 \cdot 1.0 + 1.80 + 1.60 \\ &= 4.4 \end{aligned}$$

Based on current velocity value of Particle A $V_A = (1.5, 4.4)$, the particle position is updated using Equation 3.2

$$\begin{aligned} x_{A,t+1} &= x_{A,t} + V_A \\ &= 10.0 + 1.5 \\ &= 11.5 \end{aligned}$$

$$\begin{aligned} y_{A,t+1} &= y_{A,t} + V_A \\ &= 5.0 + 4.4 \\ &= 9.4 \end{aligned}$$

The new position for Particle A (11.5, 9.4) is shown in Figure 3.2(b).

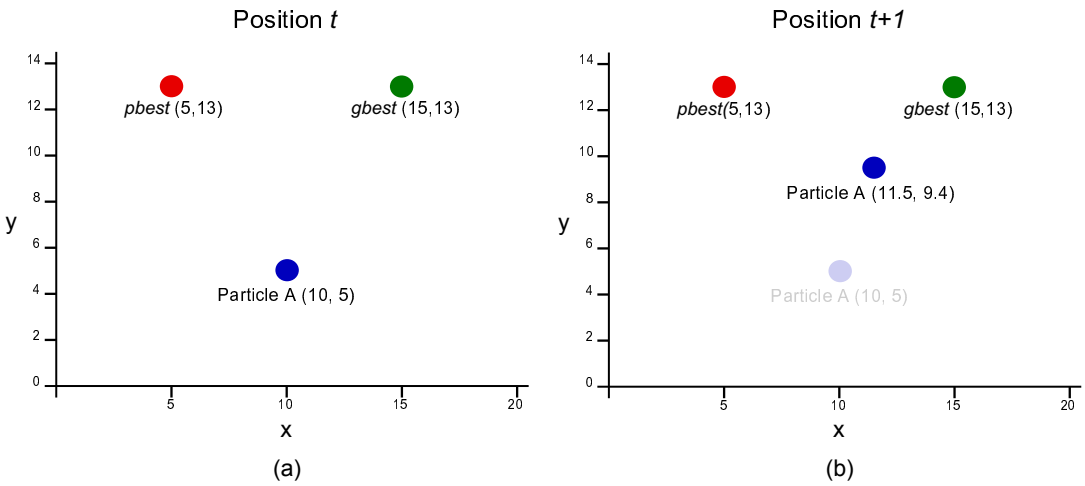

Figure 3.2: Particle movement in 2D space problem.

It is apparent that PSO shares many common features with GA. Both algorithms start with a randomly generated population, and use a fitness function to evaluate the population. Both methods update the population and all individual search for the optimum solutions. However, PSO particles update themselves with the internal velocity, and have memory as storage of history. In PSO, $gbest$ shares the information with others in the population. This makes parti-

cles in PSO more intelligent due to the information sharing mechanism and all the particles tend to converge to the solution represented by $gbest$ quickly.

3.2.3 *Applications*

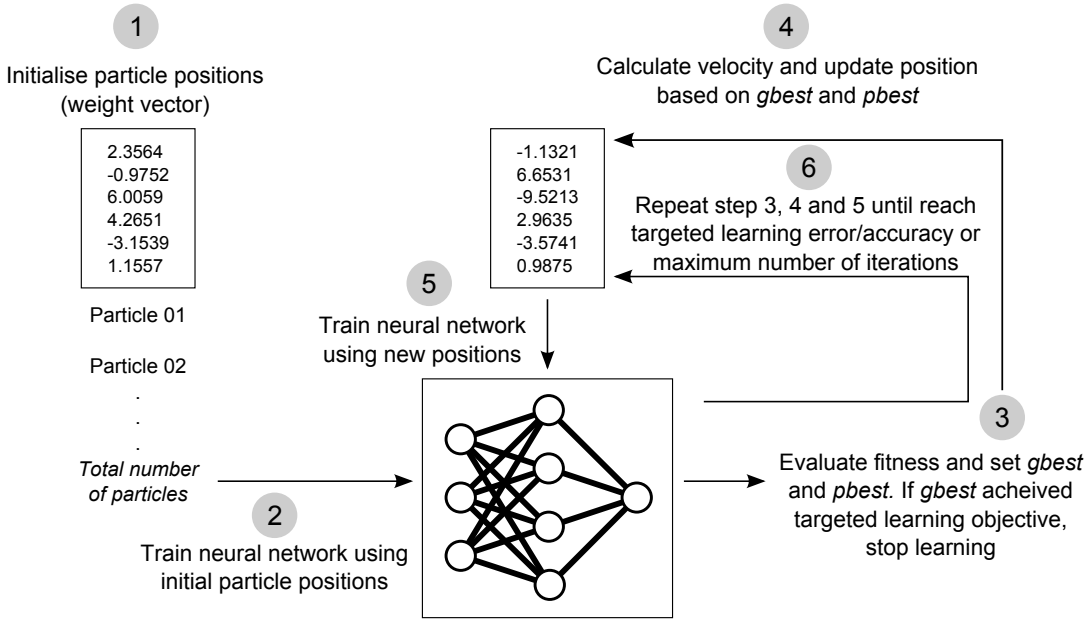

Figure 3.3: A diagram of neural network learning using PSO.

Initially PSO was tested on parameter and function optimisation (Shi & Eberhart, 1999; Angeline, 1998; Clerc, 1999). Later, the algorithm has been widely applied for the neural network learning such as in Bergh (1999); Bergh and Engelbrecht (2000); Zhang, Shao, and Li (2000); Mendes, Cortez, Rocha, and Neves (2002); Gudise and Venayagamoorthy (2003); Meissner, Schmuker, and Schneider (2006). The particle's position represents the problem to be optimised such as weight in neural network learning. The particle moves within the weight space attempting to minimise the learning error. Changing the position implies updating the weight of the network in order to reduce the error of the current iteration. The new position thus uses a set of new weights to obtain the new error. The particle with the lowest error is normally considered as the global best solution. The training process continues until satisfactory error is

achieved by the best particle, or when computational limits are reached. When the training ends, the weights are used to calculate the classification error for the training patterns. The same set of weights is used then to test the network using the test patterns. A diagram representing ANN with PSO learning process is shown in Figure 3.3. Figure 3.4 shows the particle movement during learning when the ANN-PSO is applied to a simple Exclusive OR (XOR) problem. All particles are trying to reach the lowest error possible.

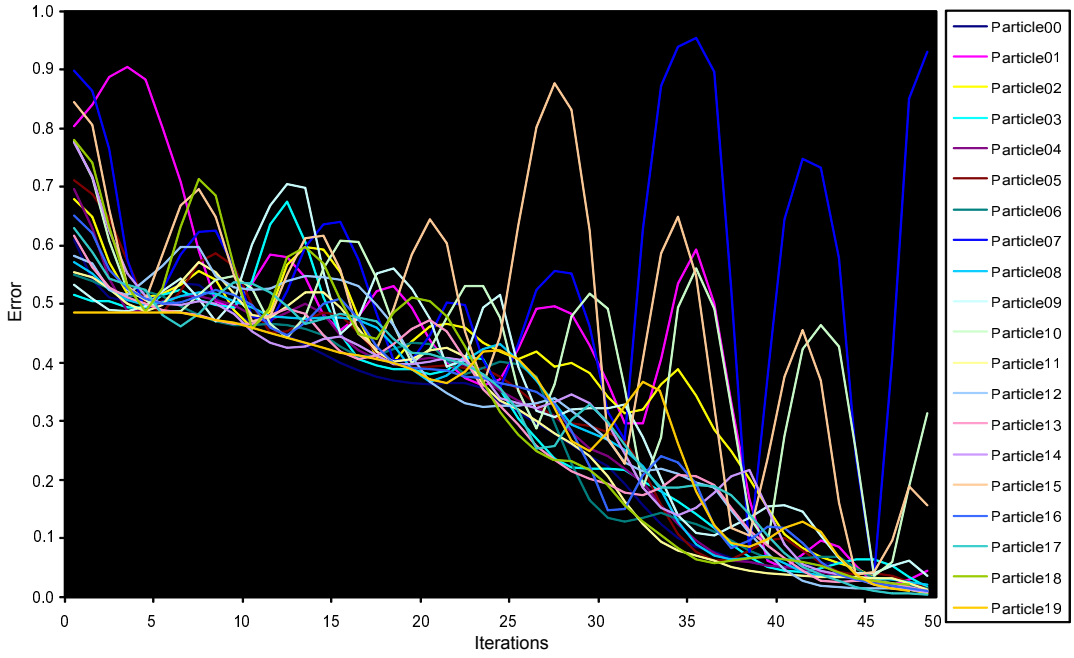

Figure 3.4: Particle behaviour during learning process.

Because of its efficiency and simplicity, PSO has also been applied as an optimiser in many other applications such as in engineering control systems (Yoshida, Kawata, Fukuyama, Takayama, & Nakanishi, 2000; Khodier & Christod, 2005; Valle, Venayagamoorthy, Mohagheghi, Hernandez, & Harley, 2008), multiobjective optimisation (Coello, Pulido, & Lechuga, 2004; Hu & Eberhart, 2002), biomedical applications (Wachowiak, Smolikova, Zheng, Zurada, & Elmaghraby, 2004; Zainud-Deen, Hassen, Ali, Awadalla, & Sharshar, 2008), image classification and clustering (Omran, Engelbrecht, & Salman, 2006; Peng-Yeng, 2004) and others. However, despite recent research and development,

there is still an opportunity to enhance the methods for parameter optimisation and feature selection tasks.

3.3 QUANTUM-INSPIRED ALGORITHMS

Information in the physical world is represented by some physical system. Quantum information represented by quantum physical systems differs from its classical counterpart in many notable ways. For example, quantum information cannot be cloned arbitrarily (Wootters & Zurek, 1982). Since classical computing can be described as manipulating classical information, quantum computing is, in the same spirit, manipulation of quantum information. It is possible that the properties of quantum information can help to resolve some computational tasks more efficiently than when classical information representation is used. In fact this was suggested already in Feynman (1982), but the most interesting example was given in a very remarkable discovery where Shor (1994) demonstrated that quantum computers would allow efficient integer factorisation, a task assumed impossible for classical information processing. Several notable quantum algorithms are presented in Hirvensalo (2001) and Nielsen and Chuang (2000). It is worth emphasising here that the efficiency of quantum computing comes from the ingenious use of the superposition principle, not from the high ”clock frequency” of quantum computers.

While there are certain technological limitations and accessibility problems related to quantum computers, the quantum information principles have been proved to be useful for the development of new evolutionary optimisation algorithms that run on contemporary computers (Nielsen & Chuang, 2000; Hirvensalo, 2001; Han & Kim, 2002; Jang, 2004; Talbi, Draa, & Batouche, 2006; Defoin-Platel, Schliebs, & Kasabov, 2007; Abs Da Cruz, Vellasco, & Pacheco, 2007; Luitel & Venayagamoorthy, 2010). As shown in Narayanan (1999) in relation to classical neural networks and Kasabov (2007, 2009) in relation to spiking neural networks, quantum computing principles have been seen as a source of inspiration for novel computational methods. Two famous quantum applications are the factorisation problem (Shor, 1994) and the Grover’s database search algorithm (Grover, 1996).

3.3.1 *Quantum Computation Principles*

In classical computing, information is represented in bits where each bit must hold a value of either 0 or 1. However, in quantum computing, information is instead represented by a quantum bit (qubit) where the value of a single qubit could be 0, 1, or a superposition of both (Hey, 1999). Superposition is a state that represents both 0 and 1 simultaneously based on their probability. The quantum state is modelled by the Hilbert space of wave functions and is defined as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{3.3}$$

where α and β are complex numbers defining probabilities at which the corresponding state is likely to appear when a qubit collapses, for instance, when reading or measuring the state. Probability fundamentals state that $|\alpha|^2+|\beta|^2 = 1$, where $|\alpha|^2$ gives the probability that a qubit is in the OFF (0) state and $|\beta|^2$ gives the probability that a qubit is in the ON (1) state.

3.3.2 *Quantum Gates*

The probability of α and β can be modified by applying quantum gates. Similarly to classical logic gates that perform conversion operations in classical logic computation, quantum gates execute conversion operation on qubits. Several quantum gates are available and some are designed for a specific n-bit problems. The NOT-gate is the most commonly used gate in classical circuits. This gate simply inverts the qubit value, changing it from 0 to 1 or from 1 to 0. For qubit problems, a NOT gate has the capability to exchange the probability in the superpositioned states. For example, changing the probability that the qubit will collapse at $|0>$ with the probability of collapse at $|1>$, $\alpha|0> +\beta|1>$ becomes $\beta|0> +\alpha|1>$.

The Walsh-Hadamard Transformation Gate is simply refered to as the Hadamard Gate. This gate is designed for one bit transformation problems and is widely used in quantum computation problems. The objective is to place the

un-superpositioned qubit into a superposition of $|1>$ and $|0>$ states. The transformation is defined by the Hadamard matrix in Equation 3.4. The role of Hadamard gates in quantum computation is discussed in Shepherd (2006).

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.4}$$

The second quantum gate is the general-purpose Rotation Gate. The Rotation Gate is dependent on the value θ and the transformation is conducted using Equaition 3.5. θ is the quantum angle that will be discussed in Section 3.4.2. In Zhang, Zhang, Rong, and Cheng (2010), the authors present six types of rotation gates that are derived from the standard architecture. An interesting investigation is conducted to compare all gates on image sparse decomposition problems. The experimental result shows that the QR-Gate5 gives the best results in term of the best visual quality and the highest peak signal to noise ratio on the constructed images.

$$U_\theta = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.5}$$

Apart from these two commonly used gates, there are some other gates, for example, the Phase shift Gate where one component is changed when there is dissimilarity between two events. The Toffoli gate (Toffoli, 1980) applies NOT operation and is mostly used in reversal tasks where input and output matrix must be of the same dimension. This gate is normally formed in three qubit format and only reverses the third qubit if the two qubits are in the $|1>$ state. Similarly to the Toffoli gate, the Fredkin gate (Fredkin & Toffoli, 1982) is also designed for reversal operation. This three-qubit gate swaps the last two qubit if the first qubit is in the $|1>$ state. However, there is no swap operation if the first qubit is $|0>$. There are a few other quantum gates such as Controlled NOT gate, Square-Root-NOT gate, Pauli gate and Swap gate. All these gates are explained in many quantum computation publications such as Barenco et al. (1995) and Nielsen and Chuang (2000).

3.4 QUANTUM EVOLUTIONARY ALGORITHM

QEA was inspired by the concept of quantum principle and was popularised by Han and Kim (2000). Since then, this technique has attracted the attention of many researchers around the world due to its many advantages when compared to the classical EA. Building up on the basic EA concept, QEA is a population-based search method that simulates a biological evolution process and mechanisms, such as selection, recombination, mutation and reproduction. Each individual in a population plays a role as a candidate solution and its fitness to solve a given task is evaluated. However, instead of using real number, information in QEA is represented in qubit. The value of a single qubit could be 0, 1, or a superposition of both. A single qubit is the smallest information unit and can be defined as $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$ which satisfies the probability fundamentals stating that $|\alpha|^2 + |\beta|^2 = 1$ as explained in Section 3.3.1. A QEA individual is represented as a qubit vector $\begin{bmatrix}\alpha_1,\alpha_2,\ldots,\alpha_N\\ \beta_1,\beta_2,\ldots,\beta_N\end{bmatrix}$ where α and β are complex numbers defining probabilities at which the corresponding states are likely to appear when a qubit collapses, for instance, when reading or measuring its value. N represents the problem dimension. Figure 3.5 describes QEA as it was originally explained in Han and Kim (2002) and later applied in Defoin-Platel et al. (2007).

The three levels of QEA are the individual, group and population level. An individual i generated at time t holds a string of qubit of N number - $Q_i(t)$ as in Equation 3.6.

The probability value of $|\beta_i^N|^2$ will determine the state when a qubit collapsed. $C_i(t)$ represents the collapsed value that will be used to determine the fitness of i. Attractor $A_i(t)$ always keeps the best solution for a particular individual i. In every iteration, the fitness of $C_i(t)$ and $A_i(t)$ are compared. If the fitness value of $A_i(t)$ is better than $C_i(t)$, then the qubit of $Q_i(t)$ will be updated using the rotation gate update technique described by Equation 3.7. In this situation, the value of $Q_i(t)$ will be moving to attractor $A_i(t)$. In contrast, if $C_i(t)$ fitness is better than $A_i(t)$, $A_i(t)$ will simply be replaced by $C_i(t)$. The rotation angle θ determines the direction of rotation (clockwise for negative values and bounded in the range of $[0, \pi/2]$(Han & Kim, 2003). In the second level which is referred to as a quantum group, there will be several k individuals. The

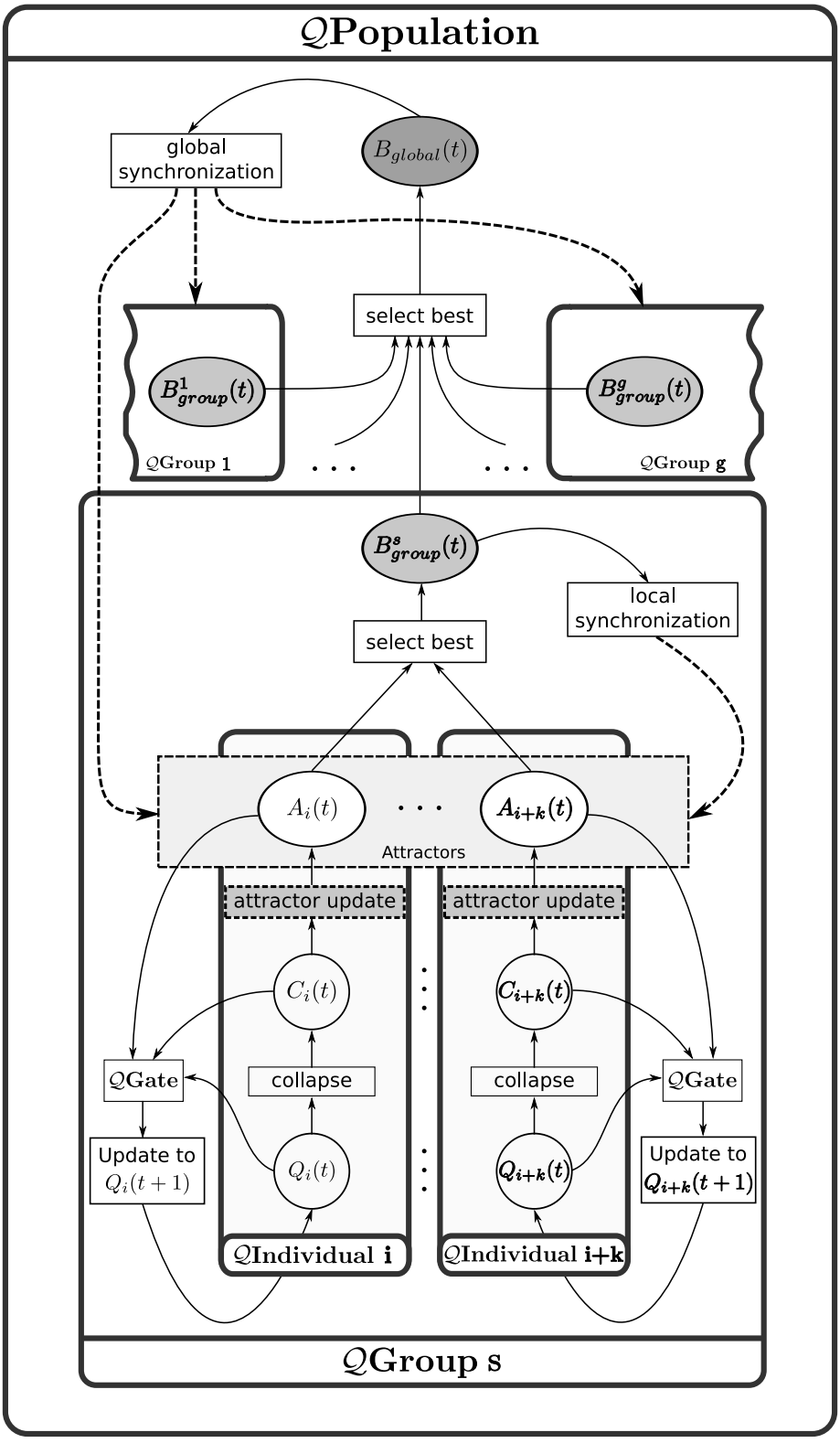

Figure 3.5: The three levels of QEA: individual (at bottom), group (in the middle) and population (at top).

best attractor from the group B_{group}, will be stored and used to update another individual attractor in the group. Finally, in the top population level several groups create a population and the best attractor among the groups B_{global} is stored. B_{group}, and B_{global} will be evaluated periodically during the optimisation process. QEA have been reported to be successful for solving complex benchmark problems (Feng, Wang, Ge, Zhou, & Liang, 2006; Abs Da Cruz et al., 2007), multiobjective optimisation (Talbi et al., 2006; Kim, Kim, & Han, 2006) and several real world problems (Jang, 2004; Fan, Brabazon, O'Sullivan, & O'Neill, 2007; Gu, Gu, Cao, & Gu, 2010).

$$Q_i = Q_i^1 Q_i^2 \dots \ Q_i^N = \begin{bmatrix} \alpha_i^1 & \alpha_i^2 & \dots & \alpha_i^N \\ \beta_i^1 & \beta_i^2 & \dots & \beta_i^N \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} \alpha_i^j(t+1) \\ \beta_i^j(t+1) \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \begin{bmatrix} \alpha_i^j(t) \\ \beta_i^j(t) \end{bmatrix} \tag{3.7}$$

Several modification on QEA have been proposed. Some new elements have been added to improve search robustness and to provide better convergence during optimisation (Han & Kim, 2004; You, Liu, & Shuai, 2006; Sun, Xu, & Fang, 2006; Xiao, Xu, Chen, Zhang, & Pan, 2009). Defoin-Platel et al. (2007) and Schliebs, Defoin-Platel, Worner, and Kasabov (2009a) proposed an extended version of QEA called Versatile QEA (vQEA) and applied it into ESNN optimisation. The result produced a faster convergence to the optimal solution with better accuracy when compared to traditional neural networks such as MLP and Naïve Bayesian Classifier (NBC). Some principles of quantum computation and QEA have been also implemented in other well known optimisers such as GA and PSO.

3.4.1 *Quantum Genetic Algorithm*

Adapting the principle of GA and quantum computation, the Quantum-inspired Genetic Algorithm (QiGA) was first discussed in Narayanan and Moore (1996).

Algorithm 2 QiGA procedure

1: Create population with n-number of qubit chromosome
Do
2: Evaluate each individual's fitness
3: Select two individual with the best fitness
4: Execute the quantum crossover operation
5: Execute the quantum mutation operation
6: Check for termination criteria
While (stopping criteria not met)

The development of QiGA has been described in Zhang, Li, Jin, and Hu (2006), starting with the introduction of QiGA concept and theory (Narayanan & Moore, 1996), implementation of quantum operator (Han & Kim, 2000), introduction of parallel QiGA (Han, Park, Lee, & Kim, 2001), improving performance with quantum crossover and quantum mutation (Li & Zhuang, 2002), multiqubit encoding and dynamic rotation angle mechanism (Yang, Li, & Zhuang, 2003) plus other improvements on QiGA such as in Zhang et al. (2006); Jian, Li-juan, Ru-chuan, and Zhong-gen (2009).

The solution candidates or the chromosomes are presented as a string of qubits and the quantum operations are applied to the chromosomes. A series of chromosomes assemble the population. Then, every chromosome is guided by the classifier or other algorithms to solve the given problem. The chromosome update strategy is based on the standard GA. The parents are selected based on their fitness to create new offspring chromosomes. Algorithm 2 explains the general QiGA procedure and further detail about QiGA algorithms can be found in Li and Wang (2007); Layeb and Saidouni (2007); Gu, Gu, and Gu (2009).

Several advantages of QiGA, such as good searching capability, rapid convergence, required small population size and short computation time are discussed in Narayanan and Moore (1996); Han and Kim (2000); Li and Zhuang (2002); Zhang, Jin, and Li (2003); Jian et al. (2009). The first application of QiGA was to solve travelling salesman problem (Narayanan & Moore, 1996). Other applications of QiGA are engineering problems (Vlachogiannis & stergaard, 2009; Lee, Lin, Liao, & Tsao, 2011), image processing (Talbi, Ba-

touche, & Draa, 2004; Benatchba, Koudil, Boukir, & Benkhelat, 2006) and scheduling (Li & Wang, 2007; Gu et al., 2009).

3.4.2 *Quantum-inspired Particle Swarm Optimisation*

Quantum principles have been embedded into PSO as a mechanism for the probability calculation and normally referred to as QiPSO. The QiPSO concept was first introduced in Sun et al. (2004). This concept has been extended in Gao and Diao (2009) who employed the quantum principles explained by Han and Kim (2002). The main idea of QiPSO is to use the standard PSO function to update the particle position represented as a quantum angle (θ). The quantum angle θ has normally been used in quantum-inspired optimisation algorithms to calculate and update probability and is represented as $\left[\begin{smallmatrix}\alpha\\ \beta\end{smallmatrix}\right]$. θ corresponds to the angle in the trigonometry and is bounded to the first quadrant. Figure 3.6 shows an example where the probability is computed when the $\theta = 40^{\circ}$. Coordinates x and y represent the cosine (cos) and sine (sin) value respectively. $\left[\begin{smallmatrix}\alpha\\ \beta\end{smallmatrix}\right] = \left[\begin{smallmatrix}cos(\theta)\\ sin(\theta)\end{smallmatrix}\right]$ satisfies the probability fundamental of $|\alpha|^2 + |\beta|^2 = 1$ and can be substituted with Equation 3.8.

$$|sin(\theta)|^2 + |cos(\theta)|^2 = 1 \tag{3.8}$$

For $\theta = 40^{\circ}$, $cos(\theta) = 0.766$ and $sin(\theta) = 0.643$. Following the consideration of Equation 3.8, new probability of α obtained is 0.587 and β is 0.413.

In QiPSO, changes of the θ during learning process is crucial to measure next qubit collapse state. The formula for velocity update in standard PSO is modified to get a new quantum angle which is translated to the new probability of the qubit by using Equation 3.9. The θ_{gbest} represents the best angle stored in as $gbest$ and θ_{pbest} is the best angle found by the particle.

$$\Delta\theta_{n,t+1} = w \cdot \Delta\theta_{n,t} + c_1 \cdot rand_1 \cdot (\theta_{pbest} - \theta_{n,t}) + c_2 \cdot rand_2 \cdot (\theta_{gbest} - \theta_{n,t}) \tag{3.9}$$

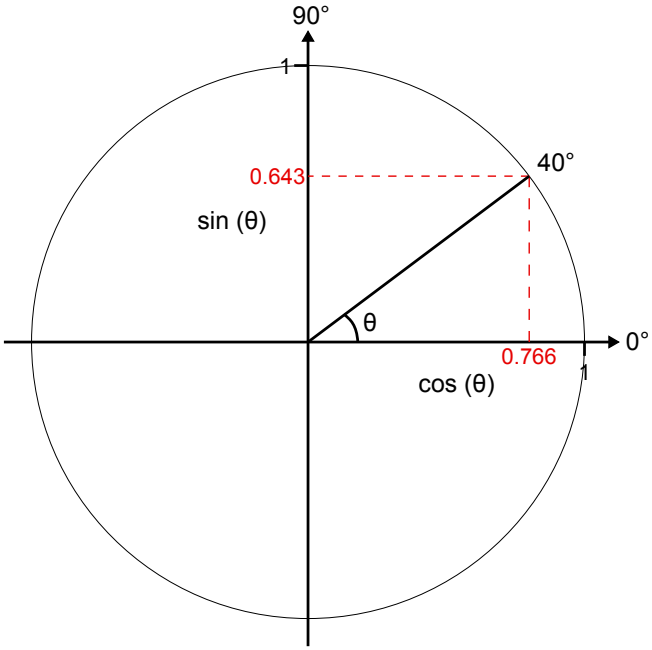

Figure 3.6: Quantum computation of probability.

There are two approaches to updating the current θ. In the first approach, the new quantum angle is determined by using a modified standard PSO position update formula (Equation 3.10). Then, based on the new θ angle, the new probability of α and β can be calculated using Equation 3.8.

$$\theta_{n,t+1} = \theta_{n,t} + \Delta\theta_{n,t} \tag{3.10}$$

The second approach uses a quantum gate to compute the new θ. The most commonly used quantum gate in quantum probability computation is the rotation gate. Based on the new θ velocity, the new probability of α and β is calculated using a rotation gate as shown in Equation 3.11.

$$\begin{bmatrix} \alpha_{t+1} \\ \beta_{t+1} \end{bmatrix} = \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) \\ sin(\Delta\theta) & cos(\Delta\theta) \end{bmatrix} \cdot \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} \tag{3.11}$$

Figure 3.7 presents an example of the qubit update in QiPSO using rotation gate. In this scenario, the previous angle θ at time t was assumed to be $22°$. After computing direction changes using Equation 3.9, change in the angle $\Delta\theta$

is found $18°$. The new probability of α and β is calculated after applying the Rotation Gate in Equation 3.11

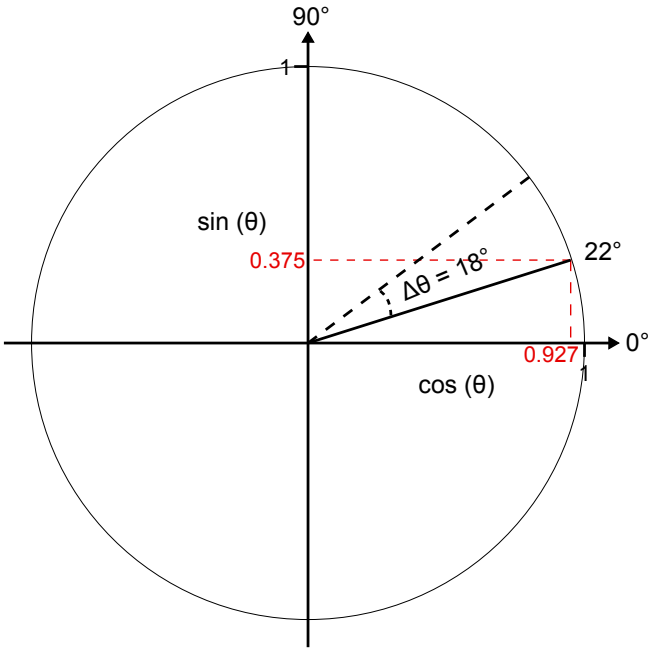

Figure 3.7: Quantum angle update in Trigonometry.

$$
\begin{aligned}
\begin{bmatrix} \alpha_{t+1} \\ \beta_{t+1} \end{bmatrix} &= \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) \\ sin(\Delta\theta) & cos(\Delta\theta) \end{bmatrix} \cdot \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} \\
&= \begin{bmatrix} cos(18°) & -sin(18°) \\ sin(18°) & cos(18°) \end{bmatrix} \cdot \begin{bmatrix} 0.927 \\ 0.375 \end{bmatrix} \\
&= \begin{bmatrix} 0.951 & -0.309 \\ 0.309 & 0.591 \end{bmatrix} \cdot \begin{bmatrix} 0.927 \\ 0.375 \end{bmatrix} \\
&= \begin{bmatrix} 0.766 \\ 0.643 \end{bmatrix}
\end{aligned}
$$

Derived from Equation 3.8, the final probability is:

$$\begin{bmatrix}\alpha_{t+1}\\ \beta_{t+1}\end{bmatrix}^2 = \begin{bmatrix}0.766\\ 0.643\end{bmatrix}^2 = \begin{bmatrix}0.587\\ 0.413\end{bmatrix}$$

After the probability of α and β is obtained, a random value within the interval $[0, 1]$ is generated to observe the collapse state. A collapse state is determined using the rule described in Equation 3.12 and is used to solve the given problems. Similarly to the standard PSO, the particle with the best fitness in the entire swarm will be assigned as $gbest$ particle. The best fitness found during learning for every individual particle is stored as $pbest$. These two solutions will be used for new α and β probability computation in the next iteration.

$$C = \begin{cases} 0, & \text{if random} > |\alpha|^2 \\ 1, & \text{if otherwise} \end{cases} \tag{3.12}$$

QiPSO has been tested with promising results on several problems such as benchmark functions (Yang, Wang, & Jiao, 2004; Wang & Zhou, 2007; Pant, Thangaraj, & Abraham, 2008; Liu & Ma, 2008), benchmark problems (Yu, Tian, & Yin, 2006; Wang, Zhang, Niu, & Yao, 2011), control systems (Mikki & Kishk, 2005; Ma, Liu, & Lin, 2007; Jeong, Park, Jang, & Lee, 2009), economic applications (Meng, Wang, Dong, & Wong, 2010), image segmentation (Lu, Liao, & Chen, 2007) and fuzzy system (Wang, Yang, Xu, & Sun, 2007; Tang & Xue, 2008)

3.5 PRINCIPLES OF QUANTUM-INSPIRED SNN

The quantum computation principles are explained in Section 3.3.1. Qubit information representation can be applied to the applications that require changes in its states during learning. The principle of feature selection and parameter optimisation of SNN using quantum representation and quantum operations

was first introduced by Kasabov (2009). As illustrated in Equation 3.13, a string of qubits (Q_t) is used to represent the whole feature set (x_t) and each feature is mapped into a single qubit. α_t defines the probability of the collapse state measured using Equation 3.12. Collapse value of 1 represents the feature is used for the learning process while 0 means the feature is discarded. The same representation is employed for parameter optimisation. A population of qubits is used to represent a real value parameter. The collapse qubit states correspond to a set of binary strings, which are later translated into a real value. Further discussion on the implementation of QiSNN principles with the proposed integrated framework between ESNN, PSO and QiPSO, whereby parameters and features are optimised simultaneously is discussed in Chapter 4 and Chapter 5.

$$\begin{array}{cccc} Q_{t,1} & Q_{t,2} & \dots & Q_{t,N} \\ & \downarrow & & \\ \alpha_{t,1} & \alpha_{t,1} & \dots & \alpha_{t,N} \\ \beta_{t,1} & \beta_{t,2} & \dots & \beta_{t,N} \\ & \downarrow & & \\ x_{t,1} & x_{t,1} & \dots & x_{t,N} \end{array} \tag{3.13}$$

Quantum operation can also be applied for spike representations. This principle has been applied in PSNM as explained in Section 2.3.6. The probability of spike emission depends on the evaluation of the qubit states. A collapse value of 1 means the spike is present while value of 0 means otherwise. Two other probability elements, the probability of connection existence and that of spike contribution to PSP calculation in pSNM can also be represented using qubit operations as explained in Kasabov (2010). Chapter 6 will explain the implementation of PSNM in ESNN framework.

3.6 SUMMARY

This chapter presented a review of quantum-inspired PSO. It first discussed the concept of a biologically-inspired optimiser, the PSO. The strategy is simple yet very efficient. Unlike GA, each particle in a swarm interacts with each other and contributes to the solution through $gbest$ particle. Every particle has its chance to be $gbest$ particle and the $gbest$ information is shared with other particles. At the same time, every particle memorises its own best solution called $pbest$. $gbest$ and $pbest$ transform a swarm of particles to an effective and intelligent solution finder. Second, the chapter explains the QiSPO that is derived from PSO where the main objective is to solve binary problems. The QiPSO update the quantum angle to determine a particular state.

With the understanding of ESNN, PSO and QiPSO methods and together with the concept of ECoS and quantum probability, the first integration between these methods will be discussed in Chapter 4. Since ESNN requires optimisation in order to work effectively, the PSO and QiPSO optimisers will be investigated for this task.

Chapter 4

PROPOSED METHODS FOR INTEGRATING ESNN, PSO AND QIPSO

Many real world problems require optimisation for several reasons. Two common reasons are difficulty in tuning the optimal values and speeding up the learning process. The same is also valid for ESNN where parameter optimisation is necessary as explained by Wysoski et al. (2006b) and Schliebs, Defoin-Platel, Worner, and Kasabov (2009a). In this chapter, novel methods for the integration of PSO and QiPSO with ESNN are introduced for the first time and results are compared with well known classifier algorithms. The chapter explains first the integration of ESNN and PSO and then explains the integration of ESNN and QiPSO. Discussion of the method and obtained results is presented later in the chapter.

4.1 INTEGRATED ESNN-PSO FOR PARAMETER OPTIMISATION

In neural network models, an optimal combination of parameters can influence their performance. It is not feasible to manually adjust the parameters, particularly when dealing with different combinations for different datasets. Consequently, parameter optimisation is vital and much research has been conducted on it (Bäck & Schwefel, 1993). Like other models, ESNN is sensitive to its parameters. Optimal combinations of parameters lead to better classification accuracy. It is inappropriate to manually adjust the parameters to find the correct combination since this process would be inefficient and unsystematic. The performance, advantages and capabilities of PSO in solving problems effectively, have drawn much attention to this technique. Therefore, this chap-

ter proposes how PSO can be used for ESNN optimisation. ESNN is known as a fast and efficient online processing method and its first application is for fast visual processing application (Wysoski et al., 2006b). For all proposed integrated frameworks in this study, the optimisation of the ESNN classifier is performed in offline mode before the optimised ESNN can be used for online classification.

4.1.1 *Framework*

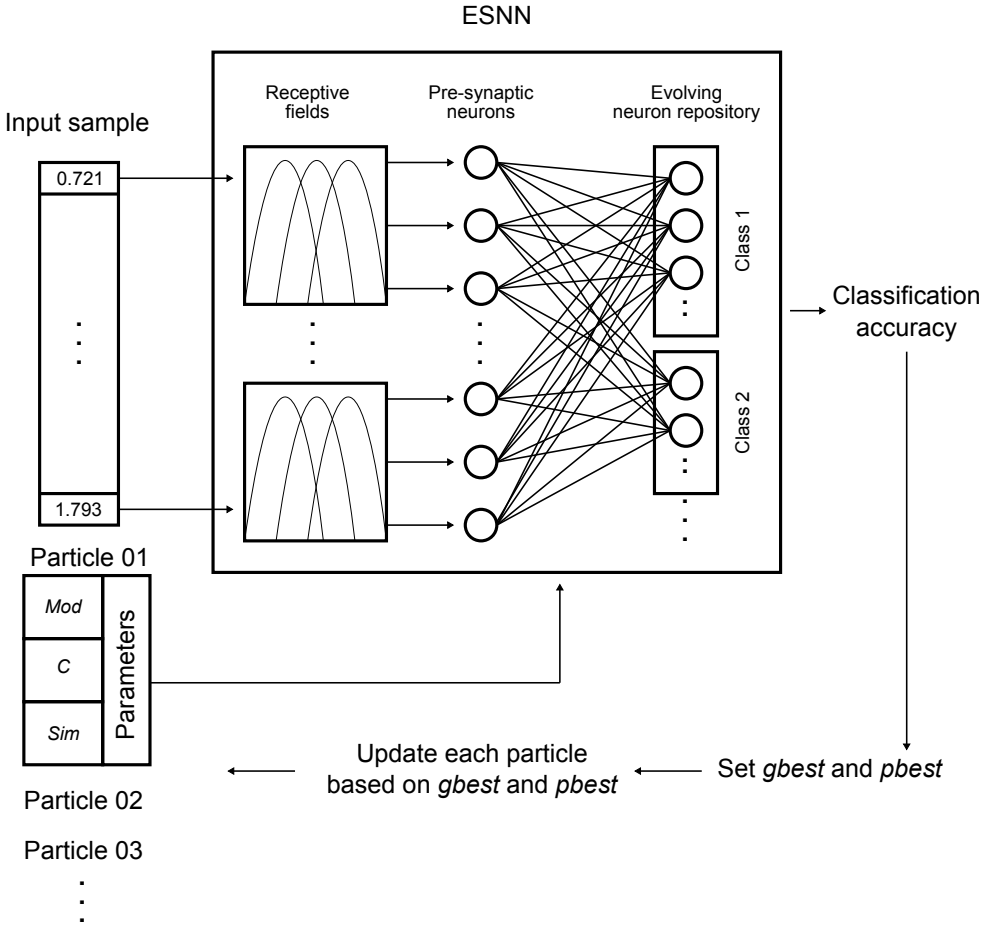

Figure 4.1: The integrated framework of ESNN-PSO.

The proposed ESNN-PSO framework is shown in Figure 4.1. The framework integrates ESNN (the classifier) and PSO (the parameter optimiser for

ESNN). The middle part of the diagram represents ESNN as comprehensively explained in Chapter 2. Since information in ESNN is represented as spikes, input data must be encoded in spike pulses. Population Encoding in ESNN distributes a single input value to multiple pre-synaptic input neurons. Each pre-synaptic neuron generates a spike at a certain firing time using Equation 2.4 and the illustration of this encoding process is shown in Figure 2.5. ESNN utilises the Thorpe's neuron model (Thorpe, 1997) because of its effectiveness and simplicity. The fundamental concept of this model is that the earlier spikes received by a neuron have a stronger weight compared to the later spikes. Once the potential reaches a certain amount of spikes and the PSP exceeds the threshold value, the neuron fires and becomes disabled. The neuron in this model can only fire once. The computation of the PSP in this model is explained in Equation 2.17. Since the first intention of ESNN is to introduce a fast classifier for online classification, the one-pass learning has been utilised in the classifier.

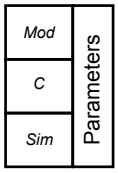

Figure 4.2: The particle structure in ESNN-PSO framework.

The integration is performed using the well known Wrapper approach. This approach was first introduced in John, Kohavi, and Pfleger (1994) and comprehensively discussed later in Kohavi and Sommerfield (1995) and in Kohavi and John (1997). The Wrapper approach combines the classifier with an optimisation algorithm. In this case, PSO interacts with ESNN to optimise the ESNN parameters, namely modulation factor (Mod), proportion factor (C) and neuron similarity value (Sim). All particles are initialised with a set of random values and they subsequently interact with each other based on classification accuracy. Every particle holds certain parameter values as shown in Figure 4.2 and uses these values to construct ESNN. Then, the constructed ESNN takes the input samples and classifies them according to their targeted classes. A fitness function to evaluate particle's performance is calculated based on the

classification accuracy. Valko, Marques, and Castelani (2005) identified the fitness function as an important element in such integrated approach. A particle with the best classification accuracy among all particles will be saved as $gbest$ particle. The parameter value in $gbest$ is later used by other particles to update its position. However, each particle also keeps track of its own best solution found during the learning. This value is called $pbest$ and is also an important attractor in the particle's next position update. The learning stops either when the system reaches the predefined maximum number of iterations or when one of the particles finds the desired classification accuracy. Algorithm 3 explains the proposed integrated ESNN-PSO and the detailed descriptions are presented in Appendix A.

Algorithm 3 Integrated ESNN-PSO

```
1: for all particle do
2:    initialise all ESNN parameters
3:    initialise fitness
4: end for
5: while not reaching maximum iteration do
6:    for all particle do
7:       get fitness from ESNN (Algorithm 1)
8:       if current fitness better than pbest fitness then
9:          assign current particle as pbest
10:         if current pbest fitness better than gbest fitness then
11:            assign pbest as gbest
12:         end if
13:      end if
14:      for all ESNN parameters do
15:         calculate velocity using Equation 3.1
16:         update parameter using Equation 3.2
17:      end for
18:   end for
19: end while
```

4.1.2 *Experimental Datasets*

Two synthetic datasets, the Uniform Hypercube dataset and Two Spirals problems have been used to evaluate the performance of the proposed framework.

Uniform Hypercube

The proposed integrated ESNN-PSO method was tested on a Uniform Hypercube dataset (Estévez, Tesmer, Perez, & Zurada, 2009). Thirty features were created where only 10 made up the relevant features vector $(r_1, r_2, ..., r_{10})$, whereby a sample belongs to class 1 when $r_i < \gamma^{i-1} * \alpha$ for i = 1 to 10. The chosen parameters were $\gamma = 0.8$ and $\alpha = 0.5$. The features that were not relevant to determine the output class consisted of 10 random features with the random value in interval [0,1], and 10 redundant features were copied from relevant features with an addition of noise calculated from Gaussian function. These redundant dimensions were generated by adding a Gaussian noise using standard deviation of $\sigma = |p| * s$ with $|p|$ being the absolute value of vector p while s is a parameter controlling the noise strength to the original data. s was set to 0.3 for this dataset. The features were arranged randomly to simulate a real world problem, and the relevant features were scattered in the dataset as presented in Table 4.1. The problem consisted of 500 samples that were equally distributed into two classes. Details of Hypercube data generation can be found in Estévez et al. (2009).

Table 4.1: Uniform hypercube feature arrangement

Features	**Arrangement**									
Relevant	02	04	09	10	11	15	19	20	26	30
Redundant	03	07	12	14	17	18	21	25	27	28
Random	01	05	06	08	13	16	22	23	24	29

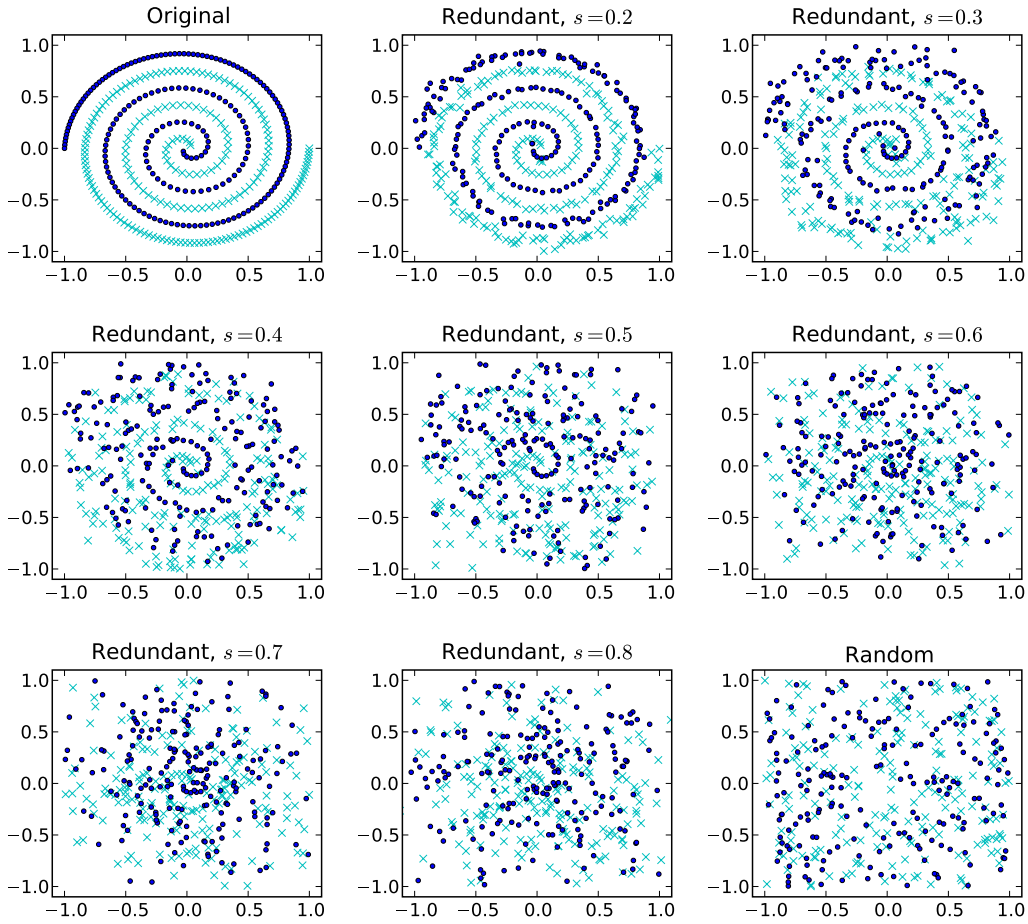

Figure 4.3: The Two Spirals data. Colours represent different class labels. Redundant data are copied from the original data with added noise while the random data is generated randomly. As noise was added, the data became more difficult to be distinguished between classes.

Two Spirals

The second synthetic dataset is the one from Two Spirals problem, well known to be a difficult non-linear separation problem first introduced in Lang and Witbrock (1988). In order to evaluate the performance in feature selection task, two relevant data were copied with some noise added to the original data. These redundant data were generated similarly to redundant data in Hypercube dataset by adding a Gaussian noise to the original spiral points $p = (x, y)^T$. The noise increased linearly according to the distance from the spiral origin

$(0,0)^T$. The noise value was calculated as the p_i centered Gaussian distributed random variable $N(p_i, \sigma^2)$. The information available in a feature decreased when a higher level of noise was applied as shown in Figure 4.3. In addition to this, several irrelevant features with random dimension values within the interval [0,1] were also added to the dataset. The dataset in this experiment consisted of 20 features with two relevant features, 14 redundant features with the noise level s from 0.2 to 0.8 and four random features. Detailed explanation of the data generation can be found in Schliebs, Defoin-Platel, and Kasabov (2009a). The features were then arranged in a random order to simulate a scenario where relevant features were scattered in the dataset, as shown in Table 4.2. Four hundred samples that were equally distributed between two classes were generated.

Table 4.2: Two Spirals feature arrangement

Position	**Features**
01	Redundant 0.3
02	Original
03	Redundant 0.7
04	Redundant 0.5
05	Random
06	Redundant 0.4
07	Redundant 0.4
08	Random
09	Redundant 0.2
10	Redundant 0.3
11	Random
12	Redundant 0.6
13	Redundant 0.2
14	Redundant 0.6
15	Redundant 0.7
16	Random
17	Redundant 0.8
18	Original
19	Redundant 0.8
20	Redundant 0.5

4.1.3 *Setup*

Receptive fields were used to produce a weight pattern or weight vector of a particular sample that could identify the output class. Different numbers of receptive fields for each dataset influenced the accuracy of the results. From preliminary experiments and as suggested by Schliebs, Defoin-Platel, Worner, and Kasabov (2009a), 20 receptive fields were chosen.

There is no definite answer on how many particles should be used to solve a certain problem. Generally, higher numbers of particles are required for more complex problems and lower number of particles for simple problems. Additionally, the balance between solution exploration (searching for good solutions) and exploitation (refining the solutions by combining information gathered during the exploration phase) must be taken into account (Shi & Eberhart, 1998b). In this experiment, 20 particles were used to explore the solution. Both PSO parameters c_1 and c_2 were set to 1.2, which meant a balanced exploration between $pbest$ and $gbest$ as well as the inertia weight w = 2.0. The integrated ESNN-PSO was tested on the Uniform Hypercube and Two Spirals datasets and the average result was computed in 1000 iterations using 10 fold cross validation.

These datasets were also used for obtaining test results from MLP and Support Vector Machine (SVM) with the same conditions for comparison purposes. Details of MLP with BP learning procedure is described in Appendix G and also can be found in Chauvin and Rumelhart (1995). The computation example is explained in Jones (2005). In this study. the NeuCom software is used to simulate the MLP for the given problems. The software has been described in Section 2.6. After preliminary experiments with tuning the parameters, the following values are found to be the optimal setting for this experiment: the learning rate was set to 0.3 and the momentum rate at 0.9. Number of hidden units for Hypercube data was set to 30. Because of the high level of noise injected into the Two Spirals problem, 40 hidden neurons were used. Both problems have been trained for 1000 iteration with 10-fold cross validation procedure.

The Hypercube and Two Spirals datasets were also applied to SVM using the NeuCom software. SVM is a statistical method and was originally designed

for linear separable problems. Nevertheless, the introduction of kernel function gives the method the capability of handling non-linear separable problems. Support vector is the data points that form a decision line for classification problem that helps classify the data in specific output classes. During learning, this method tries to formulate an equation to represent all data points with respect to their classes. The key features of SVMs are the use of kernels, the absence of local minima, the sparseness of the solution and the capacity control obtained by optimising the margin (Shawe-Taylor & Cristianini, 2004). Comprehensive description and review of SVM can be found in Shawe-Taylor and Cristianini (2000). Polynomial kernel function has been selected in both experiments. The only parameter in polynomial kernel is the kernel degree that was set to 1 after several attempts of fine-tuning.

4.1.4 *Performance Analysis*

In both experiments conducted on two separate datasets, the three ESNN parameters as explained in Section 2.7 evolve steadily until reaching certain optimal values. The combination of these values leads to better classification results. As shown in Figure 4.4 and Figure 4.5, for both datasets, the value of Mod is in the range of 0.9 and 1.0. The parameter is important because it represents the connection weight in the ESNN and should not be too low. If a low value was selected, it would end up with most connections assigned with the weight value of 0 due to the nature of weight computation as shown in Equation 2.25. In contrast, a higher value means most weights will have a connection value, which can be translated into well-presented weight patterns according to their output class. On the other hand, the proportion value C which controls the PSP threshold value is between 0.55 and 0.85. As for the evolving part, value of Sim is observed to be between 0.3 and 0.6. Higher value means there are many neurons within the similarity range that are merged while lower value means otherwise. The similarity is calculated using Euclidean distance. It can be concluded that based on the value with 20 receptive fields used, most of the neurons are within the similarity range and are merged.

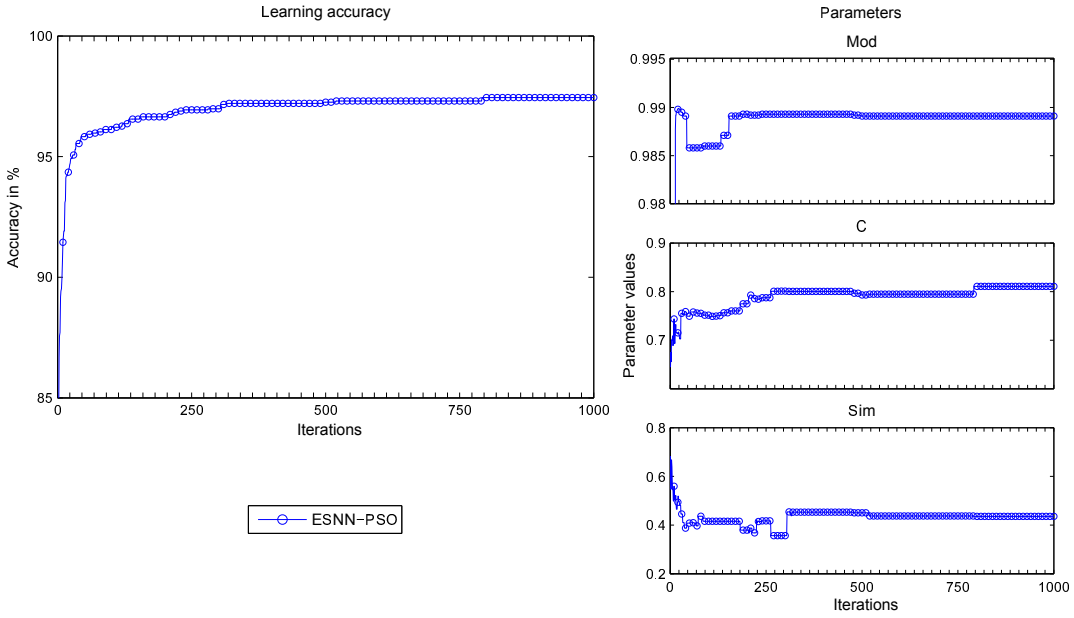

Figure 4.4: Evolution of accuracy and parameters on Hypercube dataset.

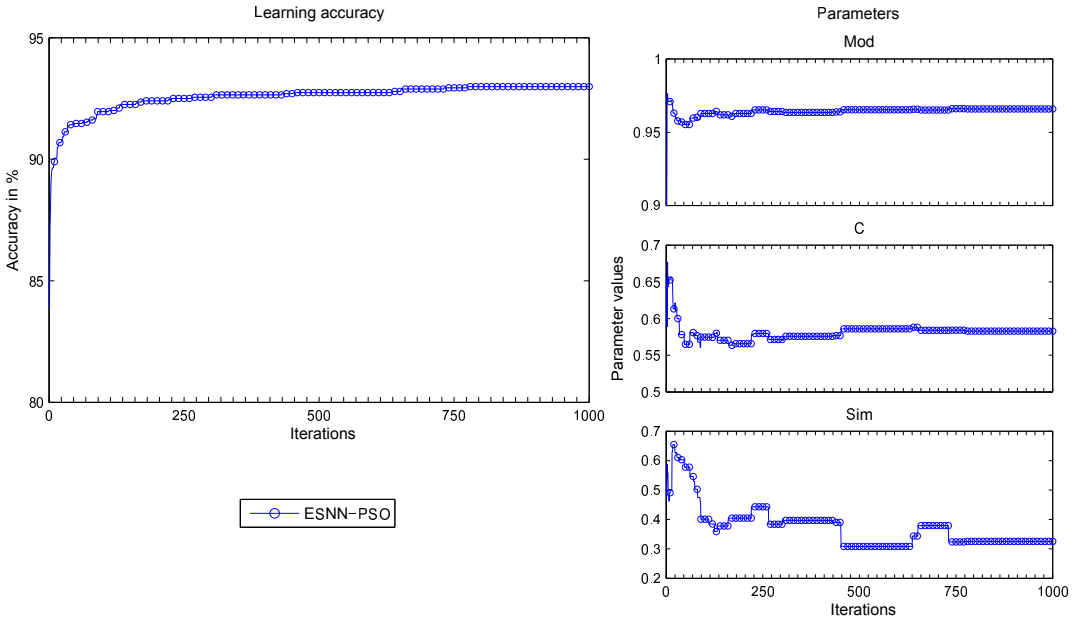

Figure 4.5: Evolution of accuracy and parameters on Two Spirals dataset.

The testing accuracy recorded from the proposed integrated method is 93.81% for the Hypercube dataset and 73.26% for the Two Spirals. MLP gives 89.40% and 52.40% respectively, while SVM achieves results comparable with those from the proposed method at 93.60% for the Hypercube and 62.00% for the Two Spirals. Table 4.3 shows the overall testing results. Both MLP and SVM

parameters also have been manually tuned to get the best setting in this experiment. Nevertheless, the result shows that ESNN with optimised parameters gives better classification accuracy than these two traditional methods.

Table 4.3: Comparison of classification accuracy

Method	Hypercube	Two Spirals
ESNN-PSO	93.81% $\pm$ 3.49	73.26% $\pm$ 7.33
MLP	89.40% $\pm$ 4.12	52.40% $\pm$ 7.11
SVM	93.60% $\pm$ 4.76	62.00% $\pm$ 7.12

Due to a higher number of irrelevant features in both datasets which may lead to misclassification of some samples and subsequently resulted lower classification accuracy, feature selection is necessary to select only few relevant features that are significant for the classification. A feature optimisation method utilising the principle of quantum computation in PSO is proposed in the next experiment.

Table 4.4: Comparison of computational time

Method	Time in Sec	Computational relative to ESNN
ESNN	5	1.0%
MLP	18	3.6%
SVM	2	0.4%

Table 4.4 shows the computational time required to run these three algorithms. The unoptimised ESNN is used for this comparison since two other algorithms are not optimised. All methods have been tested using the Hypercube dataset. In order to execute a single run, ESNN requires 5 seconds while MLP and SVM need 18 seconds and 2 seconds respectively. Therefore, MLP needs 3.6% more while SVM used 0.4% less computational time compared to ESNN. MLP requires more computational time compared to ESNN and

SVM because learning mechanism in MLP involves cycle of iteration. One-pass learning in ESNN framework involves only single feed forward learning which requires less computational time. SVM is a statistical method and does not require any repetition in learning which make the learning fast.

4.2 INTEGRATED ESNN-QIPSO FOR SIMULTANEOUS FEATURE SELECTION AND PARAMETER OPTIMISATION

The proposed framework of ESNN-PSO used in the previous experiment is able to obtain the best combination of ESNN parameters and better results compared to the traditional methods. However, due to the high number of irrelevant features in the datasets, results are believed to be much better if these features can be removed. In contrast, classification with only relevant features may lead to better outcomes. This section proposes a novel ESNN-QiPSO framework that integrates ESNN with QiPSO where features and parameters are optimised simultaneously into a single framework.

4.2.1 *Framework*

The proposed ESNN-QiPSO framework is very similar to the previous framework. However, the major difference is in the particle structure. Since there are two components to be optimised, each particle is divided into two parts. The first part of each particle holds the feature mask value for feature optimisation, while the other part holds a binary string for parameter optimisation.

The binary string mask is introduced in this experiment for the feature selection task. The mask determines which feature is going to be selected from the entire sample. Every feature is represented by a qubit in the mask. The probability of selecting features depends on the final value to which the qubit collapses. Collapse value 1 means that the feature is selected while value 0 means it is not selected. The selected features are then transformed into spike trains using Population encoding as described in the previous framework.

For parameter optimisation, a set of qubits represents the parameters values. Because the information held by particles is in binary representation, conver-

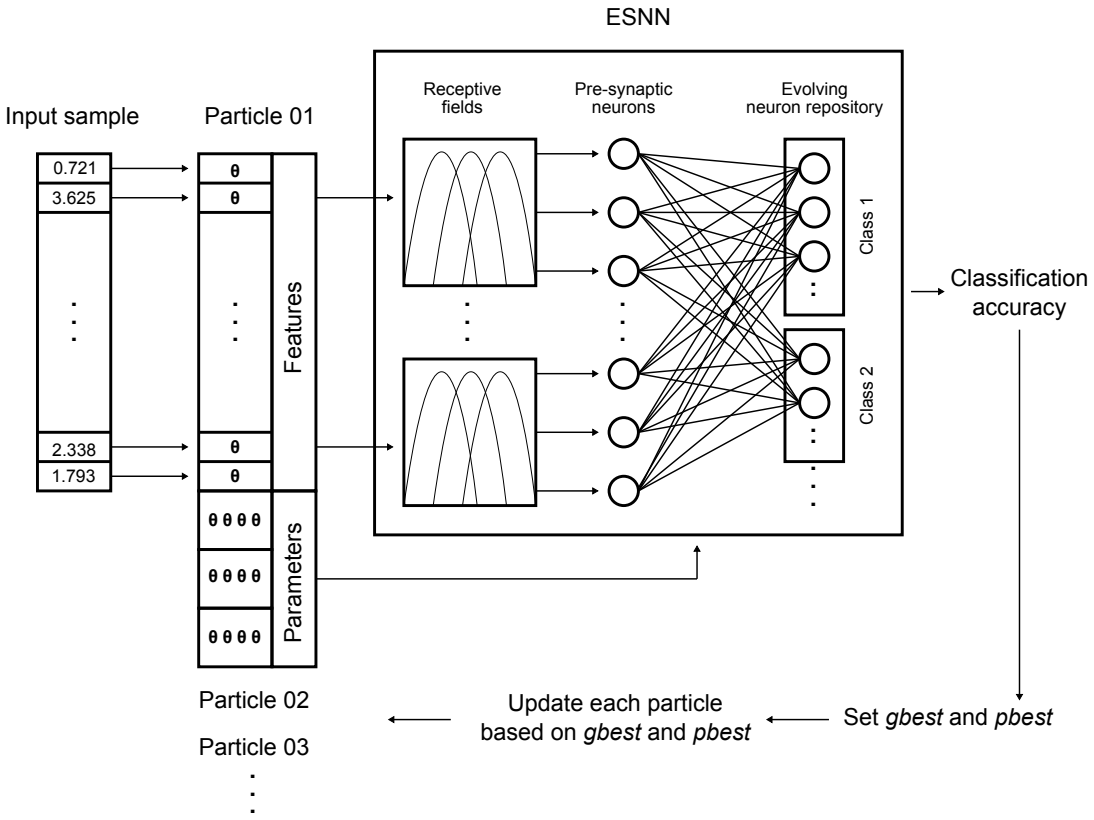

Figure 4.6: A Framework of ESNN-QiPSO. The real value input features are coupled with a qubit feature mask. Final qubit value when collapsed is 1 or 0 indicating a feature being selected or non-selected respectively. The selected features are then mapped into spike trains for learning. For parameter optimisation, the string of collapsed values are then translated into real values using the Gray code function.

sion into real value is required. For this task, the Gray code method is chosen since it is proven to be a simple and effective way to convert a binary representation into a real value (Gray, 1953). Following the procedure of one-pass learning, the connection weights in ESNN are trained according to the parameters obtained by the particles. All particles are initialised with a random set of binary values and subsequently interact with each other based on their fitness in classification accuracy. Figure 4.6 illustrates the proposed framework. Algorithm 4 describes the proposed integrated ESNN-QiPSO framework and the detailed implementations are described in Appendix B.

Algorithm 4 Integrated ESNN-QiPSO

for all particle **do**
  for all ESNN parameters **do**
    for all qubit **do**
      initialise θ
      get collape state using Equation 3.12
    end for
    convert binary string to real value using Gray code
  end for
  for all feature qubit **do**
    initialise θ
    get collape state using Equation 3.12
  end for
  initialise fitness
end for
while not reaching maximum iteration **do**
  for all particle **do**
    get fitness from ESNN (Algorithm 1)
    if current fitness better than $pbest$ fitness **then**
      assign current particle as $pbest$
      if current $pbest$ fitness better than $gbest$ fitness **then**
        assign $pbest$ as $gbest$
      end if
    end if
    for all ESNN parameters **do**
      for all qubit **do**
        calculate velocity using Equation 3.9
        apply rotation gate in Equation 3.11
        get collape state using Equation 3.12
      end for
      convert binary string to real value using Gray code
    end for
    for all feature qubit **do**
      calculate velocity using Equation 3.9
      apply rotation gate in Equation 3.11
      get collape state using Equation 3.12
    end for
  end for
end while

4.2.2 *Setup*

The QiPSO utilised the same searching method as the PSO and used a set of candidates in a population. Twenty particles were defined to find the solution. The number of features mask was set to the same number of input features. Since all three ESNN parameters ranged between 0 and 1, each parameter is represented by six qubits and were sufficient to represent the real value.

The proposed ESNN-QiPSO method was tested using the same datasets as before - the Uniform Hypercube and the Two Spirals. All other parameters were set to the same value as in previous experiments. Twenty receptive fields were chosen for data encoding, c_1 and c_2 were set to 0.05 for this binary optimisation and inertia weight w = 2.0. Particle dimension vector refers to the number of variables to be optimised by QiPSO. Therefore, the Hypercube problem dimension had three ESNN parameters and 30 features. On the other hand, there were 23 dimensions for the Two Spirals dataset.

A total of 1000 iterations were performed for learning with 10-fold cross validation was used. The datasets were applied to ESNN-QiPSO for comparison with the previous proposed methods.

4.2.3 *Performance Analysis*

All particles were initialised with a random set of binary values and they subsequently interacted with each other based on classification accuracy. The two main elements that contributed to classification accuracy were feature and parameter optimisation and they are discussed below.

Feature Selection

Figure 4.7 illustrates the average number of selected features during the learning process on the Hypercube dataset. The colours in the diagrams reflect how frequently the features were selected at a specific iteration. The brighter colour means the corresponding samples were selected more often, while the darker colour means otherwise. The bar chart simplifies the final number of samples being selected from 10 runs of 10 folds cross validations. For the Hypercube

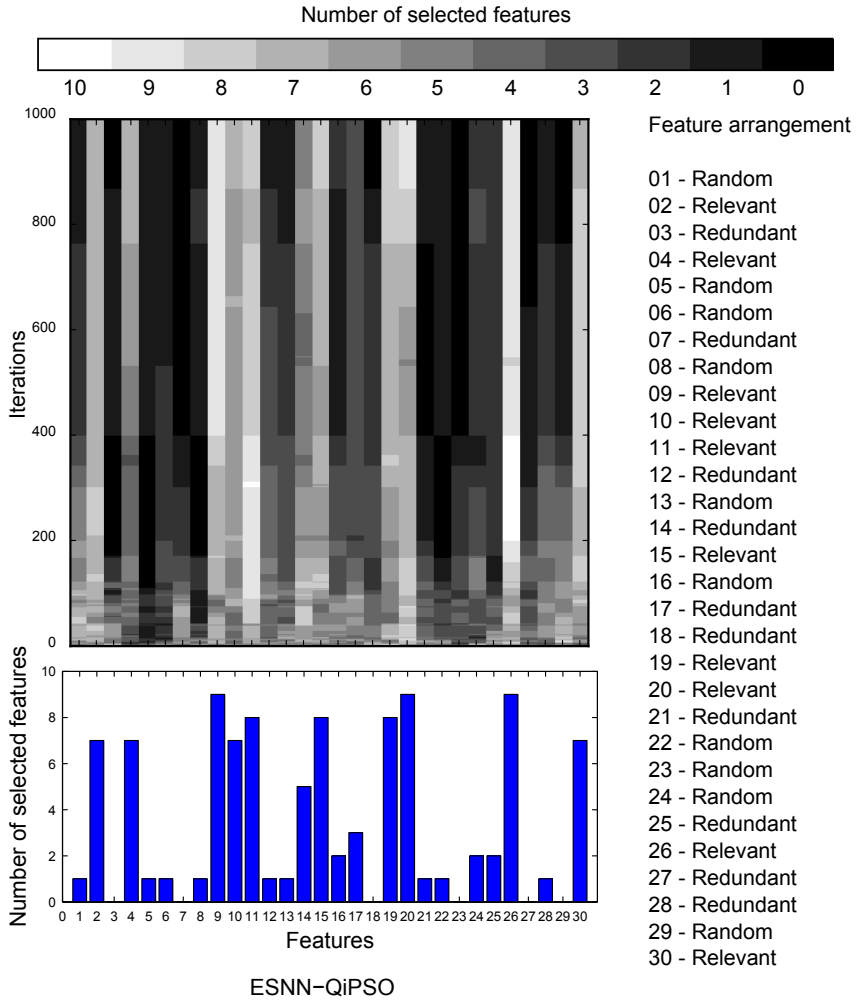

Figure 4.7: Feature selection evolution on the Hypercube dataset. Ten relevant features are selected as indicated by the bright colour representation. At the same time, the redundant and random features are gradually been removed by QiPSO during learning.

dataset, it is evident that in the early learning phase more features were used by the particles. This contributed to the low accuracy, as shown in previous experiments when many irrelevant features were used for classification. The irrelevant features that consisted of random and redundant features were consistently discarded during the learning process. The final result shows that all 10 relevant features were selected between seven and nine times from the 10 runs. Meanwhile the irrelevant features, especially the random ones that contained no information to distinguish between classes, were rarely selected in the final stage of learning.

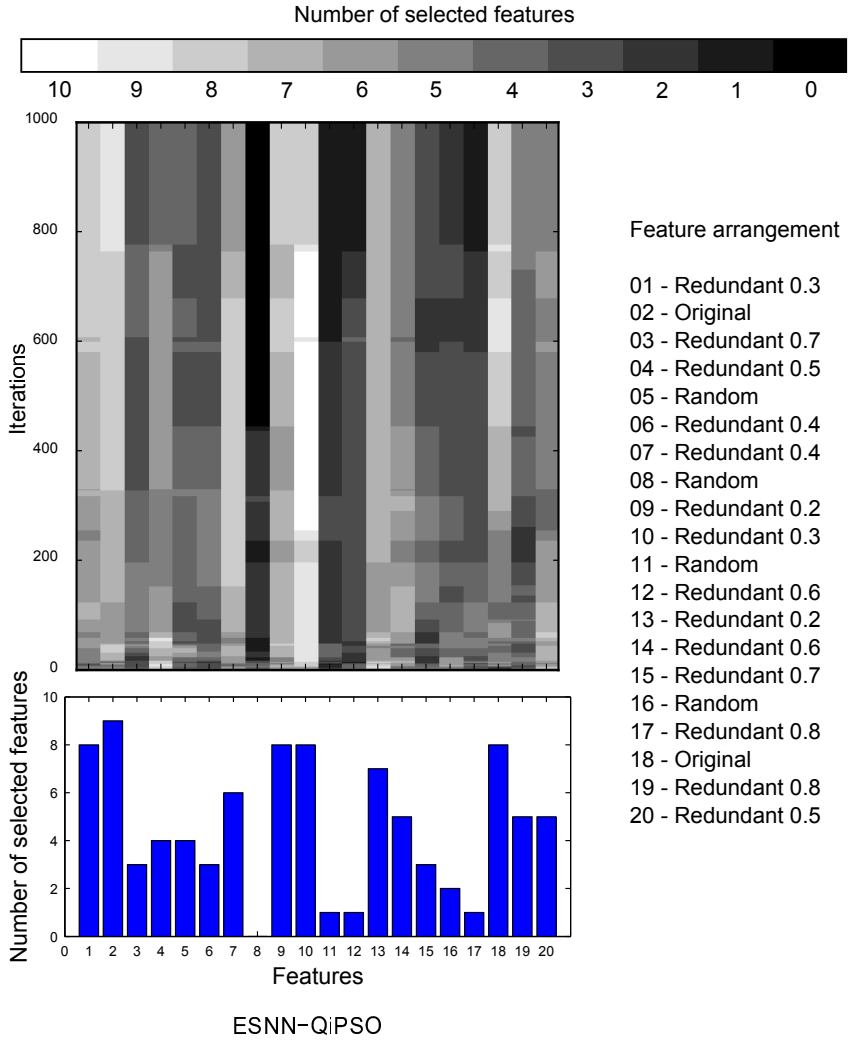

Figure 4.8: Feature selection on the Two Spirals dataset. Two relevant features were selected nine and eight times from 10 cross validation runs, while other features were removed by QiPSO during learning.

On the other hand, because of the different levels of noise that was injected into the Two Spirals dataset, QiPSO was not able to simply remove the irrelevant features. Samples with noise level of 0.2, 0.3 even 0.4 still contain information that can be used to differentiate between classes, as shown in Figure 4.3. At noise level of 0.5, the starting point of the spirals can still be distinguished and datasets with higher noise levels were considered as irrelevant and began to scatter. Figure 4.8 shows that while the two original features were selected in almost all experiment runs, the redundant features were also selected often. In contrast, the random features were steadily removed during learning. However, due to the reduced number of features and the final selected

features mostly containing information for the classification, the accuracy of the result shows some improvement in this experiment compared to a previous experiment that was using all features, as depicted in Figure 4.10.

Parameter Optimisation

The correlation between feature selection and classification accuracy are clearly observed in the previous section. However, it is also clear that the result is not only affected by feature selection, but also by parameter optimisation. Figure 4.9 and Figure 4.10 show the parameter optimisation by QiPSO for the Hypercube and Two Spirals datasets. The results show that all parameters are optimised into the same range, as observed in the previous experiment. All parameters are dependent on each other for optimal solution. Often, changes in a certain parameter require changes in other parameters as well.

The modulation factor Mod is optimised in the range of 0.9 to 1.0 by QiPSO. The value evolves steadily to a certain optimal value, especially in Hypercube data, where the optimal value after the learning is almost similar. When examining value of firing threshold C, it starts almost at the same value at the beginning and then advances towards an optimal value. Finally, for the similarity threshold Sim, the value is significantly different compared to the value optimised by PSO. This is because the connection similarity is low when a high number of connections exists. In ESNN-PSO experiments, all features are used. In contrast, when the feature selection process reduces the number of features, number of pre-synaptic neurons and connections decreases as well. This improves the chance that more connections are in the similarity range and subsequently improves the results.

The average accuracy for ESNN-QiPSO with simultaneous feature selection and parameter optimisation is 94.74% for Hypercube and 84.09% for Two Spirals. This clearly shows that results can be improved not only by optimising the parameters. Better results have also been achieved by reducing the problem noise by eliminating the irrelevant features and selecting features that contain the most information for classification. The accuracy during early learning is relatively low for ESNN-QiPSO. The reason is that the random selection approach by particles selects features that contain little information to enable

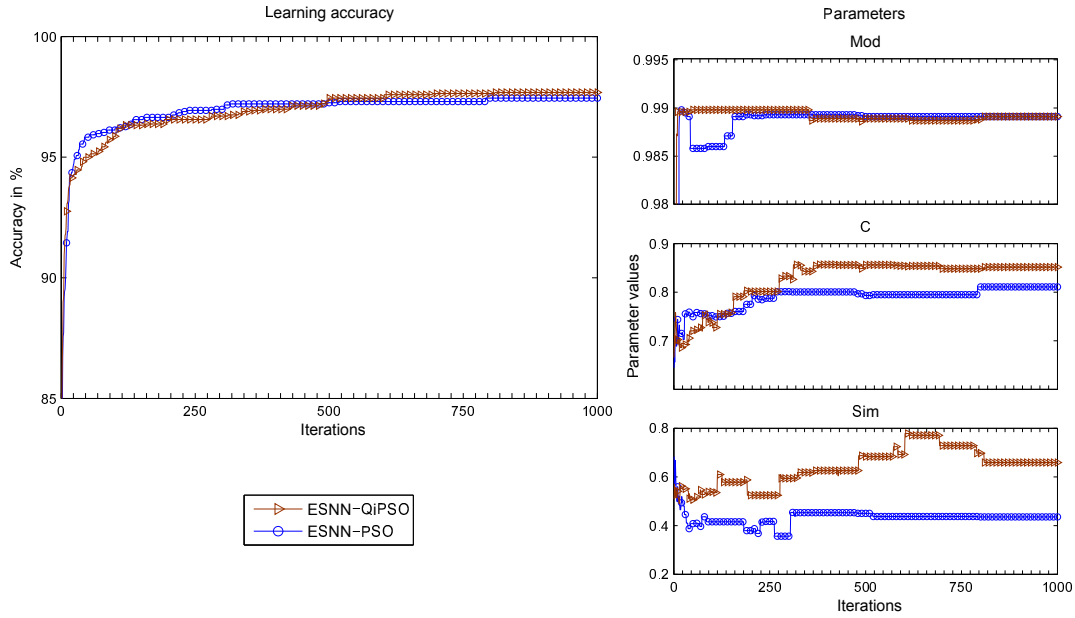

Figure 4.9: Evolution of accuracy and parameters on Hypercube dataset from the integrated ESNN-QiPSO framework.

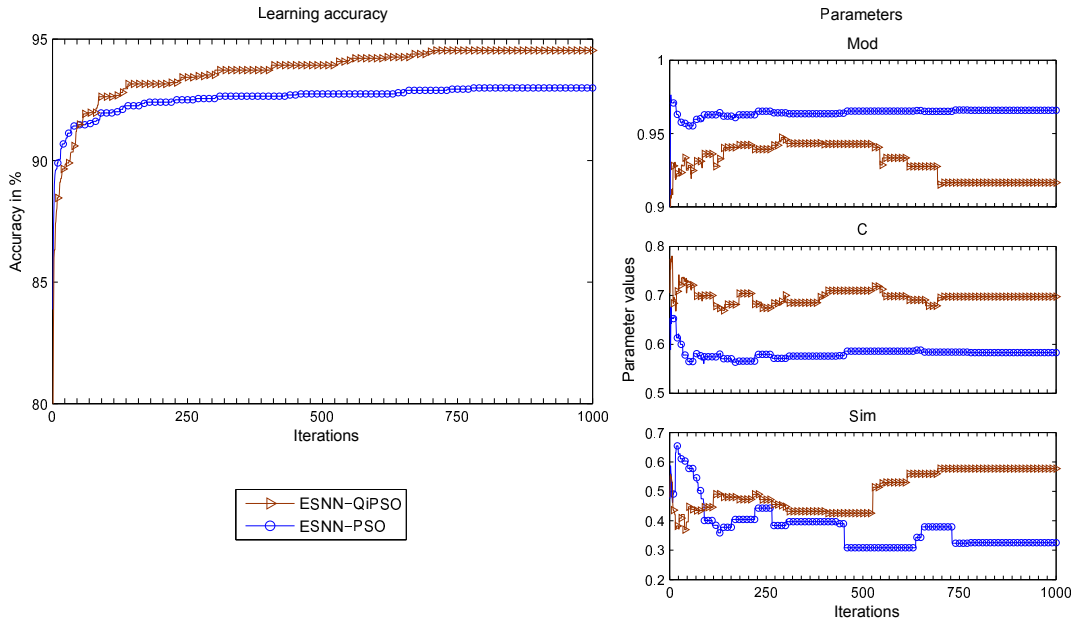

Figure 4.10: Evolution of accuracy and parameters on Two Spirals dataset from the integrated ESNN-QiPSO framework.

better classification. Gradually, learning is improved and the particles start to find better features and parameter combinations. In addition to parameter optimisation, this experiment also demonstrates that better classification accuracy

can be achieved by selecting relevant features and eliminating the irrelevant ones. The overall results are shown in Table 4.5 for easy comparison.

Table 4.5: Comparison of classification accuracy

Method	Hypercube	Two Spirals
ESNN-QiPSO	94.74% $\pm$ 4.34	84.09% $\pm$ 6.43
ESNN-PSO	93.81% $\pm$ 3.49	73.26% $\pm$ 7.33
MLP	89.40% $\pm$ 4.12	52.40% $\pm$ 7.11
SVM	93.60% $\pm$ 4.76	62.00% $\pm$ 7.12

Knowledge Discovery

Machine learning techniques involve the development of algorithms and computer programs that would enable the machine to learn and produce solutions to the given problems. Analysis of the results obtained from the learning process is performed in order to understand the problems.

In feature selection task, the features selected in this case by QiPSO are normally considered as the relevant features. From this outcome, feature selection expresses some knowledge on which features that are important in decision making process. This is important for understanding the problems and especially for achieving a good classification result. Some high-dimensional real world problems consist of a large number of variables. Correlation between variables can also be identified based on the final selected features. Running the experiment 10 times shows that some features will be selected only when certain other features are selected as well. For instance, as shown in Figure 4.7, whenever Feature 9 is selected, almost certainly Feature 20 and Feature 26 are also selected. Therefore, it can be concluded that these three features are relevant and also interrelated. It also shows that the selected features are important in order to get higher classification accuracy.

Knowledge discovery from parameter optimisation can be found in the range of optimal value. The higher Mod value represents early connections have higher weight and more importance in deciding the output class of a sam-

ple. On the other hand, the smaller value means early connections have lower weight. Therefore, more spikes are required to decide the output class. For the threshold C, it has been discovered that if the value is higher, more spikes and time are required to make a decision. However, if smaller value is selected, few spikes are enough to fire an output spike and determine the class of the sample. Merge rate of output neurons is controlled by the parameter Sim. A higher Sim value means more output neurons are merged. This leads to the creation of a robust network architecture, where a small number of output neurons represent entire samples in the dataset. From both experiments, the optimal range of Mod is between 0.9 and 1.0. On the other hand, C and Sim range from 0.5 to 0.9 and 0.3 to 0.7, respectively. It can be concluded that each ESNN parameter converges towards a certain range of values. However, there is no specific value range that can be applied to all problems where each problem will have its own combination of parameters.

In general, every dataset requires specific analysis in order to understand the problems it represents. Comprehensive understanding of the problem may improve efficiency in decision making and also make future decision making easier and faster. Knowledge can be extracted from many sources such as time prediction result, visualisation, classification, clustering and others. Many studies have examined data mining and knowledge discovery subject (Fayyad, Piatetsky-Shapiro, Smyth, & Uthurusamy, 1996; Bramer, 1999; Cios, Pedrycz, Swiniarski, & Kurgan, 2007).

4.3 IMPLEMENTATION ISSUES

4.3.1 *Qubit Representation of Parameters*

Some problems persisted when using QiPSO algorithms for parameter optimisation and feature selection The problems include the possibility of missing the optimal parameter value when using only binary QiPSO. As the information is represented in a binary structure, the conversion from binary to real value can cause such problems, especially if the selected number of qubits representing the parameter value is insufficient.

Table 4.6: Example of conversion from qubit to real value

Qubit	Real value
[00]	0.0
[01]	0.3
[10]	0.7
[11]	1.0

Table 4.6 shows an example where two qubits represent parameter values within the interval [0,1]. In this example, if the optimal value is 0.8, there will be no chance for the optimiser to obtain this value. An easy solution is to increase the number of qubits to cover more real values. However, this will lead to longer computation time.

4.3.2 *Feature Selection*

In addition, QiPSO's search strategy is based on random selection of features at the beginning of the process. Each particle will update itself based on the best solution subsequently found. A major problem with this approach is that the randomly selected features at the beginning may not be the relevant. Other particles that take part in the entire process are thus affected. This is due to each particle updating its information without relevant features as illustrated in Figure 4.11.

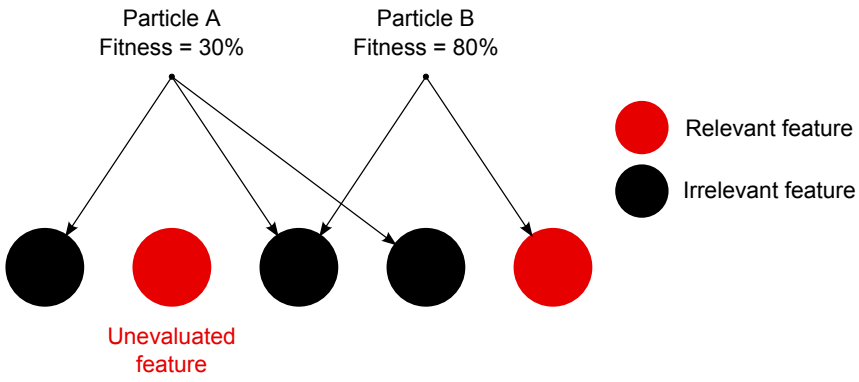

Figure 4.11: Example of feature evaluation.

The best solution is to explore all possibilities, but it is unrealistic for high dimensional problems. Figure 4.12 shows the search tree where all possibilities are taken into account. These 2^4 problems resulted in 16 possibilities. For a problem with 20 features, the search tree have 1,048,576 possibilities, a number that would make the computation far too slow.

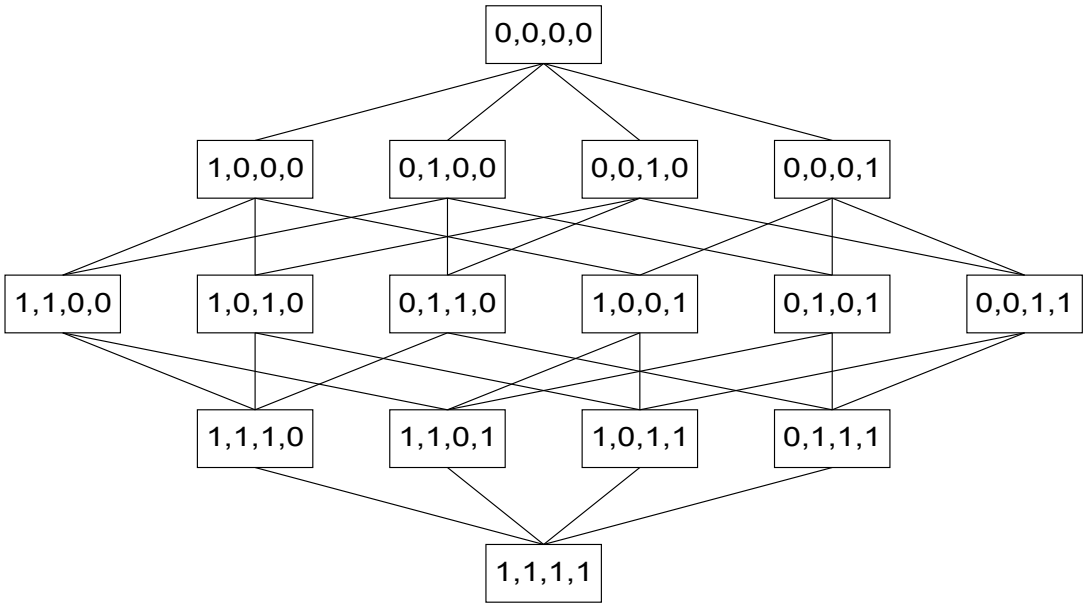

Figure 4.12: Search tree for four qubits with 16 possible combinations.

4.3.3 *Number of Connections*

The study of the feature selection process shows that the network performs well when the number of features is reduced, which also reduces the number of connections. In the experiment, fewer connections result in faster learning process with better accuracy. This suggests that reducing the number of connections can be used to enhance the learning.

Motivated by this finding, a new enhancement to the optimiser and ESNN will be introduced in the next two chapters. A new search strategy will be introduced for the optimiser and this enhancement is discussed in Chapter 5. The new evolving connection structure will be proposed in Chapter 6 as it

appears that not all connections are necessary. This finding also aligns with the probabilistic neuron model as discussed in Chapter 2.

4.4 EXTENDED ESNN-QIPSO FRAMEWORK FOR STRING PATTERN CLASSIFICATION

In a collaboration between Knowledge Engineering and Discovery Research Institute (KEDRI) where this research has been studied and the National Institute of Information and Communications Technology (NiCT, Japan), the proposed ESNN-QiPSO method has been tested on a string pattern recognition problem. String pattern recognition is an approach for determining which group of a string belongs to, based on analysing its contents. This task, despite being quite challenging, is very important to certain areas such as internet security and virus detection. Strings can be texts, musical symbols or others that are not necessarily in numerical formats. Since most classifier algorithms can only accept numerical values, transformation from strings to numerical values is required. String kernels are a well-known method for transforming string input values into high dimensional input vectors (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002). The kernel provides the classifier algorithm with the capability of mapping the original non-linear separable data into a higher-dimensional space which is linearly separable. There are several well-known string kernels such as Bag of Words and n-grams. The output from the kernel process is the kernel matrix which is used as input to the algorithm for classification, clustering or ranking tasks. This technique is quite simple yet effective in transforming the input from string format into numerical values.

4.4.1 *Framework*

This experiment has extended the ESNN-QiPSO framework for string pattern recognition with the addition of a string kernel. The aim is to investigate the efficiency of QiPSO when it is used for optimising ESNN parameters as well as for selecting the most relevant features from the string kernel matrix, both of which have direct influence on classification accuracy. In order to allow the

proposed method to operate on string data, n-grams string kernels were used to transform the input data into the desired input format. The n-grams approach has been chosen where n-grams are n adjacent characters (substring) from an alphabet A (Lodhi et al., 2002). For example, if n=3 for the string "KEDRI", the trigrams output will be KED, EDR, DRI. Based on this approach, the similarity between strings is calculated and the result is the kernel matrix. This process is illustrated in Figure 4.14. Figure 4.13 despicts the proposed framework.

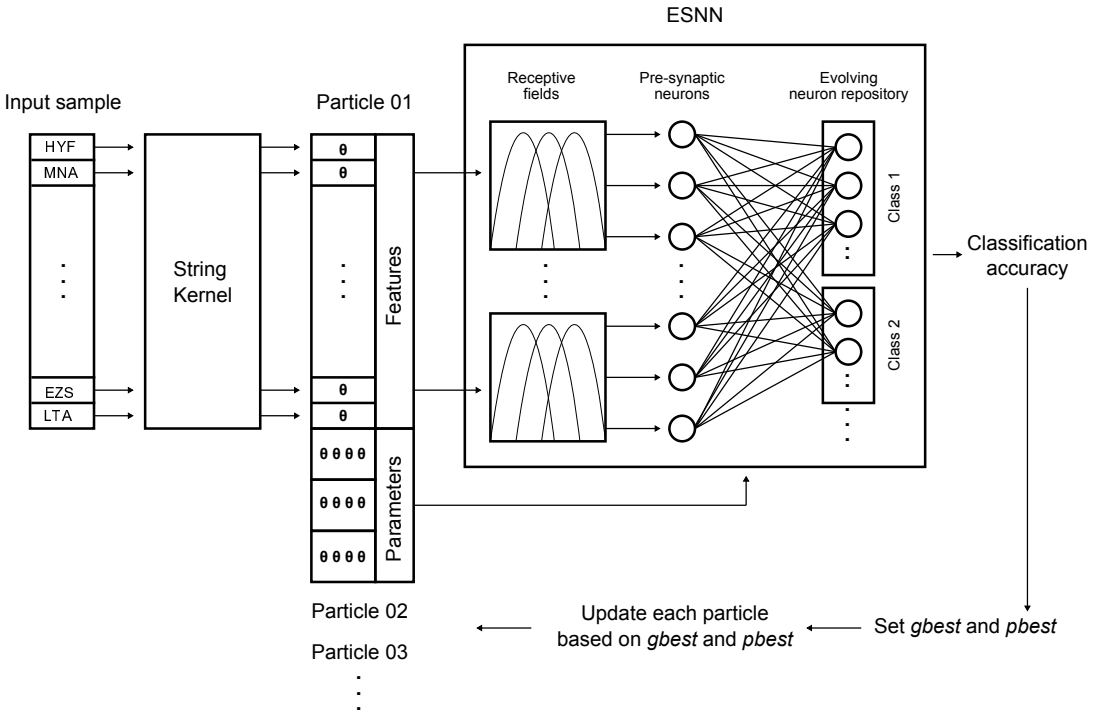

Figure 4.13: An extended ESNN-QiPSO framework for string classification.

The labeled and reformatted Reuters 21578 string dataset[1] is selected for this experiment. Only relevant information from tags topic, title and body text were extracted and some unknown tags such as "&", "$" were removed. Finally, all characters were changed to lowercase. The problem consisted of 150 samples from four classes: 38 from acquisition, 36 from corn, 38 from crude and 38 from earn. Parameters chosen were n = 4 and λ was set at the optimised

1 Reuters-21578 Text Categorization Collection Data Set, UCI Machine Learning Repository. http://www.ics.uci.edu/ mlearn/MLRepository.html

		String 1	String 2	String 3
Class 1	String 1	A	B	C
Class 1	String 2	B	A	C
Class 2	String 3	C	C	A

Figure 4.14: Kernel matrix from three strings. String 1 and 2 are class 1 and String 3 is class 2. Comparison between same strings will give the highest similarity value of A. The similarity calculation between the same classes will give a value B that is slightly lower than A, while the similarity between different classes is C, which is the lowest value. Based on this similarity values, a feature pattern between input samples can be produced.

value of 0.5 in n-gram kernels. $\lambda \in [0, 1]$ is the weight to penalising the distance between substrings. If the appearances of substrings are more coherent they receive a higher weighting than appearances with larger gaps (Saunders, Tschach, & Taylor, 2002). Twenty particles have been used to explore the solution with six qubits to represent the real value. Parameter c_1 and c_2 were set to 0.05 and the inertia weight w = 2.0. The dataset was applied to two frameworks - ESNN with feature and parameter optimisation and ESNN with only parameter optimisation. To limit the computation complexity, 10 receptive fields were chosen, the experiment was run for six continuous times and the average result was computed in 300 iterations for both algorithms.

4.4.2 *Results*

Figure 4.15 shows the evolution of classification accuracy. The average accuracy for ESNN with feature optimisation is above 70%, compared to ESNN using all features with average accuracy of 55%. The poor accuracy of ESNN with all features is due to several input features from the kernel matrix containing information that cannot be used to differentiate output classes. These irrelevant features act as a noise that leads to low classification accuracy. However, ESNN with feature optimisation is able to reduce these irrelevant features; hence higher classification accuracy is obtained. From the total number of 150 input features, QiPSO is capable of reducing the features to 70 in 300 iterations. These 70 features are the significant features because of their capa-

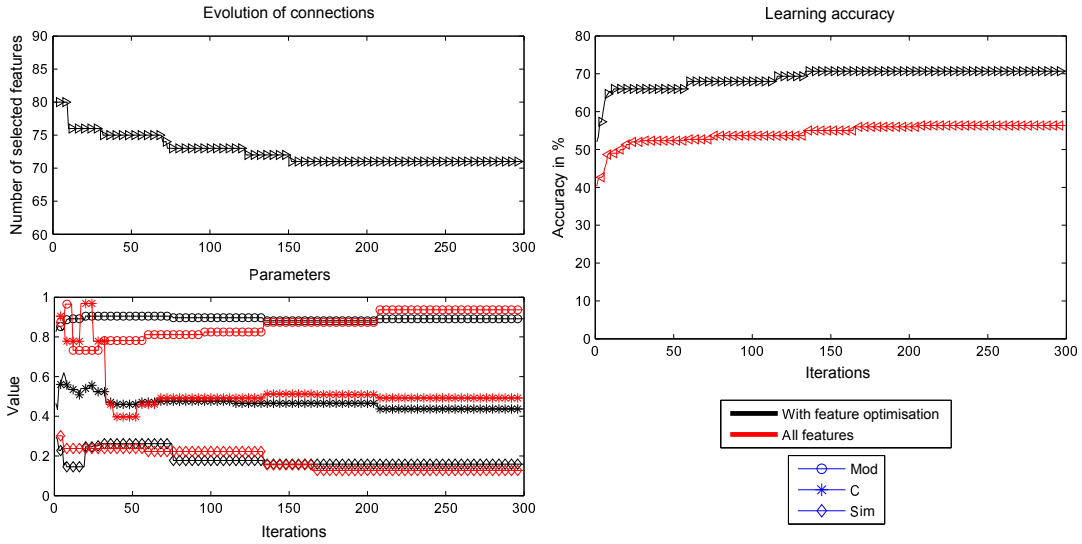

Figure 4.15: Result of string classification using ESNN-QiPSO with feature optimisation and ESNN-QiPSO with all features. The top left graph shows the evolution of connections until 70 features subset was found. Three ESNN parameters for both experiments fall into certain ranges. The ranges found are aligned to what has been discovered in previous experiment. The right graph shows the evolution of accuracy.

bility to produce higher accuracy. Since the 20 particles started the evaluation process by picking random features, the average number of initial features selected by the $gbest$ particle is around 80. The $gbest$ particle always keeps the best information and the best accuracy. Other particles updated their positions according to $gbest$ as well as $pbest$ until the new best particle was met. This procedure was repeated to eliminate irrelevant features for better identification of the most relevant features. A similar procedure was also used to find the best combination of ESNN parameters. In this study, QiPSO manages to optimise ESNN parameters in binary string format to a certain optimal values. These parameters are in pairs and are very closely related to each other. Overall, all three ESNN parameters evolve steadily towards a certain optimal value, which helps to produce better classification accuracy. This investigation continued with an experiment that fed the string kernel matrix into MLP. After manual parameters tuning, the optimal value was found and set at 0.1 for the learning rate and the momentum rate was set to 0.9 with 120 hidden neurons. The training classification accuracy is 55.4% in 300 iterations with testing result of 50.6%.

In this experiment, the data was transformed into another high-dimensional form of representation. For such difficult problems, some improvements of the framework can be investigated in future to enhance its learning capability. In addition, the string kernel can also be improved by discovering the best setting of the kernels.

4.5 SUMMARY

This chapter has proposed two novel frameworks, the integrated ESNN-PSO and ESNN-QiPSO. The integration between ESNN and PSO allows the optimiser to find the optimal parameter for ESNN automatically rather than having to adjust the parameters manually. In the second extended framework, the QiPSO can optimise both parameters and features simultaneously. The proposed method for feature selection is more effective compared to the preprocessing feature selection technique. The reason for that is because the parameter can be directly optimised and tuned based on selected features. Results show that both optimisers, the PSO and QiPSO are able to optimise ESNN parameters with QiPSO can also select the relevant features. These capabilities of QiPSO contribute to its higher classification accuracy.

This chapter also discusses the extention of the proposed ESNN-QiPSO for string classification. The nature of string classification involves an additional component (or kernel) to translate the problem into a high-dimensional problems which poses another challenge. However, the structure of ESNN can easily adapt to the high-dimensional problem and also the one-pass learning allows fast learning of the given problem. In addition to this, the feature selection in the proposed framework allows the problem dimension to be reduced, thus faster processing and better accuracy can be achieved. Overall, the results show that ESNN with parameter optimisation and using a small number of features produces promising results that can be explored in future studies.

The evaluation of the proposed integrated framework revealed several issues with the optimiser and the classifier. Therefore, some enhancement to both optimiser and the classifier are proposed in the next two chapters. The modification will be integrated with the main framework proposed in this study.

The same dataset will be used to evaluate the performance with comparison to results found in this chapter.

Chapter 5

A NOVEL DYNAMIC QUANTUM INSPIRED PARTICLE SWARM OPTIMISATION METHOD

In order to address the problems identified in Chapter 4, this chapter introduces a new structure for QiPSO. First, a hybrid particle structure is proposed for solving the problem of missing parameter value. Then, a state of the art new evaluation strategy is employed for feature selection. The improved search strategy selects the most relevant features and eliminates the irrelevant ones. This new QiPSO structure will be integrated within ESNN where features and parameters will be optimised simultaneously and more efficiently.

5.1 THE DYNAMIC QUANTUM INSPIRED PARTICLE SWARM OPTIMISATION

The proposed enhancement consists of two parts. The first part is to address the parameter problems caused by selecting an insufficient number of qubits. The conversion from qubit to the real value may lead to the possibility of missing optimal value as described in Chapter 4. At the same time, feature selection depends on qubit representation for the selection computation. Since two types of representation are required, hybridisation of information representations may solve these problems. Hence, the efficiency of the optimiser and the whole proposed framework can be improved.

5.1.1 *Hybrid Particle*

The features are modeled as a qubit vector, where probability computation is added to perform the feature selection task. Updating this value guides the optimiser to decide whether a feature is going to be selected or should be considered irrelevant. On the other hand, parameter values are presented as real numbers. As discussed earlier, the major problem using only QiPSO for parameter optimisation is that it may miss the optimal value. To overcome this problem, a simple yet effective hybrid particle structure with the combination of QiPSO and conventional PSO is proposed for these two different data types. The hybrid DQiPSO particle is divided into two parts: the first part uses quantum probability computation for feature selection and the second part holds the real value for parameters as shown in Figure 5.1. This method not only effectively solves this problem, but also eliminates the parameter that holds number of qubits representing the parameter value. Fewer parameters lead to an algorithm that is simple to use and control.

5.1.2 *Enhancement of Feature Selection Strategy*

The search strategy of QiPSO is based on random selection at the beginning of the process. Each particle will update itself based on the best solution subsequently found. A major problem with this approach is the possibility of not selecting the relevant features at the beginning of learning; other particles taking part in the entire process are thus affected. An improved search strategy has been introduced to find the most relevant features and eliminate irrelevant features. Blum and Langley (Blum & Langley, 1997) have classified the feature selection techniques into three basic approaches: Embedded approach (Almuallim & Dietterich, 1991) adds or removes features in response to prediction errors on new instances; Filter approach (Liu & Setiono, 1996) is a preprocessing method; and the Wrapper approach (John et al., 1994) which uses a learning algorithm to evaluate features. In this study, the Wrapper approaches is used with embedding of some concepts from the Filter and Embedded approach to utilise the advantages of these approaches. Generally, using a higher

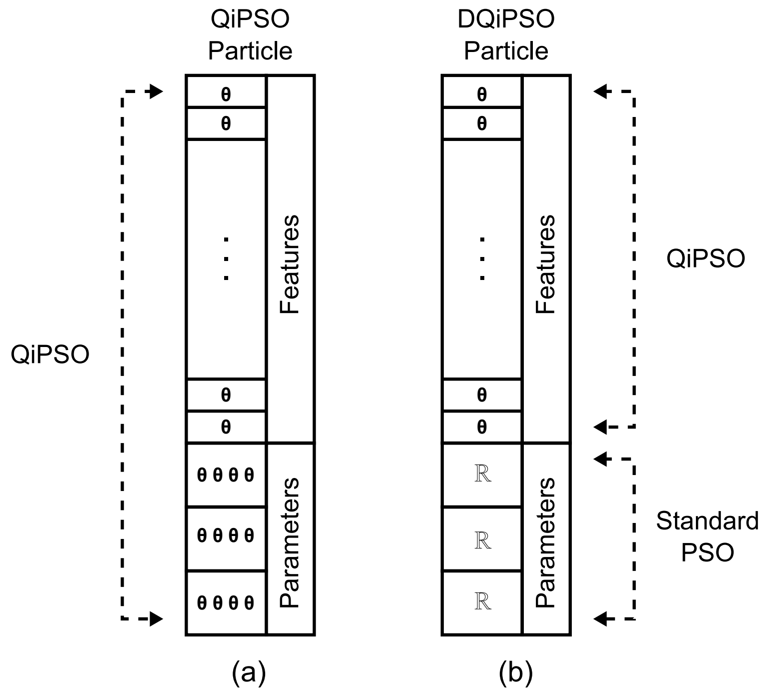

Figure 5.1: The proposed hybrid particle structure in DQiPSO. Compared with the QiPSO particle structure used in Chapter 4 (a), a particle in the proposed method (b) is divided into two parts where the first part holds quantum information and the second part holds real value information that is represented by the $\mathbb{R}$ symbol.

number of features does not necessarily translate into better classification accuracy. However, better accuracy can be acheived when a higher number of relevant features are selected.

A new strategy is proposed where five types of particles are used in the DQiPSO. Apart from the normal particle, referred to as the Update Particle, which renews itself based on and $gbest$ and $pbest$ information, four new types of particles are added to the swarm. The first type is the Random Particle, which randomly generate new sets of features and parameters in every iteration to increase the robustness of the search. The second type is the Filter Particle, which selects one feature at a time and feeds it to the network and calculates the fitness value. This process is repeated for each feature. Then, the average fitness is calculated. Any features with above average fitness will be considered as relevant. This method is targeted at linear separation problems. The third particle type is the Embed In Particle in which input features are added to the network one by one. If a newly added feature improves the

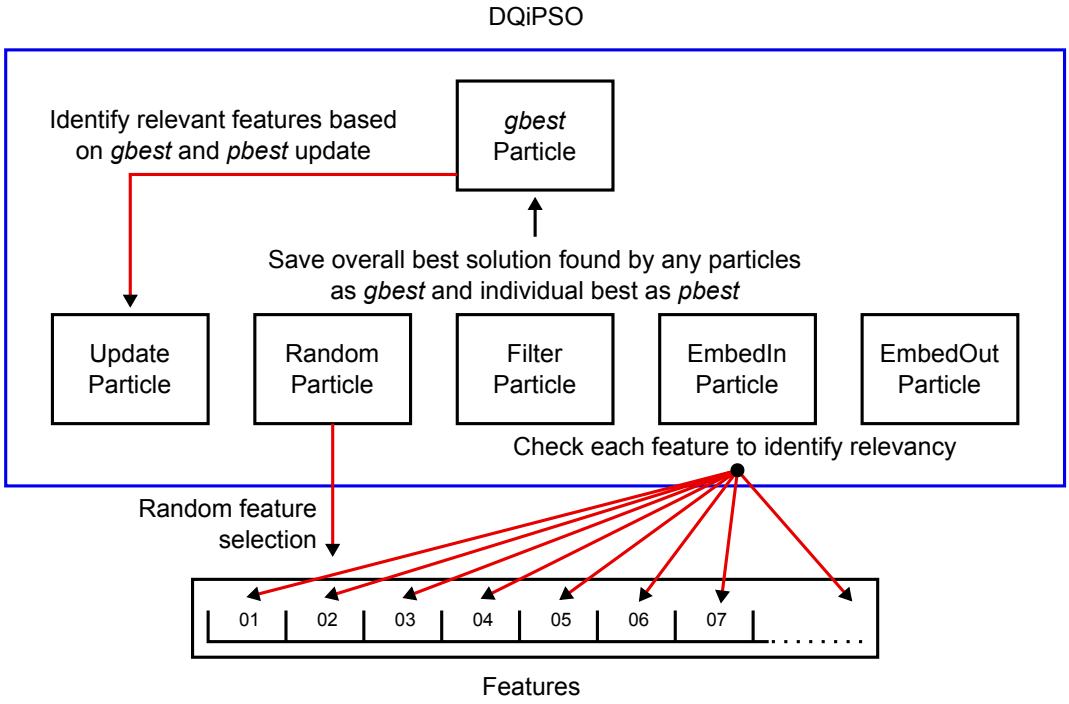

Figure 5.2: DQiPSO feature selection strategy. Update Particle next value depends on values of the $gbest$ and $pbest$; Random Particle selects features randomly while the other three particle types evaluate each feature to determine feature relavancy.

particle fitness, it will be considered as a relevant feature. Otherwise, the feature will be removed. The final particle type is the Embed Out Particle which starts the identification process with all features fed to the network to get the initial fitness value. These features are gradually removed one by one. If removing a feature causes decrement of the fitness value, then this feature will be considered as relevant and hence will be kept. Otherwise, the feature will be considered as irrelevant and removed.

The main idea behind Filter, Embed In and Embed Out particles is to identify the relevance of each feature and to reduce the number of candidates until only a small subset remains. For subsequent iterations, features considered relevant will be selected randomly to find the best combination of relevant features. This strategy helps to solve unevaluated relevant features, while reducing the search space and facilitating the optimiser to find relevant features faster. Similar to the standard PSO in updating particles, if a new particle is found to be the best solution, then it will be stored as a $gbest$. In this scenario, a new $gbest$ update rule has been implemented. A solution candidate will be assigned as

$gbest$ if it meets one of the following two criteria. The first criterion is a particle will be assigned as $gbest$ if its accuracy better than the current $gbest$ particle. The second rule for appointing a new $gbest$ is when both the candidate and the existing $gbest$ particle have the same fitness but the candidate has fewer features selected. This criteria guarantees that at all iteration cycles, $gbest$ always has the best fitness with the lowest number of features. The smallest feature subset will determine the most relevant features in the given problem. Due to the enhanced search strategies provided by DQiPSO, fewer particles are needed to perform the optimisation tasks. Hence, shorter processing time can be achieved. The summary of this strategy is illustrated in Figure 5.2.

5.2 INTEGRATING DQIPSO WITH ESNN FOR SIMULTANEOUS FEATURE AND PARAMETER OPTIMISATION

The proposed DQiPSO is able to simultaneously select relevant features and optimise the ESNN parameters. In this integrated environment, the features of the model are represented probabilistically as a qubit vector. On the other hand, parameter values are represented as real numbers. The principle of quantum superposition is used to accelerate the search for an optimal set of features. In contrast, the real value part of the particle will be updated using standard PSO update procedure.

Like all earlier frameworks, for a given classification task a swarm of particles is used to find the classification model with the best accuracy. The simulation starts with random values assigned to each particle in the swarm. Information held by the particle will be used to train the network. Then, the particles interact with each other and exchange the optimal information which ensures a faster optimisation process. The proposed integrated framework is shown in Figure 5.3. Algorithm 5 explains the integration and the pseudo code of the proposed ESNN-DQiPSO is presented in Appendix C.

The proposed ESNN-DQiPSO method was tested on the Hypercube and Two Spirals datasets, which are the same datasets employed in previous experiments. Most of the parameters are derived from the previous experiments. For the encoding, 20 receptive fields were chosen with their centers uniformly

Algorithm 5 Integrated ESNN-DQiPSO

```
1: for all particle do
2:    initialise all ESNN parameters
3:    for all feature qubit do
4:       initialise θ
5:       get collape state using Equation 3.12
6:    end for
7:    initialise fitness
8: end for
9: while not reaching maximum iteration do
10:   for all particle do
11:      get fitness from ESNN (Algorithm 1)
12:      if (current fitness better than pbest fitness) or ((current fitness == pbest
         fitness) and (feature selected less than feature selected by pbest)) then
13:         assign current particle as pbest
14:         if (current pbest fitness better than gbest fitness) or ((current pbest
            fitness == gbest fitness) and (feature selected by pbest less than fea-
            ture selected by gbest)) then
15:            assign pbest as gbest
16:         end if
17:      end if
18:      for all ESNN parameters do
19:         calculate velocity using Equation 3.1
20:         update parameter using Equation 3.2
21:      end for
22:      for all feature qubit do
23:         calculate velocity using Equation 3.9
24:         apply rotation gate in Equation 3.11
25:         get collape state using Equation 3.12
26:      end for
27:   end for
28: end while
```

distributed between the maximum and minimum values of the data. The controlling parameter β is set to 1.5. Eighteen DQiPSO particles were used, consisting of six Update, three Filter, three Random, three Embed In and three Embed Out Particles. The inertia weight was set to 2.0. θ_c_1 and θ_c_2 for probability update were set to 0.05 and c_1 and c_2 were set to 1.2 for real value update for balance exploration of $gbest$ and $pbest$. Ten-fold cross validations were used and the average result was computed based on 1000 iterations to

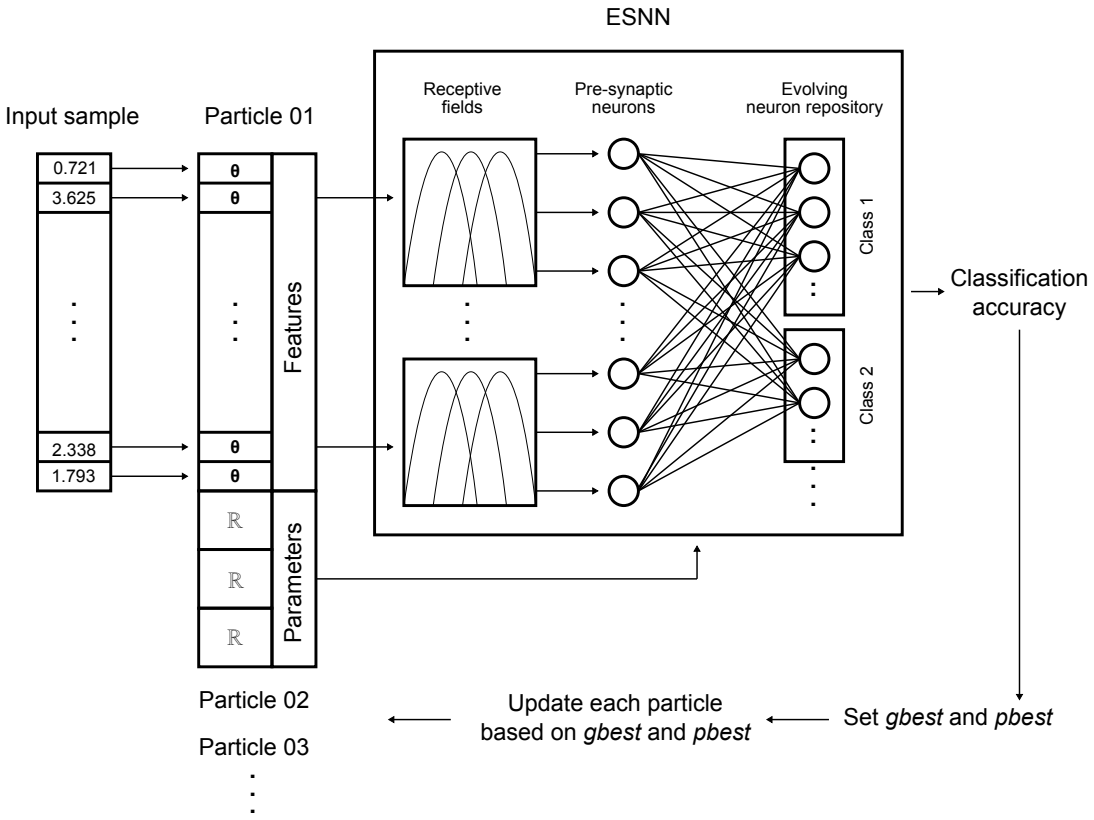

Figure 5.3: An integrated ESNN-DQiPSO framework for feature selection and parameter optimisation.

estimate the classification accuracy which determines the fitness of the model. In the next section, the result obtained for the proposed framework will be compared with results obtained in previous experiments.

5.3 PERFORMANCE ANALYSIS

The performance of DQiPSO when selecting relevant features, removing irrelevant features and optimising the parameters is discussed in this section. These crucial tasks have a direct effect on the final classification outcomes. Results from DQiPSO optimisation are compared to the previous experiments to evaluate its efficiency.

Feature selection

Figure 5.4 illustrates the comparison of selected features from DQiPSO and QiPSO for Hypercube dataset during the learning process. Lighter colours

correspond to features that are selected more often. In contrast, darker colours represent features that are eliminated during the 10 runs of the experiment. Figure 5.4(a) shows the result from a previous experiment with the integrated ESNN-QiPSO framework, while Figure 5.4(b) shows the results obtained with the new proposed integrated ESNN-DQiPSO framework.

The top diagrams illustrate the feature evolution. All ten relevant features being selected during learning by DQiPSO are clearly shown. From the figure, ten Hypercube relevant features that contain the most information can be clearly identified and are constantly being selected by DQiPSO. In contrast, the redundant and random features are completely rejected during the optimisation process. DQiPSO takes around 600 iterations to identify the relevant and irrelevant features.

All features have been ranked based on the number of selected features from 10 runs to determine their relevance after learning as shown in the bottom diagrams. Based on the feature ranking, the features found most relevant are Feature 15, Feature 19, Feature 26 and Feature 30, all of which have been selected nine times in the 10 simulation runs. They are followed by Feature 9, Feature 20, Feature 2, Feature 10, Feature 11 and Feature 4. Although the relevant features are not selected together during all 10 simulation runs, the result shows that the few combinations of relevant features are sufficient to give good classification results in a run. In addition, the major achievement of DQiPSO in this experiment is its ability to completely reject the redundant and random features. In contrast, the ability of the QiPSO to reject the irrelevant features is unsatisfactory. As shown in Figure 5.4(a), most of the irrelevant features are still being selected, which contributes to the low classification accuracy and increased computation time.

For the Two Spirals problem, Figure 5.5(b) clearly shows that the two relevant features which contained the most information, are constantly being selected at most of the time by DQiPSO. In contrast, the random value features are rejected during the learning process together with most of the redundant features. Interestingly, Feature 9 which is the redundant feature has been considered as the most relevant in the final feature ranking by DQiPSO. The reason is simply because this feature contains almost the same information as the original feature as shown in Figure 4.3. This feature is complemented by Feature

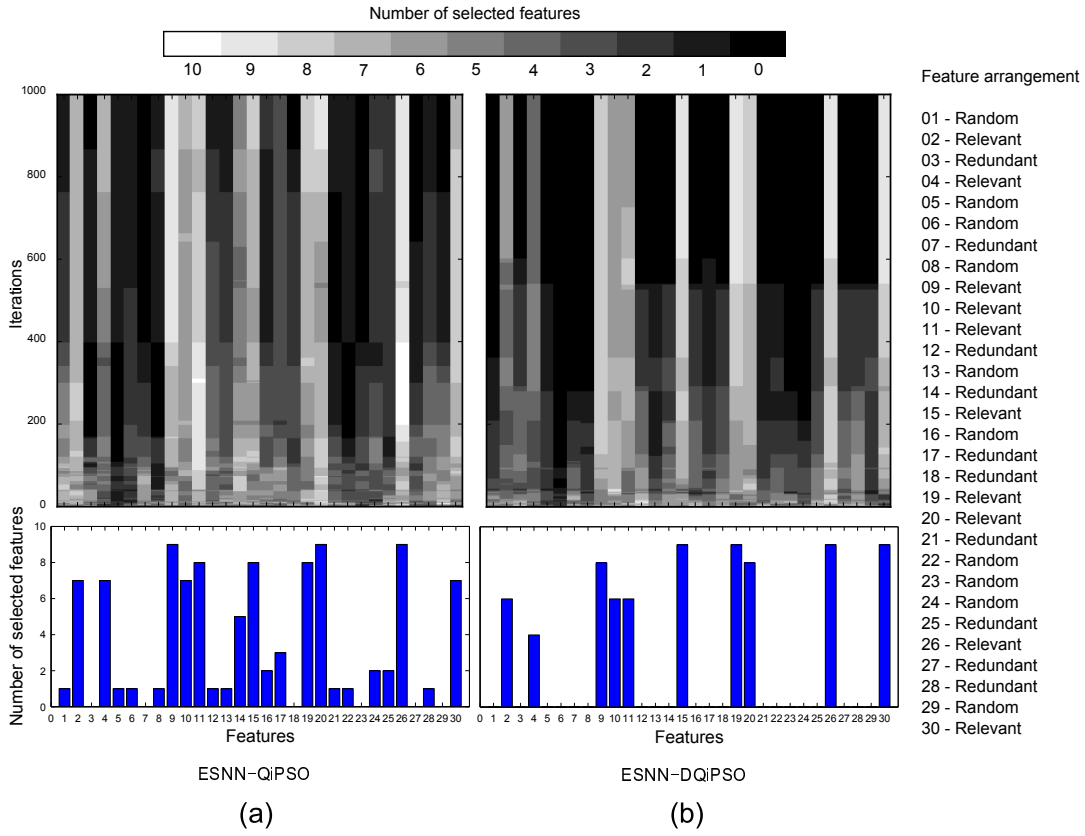

Figure 5.4: Evolution of feature selection on the Hypercube dataset. The bar graphs show the final features selected in 10 runs. (a) for ESNN-QiPSO and (b) for ESNN-DQiPSO integrated framework.

2, Feature 13 and Feature 18 in the feature ranking, all selected eight times in the 10 simulation runs. These four features contain the most information available to distinguish between two output classes. In addition, some redundant features that contain information with a noise level of 0.3 and 0.4 are still occasionally selected. Other features that are only occasionally selected or not selected at all can be considered as irrelevant features.

This situation is not much different for this dataset when compared to QiPSO. In the previous experiment, QiPSO also selected and considered features with noise level of 0.2, 0.3 and 0.4 as relevant in addition to the original features. However, the ability of QiPSO to reject the irrelevant features was unsatisfactory. Most of the redundant and random features were still selected. Also, some irrelevant features were regarded as relevant as shown by high number of times they were selected, for instance, Feature 14, Feature 19 and Feature 20 in Figure 5.5(a). This situation has affected the results and the overall classifica-

tion performance of the ESNN-QiPSO. This give slightly lower accuracy and increased computing times since more features have been selected. Because of QiPSO has no mechanism to stimulate the particle if there is no better solution found, the algorithm may converge prematurely without obtaining the optimal results.

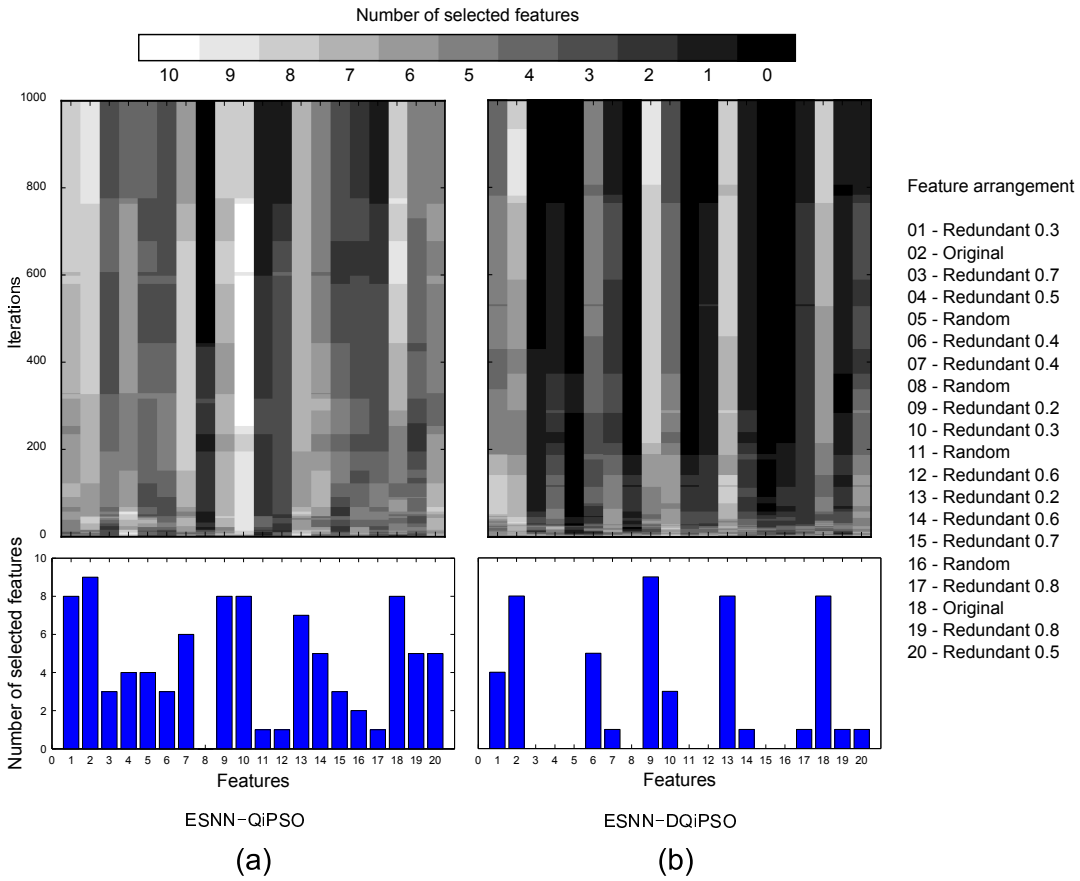

Figure 5.5: Evolution of feature selection on the Two Spirals dataset. The bar graphs show the final features selected in 10 runs. (a) for ESNN-QiPSO and (b) for ESNN-DQiPSO integrated framework.

Parameter optimisation

Parameter optimisation for the Hypercube problem is illustrated in Figure 5.6. In the experiment with ESNN-DQiPSO, the Mod value has converged to almost the same value as reported in the two previous experiments. The high

value of this parameter is due to the higher number of connections in the network. The weight assignment in ESNN as described by Equation 2.25 requires a higher value if many connections exist. Otherwise, some connections may have a weight value of 0.0. This will severely affect the network performance, especially when accumulating the PSP. Because Mod corresponds to the ESNN weight, a higher value is required to make sure all connections have weights associated to them. Both parameters C and Sim evolve to a certain optimal value.

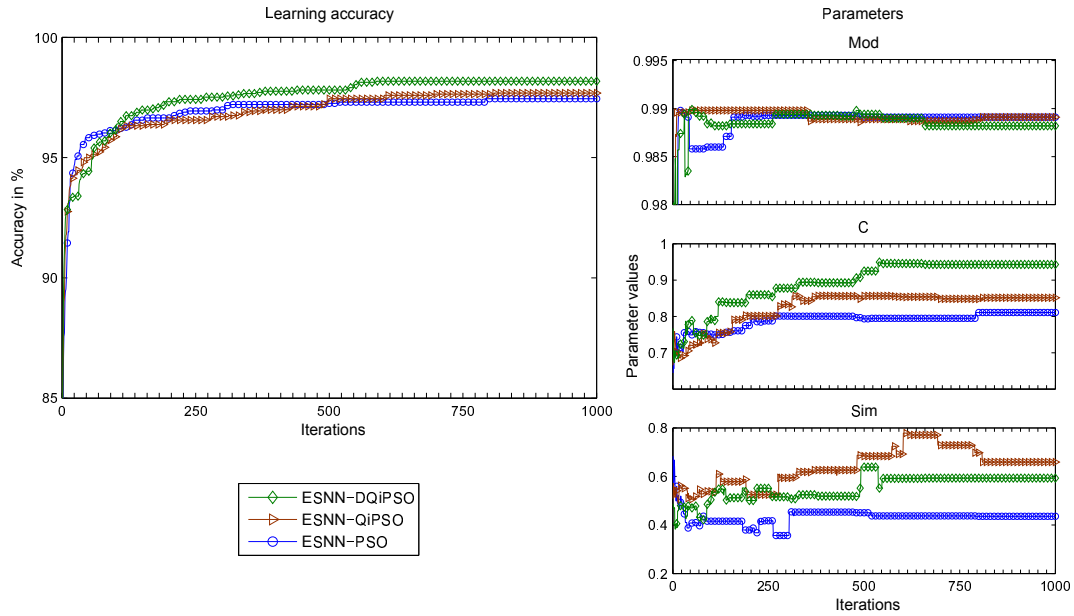

Figure 5.6: Comparison between the accuracy and parameter optimisation of the three frameworks when tested on the Hypercube dataset.

The same situations happen for the Two Spirals problem. In Figure 5.7, Mod value steadily decreases because of a lower number of generated connections due to the feature selection task. This time, DQiPSO managed to remove a significant number of irrelevant features. In line with the previous argument, a lower number of features produce a lower number of connections, therefore it requires a lower Mod value. In this experiment, the proportion parameter C is found to be high. Meaning that more input spike trains are required in order to reach the firing threshold. Lastly, the Sim parameter evolves between the

values obtained in the last two experiments to achieve the optimal combination for all ESNN parameters. ESNN parameters behaviour has been extensively studied (Wysoski et al., 2006b; Schliebs, Defoin-Platel, Worner, & Kasabov, 2009a). The results of this experiment support the finding of those studies in term of the optimal range of the parameters.

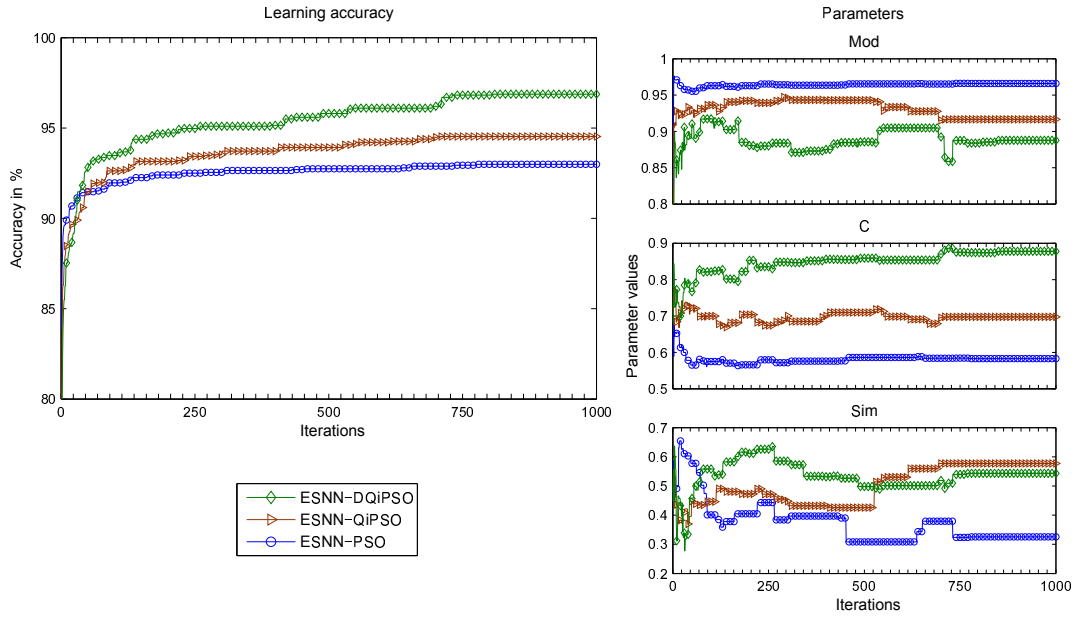

Figure 5.7: Comparison between the accuracy and parameter optimisation of the three frameworks when tested on the Two Spirals dataset.

In terms of the classification result, the average accuracy for ESNN-DQiPSO is improved by 1.00% for the Hypercube problem and 7.87% for the Two Spirals problem. Results of every single run are consistently above 80% for both problems. Table 5.1 shows a comparison of the accuracy results compared to those in the previous experiments. The results show the proposed integrated ESNN-DQiPSO outperformed results obtained with ESNN-QiPSO and ESNN-PSO, with a particularly large margin for the Two Spirals problem. The main reason for this achievement is that DQiPSO managed to select more relevant features and remove more irrelevant features in the Two Spirals dataset. For the Hypercube dataset, both DQiPSO and QiPSO methods were able to select the ten relevant features. However, DQiPSO managed to remove all irrele-

vant features which QiPSO failed to do. This experiment clearly demonstrates that the proposed DQiPSO enables the network to discover more relevant features, eliminate more irrelevant and noise features and to better optimise the parameters. For ESNN-PSO, the algorithm is entirely dependent on the parameter optimisation, which is inadequate and has affected the results. It has the lowest accuracy among all three methods proposed in this study.

Table 5.1: Comparison of classification accuracy

Method	**Hypercube**	**Two Spirals**
ESNN-DQiPSO	95.74% $\pm$ 2.69	91.96% $\pm$ 3.73
ESNN-QiPSO	94.74% $\pm$ 4.34	84.09% $\pm$ 6.43
ESNN-PSO	93.81% $\pm$ 3.49	73.26% $\pm$ 7.33

DQiPSO Performance Remarks

As demonstrated, the performance of DQiPSO is overall better than the performance of the standard QiPSO and PSO. Interestingly, DQiPSO uses a smaller number of particles in contrast to the other tested optimisers. Because of the robust search strategy, all particles in the swarm perform well both in feature selection and parameter optimisation problems. The new update strategy, which is crucial for $gbest$ selection, shows that the optimiser is continuously keeping the number of selected features at the lowest rate possible. DQiPSO managed to completely remove all irrelevant features in the linear problems. Although in DQiPSO every particle has been assigned with a different search strategy and also new selection approach, the proposed optimiser is not able to remove all irrelevant features especially in difficult problems such as Two Spirals. However, the number of irrelevant features is significantly reduced in this case.

In term of $gbest$ assignment, almost all types of particles have been selected as $gbest$ during learning. However, Update particle managed to search the best solution and selected as $gbest$ at most of the time. Other types of particles, are occasionally selected as $gbest$. Nevertheless, this shows that all proposed par-

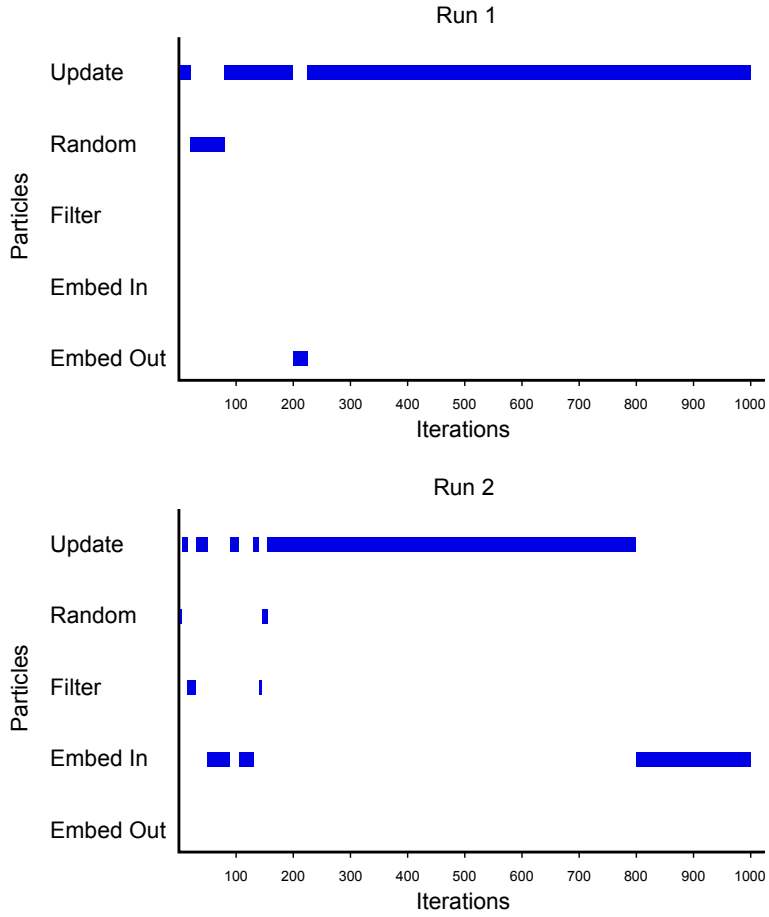

Figure 5.8: $gbest$ assignment during two validation runs. All the proposed particles managed to find the best solution at certain iterations and assigned as $gbest$ particle.

ticles are able to search better solutions which contribute to a better learning mechanism in the swarm. Figure 5.8 illustrates an example of $gbest$ assignment during learning in two validation runs. Motivated by results from this experiment, the DQiPSO was tested on a problem with greater dimension where a probabilistic element is introduced to the ESNN. Chapter 6 discusses the probabilistic ESNN architecture.

5.4 SUMMARY

This chapter has introduced a new DQiPSO model and has shown how this optimiser can be implemented for feature selection and parameter optimisation tasks. The optimiser has been applied to ESNN can be used in any problems

that require binary and the real value data to be optimised simultaneously. In addition to the particle structure, each type of particle has its own way to discover the relevant features. Some particles in this model still use the normal random selection at the beginning of the learning process. Other particles evaluate every feature to determine the relevancy. The proposed method results in the design of faster and more accurate classification models than the ones optimised with the use of standard evolutionary optimisation algorithms.

Future work is planned to apply the proposed optimiser to the modified ESNN. The modified ESNN, which will be discussed in the next chapter, requires an optimiser in order to work. DQiPSO optimises the evolved connections which may enhance the ESNN model and its classification accuracy. Only selected connections will be used during learning. It is believed that the proposed DQiPSO will be able to perform optimisation in an efficient way.

Chapter 6

A NOVEL PROBABILISTIC ESNN ARCHITECTURE AND ITS OPTIMISATION WITH THE USE OF DQIPSO

This chapter proposes a novel ESNN based on Kasabov's PSNM (Kasabov, 2010). In this modified ESNN, the connections are evolved based on the information it holds. Since the connections are dynamic, some probability computation is required, similarly to the process of feature selection task. An integrated structure is proposed, in which DQiPSO is used to optimise simultaneously connections, features and parameters. Features and connections are modeled as a qubit vector while the parameter values are presented as real numbers. The proposed method is evaluated using the datasets used for evaluating the frameworks as discussed in previous chapters. Results are compared with those obtained in previous experiments.

6.1 THE PESNN ARCHITECTURE

Kasabov (2010) introduced the probabilistic concept to the spiking neuron models to simulate biological neurons and also to enhance the model capability. This chapter introduces the extended version of the ESNN that embeds Kasabov's PSNM and is named the Probabilistic ESNN (PESNN). One major problem in ESNN is that a large number of pre-synaptic neurons are needed for each input feature. By introducing probability into the ESNN connections, the network can identify which connections are needed during the learning process. If PESNN can produce better results than ESNN, it shows that not all connections are needed. Thus, selecting certain connections and spike trains

not only leads to better results but also demonstrates that the internal learning process occurs during the selection of connections.

There are three probabilities in PSNM. The first one is the probability that a spike is emitted by pre-synaptic neurons if the connection exists. The second one is the probability of the transmitted spike to be used for PSP computation. The final probability is the probability for the output neuron to emit an output spike once the total PSP has reached the threshold. However, in this study, only the first probability is studied and applied to the Thorpe's neuron model in ESNN. In this PESNN, not only features are represented by a qubit vector, but also all connections between the neurons. A neuronal connection is either existent (1) or nonexistent (0), or in another interpretation either propagating a spike or not propagating it. A qubit vector would be a suitable representation of all connections that can be optimised using DQiPSO. In the proposed framework, each particle is divided into three parts: the first two parts use quantum probability computation for feature and connection selection and the last part holds the real value of the parameters. The PESNN structure is illustrated in Figure 6.1.

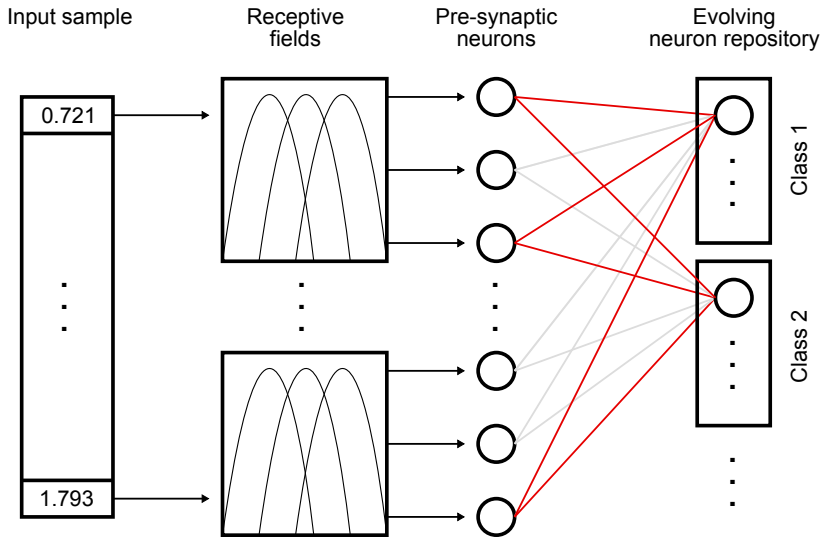

Figure 6.1: PESNN Structure with evolving connections. The darker lines represent selected connections while lighter lines indicate connections that are not selected.

The main idea is to get the best connections for the network. By selecting a different connection arrangement, a different firing time is generated for the pre-synaptic neurons and this might give a different output as shown in Figure 6.2. The figure provides a detailed illustration on how the proposed network works. There are two levels of selection, starting with selection of features and followed by selection of connections. Selected features are encoded into spike trains using population encoding. Similar to the feature selection task, all the connections are mapped into a qubit mask. Every particle in the swam selects a string of connections based on the qubit collapsed states. In the figure, two particles, Particle A and Particle B, have been assigned to select the connections during learning. The right diagrams show the example of connections selected by both particles and subsequently used for the classification. Different connections and number of selected connections may lead to different classification outcomes and accuracy.

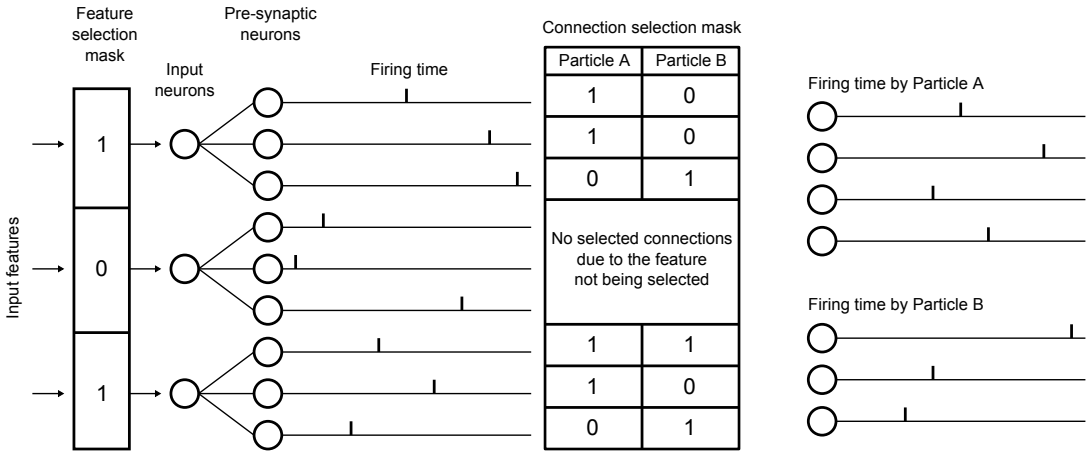

Figure 6.2: Example of connection selection task in PESNN.

6.2 A PROPOSED INTEGRATED PESNN-DQIPSO FOR SIMULTANEOUS CONNECTION, FEATURE AND PARAMETER OPTIMISATION

The proposed integrated PESNN-DQiPSO framework is almost identical to the previous ESNN-DQiPSO. However, the induction method is replaced by

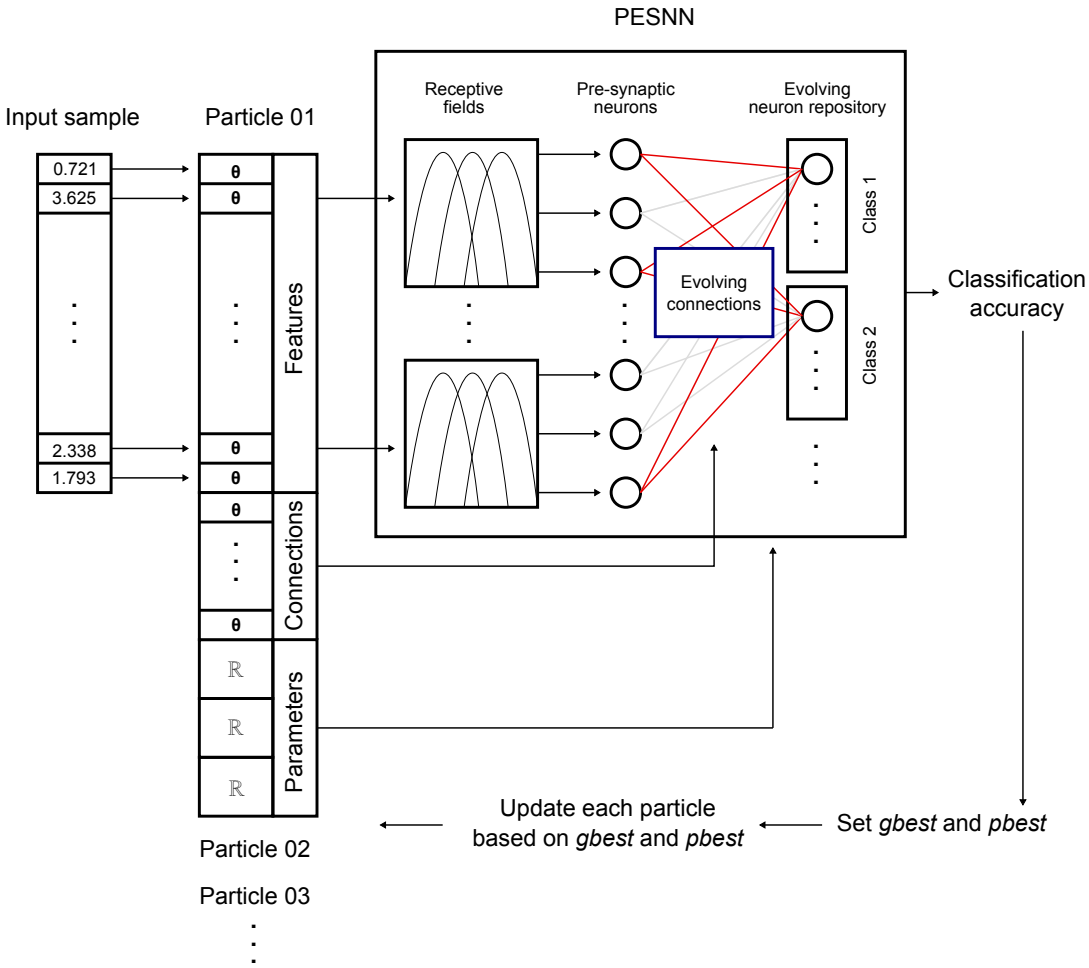

Figure 6.3: Integrated PESNN-DQiPSO framework.

the PESNN, which has evolving connections. The connections will be optimised by DQiPSO. Therefore in this framework, each particle has three parts. Two parts hold the quantum probability computation for connections and features while the third part holds the parameter optimisation. Connections and features are mapped into a string of qubits. In this case, the collapse value 1 represents the connection or feature being selected, otherwise a value 0 is assigned. Like in the previous frameworks, DQiPSO interacts with classifier and in this case PESNN, to identify the best connection structure, the most relevant features and best set of parameters. The learning starts with random probability values for connections and features and random real values for parameter. Every particle's information will be used to train the network to achieve the highest classification accuracy for a given dataset. Subsequently, every particle updates its position based on $gbest$ and $pbest$ found during learning.

Algorithm 6 Integrated PESNN-DQiPSO

1: **for all** particle **do**
2: initialise all ESNN parameters
3: **for all** feature qubit and connection qubit **do**
4: initialise θ
5: get collape state using Equation 3.12
6: **end for**
7: initialise fitness
8: **end for**
9: **while** not reaching maximum iteration **do**
10: **for all** particle **do**
11: get fitness from ESNN (Algorithm 1)
12: **if** (current fitness better than $pbest$ fitness) or ((current fitness == $pbest$ fitness) and (feature selected less than feature selected by $pbest$)) or ((current fitness == $pbest$ fitness) and (feature selected == feature selected by $pbest$) and (connection selected less than connection selected by $pbest$)) **then**
13: assign current particle as $pbest$
14: **if** (current $pbest$ fitness better than $gbest$ fitness) or ((current $pbest$ fitness == $gbest$ fitness) and (feature selected by $pbest$ less than feature selected by $gbest$)) or ((current $pbest$ fitness == $gbest$ fitness) and (feature selected by $pbest$ == feature selected by $gbest$) and (connection selected by $pbest$ less than connection selected by $gbest$)) **then**
15: assign $pbest$ as $gbest$
16: **end if**
17: **end if**
18: **for all** ESNN parameters **do**
19: calculate velocity using Equation 3.1
20: update parameter using Equation 3.2
21: **end for**
22: **for all** feature qubit and connection qubit **do**
23: calculate velocity using Equation 3.9
24: apply rotation gate in Equation 3.11
25: get collape state using Equation 3.12
26: **end for**
27: **end for**
28: **end while**

Learning continues until a termination criterion is met. Since the architecture of PESNN is derived from ESNN, the same population encoding is employed

for the data encoding, the Thorpe's model for PSP calculations with a fast one-pass learning algorithm. Algorithm 6 explains the proposed integrated PESNN-DQiPSO, Appendix D provides a detailed description of the framework and Figure 6.3 illustrates the proposed framework.

In DQiPSO, a new approach of selecting $gbest$ has been discussed in Section G.10. Any particle will only be assigned $gbest$ if its current fitness is better or with fewer selected features than the current $gbest$. However, an additional rule is appended to the optimiser for connection selections and assigned the $gbest$ particle in PESNN. The new rule imposes that any particle will be crowned as $gbest$ if it has the same accuracy and number of selected features, but lower number of selected connections than $gbest$. This is to ensure that the optimiser always holds the best set of features and connections in the lowest possible number in order to identify the optimal features and connections.

6.2.1 *Setup*

Thirty receptive fields will be assigned to every selected feature. A higher number of receptive fields is chosen here than in previous experiments in order to give more space for the optimiser to optimise PESNN's connections. As a control measure, the minimum number of connections required for a feature was set to 2. This is needed to avoid a selected connection to end up with no assigned connections. Eighteen DQiPSO particles were assigned consisting of six Update, three Filter, three Random, three Embed In and three Embed Out particles. c_1 and c_2 were set to 0.05 for probability computation and 1.2 for real value update. The inertia weight w was set to 2.0. Ten-fold cross validation was used and the average result was computed in 1000 iterations. This method was tested on the Hypercube and the Two Spirals datasets. The result of the proposed method has been compared with result obtained in previous experiments.

6.2.2 *Performance Analysis*

The performance of PESNN will be evaluated based on connection, feature and parameter optimisation.

Evolving connections

Figure 6.4 and Figure 6.5 show the results of the evolution of PESNN connections for both Hypercube and Two Spirals problems. The results are compared with the connections selected in ESNN-DQiPSO and ESNN-QiPSO. The total number of selected connections for the previous experiments are calculated by multiplying the total number of features selected by the optimiser with the total number of pre-synaptic neurons selected (which is 20) and divided by 10 runs. In contrast, the total number of selected connections for PESNN are collected directly from DQiPSO.

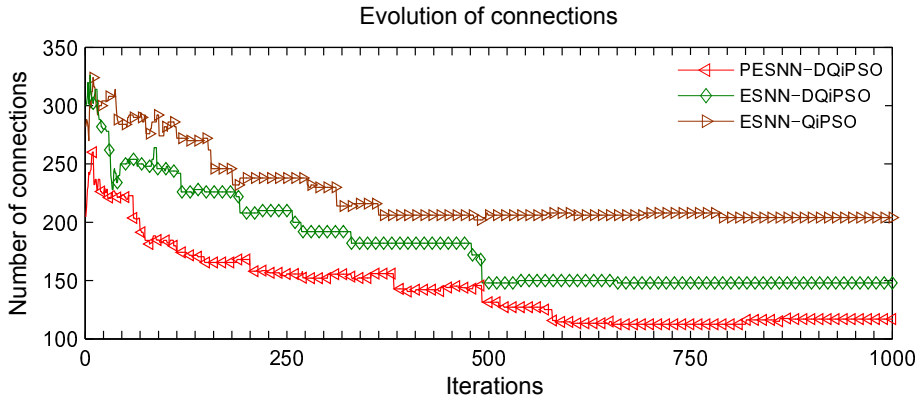

Figure 6.4: Evolving connections on Hypercube problem as derived from three proposed methods in this study. PESNN-DQiPSO algorithm steadily eliminates connections during the learning process. The method starts with a high number of selected connections and gradually decreases that number during learning. In comparison to the ESNN-DQiPSO and ESNN-QiPSO, number of selected connections is based on a number of features selected during the learning process for both algorithms. A lower number of features selected leads to a lower number of connections used.

For the proposed PESNN-DQiPSO framework, the initial number of connections randomly selected at the beginning of the learning process is 217.7

connections for the Hypercube problem. Then the connections are steadily evolved and their number starts to decrease, aligning themselves with the most informative features during learning. The final average number of connections recorded after the learning is 123.50. In a problem space with ten relevant features and 20 presynaptic neurons used, it is equivalent to 200 connections if all features have been selected. Corresponding to the final accuracy result, it demonstrates that even after almost half of the connection have been removed, the remaining connections are still able to produce a result comparable to previous ESNN results.

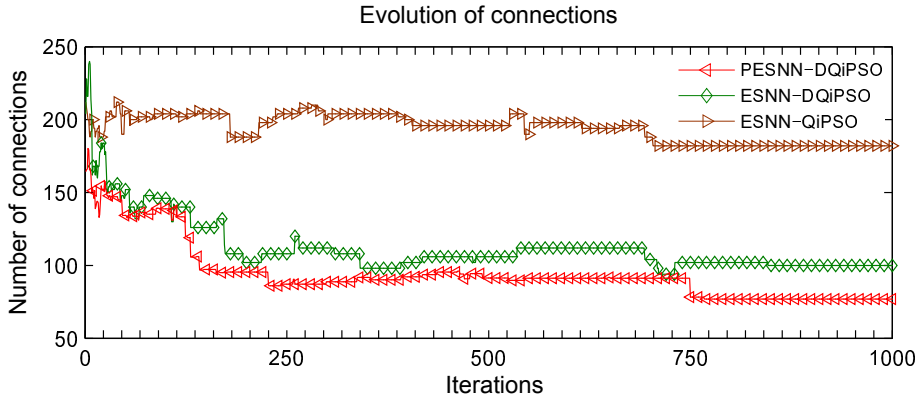

Figure 6.5: Evolving connections for Two Spirals problem with comparison between the three proposed methods. PESNN-DQiPSO optimises the connections during the learning process. The high number of features selected by ESNN-QiPSO can be translated into a high number of selected connections.

Similar results have been recorded for the Two Spirals problem as shown in Figure 6.5. The number of connections gradually decreases from the initially selected 160.10, along with the total number of selected features. The number of evolved connections in PESNN-DQiPSO and number of connections in ESNN-QiPSO was very similar due to the number of selected features by both algorithms are almost identical and fewer in number as shown in Figure 6.7. The final average number of connections selected by DQiPSO for this problem is 68.60. In contrast, the total number of connections of ESNN-QiPSO is very high due to the large number of features selected during learning. The

final classification accuracy for Two Spirals is improved due to connections evolving in line with feature and parameter optimisation.

This experiment demonstrates that ESNN with evolving connections is able to produce improved results. More importantly, the evolving connections in PSNM improves the learning capability of the standard ESNN. However, due to the sensitivity of each connection that holds some spike information, a correct combination and sufficient number of connections are crucial. Otherwise, the results may be worse than those obtained with the standard ESNN due to insufficient information supplied to the output neurons for final PSP computation.

Feature selection

The previous experiment discussed in Section 4.2 shows the accuracy of the algorithm without feature optimisation is low compared to the algorithm with feature optimisation. Figure 6.6 shows the evolution of feature selection during 1000 learning iterations and the final selected features. For the Hypercube problem, the same DQiPSO is able to select all 10 relevant features and eliminates all redundant and random features. In this specific experiment, it shows although the size of the particle is large due to three components needed to be optimised simultaneously, the feature selection ability has not been affected. In addition, the DQiPSO optimiser also managed to remove all irrelevant features around 600 iterations, the same time as in previous experiment. More iterations are required because of the problem size that each particle holds. For PESNN-DQiPSO, the most relevant features are found in Feature 4, Feature 10 and Feature 19 with nine selected time from 10 runs. They are followed by Feature 2, Feature 11, Feature 20, Feature 26, Feature 15, Feature 30 and Feature 9. In feature selection mechanism of DQiPSO, after the relevant features are found, the particles then try to reduce the number until the smallest subset of relevant features are found. This is the reason why there are still some activities after 10 relevant features are found. Overall, the duty of finding the best and smallest subset of relevant features has been successfully complied by DQiPSO for both PESNN and ESNN classifiers. For the QiPSO performance, the optimiser

has managed to select and give high ranking of all 10 relevant features even when some irrelevant features were still occasionally been selected together.

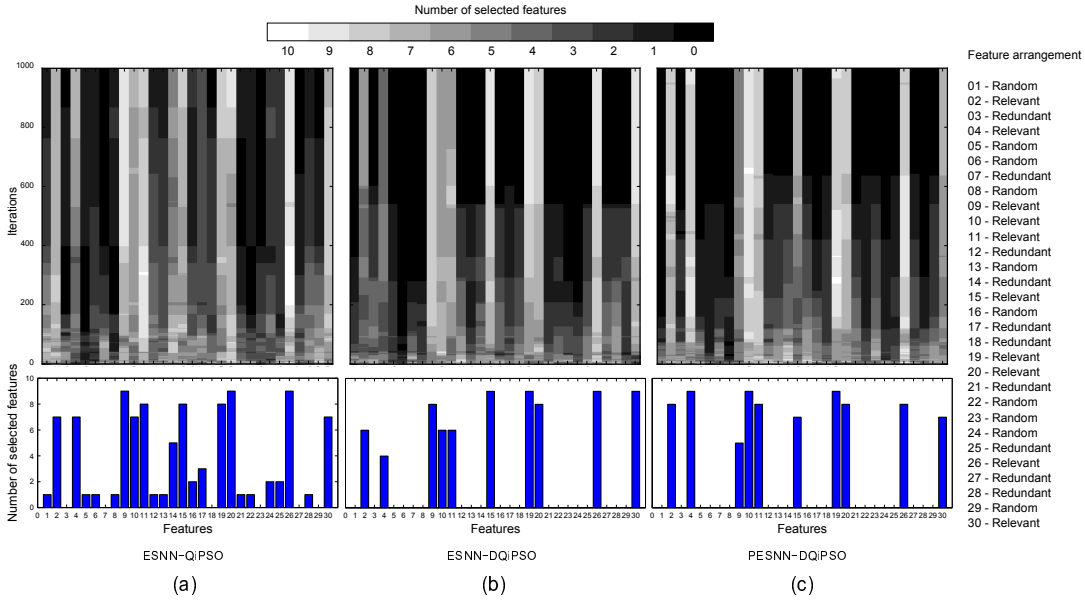

Figure 6.6: Evolution of feature selection on the Hypercube dataset. The bar graph at the bottom show the final features selected in 10 runs from: (a) ESNN-QiPSO framework, (b) ESNN-DQiPSO framework and (c) PESNN-DQiPSO integrated framework.

In the Two Spirals problem, two original features and features with the noise level of 0.2 were selected by DQiPSO. Results are depicted in Figure 6.7. In these two experiments conducted for DQiPSO optimised PESNN and ESNN, the four features - pair of original and features with noise 0.2 have been considered relevant. Due to the small noise value, features with noise level of 0.2 is very hard to be removed. Both features are almost identical and contain a lot of information that can be used to distinguish between output classes. This experiment also shows that DQiPSO is not able to remove completely other redundant features, similar to previous experiments. The irrelevant features, some with noise levels of 0.5, 0.6 and 0.7 were still occasionally selected after 1000 learning iterations. Although DQiPSO is not able to completely eliminate the irrelevant features in Two Spirals problem, its ability to significantly reduce the number of irrelevant features when compared to QiPSO is satis-

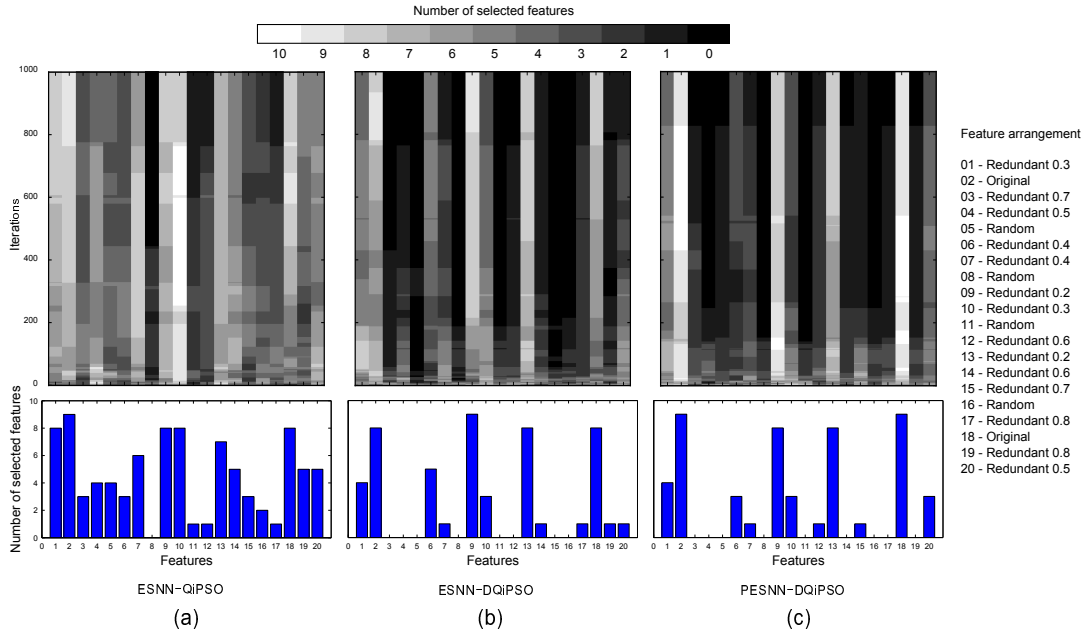

Figure 6.7: Evolution of feature selection on the Two Spirals dataset. The bar graph at the bottom show the final features selected in 10 runs from: (a) ESNN-QiPSO framework, (b) ESNN-DQiPSO framework and (c) PESNN-DQiPSO integrated framework.

factory. The final features ranked highest by DQiPSO in this experiment are Feature 2, Feature 18, Feature 9 and Feature 13 which all contains the most information. Features that have been selected less than half the times in the 10-fold cross validation runs can be considered irrelevant.

Parameter Optimisation

Figure 6.8 and Figure 6.9 show the evolution of parameter optimisation and accuracy for Hypercube and Two Spirals problems, respectively. All parameters converge to a certain value that is within the range of values from previous experiments. Parameter Mod for Hypercube dataset once again converges around the value obtained from the previous experiments. From the current and previous experimental findings, it can be concluded that the optimal range for Mod is between 0.985 to 0.990, for C is between 0.65 and 0.95 and for Sim is between 0.3 to 0.7. This means that Mod should be as high as possible, C is between the other two parameters and Sim is the lowest. Lower similarity proportion value means that fewer outputs neuron converged. This could happen

to high-dimensional and non-linear separable problems due to the complexity of their structure.

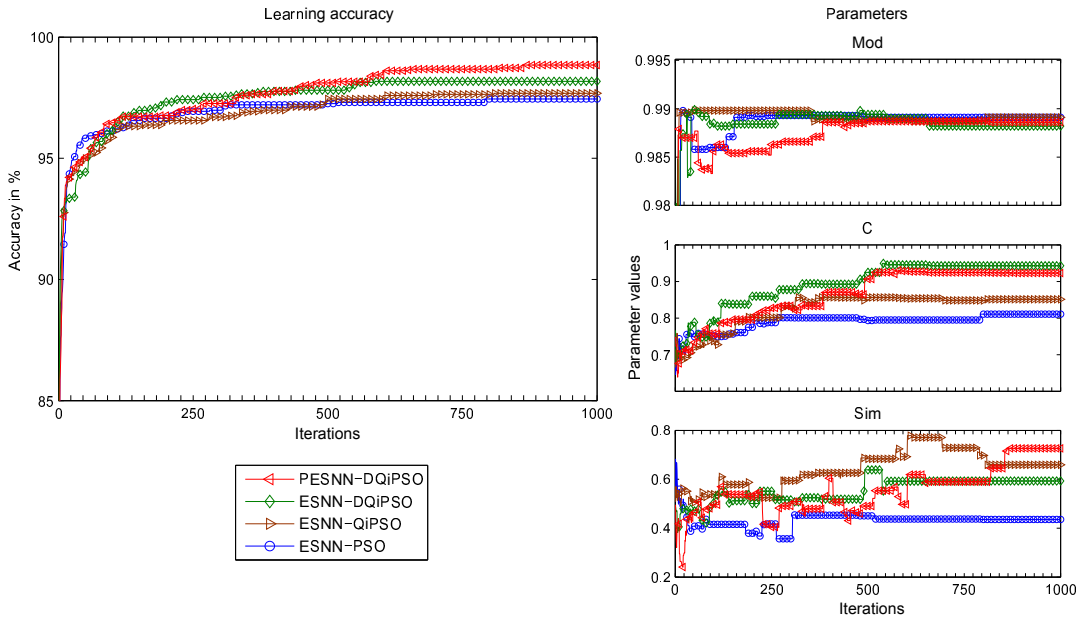

Figure 6.8: Evolution of accuracy and parameters on Hypercube dataset from the integrated PESNN-DQiPSO framework.

In both problems, a slight improvement of accuracy is achieved in comparison to the ESNN-DQiPSO experiment. The accuracy obtained for Hypercube problem is 96.29% and for the Two Spirals problems is 92.68%. Generally, a steady improvement of about 1.00% can be observed for each of the proposed methods when applied on the Hypercube dataset. When comparing PESNN-DQiPSO with ESNN-QiPSO, a significant improvement can be noticed especially for the Two Spirals problem which contains variable level of noise. This experiment has demonstrated that apart from optimising the features and parameters, classification accuracy also can be improved by selecting the right connections in the ESNN architecture.

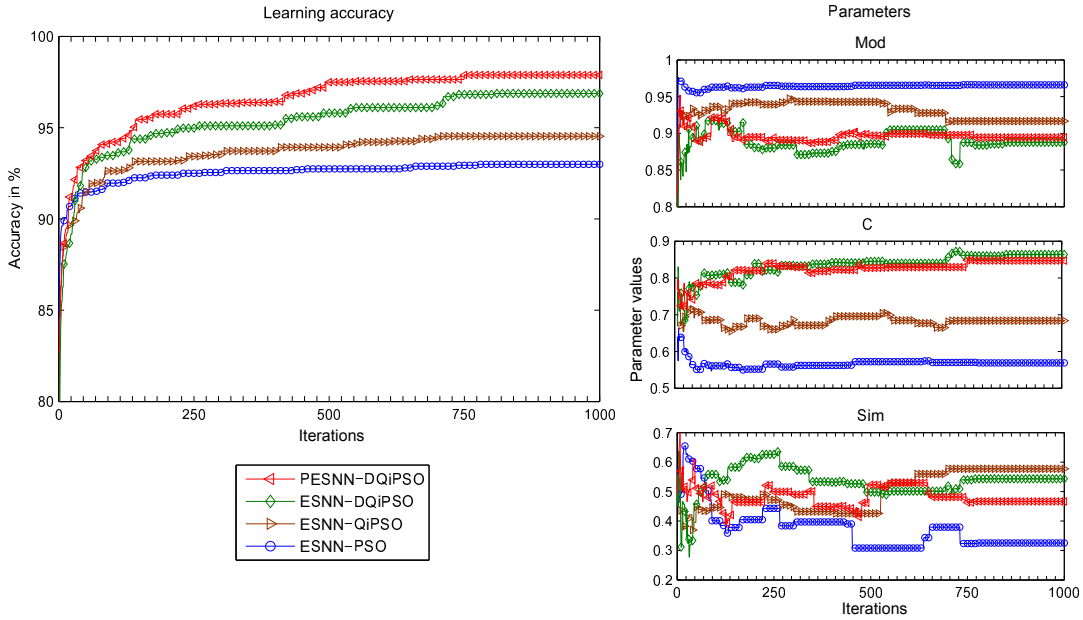

Figure 6.9: Evolution of accuracy and parameters on Two Spirals dataset from the integrated PESNN-DQiPSO framework.

Table 6.1: Comparison of classification accuracy

Method	Hypercube	Two Spirals
PESNN-DQiPSO	96.29% ± 2.99	92.68% ± 3.67
ESNN-DQiPSO	95.74% ± 2.69	91.96% ± 3.73
ESNN-QiPSO	94.74% ± 4.34	84.09% ± 6.43

6.3 COMPUTATIONAL COST

The average computational time for a single run when the proposed methods are applied to the Hypercube dataset is depicted in Table 6.2. ESNN-QiPSO and ESNN-DQiPSO optimises parameters and executes feature selection simultaneously during learning. The number of selected features may affect the computational time. Less feature selected, faster processing time can be achieved because of the fewer connections exist in ESNN. Since more features are selected by QiPSO, this resulted more computational time required compared to DQiPSO. In addition, the number of particles in QiPSO is also more than the number of particles in DQiPSO which requires more time to perform the optimisation. PESNN-DQiPSO takes significantly more computational time than other proposed methods. This is because it optimises not only

parameters and features, but also the high dimensional of ESNN connections. Although the increment of accuracy is only about 1% when PESNN-DQiPSO is tested in 10 fold cross validation for both datasets as shown in Table 6.1, the optimisation is run offline for a single time to optimise the network. The optimised network is then ready to be applied as an online classifier. It is worth to have higher computation time during learning so that the network is well trained for the online usage in the later stage.

Table 6.2: Comparison of computational time

Method	Time in minutes
PESNN-DQiPSO	49
ESNN-DQiPSO	18
ESNN-QiPSO	24

It is explicitly noted that the execution time is not the best way to compare the performance between algorithms since it may run in a different hardware specification and also different software or programming tools. Nevertheless, the discussion will be very informative when the identical hardware configurations and tools are used between all algorithms.

6.4 SUMMARY

This chapter presents a novel PESNN framework, where connections evolve based on spike information it held. The PESNN architecture allows the most informative connections to be exploited. This method does not only produce a better classification accuracy, but also enhances the learning capability and speeds up learning due to the lower number of selected connections.

Since it is unrealistic to determine manually which connections are the best, DQiPSO has been used as an optimiser for PESNN. The connections are mapped to the particle's quantum mask to determine whether a particular connection is selected or removed. Every particle has its specific selected features, parameters and connections. The particle with the best connections selected, together

with the best features and parameters that give the highest accuracy is then set as $gbest$. Other particles update their positions including their connections based on information supplied by $gbest$ and its $pbest$. From the conducted experiment, it becomes apparent that the proposed integrated PESNN-DQiPSO method has demonstrated promising results and is worthy for further exploration. The results have shown that DQiPSO is able to simultaneously identify relevant features, recognise a suitable number of connections and optimise parameters. PESNN provides accuracy that is better than the accuracy achieved with other tested methods. Thus, reveals that evolving connections improves the learning capability to the ESNN. However, this experiment has also addressed some problems regarding PESNN, especially the necessity for a suitable number of connections for better results. The lower number of selected connections produces poorer results as found in some particles during the learning process.

In the next chapter, ESNN will be modified and tested on some of the most common types of real world problems. In spatiotemporal problems, both spatial and temporal components are important for decision making. It is believed that ESNN will be suitable for solving such problems because of the temporal component found in the architecture. However, some modifications are necessary to accommodate both spatio and temporal information components.

Chapter 7

A NEW METHOD FOR SPATIOTEMPORAL PATTERN RECOGNITION BASED ON AN EXTENDED ESNN

Often real world problems are spatiotemporal. Such problems consist of space (or spatial) and time (or temporal) components, both equally important for decision making. A standard classifier is normally capable of processing only one of the components, either spatial or temporal. In this chapter, a new framework is proposed on an ESNN for spatiotemporal problems. An additional component is added to capture all information in spatiotemporal problems. The additional module exploits the principle of reservoir computing to capture both spatial and temporal information and transforms it into another data representation form that enables ESNN to classify the data efficiently. This chapter also proposes a synthetic spatiotemporal benchmark dataset used to evaluate the proposed Extended ESNN (EESNN) framework.

7.1 SPATIOTEMPORAL PROBLEMS

Spatiotemporal data relates to objects whose position, shape and size change over time (Theodoridis & Nascimento, 2000). Spatiotemporal problems normally deal with a sequence of events within a given time frame. The problem is defined by a time-evolving spatial object represented by a set of triplets (o_{id}, s_i, t_i) where o_{id} is an object with the identification number, and s_i is the location of the o_{id} at time t_i (Theodoridis, Sellis, Papadopoulos, & Manolopoulos, 1998). When solving spatiotemporal problems, spatial information is needed to represent the position of the object in space together with the temporal information indicating when an event has occurred. One of the most com-

mon spatiotemporal problems is Electroencephalography (EEG) signal processing and object recognition.

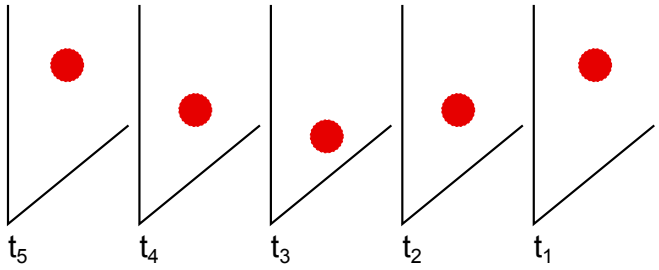

Figure 7.1: A swing ball is an example of spatiotemporal data. The figure shows the spatial position of the ball at five time points.

Figure 7.1 shows an example of spatiotemporal data. Five events occurred in a sequence of time. In spatiotemporal problems, one event is insufficient to describe the problem. For instance, no decision can be made at the single frame t_2 alone. However, when all events accumulate, then the problem can be described. In this case, the problem is portrayed as a swing ball.

There are a number of algorithms and approaches for dealing with spatiotemporal problems such as Time Delay Neural Networks (Waibel, 1989) and recurrent Elman networks (Elman, 1990). However, more biologically-inspired methods have been introduced for solving spatiotemporal problems. Many of the biological neuron properties have been studied as explained in Gerstner and Kistler (2002a), but have not been fully utilised for the creation of more efficient models for solving complex spatiotemporal problems. SNN are biologically plausible and offer some means for representing time, frequency, phase and other features of the information being processed. The main reason to study SNN for spatiotemporal problems is due to its ability to internally represent and process both spatial and temporal information adequately.

Recently, a reservoir-based method has been proposed as a solution for spatiotemporal problems (Verstraeten et al., 2007). Initial study of the combination of the SNN variant, the ESNN with the LSM has found to be promising for solving spatiotemporal problems (Schliebs, Nuntalid, & Kasabov, 2010). The study emphasises on the reservoir response after the spatiotemporal in-

put pattern is injected to the reservoir. Inspired by the findings of that study, Chapter 7 and Chapter 8 present an extended structure of the ESNN, where an additional component is introduced to deal with spatiotemporal problems. In this chapter, a simplified structure is proposed where a new module is added to capture spatiotemporal data sequence that needs to be further classified. This module utilises the standard ESNN encoding method of population rank order encoding to turn spatiotemporal input pattern into a spiking input pattern. Output from this module is then passed on to the evolving classification module that completes the classification task.

7.2 THE EESNN FRAMEWORK

The framework for the proposed EESNN is shown in Figure 7.2. EESNN incorporates two modules for information processing. The first module acts as a memory that captures the whole spatiotemporal data patterns that need to be classified. The second module is a standard ESNN used for the classification task. In the first module, both spatial and temporal components of the spatiotemporal problem are captured and transformed into high-dimensional spiking patterns. Every spatial variable value at every discrete time unit is encoded using the standard ESNN population rank-order encoding scheme. The encoded information for every time point is stored in a memory. The obtained memory of spikes is then fed into the second module for classification.

A fast one-pass time-to-first-spike learning algorithm is used that enables the new model to be more suitable for learning from the spatiotemporal streams in an adaptive and incremental manner. The high-dimensional spatiotemporal patterns are learned and classified in the evolving classification module. The Thorpe neuron model fires an output spike after sufficient spatiotemporal spike trains are received. Output from the learning for every sample is compared with the targeted output for classification accuracy computation. The proposed EESNN algorithm is described in Algorithm 7 and its detailed implementations are presented in Appendix E.

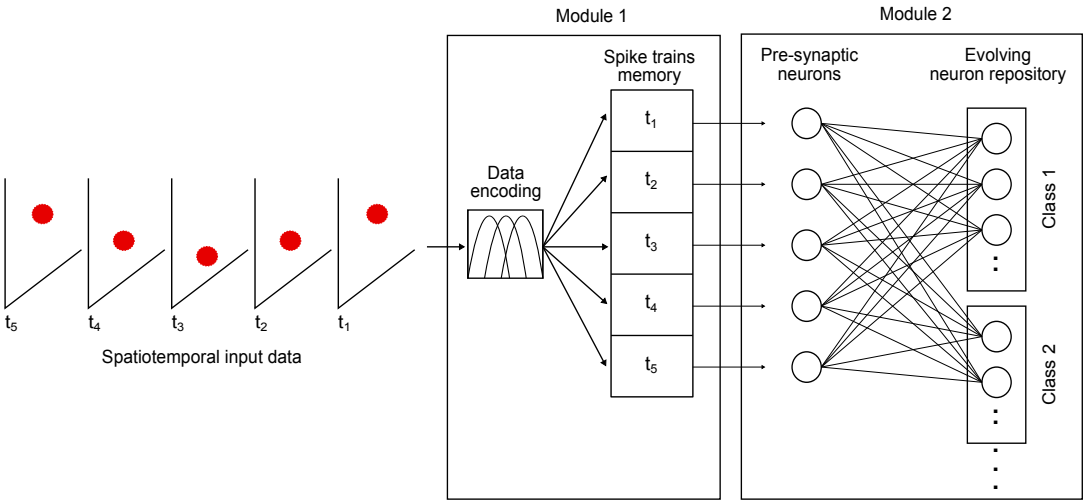

Figure 7.2: Extended ESNN framework with two modules.

Algorithm 7 EESNN

1: **for all** samples belonging to class l **do**
2: **for all** time points **do**
3: encode input samples into firing time using Equation 2.3
4: **end for**
5: accumulate all firing time for entire time points in a spike memory
6: **end for**
7: apply spike memory into ESNN (Algorithm 1)

7.3 APPLICATION ON SPATIOTEMPORAL DATASET

In order to evaluate the proposed EESNN on spatiotemporal problems, this study proposes a synthetic spatiotemporal dataset. The main reason for proposing this benchmark dataset is the need to have a problem whose complexity can be controlled. Thus, the proposed EESNN can be tested on various spatiotemporal noise levels. The efficiency of the proposed EESNN to adapt to the various noise levels is evaluated.

7.3.1 *Rotating Dot Problem*

The Rotating Dot dataset is a two-class synthetic spatiotemporal problem. The objective of this problem is to determine the moving direction of a dot. Two original patterns are created in a matrix with predefined dimension; the first

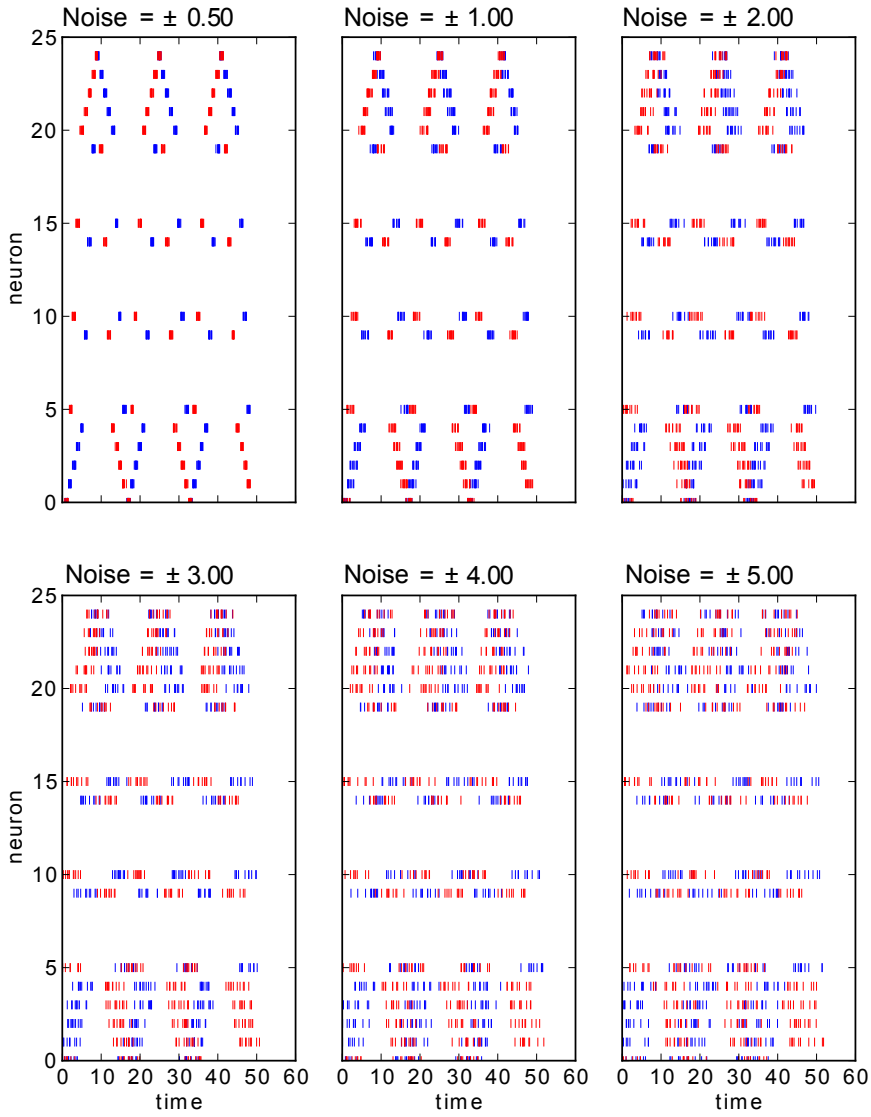

Figure 7.3: Rotating dot spatiotemporal synthetic dataset. Six datasets with different noise level are created from the original spike trains. The two colours represent the two classes.

dot rotates on clockwise direction and the second dot rotates in anticlockwise direction. Both dots use the same path when rotating. This makes the problem more interesting because at any individual time point or event, no decision can be made about whether the point turns clockwise or anticlockwise. Several or whole events have to be considered to determine the trajectories of the dot. In

this experiment, a matrix of 5×5 has been created with the dot completes two rotations for each direction.

The two original patterns have been encoded into spike trains using the population encoding method. A desired number of samples have been generated based on these spike trains. In order to control the difficulty of the problem, a uniformly distributed noise is added to the spike trains. The original data were jittered in a specific time interval. Figure 7.3 shows six datasets that have been generated each with 100 samples, 50 samples for each class. The colours indicate the two classes, one for each direction of the rotation. The problem has been sampled at 60 milliseconds simulation time. The first top diagram illustrates the spike trains with the lowest level of noise applied at 0.5 millisecond, thus the easiest problem. Increasing noise value indicates higher difficulty problems. For example, at the noise level of 5.00 milliseconds, the spike trains between two classes have been blended together and it is difficult to distinguish between the classes. The proposed method has been tested on all six problems and this is explained in the following section and also discussed in Chapter 8.

7.3.2 *Setup*

The proposed EESNN has been tested on the six Rotating Dot datasets. A rotation matrix of 5×5 with 48 frames was generated and every frame represented one millisecond of simulation time and was encoded with five receptive fields. The smaller number of receptive fields balances out the large number of rotation matrices and frames. In total, 6000 spike trains were generated and stored in the memory.

No optimiser has been embedded to the proposed framework to avoid further complexity. The high problem dimension would make the optimiser less effective, and more computation time and resources would be needed. Therefore, all parameters were manually adjusted based on the results obtained from parameter optimisation in the previous experiments. Following these considerations, Mod was set to 0.99 in this experiment. In terms of proportion factor value, the optimal values were found to be in the range from 0.65 to 0.95. However,

eight C values were tested in the range between 0.2 and 0.9 in this experiment. This was mainly because different problems might need different proportions of spike trains for better results. These eight experiments were ran with Sim set at 0.1 and 0.3 to evaluate how the similarity factor affects the problems. Lower Sim value resulted in lower merged number of output neurons, while higher value indicated otherwise. Ten-fold cross validation rule was followed in all sets of the experiments.

7.3.3 *Results*

The proposed EESNN was first tested problems were first tested on the Rotating Dot dataset with the Sim set to 0.3. The value was obtained from the parameter optimisation when DQiPSO was used in the previous experiments. From the optimal Sim range, the lowest value was selected to give some ability to the network to evolve and merge. In general, datasets with lower noise level achieved higher classification accuracy regardless of the C value. However, the value of C started to affect the accuracy when more difficult dataset were applied and higher accuracy was achieved when C was set to a higher value. On the other hand, the accuracy dropped severely when the smaller value for C was used for all datasets. For instance, at $C = 0.2$, the dataset with noise level 0.5 achieved 100% accuracy. The accuracy dropped down to 53% when EESNN was tested with a dataset with noise level 5.0. In contrast, a small decrease was recorded when a higher value for C was used. The overall results for this experiment are presented in Table 7.1.

The results for different settings of the proportion factor C for spatiotemporal problems show the best result can be obtained when C is set at around 0.6 to 0.9, confirming the range found by the optimiser on the spatial Hypercube and Two Spirals problems in previous experiments. In addition, this experiment also found that good results could also be gained when C was set to 0.4 for certain datasets. Generally, the accuracy for smaller C was lower than the accuracy achieved with a larger C value. Smaller C value showed that a lower number of spike trains were used to generate an output spike. As a result, lower accuracy was recorded when less information was supplied to the classifier.

Table 7.1: Overall result for Rotating Dot spatiotemporal problem. EESNN was tested with six datasets with different noise level. Mod was set to 0.99, C was set to values in the range of 0.2 till 0.9. Both values assigned to Sim (0.1 and 0.3) produced the same result.

Level of noise (msec)	Values of *C*							
	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.5	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00
1.0	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00	100.00% ± 0.00
2.0	95.00% ± 5.27	99.00% ± 3.16	100.00% ± 0.00	97.00% ± 4.83	100.00% ± 0.00	100.00% ± 0.00	99.00% ± 3.16	100.00% ± 0.00
3.0	73.00% ± 9.48	94.00% ± 8.43	99.00% ± 3.16	88.00% ± 10.32	100.00% ± 0.00	100.00% ± 0.00	99.00% ± 3.16	100.00% ± 0.00
4.0	62.00% ± 10.32	74.00% ± 15.05	95.00% ± 5.27	77.00% ± 9.48	96.00% ± 5.16	97.00% ± 4.83	83.00% ± 12.51	92.00% ± 6.32
5.0	53.00% ± 10.59	63.00% ± 8.23	62.00% ± 7.88	63.00% ± 10.59	85.00% ± 7.07	90.00% ± 11.54	78.00% ± 13.16	93.00% ± 8.23

This experiment was repeated for Sim = 0.1. Surprisingly, the results for all values of C produced the same accuracy as previous experiment with Sim = 0.3. The similarity value of 0.1 creates the same arrangement of output neurons as when it is set to 0.3. More output neurons are created when Sim value is low. This situation leads to the new sample to resemble with one of the trained output neuron. However, this also leads to a higher number of output neurons being created. For large problems with thousands or hundreds of thousands of input samples, more merged neurons lead to more effective classification. Although storage capacity is not an issue due to improvements in current hardware and storage capabilities, the method that uses fewer resources is preferable especially when dealing with online applications that require fast processing that uses minimal resources. Optimal parameters found in this experiment on the tested spatiotemporal problem will be considered in the next experiment where a new framework for spatiotemporal classification is proposed.

7.4 SUMMARY

This chapter has proposed an extended structure of ESNN for spatiotemporal problems along with a spatiotemporal data encoding method. This combination provides spike representation for the input patterns that are required for the classification process. The method accumulates all spatial components in the sequence of time into a memory. This method allows the classifier to easily classify the given problems. Its capability is demonstrated on a proposed benchmark dataset. Results show the proposed data encoding method is able to capture all necessary information leading to promising classification accuracy achieved by ESNN.

In the next chapter, a new version of the extended ESNN is proposed to classify spatiotemporal problems. This new framework employs the LSM for more complex reservoir construction. Both EESNN and the new extended ESNN proposed in Chapter 8 will be tested on a case study dataset for performance result comparison presented in Chapter 9.

Chapter 8

RESERVOIR-BASED ESNN FOR SPATIO-TEMPORAL PATTERN RECOGNITION

In this chapter, a novel reservoir-based ESNN (RESNN) framework utilising LSM is presented. Its suitability as a classification method is tested in the computer simulations. The goal of the study is to gain some insights into the working of the reservoir-based ESNN for classification. Its feasibility as a spatiotemporal classification method will also be evaluated in this chapter.

8.1 RESNN

Reservoir is an intermediate structure that maps an input to its high dimensional output after accumulating all the input information. A readout function is used to transform reservoir responses to the desirable output that can be used for decision making. This chapter explains how the reservoir is constructed from LIF neurons according to the preliminary study in Maass et al. (2002) and Schliebs, Nuntalid, and Kasabov (2010). The structure of LIF reservoir is studied in Section 2.3.

8.1.1 *The Reservoir*

The LIF neural model is based on an electrical circuit containing a capacitor with capacitance C and a resistor with a resistance R, where both C and R are assumed to be constant. The dynamics of a neuron i are then described by the following differential equation:

$$\tau_m \frac{du_i}{dt} = -u_i(t) + R\, I_i^{\text{syn}}(t) \tag{8.1}$$

The constant $\tau_m = RC$ is called the membrane time constant of the neuron. Whenever the membrane potential u_i crosses a threshold ϑ from below, the neuron fires a spike and its potential is reset to a reset potential u_r. The firing time $t_i^{(f)}$ of a neuron i is defined in Equation 8.2, as described in Gerstner and Kistler (2002b).

$$t_i^{(f)} : u_i(t^{(f)}) = \vartheta, f \in \{0, \ldots, n-1\} \tag{8.2}$$

where n is the number of spikes emitted by neuron i. The synaptic current I_i^{syn} of neuron i is modeled using an α-kernel:

$$I_i^{\text{syn}}(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)}) \tag{8.3}$$

where w_{ij} is the synaptic weight describing the strength of the connection between neuron i and its pre-synaptic neuron j. The α-kernel itself is defined as

$$\alpha(t) = e\, \tau_s^{-1}\, t\, e^{-t/\tau_s} \Theta(t) \tag{8.4}$$

where $\Theta(t)$ refers to the Heaviside function in Equation 8.5 and parameter τ_s is the synaptic time constant.

$$\Theta(s) = \begin{cases} 0 & \text{if } s < 0 \\ 1 & \text{if } s \geq 0 \end{cases} \tag{8.5}$$

8.1.2 *Spatiotemporal data encoding*

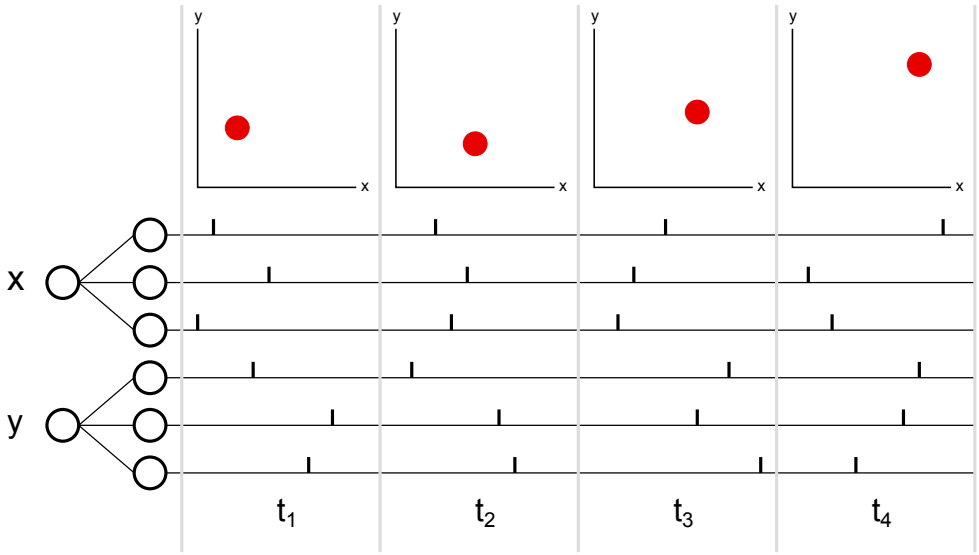

Figure 8.1: Spatiotemporal data encoding.

The first step in the framework is to encode spatiotemporal data into spike trains. Figure 8.1 illustrates the process. Each real-value of the data vector is transformed into a spike train using a population encoding. In this approach, a single input value is encoded into multiple neurons, each with a specific spike trains calculated using intersection of Gaussian function, as described in previous experiments. All frames or events in the data are encoded in the same way. As a result, a sequence of spike trains for all pre-synaptic input neurons is produced. Based on the time order, the whole series of spike trains will be injected into the reservoir. Spikes with the earliest time will be injected first followed by later ones.

8.1.3 *Framework*

Figure 8.2 shows the architecture of the proposed RESNN framework. There are four major components - spatiotemporal encoding, reservoir, liquid states as a reservoir output and the ESNN classifier.

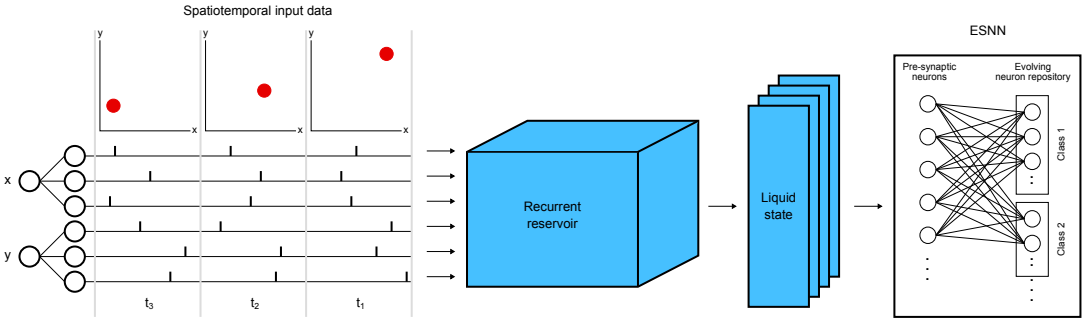

Figure 8.2: Architecture of the extended ESNN capable of processing spatio-temporal data. The coloured boxes indicate novel parts in the original ESNN architecture.

Each pre-synaptic input neuron is attached to a reservoir's input neuron. The generated spike trains are propagated into the reservoir. Since each pre-synaptic neuron has a series of spikes, the spikes are continuously being injected into the reservoir neuron. According to the LIF neuron principle, an output spike will fire and propagate a spike to the next connected neurons after it reaches its firing threshold. This process is repeated until all spike trains are completely injected.

An output spike generated by a neuron may activate other connected neurons. This activity can cause the reservoir to produce unique accumulated neuron responses. Different input spike trains from different input samples may produce different responses that could be used for discerning the sample according to its class. After the simulation is completed based on the pre-defined simulation time, the neuron responses are evaluated.

In order to perform the classification, readout functions are required to transform the reservoir responses into liquid states. Liquid states can be extracted at all time points. However, due to little available information and because some readouts require more information for the calculation, liquid states are normally calculated at certain time intervals. Different readouts produce different quality of liquid states, which affects the outcomes. The liquid states then are fed into the ESNN for classification into a desired class label. Algorithm 8 explains the proposed RESNN and the detailed descriptions are presented in Appendix F.

Algorithm 8 RESNN

```
construct reservoir with interconnected neurons using Equation 9.1
for all samples belonging to class l do
    for all time points do
        encode input samples into firing time using Equation 2.3
    end for
    store as spiketrains
end for
for all spiketrains do
    feed into reservoir
    calculate responses based on neuron spikes using Equation 8.2
    construct liquid states from reservoir responses
end for
apply liquid states into ESNN (Algorithm 1)
```

8.2 METHODS FOR RESERVOIR STATES REPRESENTATION

In order to understand the suitability of the RESNN classification method, the proposed framework was first implemented using simple synthetic spike trains. The objective of this experiment was to investigate the reservoir responses when spike trains were injected into the reservoir. The reservoir should be able to produce different responses between different output classes. Theoretically, higher dissimilarity means that better classification can be reached. Because the responses accumulate from the reservoir, this task is crucial for the classification phase.

8.2.1 *Dataset*

In this experiment, two random spike trains were generated to represent a synthetic two class problem. The original spike trains were jittered using a Gaussian function with the width of 1 millisecond where 50 samples were creates for each class. All spikes were uniformly distributed between 0 and 300 milliseconds, meaning that every sample contained the same number of spikes. The problem was simulated in 500 milliseconds simulation time. Figure 8.3 illustrates the generated spike trains.

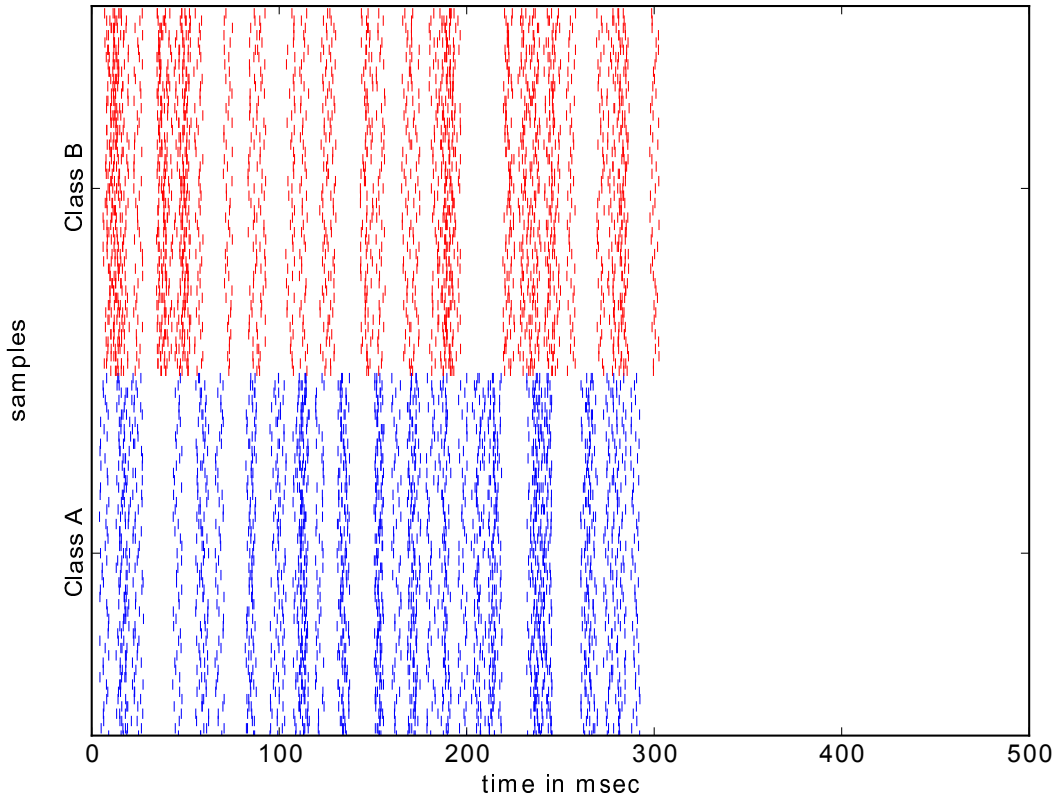

Figure 8.3: A simple two class synthetic dataset. The original data in each class were jittered to produce 50 samples.

8.2.2 *Setup*

In this experiment, the LSM reservoir was constructed with three dimensional network grid $4 \times 4 \times 4$, which is equivalent to 64 interconnected neurons. In the network, two neurons A and B were interconnected with a connection probability according to Equation 9.1

$$P(A, B) = C \times e^{\frac{-d(A,B)}{\lambda^2}} \tag{8.6}$$

where $d(A, B)$ denotes the Euclidean distance between two neurons and λ corresponds to the density of connections which was set to $\lambda = 2$. Higher or smaller values of λ represented higher or lower probability that a connection exists between the two neurons. The reservoir contained 80% excitatory (ex) neurons and 20% inhibitory (inh) neurons, which were randomly assigned

to neurons. Parameter C depends on the type of the neurons, $C_{ex-ex} = 0.3$, $C_{ex-inh} = 0.2$, $C_{inh-ex} = 0.5$ and $C_{inh-inh} = 0.1$. The connection weights have been randomly selected by a uniform distribution in the interval $[-8, 8]$nA. The neural parameter τ_m represents the decrease of the potential was set to 30 milliseconds, the firing threshold $\vartheta = 5$mV and reset value $u_r = 0$mV. Furthermore, a refractory period of 5 milliseconds and a synaptic transmission delay of 1 millisecond was used. Most of the parameters were directly adopted from a study by Grzyb, Chinellato, Wojcik, and Kaminski (2009).

For the ESNN classifier, parameter Mod was set to 0.99, C = 0.6 and Sim = 0.1. These values were based on the optimal range found in the previous experiment.

8.2.3 *Results Analysis*

Figure 8.4 shows the responses after the input spike trains have been injected into the constructed reservoir. The top two diagrams represent the averaged responses for 50 samples from each class, A and B respectively. The figure clearly shows that some of the neurons were still activated after all input spikes had been completely fed into the reservoir in 300 milliseconds. However, the difference between classes is hard to notice due to the density of the responses. In order to investigate the quality of both responses, in terms of distinguishability between classes, these two average responses were subtracted. The diagram at the bottom represents the whole responses. The blank area means no response while the colours correspond to the value after the subtraction operation. The figure shows that the responses from each of the two classes are dissimilar; this may lead to easier classification process later.

In order to perform classification, the state of the liquid at a given time t has to be read out from the reservoir. The way the liquid state is defined is critical for the proper working of the method. In this experiment, three different types of readouts have been investigated - cluster, frequency and analog readouts.

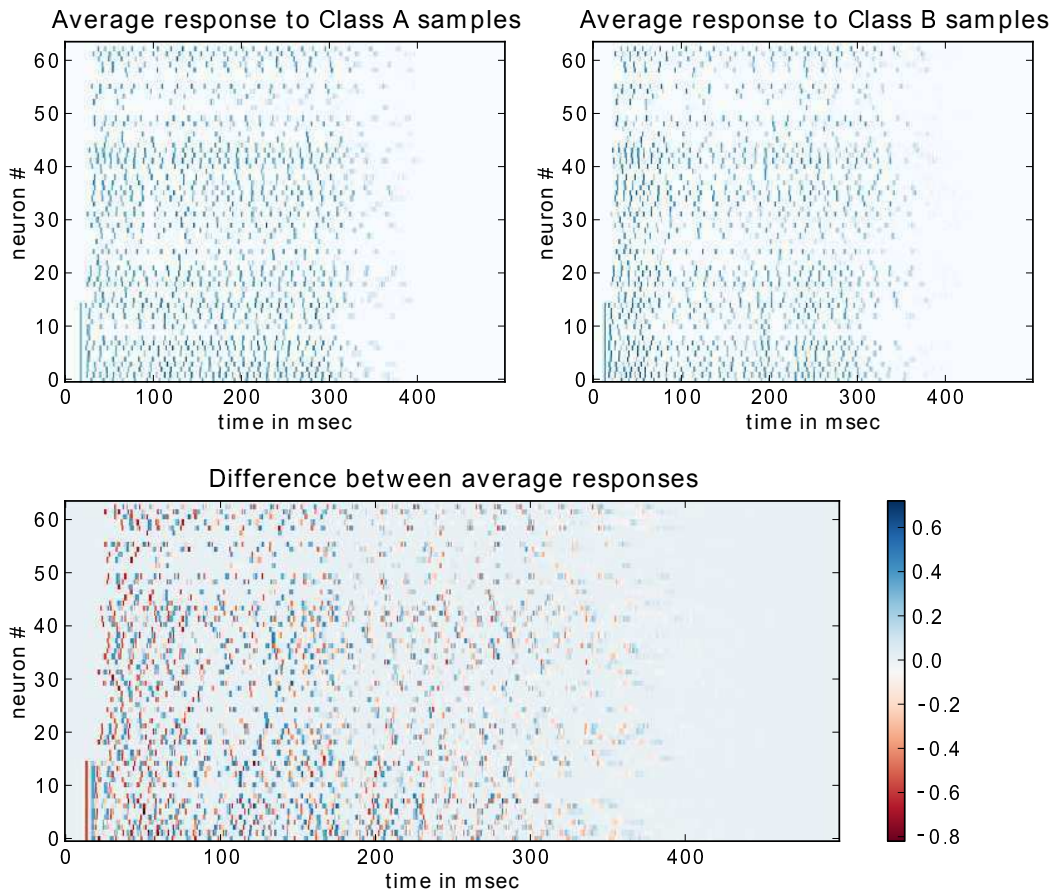

Figure 8.4: Reservoir response on synthetic dataset.

Cluster Readout

The first type of readout is called cluster readout. In this method, neurons in the reservoir are grouped into clusters and then the activity of the neurons in the clusters is determined. The population activity is defined as the ratio of neurons being active in a given time interval $[t - \Delta_c t, t]$. This follows the same principle as explained in Section 2.2.1 and also described in Gerstner and Kistler (2002b). In this experiment, 16 clusters were collected in a time window of $\Delta_c t = 10$ milliseconds. Similar readouts have also been employed in previous related studies such as in Norton and Ventura (2010).

Figure 8.5 shows the continuous accuracy when the readout was fed into the ESNN classifier. The top diagrams show the cluster readout constructed at three different time points. The readouts are sorted according to the sample class. Based on the readout setup, every sample produced a new format of

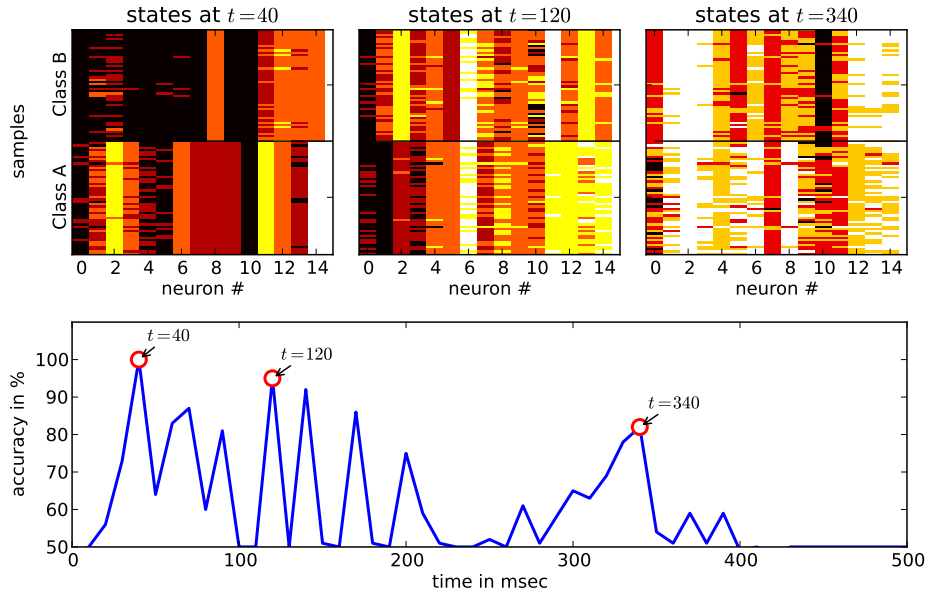

Figure 8.5: Classification accuracy from cluster readout. Three time points are selected and the readout from each point is shown on the top. At time point $t = 160$, the highest accuracy is obtained from the readout that is more distinguishable between classes compared to other time points.

data. This new data was used in the classification process. Therefore, the construction of the readout is crucial to the samples being distinguishable between classes. The same condition applies to the next two readouts.

In this experiment, although the input spike trains were spotted in only 300 milliseconds, neurons were still accumulating the spikes and activating after all input spikes trains were fed into the reservoir. These responses contained information that could be used for classification, as shown in the bottom diagram. Some accuracy can be obtained after 300 milliseconds input spike trains. This condition is referred to as the fading memory effect. Three random points of accuracy have been selected in order to visualise the readout or liquid state at t = 40, 120 and 340. It can be seen that higher accuracy requires a better distinguishable state between output classes. In this example, liquid state at t = 40 gives the highest accuracy of 100%. This corresponds to better liquid state compared to the state at t = 120 with accuracy of 95%. Surprisingly, the fading memory effect at time t = 340 still produced good accuracy at 82%.

Frequency Readout

The second readout is principally very similar to the first one. In the interval $[t-\Delta_f t, t]$, the firing frequency of all neurons in the reservoir were determined. According to the reservoir setup, this frequency readout produces a single vector with 64 continuous elements; each element refers to a single neuron over collected time window of $\Delta_f t$. In this experiment, a time window of $\Delta_f t = 30$ was used.

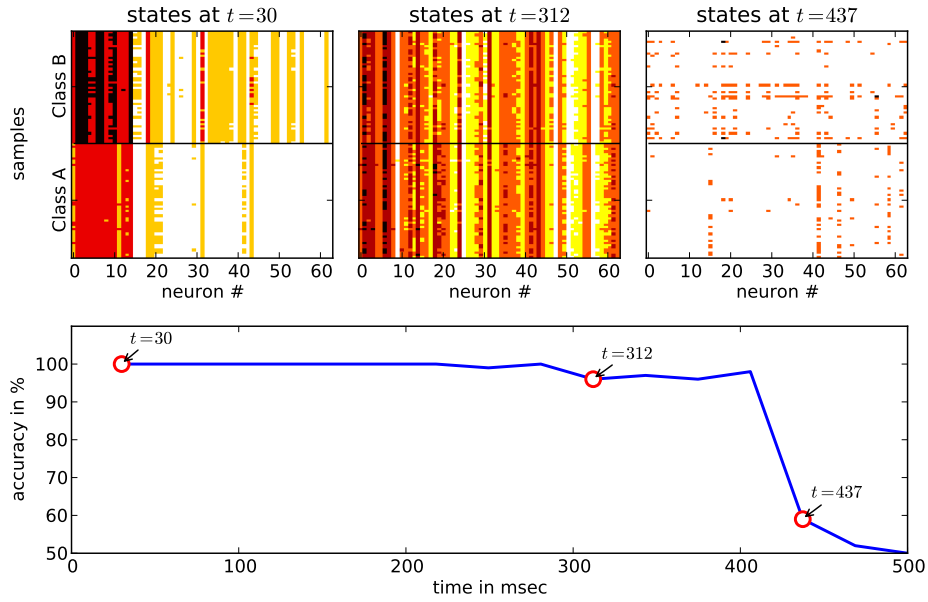

Figure 8.6: Classification accuracy from frequency readout. A more stable accuracy is achieved over time because a larger time window is required in this readout. Three time points were selected to illustrate the readout. Readouts at $t = 30$ and $t = 312$ are more distinguishable between classes and therefore provide a high accuracy. However, at $t = 437$, the fading memory effect took place and reduced the accuracy.

Similar to the previous readout, three random accuracy points were selected to investigate the quality of the reservoir readout, as shown in Figure 8.6. Low density appears at t = 30 because of the low number of responses, as can be seen in Figure 8.4. However, because the response patterns for different classes at this time are different, accuracy is high at 100%. The high density readout

at t = 312 represents many responses available for the readout. However, the accuracy slightly decreases because most of the information from the readout is resembled between two classes. At t = 437, the sharp decrease of accuracy is due to the fading memory effect where not many responses are available at the given time point. Thus, this affects the construction of the readout and the accuracy.

Analog Readout

Finally, in the analog readout, every spike is convolved by an alpha kernel function derived from Equation 8.4. A convolved spike train $\tilde{s}(t)$ is then given as

$$\begin{aligned} \tilde{s}(t) &= \sum_{t^f} \kappa(t - t^f) \\ &= \sum_{t^f} e\, \tau^{-1}\, (t - t^f)\, e^{-\frac{(t-t^f)}{\tau}} \Theta(t - t^f) \end{aligned} \tag{8.7}$$

where $\Theta(t)$ refers to the Heaviside function (Equation 8.5) and $\tau \in \mathbb{R}$ is a real-value time constant.

The responses are sampled using a time step of $\Delta_a t = 10$ milliseconds resulting in 50 time series. Similar readout has been used for example in Schrauwen, D'Haene, Verstraeten, and Campenhout (2008) for a speech recognition problem.

The accuracy of this readout is consistently high most of the time as can be seen in Figure 8.7. However, when there are fewer responses after all input spikes have been fed into the reservoir, the accuracy starts to decrease which is evident in all readouts. When the accuracy was measured, it was 100% at almost all time points, including at t = 30 and t = 150. The liquid states for both time points show different patterns for the two classes. At t = 410, the difference between the two classes is hardly noticeable due to the small number of activities. However, when the state was applied to the ESNN, a good classification result with accuracy of 89% was achieved. This is because the decay of alpha kernel function still provides some information even after several milliseconds of neuron activity. Figure 8.4 shows that after 400 milliseconds sim-

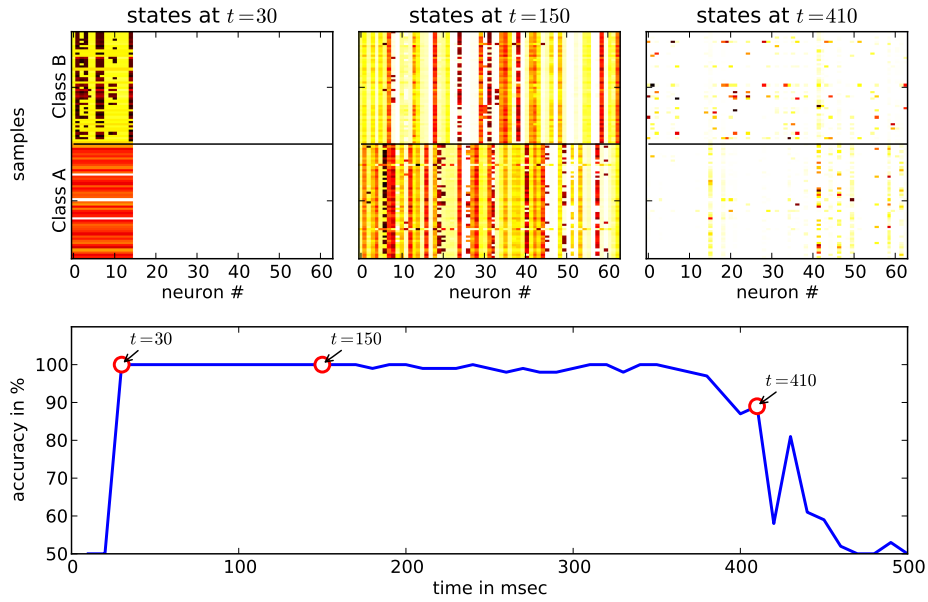

Figure 8.7: Classification accuracy from analog readout. Readouts from the selected time points are displayed at the top of the diagram. Accuracyis more stable even when few responses are left as shown at $t = 410$.

ulation time, very few responses were recorded. However, some information can still be extracted when the kernel is applied, as shown in Figure 8.7.

8.2.4 *Discussion*

Overall, three readouts came out with the highest accuracy of 100%. Among all, cluster readout results fluctuated significantly. At certain time points, cluster readout produced a high accuracy, but a few milliseconds later, a very poor one was recorded. At the time when when the input spike trains were still being injected into the reservoir (before 300 milliseconds), the other two readouts were able to generate very high accuracy. However, the cluster readout failed to utilise the responses, which consequently produced poor outcomes.

On the other hand, even though the frequency readout gave a good accuracy almost at all time points, the larger sampling time caused some delay in the accuracy measure. Figure 8.6 clearly shows that the first accuracy measurement could only be made only after 30 milliseconds. As a result, this readout would be unsuitable for rapid, online and real-time processing.

Similarly to the frequency readout, the analog readout also gave consistently high accuracy in almost the entire simulation time. But the main advantage of analog readout compared to the frequency readout is the small sampling time at only 10 milliseconds. Sampling time is important as it ensures that there is some neuron activity to be measured. The fading memory effect on this readout is also small, and therefore this readout is able to give good accuracy due to information available from the alpha kernel function.

Based on the findings in this experiment, the analog readout will be used for further analysis in the next section.

8.3 APPLICATION ON SYNTHETIC SPATIOTEMPORAL DATASETS

In order to further evaluate the performance of the proposed method, the RESNN framework was applied to the Rotating Dot dataset that is proposed in Chapter 7. All parameters in this experiment were derived from the settings in the previous experiment as described in Section 8.2.2. The LSM reservoir was adjusted to accumulate the given problem and gridded to $5 \times 5 \times 5$. The reservoir utilised 125 neurons with 80% excitatory (ex) neurons and 20% inhibitory (inh) neurons. All other reservoir parameters were left unchanged. The simulation time period was set to at 200 milliseconds. Similar to the previous ESNN setup, Mod, C and Sim were set to 0.99, 0.6 and 0.1, respectively.

8.3.1 *Performance Analysis*

In this experiment, the reservoir responses were only mapped into liquid states using analog readout. The experiment shows that this readout is more stable and gives better outcomes compared to other readouts.

Figure 8.8 shows the example of average Rotating Dot responses for the two classes, clockwise and anticlockwise. Similarly to the previous experiment, the subtraction operation was performed on the both responses and the results of the operation is shown in the bottom diagram. For the first 20 milliseconds, half of the rotation can be clearly seen by the curve shaped responses. After other neurons were excited, more activity was recorded. The difference

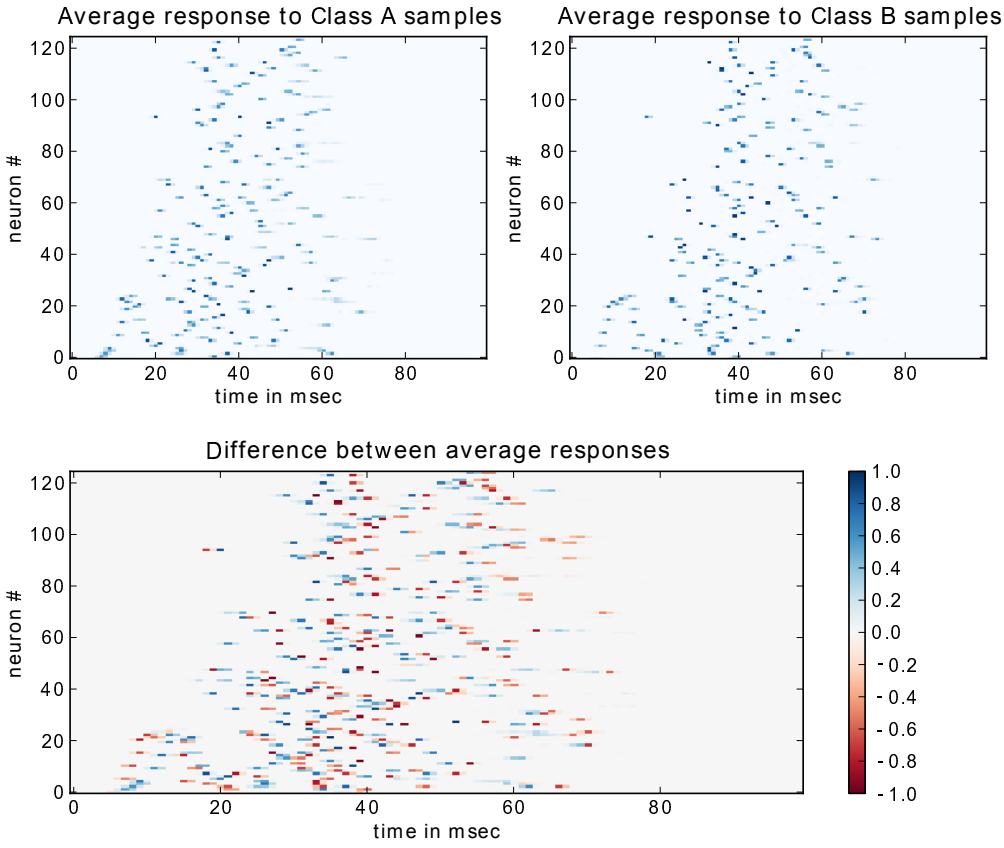

Figure 8.8: Rotating dot responses.

between output classes can be clearly identified due to data with noise of 1.0 millisecond was used.

In analog readout, the decay of the Gaussian signal that is represented by the parameter τ is very important. In this experiment, the impact of τ and the several levels of problem difficulty to the accuracy is studied. Three values for the τ parameter were selected to be studied: 2.0, 5.0 and 10.0. Results are presented in Figure 8.9. The three x-axis columns represent the different values for τ which controls the alpha kernel decay, while six rows in y-axis represent the noise level ϵ.

The first three top diagrams show the results for dataset with noise level of 0.5, which is the easiest data. It clearly shows that τ does not affect the results. the experiment gave consistent high accuracy over the simulation time for all values of τ. Results at noise level 1.0 millisecond also show the same accuracy pattern and τ has no effect on the results. The next two rows of diagrams

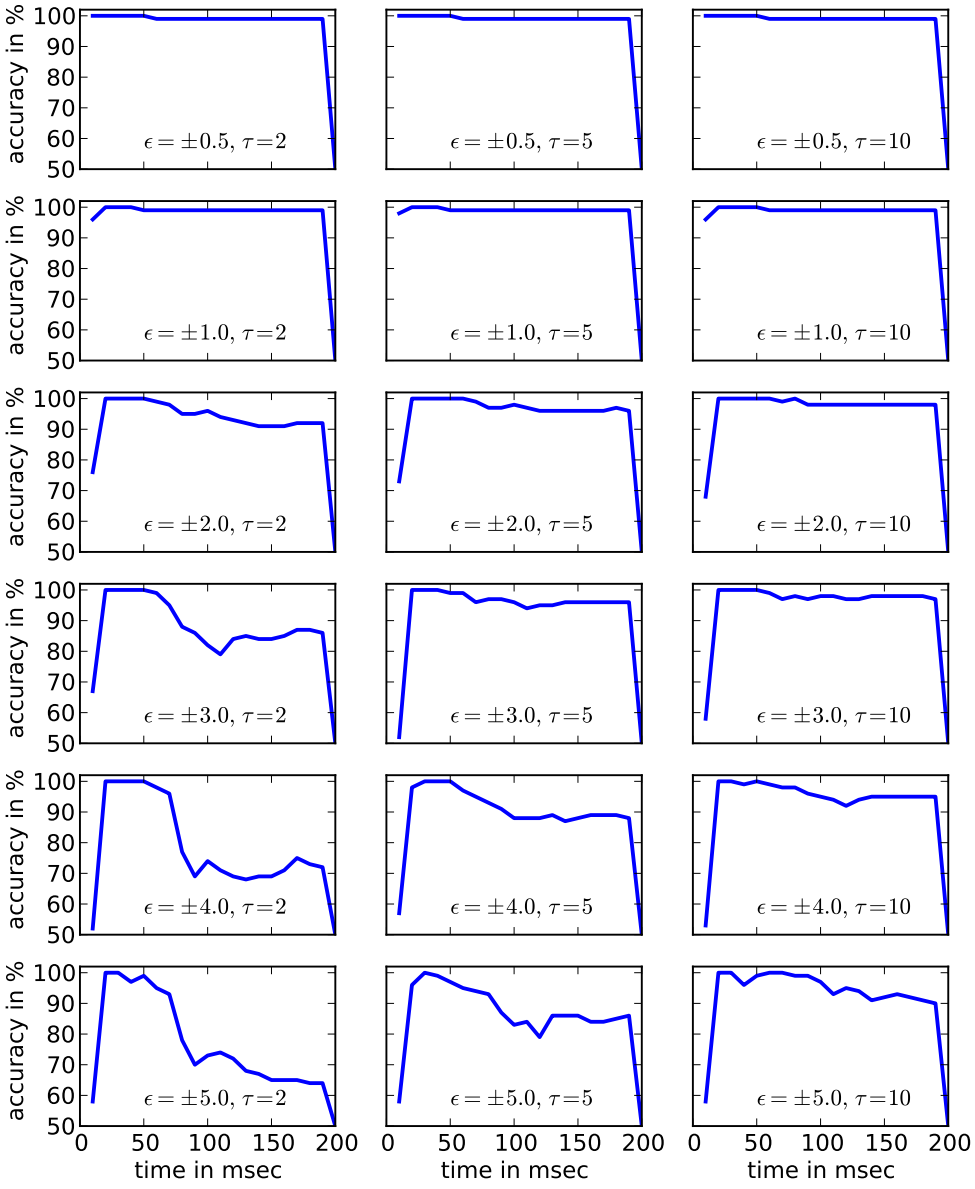

Figure 8.9: Analog readout accuracy over simulation time. The problem is presented with different level of difficulty (ϵ)and τ value for kernel to evaluate the correlation between these two aspect.

represent the problem with difficulty of 2.0 milliseconds and 3.0 milliseconds, where the smallest τ gave a lower accuracy over time compared to higher τ values. For the more difficult problems (ϵ is 4.0 and 5.0), the highest τ provides a better accuracy.

Interestingly, good results were obtained for simulation time between 30 and 60 milliseconds in all experiments. This is when the input spikes were injected

into the reservoir. It can be concluded that the responses from the reservoir are sufficient to classify the data after partial input was fed into the reservoir. The Rotating Dot dataset consisted of several rotations of the dot. From Figure 7.3, it is apparent that the classifier should be able to identify the movement of the dot, i.e. to distinguish between the two classes, clockwise and anticlockwise in the first rotation. The accuracy starts to drop when the process of feeding input spikes is completed. However, smaller decrement can be retained when higher value for τ are used. Thus, it produces higher accuracy compared to lower values of τ.

It can be concluded that τ has no effect on accuracy in simple problems. However, when the problems have higher difficulity, τ starts to play an important role. The kernel function continuously provides some information depending on the τ value. Smaller values mean that the decay is very short and less information is available in the next several milliseconds. In contrast, long time of decay provides more information for the next few milliseconds when the value for τ is larger and might cause the liquid state poured with massive information. Therefore, a careful selection of τ value is necessary for optimal outcomes.

8.3.2 *Comparison of results obtained with EESNN*

The overall results for the EESNN and the RESNN methods are presented in Figure 7.1 and in Figure 8.9 respectively. A discussion on these results and the related findings are presented in Chapter 7 and Chapter 8. While EESNN only provides the final accuracy after the learning process is completed, RESNN provides the accuracy continuously during the simulation time. The explanation in this section refers to both figures.

Classification accuracy of 100% is obtained when the EESNN method is applied to the Rotating Dot dataset with noise level of 0.5 and 1.0. The RESNN method achieved 99% to 100% accuracy at almost all time points except for the first and last few milliseconds when no responses were available for classification. In fact, this situation occurred in all experiments with RESNN. Generally, both methods can achieve high accuracy for simple datasets with low

noise level at any C value for EESNN and τ value for RESNN. When EESNN was tested with the dataset with noise level 2.0, the smallest C value gave the lowest accuracy compared to others. Similarly to RESNN, results recorded from the smallest τ value shows that the accuracy at most of the time points is low compared to higher τ value.

When EESNN is tested with a more difficult dataset with noise levels of 3.0, 4.0 and 5.0, a larger C value gives a good accuracy most of the time. However, there is also a certain time when a high accuracy can be obtained and when C is set to 4.0 as shown in the results when a dataset with noise level of 3.0 and 4.0 is tested. Therefore, tuning the C value in EESNN for difficult problems has to be done carefully since not only the high value of C, but some lower values could also give a good accuracy. Larger τ value provides better accuracy compared to the smaller τ value as shown in the results when RESNN is tested with a dataset with noise level of 3.0, 4.0 and 5.0. Only a few settings achieved 100% accuracy. It can be concluded that the RESNN method always needs to use larger τ value when dealing with the more difficult problems. It can be concluded that larger τ value always provide high accuracy. In contrast, larger C values do not necessarily mean that EESNN can achieve high accuracy results with difficult problems, since lower values may give a good result as well. This makes finding the optimal C value more difficult in EESNN.

The advantage of EESNN method over the RESNN method is that it is faster as it has the simple spike trains memory construction that does not require any internal computation. On the other hand, RESNN has a more complex reservoir structure with the integrated recurrent network and LIF neurons. The method also requires more computational time and resources to perform classification. Nevertheless, the advantage of the RESNN method is that it has the capability to do classification at any selected time point. The fading memory effect gives the network an extra ability to classify the given problem even after all spike trains are fed to the reservoir. Interestingly, all experiments with RESNN deliver 100% classification accuracy at certain time points. It can be concluded that each of the two methods has its strengths and weaknesses that need to be further explored.

8.4 SUMMARY

This study has proposed an extension of the ESNN architecture, called RESNN, that enables the method to process spatiotemporal data. Using a reservoir computing approach, a spatiotemporal signal is projected into a single high-dimensional pattern that can be learned by the ESNN training algorithm. Configuring the reservoir is not an easy task. However, once the reservoir is configured properly, ESNN can be an efficient classifier of liquid states extracted from the reservoir.

This chapter has also discussed three types of the readouts. The readout construction is a crucial task that can affect the classification accuracy. The study found that the analog readout has some advantages and has been chosen for the next experiment. In the analog readout experiment, several scenarios have been set up to investigate the optimal value for τ - the analog alpha kernel that provides information to the classifier. Results show that a higher value is required for difficult problems, while any values can solve simple problems.

The next chapter will discuss the implementation of RESNN together with EESNN in a case study. Results of both proposed methods will be compared and analysed.

Chapter 9

A CASE STUDY ON A SPATIOTEMPORAL PROBLEM - SIGN LANGUAGE GESTURE RECOGNITION

In order to investigate the performance of both novel EESNN and RESNN methods for classification of spatiotemporal data, a real-world spatiotemporal dataset is studied in this chapter. In the next few sections, the LIBRAS sign language dataset is explained. The experimental setup and the obtained results are discussed in the last part of this chapter.

9.1 THE DATASET

LIBRAS is the acronym for LIngua BRAsileira de Sinais, which is the official Brazilian sign language. The LIBRAS dataset contains data describing 15 hand movements (signs) that can be learned and classified by the two studied methods. The movements data are obtained from recorded videos of four different people performing the movements in two sessions. In total 360 videos have been recorded, where each video recorded one movement that lasts for about seven seconds. Forty five frames have then been extracted from the videos according to uniform distribution. In each frame, the centroid pixels of the hand are used to determine the movements. All samples have been organised in 10 sub-datasets, each representing a different classification scenario. Datasets 1 to 7 contain all samples while Datasets 8, 9 and 10 contain selected samples. More comprehensive details about the datasets can be found in Dias, Madeo, Rocha, Bíscaro, and Peres (2009). The data can be obtained from the UCI machine learning repository [1].

1 Available at http://www.ics.uci.edu/ mlearn/

In this experiment, Dataset 10 has been chosen. It contains the hand movements recorded from three different people. The objective of using this dataset is to train and test the proposed model for user-independent movement classification and recognition, where the hand movement of one or several persons can be used to train the system to identify the same movements of other people. This dataset consists of 270 videos with 18 samples for each of the 15 classes. An illustration of the dataset is given in Figure 9.1. The diagrams show a single sample of each class.

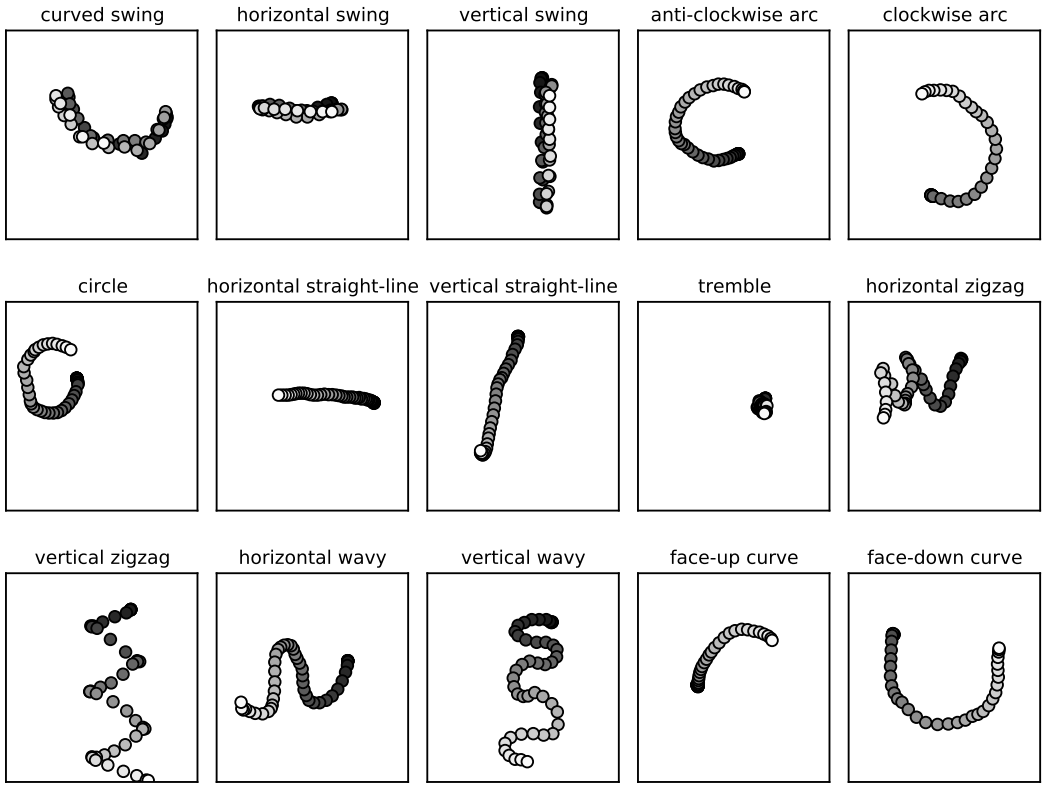

Figure 9.1: The LIBRAS data set. A single sample for each of the 15 classes is shown. The colours indicate the time frame of a given data point (black/white corresponds to earlier/later time points).

9.2 EXPERIMENT SETUP

The LIBRAS dataset is used to evaluate and compare the performance of ESNN and RESNN. The results will be also compared with the results obtained from MLP when used on the same LIBRAS dataset. All three algorithms have different sets of parameters that need to be defined as shown in the three subsections below.

9.2.1 *EESNN*

The three ESNN parameters, Mod, C and Sim, were set to 0.99, 0.65 and 0.05, respectively based on considerations derived from previous experiments in this study and also in Schliebs, Defoin-Platel, and Kasabov (2010). Parameter β, which controlled the Gaussian width was set to 1.5. The two LIBRAS spatial variables, x and y, represent the coordinates of each frame centroid. All 45 frames formed an input pattern related to one of the 15 classes (signs or movements). The input data range was normalised and set to values in the interval [-0.5, 1.5]. Each of the spatiotemporal input patterns of 90 spatiotemporal variables was encoded using population encoding. Every variable was encoded using 20 Gaussian receptive fields. The EESNN was trained and tested in a 9-fold cross-validation mode.

9.2.2 *RESNN*

As described in Chapter 2, population encoding is used to transform the input data into spike trains. This method is characterised by the number of receptive fields used for encoding along with the width β of the Gaussian receptive fields. After some initial experiments, 30 receptive fields were selected and a width $\beta = 1.5$. More details of the method can be found in Bohte et al. (2002).

In this experiment, a liquid with a small-world inter-connectivity pattern was constructed as described in Maass et al. (2002). A recurrent SNN was generated by aligning 100 neurons in a three-dimensional grid of size $4 \times 5 \times 5$.

Two neurons A and B in this grid were connected with a connection probability of:

$$P(A,B) = C \times e^{\frac{-d(A,B)}{\lambda^2}} \tag{9.1}$$

where $d(A,B)$ denotes the Euclidean distance between two neurons and λ corresponds to the density of connections which was set to $\lambda = 2$ in all simulations. Parameter C depends on the type of the neurons. The neurons were discriminated into excitatory (ex) and inhibitory (inh) neural types resulting in the following parameters for C: $C_{ex-ex} = 0.3$, $C_{ex-inh} = 0.2$, $C_{inh-ex} = 0.5$ and $C_{inh-inh} = 0.1$. The network contained 80% excitatory and 20% inhibitory neurons. The connections weights were randomly selected by a uniform distribution and scaled in the interval $[-8, 8]$nA. The neural parameters were set to $\tau_m = 30$ms, $\vartheta = 5$mV, $u_r = 0$mV. Furthermore, a refractory period of 5ms and a synaptic transmission delay of 1ms was used. Using this configuration, the recorded liquid states did not exhibit the undesired behaviour of over-stratification and pathological synchrony - effects that are common for randomly generated liquids (Norton & Ventura, 2006). Similarly to the experiments conducted in Chapter 8, three reservoir readouts were evaluated, namely the cluster, frequency and analog readouts.

9.2.3 *MLP*

The results were compared with results obtained from an experiment with a traditional Time Delay MLP, trained and tested in the same way. From the preliminary experiments with some parameter tuning to find the best combinations, the optimal number of hidden nodes in the MLP was found to be 45, learning rate 0.3. The original unprocessed LIBRAS dataset was used and the learning in MLP was performed in 500 iterations. Similarly to EESNN and RESNN, 9-fold cross validation was performed to the dataset.

9.3 PERFORMANCE ANALYSIS

Both algorithms are evaluated to find out how well they can capture the spatiotemporal data and transformed it into information that the classifier is able to process. The overall accuracy results are also compared.

9.3.1 *EESNN*

Since in spatiotemporal problems both the spatial and temporal components are needed for decision making, the entire dataset was translated into spike trains and fed into a spike train memory, as mentioned in Chapter 7. The entire spike trains from the memory were then fed into ESNN for classification. The average training accuracy of the EESNN in this experiment was 99.35% ± 0.30, while for the testing, accuracy of 88.15% ± 6.26 was achieved.

Results show that although EESNN is a relatively simple approach, it can perform very well, considering that this is a 15-class problem. In addition to this, every sample that represents a movement within a class sometimes contains incomplete information to represent the movement. For instance in the circle movement, each person produced six samples in a movement class. However, not all samples made a complete circle movement. This problem is more complicated since Dataset 10 contained movements performed by three persons. Thus, there are a lot of movement varieties although they are in the same movement class. The proposed EESNN demonstrated that it can be applied not only on a user dependent dataset, which means recognising the movements of only one person, but also on user independent dataset that contains movements of more than one person. This is because the extended first layer of the EESNN captures the entire information in the first phase of the operation, so that the complex patterns are better classified in the second layer.

9.3.2 *RESNN*

The LIBRAS dataset was encoded into spike trains and then fed into the reservoir. Different samples produced different spike trains which generated dif-

ferent sets of reservoir responses. Figure 9.2 shows the responses from four different samples, each corresponding to a different class. Responses from the samples were extracted to form the readout for classification.

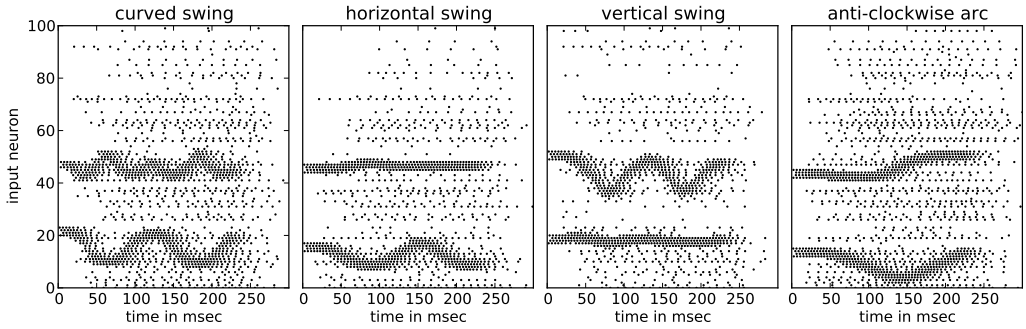

Figure 9.2: Raster plot of some typical neural responses recorded from a reservoir of 100 neurons. Each diagram shows the response when stimulated by samples belonging to different classes. It can clearly be seen that different response patterns were recorded for different classes.

All readouts extracted at a given time were fed to the standard ESNN for classification. Based on preliminary experiments, some initial ESNN parameters were chosen. The modulation factor was set to $Mod = 0.99$, the proportion factor $C = 0.46$ and the similarity threshold $Sim = 0.01$. Using this setup, the extracted liquid states over all possible readout times were classified.

The evolution of accuracy over time for each of the three readout methods is presented in Figure 9.3. Clearly, the cluster readout (a) is the least suitable readout among the three methods. The best accuracy found is 60.37% for the readout extracted at time 40ms, the marked time point in the diagram The readouts extracted at time 40ms are presented in the bottom diagram (a). Each row in this diagram presents the readout vector of one of the 270 samples, the colour indicating the real value of the elements in that vector. Darker colour represents higher readout value, while lighter colour represents lower value. The samples are ordered in a way that allows a visual discrimination of the 15 classes. The first 18 rows belong to class 1 (curved swing), the next 18 rows to class 2 (horizontal swing) and so on. Given the extracted readout vector, it is possible to even visually distinguish between certain classes of

samples. However, there are also some significant similarities between classes of readout vectors, which clearly has a negative impact on the classification accuracy.

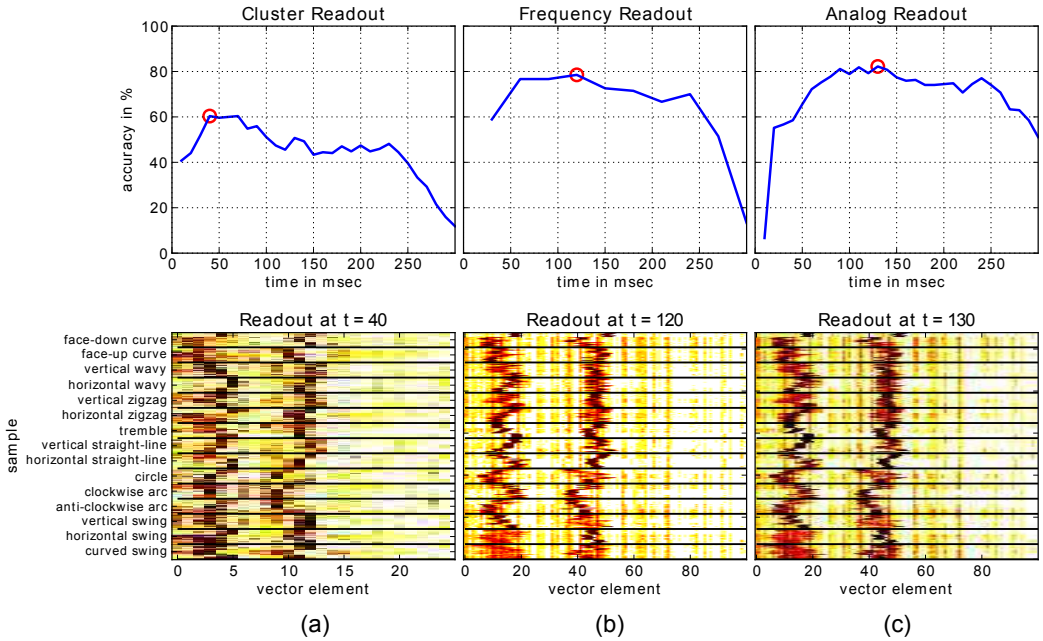

Figure 9.3: Classification accuracy of ESNN for three readouts extracted at different times during the simulation of the reservoir (top row of diagrams). The best accuracy obtained is marked with a small (red) circle. For the marked time points, the readout of all 270 samples of the data are shown (bottom row).

The situation improves when the frequency readout is used resulting in a maximum classification accuracy of 78.51% for the readout vector extracted at time 120ms, as shown in Figure 9.3(b). It clearly shows that the readout vectors are much better discriminationed into output classes as shown in bottom diagram (b): The intra-class distance between samples belonging to the same class is small, but inter-class distance between samples of other classes is large. However, the best accuracy is achieved using the analog readout extracted at time 130ms as shown in Figure 9.3(c). Patterns of different classes are clearly distinguishable in the readout vectors resulting in a good classification accuracy of 82.22%.

Off-line parameter and feature optimisation of RESNN

The discussion above already demonstrates that many parameters of the RESNN need to be selected optimally in order to achive satisfactory results. This is a difficult task since the number of these parameters is high and also that some of them are correlated to each other. To further improve the classification accuracy of the analog readout vectors, DQiPSO proposed in Chapter 5 is used. The parameters of the ESNN classifier are optimised along with the input features (the vector elements that represent the state of the reservoir). The readout vectors were extracted at time 130ms, since this time point had reported the most promising classification accuracy in the experiment shown in Figure 9.3(c). In DQiPSO, 20 particles were used, consisting of eight update, three filter, three random, three embed-in and three embed-out particles. Parameters c_1 and c_2 which control the exploration corresponding to the $pbest$ and the $gbest$ respectively, were both set to 0.05 for probability update and 1.2 for real value update. The inertia weight was set to w = 2.0. Eighteen-fold cross validations were used and results were averaged in 500 iterations in order to estimate the classification accuracy of the model.

Figure 9.4 presents the evolution of the selected features during the optimisation process. The colour of a point in this diagram reflects how often a specific feature was selected at a certain generation. The lighter the color, the more often the corresponding feature was selected at the given generation. It can clearly be seen that a large number of features have been discarded during the evolutionary process (the darker colour). The pattern of relevant features matches the elements of the readout vector that have larger values, *cf.* the dark points in Figure 9.3 and compared to the selected features in Figure 9.4.

The evolution of accuracy obtained from the global best particle during the PSO optimisation process is presented in Figure 9.5. The optimisation clearly improves the classification abilities of ESNN. After the DQiPSO optimisation, accuracy of 88.59% $(\pm 2.34\%)$ is achieved.

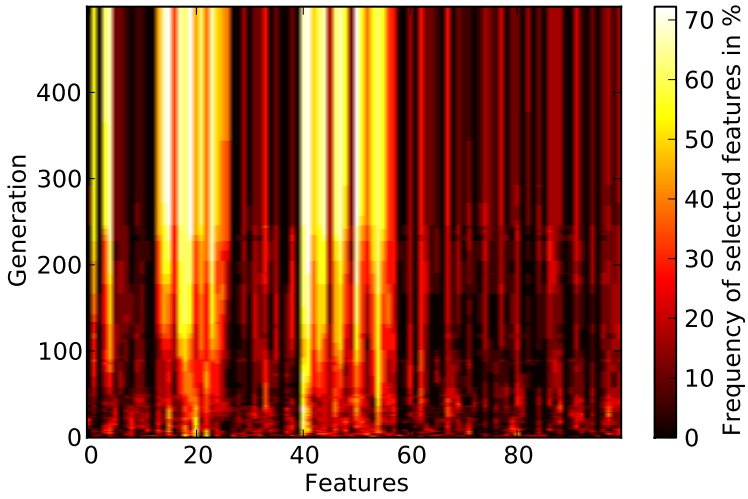

Figure 9.4: Evolution of feature subsets. The objective is to obtain input vectors that contain the most information. Compared with Figure 9.3, this figure shows the feature selection process is able to select vectors in the range of 15 to 25 milliseconds and 40 to 55 milliseconds which contain most information. Other vectors that hold less information are also occasionally being selected during the learning process.

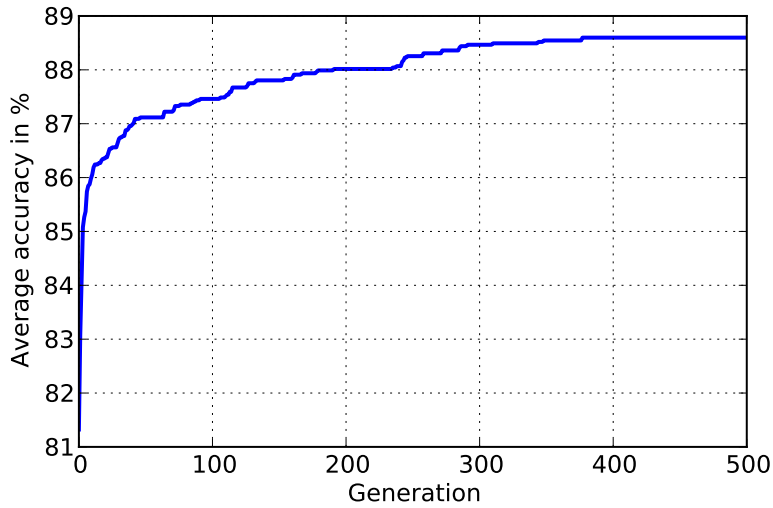

Figure 9.5: Evolution of the accuracy based on the global best solution during the optimisation with DQiPSO.

9.3.3 *Overall Comparison*

The test accuracy of an MLP under the same conditions of training and testing As those used for EESNN and RESNN is found to be 82.96% $(\pm 5.39\%)$. In

comparison, EESNN achieves 88.15% ± 6.26, while RESNN obtains 88.59% $(\pm 2.34\%)$. Although it is clear that both proposed methods performed better than MLP, there are some issues that need to be addressed.

In terms of computational speed, it is clear that EESNN is faster than RESNN because of the advantages of one-pass learning that is fully utilised in EESNN. However, in order to obtain the best results, several adjustments to the parameters have to be made manually. This is because the EESNN is not embedded with any optimiser. The main objective of EESNN is to propose a method to classify spatiotemporal problems faster. The method is intended to be used for online data processing that requires faster computation capability.

In contrast, RESNN is more complicated and slower than EESNN. However, the main advantages of this methods is the ability to classify at any time even while the input trains are still being fed into the reservoir. As shown in Figure 9.2, although reservoir responses to the LIBRAS input spikes are within 300 milliseconds, the RESNN method is able to start classifying the problems immediately from when the data are fed into the reservoir (virtually at 1 millisecond as shown in Figure 9.3). The classification accuracy at the beginning is low because the system is supplied with very little information. As soon as the information is sufficient, the method is able to give good results without waiting for all spike trains to be fed into the reservoir in 300 milliseconds. The fading memory effect is also another advantage of RESNN. The reservoir is able to produce some responses that can be used to classify even after all spike trains have been completely fed into the reservoir, as shown for the last 30 milliseconds in Figure 9.3. Interconnected recurrent neurons in the reservoir keep activating other neurons when they receive enough spikes to generate the responses and the fading memory effect.

Generally, the advantages of the proposed methods when compared to the MLP and other traditional NNs are:

1. Fast, one-pass, incremental learning, rather than multiple and batch learning, i.e. hundreds and thousands of iterations for MLP.

2. Evolvability - both models can be incrementally trained on new data and new classes without the need to be retrained on old data.

On the other hand, setting the parameters for the reservoir is a complex process. For the offline classification, the DQiPSO optimiser has been used for feature and classifier optimisation. This is the first attempt to apply this method to real spatiotemporal problems and some improvements could be made to solve this issue. These improvements are discussed in the next chapter.

9.4 SUMMARY

This chapter describes and discusses how two proposed methods, EESNN and RESNN are tested on a LIBRAS sign language dataset. The aim is to demonstrate how these two methods can be used for a real-world spatiotemporal classification problem. The movement recognition is crucial when developing a system that will be used by a disabled person. For instance, such a system can be built into a computer interface that enables deaf people to interact with computers. The result from this experiment shows that the accuracy of identification is satisfactory, even though the movement has been recorded from three different actors. Every actor has different hand movement even for the same sign class. In addition, the dataset also contains a few imperfect movements such as incomplete movement of circle and other shapes.

Both methods have their advantages and disadvantages. In addition, each method is intended for different use - EESNN for online and RESNN for offline classification. From the experimental analysis, it can be concluded that the suitable setup of the reservoir is not an easy task and future studies should identify ways to automate or simplify this procedure. However, once the reservoir is configured properly, the ESNN classifier is shown to be an efficient classifier of the liquid states extracted from the reservoir and satisfying classification results can be achieved.

The next chapter presents a summary of this study and discusses directions for some future work.

Chapter 10

CONCLUSION AND FUTURE DIRECTION

This chapter summarises the work that has been done to achieve the research objectives as specified in Chapter 1 and to answer the research questions posed in Chapter 1. Several suggestions for future work are also discussed in this chapter.

10.1 CONCLUSION

This thesis has studied two major areas of computational intelligence: the neural information processing and bio-inspired evolutionary computation. Each of these two areas has its own direction and is exciting to explore. Recent findings by researchers around the world have made these areas even more interesting and many applications have been proposed. Combination and hybridisation of methods is a current trend that is commonly practiced. Yet, there is no absolute answer to which method is the best to solve the problems that the scientific community faces. Hopefully, this research has contributed small but significant pieces of knowledge to the community and can be used as a useful reference for future research.

Aligned with the research objectives, the ultimate goal of this research is to develop an effective method for spatial and spatiotemporal pattern recognition by proposing integrated frameworks. This method integrates several computational principles - ECoS, quantum computation and reservoir computing to address some specific problems of:

- Parameter optimisation,

- Feature selection,
- Model optimisation,
- Adaptation to new data,
- Spatial classification, and
- Spatiotemporal pattern recognition.

In order to carry out the research, the first crucial step is to understand the ESNN architecture. Most of the literature review presented in Chapter 2 is focussed on understanding this unique ECoS-inspired classifier. The review started by introducing the SNN from which ESNN inherits most of its components. These components include the data encoding methods, neuron models and learning algorithms. The chapter also explains that the construction of LSM is based on the recurrent spiking neurons. All tools and applications of SNN are also described in this chapter. The need for parameter optimisation is identified as some unsatisfactory results caused by inaccurate parameter combinations are explained in Chapter 3. This chapter also describes the PSO algorithm and its variant, the QiPSO, which is used for binary optimisation.

The first attempt in this thesis to build an integrated structure is discussed in Chapter 4. It integrates the standard PSO with ESNN. The main purpose is to understand how the ESNN model can be optimised by tuning its parameters. Every particle in the PSO optimiser holds the ESNN parameters that need to be optimised. Utilising the Wrapper approach, the particles interact with each other based on their fitness function and share the best information found trough the $gbest$ particle. This method was tested using two synthetic datasets, namely the Hypercube and the Two Spirals which is explained in detail in Section 4.1.2. The results from the proposed method were compared with two commonly used classifiers, the MLP and SVM. The analysis of this framework shows that the optimised model can produce better classification accuracy.

Very often in real world problems, not all input features are relevant for good classification. Therefore, a feature selection process can be considered compulsory along with parameter optimisation. In the next step of this study, the

quantum principle is added to the optimiser for the probability computation in the feature selection task as explained in the second framework proposed in Chapter 4. This time, both parameters and input features are simultaneously optimised in the integrated environment. The particle is divided into two parts, where one part holds the ESNN parameter values and another part holds the qubit information for feature selection. This leads to a better classification result than the one obtained with the previous framework where only the parameters are optimised. The classification result (Table 4.5) shows that only some of the features are needed in order to achieve better accuracy results. Normally, only some of the features contain relevant information that can help to produce better outputs. For high dimensional problems, feature selection also helps to reduce both the processing time and the complexity of the problems. Analysis of the selected features leads to a better understanding of the problem at hand and enables the discovery of new knowledge as explained in Section 4.2.3. In collaboration with NICT, the proposed method was applied for string pattern recognition. An additional component, a string kernel is added into the proposed framework to transform string data to numerical values required by the classifier. The framework is depicted in Figure 4.13. A promising result is obtained with the proposed framework as it managed to optimise the parameter and to reduce the problem size.

A few problems have been identified when analysing the experiments in Chapter 4. Section 4.3 explained the issues. This includes the possibility of missing the optimal parameter value due to data representation. The second problem is the lack of ability of the optimiser to find the relevant features because of the random evaluation method. Therefore, several enhancements have been proposed involving both the optimiser and the classifier. For the optimiser, a hybrid particle structure is proposed in Chapter 5 where standard PSO particles and QiPSO particles are combined in order to solve the problem effectively. This allows the standard PSO to optimise the parameter as a real value while the QiPSO part handles feature and connection optimisation that involves probability computation. A new strategy for finding the relevant features is also proposed and it involves several types of particles. Each particle has its own way to identify the relevant features. With the combination of these types of particles, the relevant features can be identified and the irrelevant

ones can be eliminated earlier, as shown in Figure 5.4 and Figure 5.5. These improvements allow the proposed framework to achieve better classification results.

On the other hand, it appears that adding some probabilistic elements in the classifier may lead to a model that is more efficient than a deterministic model. In Chapter 6, the ESNN is extended based on the concept of the probabilistic neuron model, whereby probabilistic connection is introduced. This probabilistic connection can also be considered as an evolving connection where two principles, namely ECoS and quantum computation are combined. This experiment is conducted mainly to find out if the ESNN learning can be enhanced. In this method, the connections are not static, but rather evolve based on information that the connections hold as shown in Figure 6.1 and Figure 6.2. A connection is selected if the firing time information it holds is significant for the classification. Connections with least information will not be selected by the optimiser. The results show a slight improvement when this probabilistic principle is applied into the ESNN connections. The results also show that the optimiser is able to reduce a significant number of connections which leads to a faster learning for the PESNN.

The next phase in this study is extending ESNN to solve spatiotemporal classification problems. Two approaches have been proposed: EESNN and RESNN. Chapter 7 starts with an explanation of the spatiotemporal problems and this is followed by proposing the EESNN framework. This method captures both the spatial and temporal information and stores both types of information in an additional memory. This information is then propagated to the second module for classification. This chapter also proposes a synthetic spatiotemporal dataset as a benchmark dataset to test the framework. The Rotating Dot dataset explained in Section 7.3.1 is based on a two-class problem to determine the movement of a dot either clockwise or anticlockwise where the same path is used by the dot in both directions. The result from the experiments demonstrates that EESNN is able to classify the spatiotemporal data. High classification results are obtained when correct parameters are used even when datasets with high noise rates are used for testing. Quite a similar approach is used in RESNN. However, this approach applies the complex LSM as a reservoir to capture both spatial and temporal information as explained

in Chapter 8. The RESNN framework is depicted in Figure 8.2. The spatiotemporal data is encoded into a sequence of spike trains and then injected into the reservoir. The responses from the reservoir is measured by the readout functions and translated into liquid states for the classification process.

Section 8.2.1 describes a simple series of spike trains and applied to RESNN for the feasibility study of the proposed framework as spatiotemporal classifier. In this experiment, three readouts are evaluated: Cluster, Frequency and Analog readout. The results show the Analog readout is more stable and consistently produces high classification accuracy over the entire simulation time. Analog readout is further studied when RESNN is tested on the Rotating Dot dataset proposed in Chapter 7. Figure 8.9 shows the result when the Analog readout is tested on the dataset with different levels of difficulty and with different parameter values for the Analog readout. It can be concluded that higher τ value provides better classification accuracy. The result on the Rotating Dot dataset obtained by RESNN is compared with that obtained by EESNN in the last part of the chapter. Each method has its own strength: EESNN is a fast algorithm while RESNN is capable of classifying continuously at any time point. The fading memory effect in RESNN helps the algorithm to perform classification even after all spike trains have been injected into the reservoir.

Chapter 9 discusses the results from testing both EESNN and RESNN on a case study with spatiotemporal problems. The dataset for the case study is LIBRAS, which contains 15 sign language movements that need to be classified. An example of the movement for each class recorded is shown in Figure 9.1. Both methods show good capabilities to classify the given case study problems effectively. EESNN gives good classification accuracy with fast learning capability. On the other hand, all three RESNN readouts are studied for the LIBRAS dataset. The result is consistent with findings in Chapter 8 where Analog readout gives the best accuracy as shown in Figure 9.3. The specific liquid states are then further tested with offline RESNN optimisation with an improved result is recorded as described in Section 9.3.2. The summary of contributions is depicted in Figure 1.2. Based on the objectives and the work done in this study, the contributions of this study are described in the next section.

Along with achieving the research objectives, this study has also been aimed at finding the answers to the related research questions. The following paragraphs provide brief and concise answers to the research questions.

1. How to integrate and optimise ESNN with PSO and QiPSO?

   PSO can act as an optimiser for ESNN parameters. On the other hand, QiPSO can be used not only for parameter optimisation, but also for simultaneous feature optimisation. To integrate these methods, the Wrapper approach has been applied. In the Wrapper approach, the optimiser supplies the information to the induction algorithm which is ESNN, or PESNN, in this case. The induction algorithm use the information provided by the optimiser solution candidates and return the fitness function to be evaluated by the optimiser. Detailed explanations are provided in Chapter 4.

2. Can the learning of ESNN be improved by introducing probabilistic elements?

   The experiments have proven that some improvements of the classification results can be obtained from the modified ESNN - the PESNN. Evolving connections in PESNN allows the network to select the most informative connections that can contribute significant spikes to achieve better results as explained in details in Chapter 6.

3. How may the current ESNN be extended in order to solve spatiotemporal pattern recognition problems?

   In order to conduct classification on spatiotemporal problems, both time and space have to be considered. Therefore, an additional component has to be added into the framework to capture both types of information. This study has proposed two solutions. The first solution is the framework proposed in Chapter 7, where a spike memory is introduced to collect all information before sending it to the classifier. The second solution is the framework proposed in Chapter 8, where the LSM has been introduced to accumulate all spatiotemporal information for classification. The responses from the reservoir are measured and transformed into a liquid state format that can be interpreted by the classifier.

10.2 THESIS CONTRIBUTIONS

Contributions of this study are described as follows:

1. A major contribution of this study is the development of the following algorithms:
    - The development of DQiPSO. This thesis has developed a new PSO structure and has shown how this optimiser can be implemented for model optimisation of both parameters and connections and for feature selection. The proposed optimiser provides a more efficient classification with optimal features selection, parameter and connection optimisation as explained in Chapter 5.
    - The study presents a new ESNN structure as described in Chapter 6 - the PESNN, which introduces the evolving connections that provide the internal learning capability to ESNN.
    - This study has proposed an extension of the ESNN architecture that enables the method to process spatiotemporal data. The spatiotemporal data signal is transformed into a single high-dimensional network state that can be learned by the ESNN training algorithm. This study proposes in Chapter 7 EESNN as an extended structure of ESNN, where an additional module is integrated with ESNN to provide spike representation of the input patterns required for the classification task.
    - This study also proposed the RESNN framework for spatiotemporal pattern recognition explained in Chapter 8. The LSM has been experimentally demonstrated as reservoir that accumulates both spatial and temporal data components and transforms it into the liquid states that can be learned by the classifier. Furthermore, the framework also shows an enhanced capability to separate the output classes which leads to better classification.
2. This study has proposed and developed several novel integrated frameworks for simultaneous feature selection and model optimisation, which are:

- Integrated ESNN-PSO where PSO optimises all ESNN parameters to get the optimal values (Chapter 4).
- Integrated ESNN-QiPSO, a novel integration between ESNN and QiPSO. This framework allows features and ESNN parameters to be optimised simultaneously to improve classification accuracy. The extended version with string kernel allows the string dataset to be classified by the proposed framework. (Chapter 4).
- Integrated ESNN-DQiPSO framework for simultaneous feature and ESNN parameter optimisation. The improvements proposed in the optimiser allow this framework to optimise features, parameters and connections effectively and lead to better classification accuracy (Chapter 5 and Chapter 6).
- Integrated PESNN-DQiPSO, where evolving connections were introduced in the ESNN as part of the internal learning mechanism. A novel integration between PESNN and DQiPSO is proposed for simultaneous connection, feature and ESNN parameter optimisation (Chapter 6).

3. This study has developed two new solutions to the real world problems based on ESNN. The application of string classification has been explained in Chapter 4. LIBRAS hand movement recognition might lead to the development of a system that helps members of the deaf community to interact with computers has been discussed in Chapter 9.

4. In additon, a synthetic spatiotemporal dataset called Rotating dot is proposed in this study. The difficulty of the dataset can be controlled. This contributes to the better analysis of results obtained from the proposed EESNN and RESNN method (Chapter 7).

5. As intended in the objective of the study, all experiments in this study have been shared with the scientific community by producing eight blind peer-review international academic papers as listed in Chapter 1.

10.3 FUTURE DIRECTION

While the methods proposed in this study deliver promising results, efforts to improve their performance should still continue. Below are some suggestions for future work in the researched area:

10.3.1 *Optimisation Strategy*

As explained in chapter 6, optimisation of connections, features and parameters for PESNN is quite slow because of the large dimension of the problems. This situation does not only happen to the PSO, but also to other optimisers when dealing with high dimensional problems. Several approaches are suggested to deal with PESNN optimisation. The first approach is to allow only certain connections or a cluster area for optimisation. This is because not all connections have enough weight information to be used. For instance, if an input is distributed into 20 pre-synaptic neurons, normally the first five and the last five neurons hold insignificant weight values. Thus, this connection can be eliminated and the rest can be used for optimisation. This approach not only manages to reduce significantly the number of connections to be optimised, but also eliminates unnecessary connections that contain less information.

The second approach is to use the parallel computing as a possible solution to this problem. A problem can be divided into several sub-tasks with each task solved a specific problem. This approach may produce faster and effective method that can be applied in real world high dimensional problems. Another approach is to use the *BlueFern supercomputer*[1] which is currently available in New Zealand for research purposes. Using a supercomputer for neural information processing as in the Blue Brain Project (Markram, 2006) is an option on problems involves huge computation time and requirements. In terms of feature optimisation strategy, although the proposed DQiPSO optimiser is able to enhance the selection of relevant feature compared to QiPSO, there is still a gap to improve the strategy. Future works could focus on finding more effective methods for eliminating less relevant features. These may in-

1 More information about BlueFern is available at http://www.bluefern.canterbury.ac.nz

clude applying the three levels optimisation as proposed in QEA Defoin-Platel et al. (2007).

10.3.2 *Classifier*

This study has demonstrated the proposed ESNN framework in spatial and spatiotemporal classification problems. Future studies in this area should focus on the characteristics of ESNN and PESNN and how to apply them to other possible problems such as prediction and clustering. ESNN is well known for its fast learning and is very suitable for online or real time problems. Furthermore, the concept of recurrence in ESNN can also be an interesting subject to be explored. A recurrent network allows the network to memorise some of the previous information that could be used to solve the given problems.

One of the main challenges in ESNN application is to determine the optimal number of pre-synaptic neurons for a given dataset. Number of pre-synaptic neurons is required before the ESNN structure can be constructed. This problem is similar to identify the number of hidden nodes in MLP. Lower number of pre-synaptic neurons causes less input spikes generated and subsequently may affect learning accuracy, while higher number increases computational time. Although the ESNN uses the Thorpe model for its neuron model, there are some other potential neuron models to be explored. In addition to this, one of the probabilistic elements from Kasabov's model has been applied which is the probabilistic connection for the development of PESNN. There are two other probabilistic elements that can be further explored as explained in Section 2.3.6.

10.3.3 *Reservoir*

The LSM reservoir is a rather complex structure and future studies could investigate how to simplify it. The dynamic threshold (Schliebs, Nuntalid, & Kasabov, 2010) in the reservoir is interesting to investigate. Applying the dynamic threshold into the reservoir may create a different set of responses, which could lead to better classification. Future directions include the devel-

opment of new learning algorithms for the reservoir and the application of the method on other spatiotemporal real-world problems such as video or audio pattern recognition tasks. Furthermore, future studies could also focus on developing and implementating specialised SNN hardware (Indiveri, Chicca, & Douglas, 2009; Indiveri, Stefanini, & Chicca, 2010) to allow the classification of spatio-temporal data streams in real time.

10.3.4 *Neurogenetic optimisation*

One of the future applications to be considered is in the area of neurogenetics. The majority of existing models of neural development are molecular and biochemical models that do not take into account the role and dynamics of genes (Benuskova & Kasabov, 2007). Kasabov, Benuskova, and Wysoski (2004) introduced a novel connectionist approach to neural network modelling by integrating a neural network model with dynamic gene networks. Interaction of genes in the neurons affect the whole neural network. Therefore, the optimisation of the gene interaction network and the gene expression values are necessary for achieving the optimal state of neural network. Future studies could investigate the suitability of the proposed optimiser in this study for such problems.

Appendix A

DESCRIPTION OF THE INTEGRATED ESNN-PSO

This section formalise the description of the proposed integrated ESNN-PSO. The pseudo code is split into several sub functions.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- i iterations
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- $parNo$ number of particle in the swarm
- w inertia weight
- c_1 control exploration towards $pbest$
- c_2 control exploration towards $gbest$
- rec number of receptive fields for input neurons
- β for gaussian width

construct $pbest$, $gbest$ and $parNo$ of $particle$ with a structure of:
- $parameter$ array to store ESNN parameters for optimisation
- R array for constructed output repository
- $fitness$ value for $particle$

define functions:
- *readData()* encode dataset to spike trains
- *pso()* particle update for training
- *training()* get fitness value from ESNN
- *storeBest()* store $gbest$ and $pbest$
- *testing()*

readData()

for all n **do**
pso()
testing()
end for

Pseudocode 2 *readData()* function

---

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
for all input data field **do**
create rec number input neurons using Gaussian function in Equation 2.3 bounded between $inpMin$ and $inpMax$, with:
1) centre is calculated in Equation 2.4
2) width calculated in Equation 2.5 with control parameter β is applied
store input neurons in $spiketrains$
end for
end for

allocate $spiketrains$ according to n, and store in $nfoldspikes$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 3 *pso()* function

---

for all $particle$ **do**
initialise all $parameter$
initialise $fitness$
end for

while not reaching maximum i **do**
for all $particle$ **do**
training()
if $fitness$ **better than** $fitness$ of $pbest$ **then**
storeBest()
end if

for all $parameter$ **do**
calculate velocity using Equation 3.1 with consideration of

w, c_1, c_2, $gbest$ and $pbest$
update $parameter$ position using Equation 3.2
end for
end for
end while

Pseudocode 4 *storeBest()* function

---

replace $pbest$ with $particle$

if $fitness$ of $pbest$ **better than** $fitness$ of $gbest$ **then**
replace $gbest$ with $pbest$
end if

Pseudocode 5 *training()* function

---

retrieve $parameter$ from $particle$
initialise neuron repository $R = \{\}$

for all output class **do**
for all $nfoldspikes$ **do**
calculate the connection $weight$ using Equation 2.25
get the maximum possible potential according to Equation 2.26
calculate firing time threshold ϑ based on Equation 2.27
if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
$weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
$\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
else
add the new neuron $R \leftarrow R \cup \{weight\}$
end if
end for
end for

compare training output with $nfoldoutput$ and calculate accuracy
$fitness \leftarrow$ accuracy
return $fitness$

Pseudocode 6 *testing()* function

retrieve R and $parameter$ from $gbest$

for all testing $nfoldspikes$ **do**
    calculate the connection $weight$ using Equation 2.25
    while PSP **less than** ϑ **do**
        calculate PSP using Equation 2.17
    end while
    get output class from output neuron which spike first
end for

compare testing output with $nfoldoutput$ and calculate accuracy

Appendix B

DESCRIPTION OF THE INTEGRATED ESNN-QIPSO

The proposed integrated ESNN-QiPSO is described in this section.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- i iterations
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- Q number of qubit
- $parNo$ number of particle in the swarm
- w inertia weight
- c_1 control exploration towards $pbest$
- c_2 control exploration towards $gbest$
- rec number of receptive fields for input neurons
- β for gaussian width

construct $pbest$, $gbest$ and $parNo$ of $particle$ with a structure of:
- **for all** $parameter$ **do**
  - assign Q size par_θ to store parameter probability
  - assign Q size par_col to store parameter collape bit
- **end for**
- **for all** $feature$ **do**
  - assign $feat\_\theta$ array to store feature probability
  - assign $feat\_col$ array to store feature collape bit
- **end for**
- R array for constructed output repository
- $fitness$ value for $particle$

define functions:
- *readData()* encode dataset to spike trains
- *qipso()* particle update for training

training() get fitness value from ESNN
storeBest() store $gbest$ and $pbest$
testing()

readData()

for all n **do**
qipso()
testing()
end for

Pseudocode 2 *readData()* function

---

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
for all input data field **do**
create rec number input neurons using Gaussian function in Equation 2.3
bounded between $inpMin$ and $inpMax$, with:
1) centre is calculated in Equation 2.4
2) width calculated in Equation 2.5 with control parameter β is applied
store input neurons in $spiketrains$
end for
end for

allocate $spiketrains$ according to n, and store in $nfoldspikes$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 3 *qipso()* function

---

for all $particle$ **do**
for all $parameter$ **do**
for all Q **do**
initialise par_θ
get collape bit par_col using Equation 3.12
end for
convert binary string par_col to real value
end for
for all $feature$ **do**
initialise $feat\_\theta$
get collape bit $feat\_col$ using Equation 3.12

```
    end for
    initialise fitness
end for

while not reaching maximum i do
    for all particle do
        training()
        if fitness better than fitness of pbest then
            storeBest()
        end if

        for all parameter do
            for all Q do
                calculate par_θ velocity using Equation 3.9 with consideration
                of w, c1, c2, gbest and pbest
                apply rotation gate in Equation 3.11
                get collape bit par_col using Equation 3.12
            end for
            convert binary string par_col to real value
        end for

        for all feature do
            calculate feat_θ velocity using Equation 3.9 with consideration of
            w, c1, c2, gbest and pbest
            apply rotation gate in Equation 3.11
            get collape bit feat_col using Equation 3.12
        end for

    end for
end while
```

Pseudocode 4 *storeBest()* function

```
replace pbest with particle

if fitness of pbest better than fitness of gbest then
    replace gbest with pbest
end if
```

Pseudocode 5 *training()* function

retrieve $parameter$ from $particle$
initialise neuron repository $R = \{\}$

for all output class **do**
  if $feat\_col$ == 1 **then**
    get $nfoldspikes$ of selected $feature$ and store in $selectspikes$
    for all $selectspikes$ **do**
      calculate the connection $weight$ using Equation 2.25
      get the maximum possible potential according to Equation 2.26
      calculate firing time threshold ϑ based on Equation 2.27

      if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
        $weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
        $\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
      else
        add the new neuron $R \leftarrow R \cup \{weight\}$
      end if

    end for
  end if
end for

compare training output with $nfoldoutput$ and calculate accuracy
$fitness \leftarrow$ accuracy
return $fitness$

Pseudocode 6 *testing()* function

retrieve R and $parameter$ from $gbest$

if $feat\_col$ in $gbest$ == 1 **then**
    get testing $nfoldspikes$ of selected $feature$ and store in $selectspikes$
    for all $selectspikes$ **do**
        calculate the connection $weight$ using Equation 2.25
        while PSP **less than** ϑ **do**
            calculate PSP using Equation 2.17
        end while
        get output class from output neuron which spike first
    end for
end if

compare testing output with $nfoldoutput$ and calculate accuracy

Appendix C

DESCRIPTION OF THE INTEGRATED ESNN-DQIPSO

This section describe the proposed integrated ESNN-DQiPSO.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- i iterations
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- $parNo$ number of particle in the swarm
- w inertia weight
- c_1 control exploration towards $pbest$ for real value
- c_2 control exploration towards $gbest$ for real value
- θ_c_1 control exploration towards $pbest$ for quantum angle
- θ_c_2 control exploration towards $gbest$ for quantum angle
- rec number of receptive fields for input neurons
- β for gaussian width

construct $pbest$, $gbest$ and $parNo$ of $particle$ with a structure of:
- $parameter$ array to store ESNN parameters for optimisation
- **for all** $feature$ **do**
  - assign $feat\_\theta$ array to store feature probability
  - assign $feat\_col$ array to store feature collape bit
- **end for**
- R array for constructed output repository
- $fitness$ value for $particle$

define functions:
- *readData()* encode dataset to spike trains
- *dqipso()* particle update for training
- *training()* get fitness value from ESNN

storeBest() store $gbest$ and $pbest$
testing()

readData()

for all n **do**
dqipso()
testing()
end for

Pseudocode 2 *readData()* function

---

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
for all input data field **do**
create rec number input neurons using Gaussian function in Equation 2.3 bounded between $inpMin$ and $inpMax$, with:
1) centre is calculated in Equation 2.4
2) width calculated in Equation 2.5 with control parameter β is applied
store input neurons in $spiketrains$
end for
end for

allocate $spiketrains$ according to n, and store in $nfoldspikes$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 3 *dqipso()* function

---

for all $particle$ **do**
initialise all $parameter$
for all $feature$ **do**
initialise $feat\_\theta$
get collape bit $feat\_col$ using Equation 3.12
end for
initialise $fitness$
end for

while not reaching maximum i **do**
for all $particle$ **do**

training()

if ($fitness$ **better than** $fitness$ of $pbest$) **or**
(($fitness$ == $fitness$ of $pbest$) **and**
(selected $feature$ **less than** selected $feature$ of $pbest$)) **then**
storeBest()
end if

for all $parameter$ **do**
calculate velocity using Equation 3.1 with consideration of
w, c_1, c_2, $gbest$ and $pbest$
update $parameter$ position using Equation 3.2
end for

for all $feature$ **do**
calculate $feat\_\theta$ velocity using Equation 3.9 with consideration of
w, θ_c_1, θ_c_2, $gbest$ and $pbest$
apply rotation gate in Equation 3.11
get collape bit $feat\_col$ using Equation 3.12
end for

end for
end while

Pseudocode 4 *storeBest()* function

replace $pbest$ with $particle$

if ($fitness$ of $pbest$ **better than** $fitness$ of $gbest$) **or**
(($fitness$ of $pbest$ == $fitness$ of $gbest$) **and**
(selected $feature$ of $pbest$ **less than** selected $feature$ of $gbest$)) **then**
replace $gbest$ with $pbest$
end if

Pseudocode 5 *training()* function

retrieve $parameter$ from $particle$
initialise neuron repository $R = \{\}$

for all output class **do**
if $feat\_col$ == 1 **then**
get $nfoldspikes$ of selected $feature$ and store in $selectspikes$

for all $selectspikes$ **do**
calculate the connection $weight$ using Equation 2.25
get the maximum possible potential according to Equation 2.26
calculate firing time threshold ϑ based on Equation 2.27

if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
$weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
$\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
else
add the new neuron $R \leftarrow R \cup \{weight\}$
end if

end for
end if
end for

compare training output with $nfoldoutput$ and calculate accuracy
$fitness \leftarrow$ accuracy
return $fitness$

Pseudocode 6 *testing()* function

---

retrieve R and $parameter$ from $gbest$

if $feat\_col$ in $gbest$ == 1 **then**
get testing $nfoldspikes$ of selected $feature$ and store in $selectspikes$
for all $selectspikes$ **do**
calculate the connection $weight$ using Equation 2.25
while PSP **less than** ϑ **do**
calculate PSP using Equation 2.17
end while
get output class from output neuron which spike first
end for
end if

compare testing output with $nfoldoutput$ and calculate accuracy

---

Appendix D

DESCRIPTION OF THE INTEGRATED PESNN-DQIPSO

This section describe the proposed integrated PESNN-DQiPSO.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- i iterations
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- $parNo$ number of particle in the swarm
- w inertia weight
- c_1 control exploration towards $pbest$ for real value
- c_2 control exploration towards $gbest$ for real value
- θ_c_1 control exploration towards $pbest$ for quantum angle
- θ_c_2 control exploration towards $gbest$ for quantum angle
- rec number of receptive fields for input neurons
- β for gaussian width

construct $pbest$, $gbest$ and $parNo$ of $particle$ with a structure of:
- $parameter$ array to store ESNN parameters for optimisation
- **for all** $feature$ **do**
  - assign $feat\_\theta$ array to store feature probability
  - assign $feat\_col$ array to store feature collape bit
- **end for**
- **for all** $connection$ **do**
  - assign con_θ array to store connection probability
  - assign con_col array to store connection collape bit
- **end for**
- R array for constructed output repository
- $fitness$ value for $particle$

define functions:
readData() encode dataset to spike trains
dqipso() particle update for training
training() get fitness value from ESNN
storeBest() store $gbest$ and $pbest$
testing()

readData()

for all n **do**
dqipso()
testing()
end for

Pseudocode 2 *readData()* function

---

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
for all input data field **do**
create rec number input neurons using Gaussian function in Equation 2.3
bounded between $inpMin$ and $inpMax$, with:
1) centre is calculated in Equation 2.4
2) width calculated in Equation 2.5 with control parameter β is applied
store input neurons in $spiketrains$
end for
end for

allocate $spiketrains$ according to n, and store in $nfoldspikes$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 3 *dqipso()* function

---

for all $particle$ **do**
initialise all $parameter$
for all $feature$ **do**
initialise $feat\_\theta$
get collape bit $feat\_col$ using Equation 3.12
if ($feat\_col$ == 1) **then**
initialise con_θ
get collape bit con_col using Equation 3.12

```
        end if
    initialise fitness
end for

while not reaching maximum i do
    for all particle do

        training()

        if (fitness better than fitness of pbest) or
        ((fitness == fitness of pbest) and
        (selected feature less than selected feature of pbest)) or
        ((fitness == fitness of pbest) and
        (selected feature == selected feature of pbest)) and
        (selected connection less than selected connection of pbest)) then
            storeBest()
        end if

        for all parameter do
            calculate velocity using Equation 3.1 with consideration of
            w, c1, c2, gbest and pbest
            update parameter position using Equation 3.2
        end for

        for all feature do
            calculate feat_θ velocity using Equation 3.9 with consideration of
            w, θ_c1, θ_c2, gbest and pbest
            apply rotation gate in Equation 3.11
            get collape bit feat_col using Equation 3.12

            if (feat_col == 1) then
                for all connection do
                    calculate con_θ velocity using Equation 3.9 with consideration
                    of w, θ_c1, θ_c2, gbest and pbest
                    apply rotation gate in Equation 3.11
                    get collape bit con_col using Equation 3.12
                end for
            end if
        end for

    end for
end while
```

Pseudocode 4 *storeBest()* function

replace $pbest$ with $particle$

if ($fitness$ of $pbest$ **better than** $fitness$ of $gbest$) **or**
(($fitness$ of $pbest$ == $fitness$ of $gbest$) **and**
(selected $feature$ of $pbest$ **less than** selected $feature$ of $gbest$)) **or**
(($fitness$ of $pbest$ == $fitness$ of $gbest$) **and**
(selected $feature$ of $pbest$ == selected $feature$ of $gbest$)) **and**
(selected $connection$ of $pbest$ **less than** selected $connection$ of $gbest$)) **then**
    replace $gbest$ with $pbest$
end if

Pseudocode 5 *training()* function

retrieve $parameter$ from $particle$
initialise neuron repository $R = \{\}$

for all output class **do**
    if $feat\_col$ == 1 **then**
        get $nfoldspikes$ of selected $feature$ and store in $selectspikes$
        for all $selectspikes$ **do**
            if con_col == 1 **then**
                get $selectspikes$ and store in $finalspikes$
            end if
        end for

        for all $finalspikes$ **do**
            calculate the connection $weight$ using Equation 2.25
            get the maximum possible potential according to Equation 2.26
            calculate firing time threshold ϑ based on Equation 2.27
            if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
                $weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
                $\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
            else
                add the new neuron $R \leftarrow R \cup \{weight\}$
            end if

        end for
    end if

end for

compare training output with $nfoldoutput$ and calculate accuracy
$fitness \leftarrow$ accuracy
return $fitness$

Pseudocode 6 *testing()* function

---

retrieve R and $parameter$ from $gbest$

if $feat\_col$ in $gbest$ == 1 **then**
    get testing $nfoldspikes$ of selected $feature$ and store in $selectspikes$
    for all $selectspikes$ **do**
        if con_col in $gbest$ == 1 **then**
            get $selectspikes$ and store in $finalspikes$
        end if
    end for

    for all $finalspikes$ **do**
        calculate the connection $weight$ using Equation 2.25
        while PSP **less than** ϑ **do**
            calculate PSP using Equation 2.17
        end while
        get output class from output neuron which spike first
    end for
end if

compare testing output with $nfoldoutput$ and calculate accuracy

---

Appendix E

DESCRIPTION OF THE INTEGRATED EESNN

This section formalise the description of the proposed integrated EESNN for spatiotemporal pattern recognition.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- mod modulation factor
- c proportion factor
- sim similarity value
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- rec number of receptive fields for input neurons
- β for gaussian width

define functions:
- *memory()* encode dataset to spike trains
- *training()* get fitness value from ESNN
- *testing()*

memory()

for all n **do**
- *training()*
- *testing()*

end for

Pseudocode 2 *memory()* function

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
  for all time point **do**
    for all input data field **do**
      create rec number input neurons using Gaussian function in Equation 2.3 bounded between $inpMin$ and $inpMax$, with:
      1) centre is calculated in Equation 2.4
      2) width calculated in Equation 2.5 with parameter β is applied
    end for
  end for
  accumulate all input neurons for entire time points in $spikeMemory$
end for

allocate $spikeMemory$ according to n, and store in $nfoldspikeRsv$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 3 *training()* function

require mod, c, and sim
initialise neuron repository $R = \{\}$

for all output class **do**
  for all $nfoldspikeRsv$ **do**
    calculate the connection $weight$ using Equation 2.25
    get the maximum possible potential according to Equation 2.26
    calculate firing time threshold ϑ based on Equation 2.27
    if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
      $weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
      $\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
    else
      add the new neuron $R \leftarrow R \cup \{weight\}$
    end if
  end for
end for

compare training output with $nfoldoutput$ and calculate accuracy

Pseudocode 4 *testing()* function

require mod
retrieve R

for all testing $nfoldspikeRsv$ **do**
    calculate the connection $weight$ using Equation 2.25
    while PSP **less than** ϑ **do**
        calculate PSP using Equation 2.17
    end while
    get output class from output neuron which spike first
end for

compare testing output with $nfoldoutput$ and calculate accuracy

Appendix F

DESCRIPTION OF THE INTEGRATED RESNN

This section describe the proposed RESNN.

Pseudocode 1 *main()* function

initialise:
- n fold cross validation
- $matrix$ dimension of reservoir
- mod modulation factor
- c proportion factor
- sim similarity value
- $inpMin$ minimum input value
- $inpMax$ maximum input value
- rec number of receptive fields for input neurons
- β for gaussian width

define functions:
- *spiketrains()* encode dataset to spike trains
- *lsmreservoir()* accumulate spike trains
- *training()* get fitness value from ESNN
- *testing()*

spiketrains()
lsmreservoir()

for all n **do**
- *training()*
- *testing()*

end for

Pseudocode 2 *spiketrains()* function

read input samples and store in $sample$ array
read output file and store in $output$ array

for all $sample$ **do**
  for all time point **do**
    for all input data field **do**
      create rec number input neurons using Gaussian function in Equation 2.3 bounded between $inpMin$ and $inpMax$, with:
      1) centre is calculated in Equation 2.4
      2) width calculated in Equation 2.5 with parameter β is applied
    end for
    store input neurons in $spiketrains$
  end for
end for

Pseudocode 3 *lsmreservoir()* function

construct $matrix$ size $reservoir$ with interconnected neurons using Equation 9.1

for all $sample$ **do**
  for all $spiketrains$ **do**
    feed into $reservoir$
    calculate neuron spikes using Equation 8.2 and record into $response$
    construct $liquidstates$ from $response$
  end for
end for

allocate $liquidstates$ according to n, and store in $nfoldliquidstates$
allocate $output$ according to n, and store in $nfoldoutput$

Pseudocode 4 *training()* function

require mod, c, and sim
initialise neuron repository $R = \{\}$

for all output class **do**
  for all $nfoldliquidstates$ **do**
    calculate the connection $weight$ using Equation 2.25
    get the maximum possible potential according to Equation 2.26
    calculate firing time threshold ϑ based on Equation 2.27

if $\min(d(weight, weight^{(k)}) < sim, \quad weight^{(k)} \in R$ **then**
$weight^{(k)} \leftarrow$ merge $weight$ and $weight^{(k)}$ according to Equation 2.28
$\vartheta^{(k)} \leftarrow$ merge ϑ and $\vartheta^{(k)}$ according to Equation 2.29
else
add the new neuron $R \leftarrow R \cup \{weight\}$
end if
end for
end for

compare training output with $nfoldoutput$ and calculate accuracy

---

Pseudocode 5 *testing()* function

---

require mod
retrieve R

for all testing $nfoldliquidstates$ **do**
calculate the connection $weight$ using Equation 2.25
while PSP **less than** ϑ **do**
calculate PSP using Equation 2.17
end while
get output class from output neuron which spike first
end for

compare testing output with $nfoldoutput$ and calculate accuracy

---

Appendix G

DESCRIPTION OF THE MLP

MLP can be considered the most established machine learning algorithm available and is widely used in many real world applications. All neurons in MLP are interconnected and organised in several layers - input, hidden and output layers. BP is the common learning algorithm in the MLP network. The algorithm starts with assigning random weights to all connections. The goal is to adjust the weight so that the targeted output can be achieved. In the first phase of learning, samples presented to the input layer are propagate to the hidden and output layers. Output for every neuron in the hidden and output layers is obtained from the summation of all its connection weights using network activation function. Sigmoid activation function is widely adopted into BP learning. Equation G.1 shows the Sigmoid activation function. Equation G.2 and Equation G.3 explain the computation from input neurons (i) to hidden neurons (j) and hidden neurons(j) to output neurons (k), respectively.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{G.1}$$

$$x_j = (\sum w_{ij} I_i)\phi_j \tag{G.2}$$

where x_j is the output at the hidden layer, w_{ij} is the weight between input and hidden layers, I_i is the input data and ϕ_j is the bias.

$$x_k = (\sum w_{jk} I_j)\phi_k \tag{G.3}$$

where x_k is the output at the output layer, w_{jk} is the weight between input and hidden layers, I_j is the output computed at hidden layer and ϕ_k is the bias at output neuron. The error between feed forward output neuron and actual desired output is calculated using Equation G.4. The mean square error is normally used for computing the network error.

$$error = \frac{1}{2}(O_t - O_c)^2 \tag{G.4}$$

where O_d is the target output and O_c is the computed output during the feed forward phase.

In the second phase, the computed error is propagated backward from the output to the input layers. Weights are modified in order to reduce the error. Equation G.5 shows the modification of the weights between output and hidden layers.

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk} \tag{G.5}$$

where

$$\Delta w_{jk} \leftarrow w_{jk} + \eta \delta_k I_j + \alpha \Delta w_{jk} \tag{G.6}$$

$$\delta_k = O_c(O_d - O_c)(1 - O_c) \tag{G.7}$$

and w_{jk} is the weight between hidden and output layer, Δw_{jk} is the weight adjustment, η and α are the control parameters (learning rate and momentum rate respectively), I_j is the output at hidden layer, δ_k is the computed error at output neuron, O_d is the desired output and O_c is the computed output at output

layer. Then the error is computed at the hidden neurons and weights between input and hidden layers are adjusted using Equation G.8:

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \tag{G.8}$$

where

$$\Delta w_{ij} \leftarrow w_{ij} + \eta \delta_j I_i + \alpha \Delta w_{ij} \tag{G.9}$$

$$\delta_j = O_j(\sum \delta_k w_{jk})(1 - O_j) \tag{G.10}$$

and w_{ij} is the weight between input and hidden layer, Δw_{ij} is the weight adjustment, η is the learning rate, α represents momentum rate, I_i is the input data, δ_j and δ_k are the errors at hidden and output neurons respectively, O_d is the desired output and O_j is the computed output at hidden layer. Details of BP procedure can be found in Chauvin and Rumelhart (1995) and the computation example is explained in Jones (2005).

REFERENCES

Abbott, L. F., & Nelson, S. B. (2000, November). Synaptic plasticity: taming the beast. *Nature neuroscience, 3 Suppl*, 1178–1183.

Abs Da Cruz, A. V., Vellasco, M. M. B. R., & Pacheco, M. A. C. (2007). Quantum-inspired evolutionary algorithm for numerical optimization. *2006 IEEE International Conference on Evolutionary Computation, 37*(1), 2630–2637.

Adrian, E. D. (1926). The impulses produced by sensory nerve endings. *Journal of Physiology (London), 61*, 49–72.

Almuallim, H., & Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the ninth national conference on artificial intelligence (aaai-91)* (Vol. 2, pp. 547–552). AAAI Press.

Alnajjar, F., & Murase, K. (2005, nov.). Self-organization of spiking neural network generating autonomous behavior in a real mobile robot. In *Computational intelligence for modelling, control and automation, 2005 and international conference on intelligent agents, web technologies and internet commerce, international conference on* (Vol. 1, p. 1134 -1139).

Angeline, P. J. (1998). Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Proceedings of the 7th international conference on evolutionary programming vii* (pp. 601–610). London, UK: Springer-Verlag.

Arbib, M. A. (1995). *The handbook of brain theory and neural networks* (1st ed.). Cambridge, MA, USA: MIT Press.

Bäck, T., & Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation, 1*(1), 1-23.

Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., et al. (1995, November). Elementary gates for quantum computation. *Physical Review A, 52*(5), 3457–3467.

Belatreche, A., Maguire, L. P., & McGinnity, M. (2006, October). Advances in design and application of spiking neural networks. *Soft Comput., 11*, 239–248.

Benatchba, K., Koudil, M., Boukir, Y., & Benkhelat, N. (2006, nov.). Image segmentation using quantum genetic algorithms. In *Ieee industrial*

electronics, iecon 2006 - 32nd annual conference on (p. 3556 -3563).

Benuskova, L., & Kasabov, N. (2007). *Computational neurogenetic modeling* (1st ed.). Springer Publishing Company, Incorporated.

Bergh, F. van den. (1999). Particle swarm weight initialization in multi-layer perceptron artificial neural networks. In *Proceedings of the 1991 international conference on artificial neural networks, icann-91* (pp. 365–370). Elsevier Publishers B.V., North-Holland.

Bergh, F. van den, & Engelbrecht, A. P. (2000). Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, *26*, 84-90.

Bi, G., & Poo, M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, *24*(1), 139–166.

Bliss, T. V. P., & Lomo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *J Physiol.*, *232*(2), 331-356.

Blum, A., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*(1-2), 245-271.

Bohte, S. M., & Kok, J. N. (2005). Applications of spiking neural networks. *Inf. Process. Lett.*, *95*(6), 519–520.

Bohte, S. M., Kok, J. N., & Poutré, J. A. L. (2000). SpikeProp: backpropagation for networks of spiking neurons. In *ESANN* (p. 419-424).

Bohte, S. M., Kok, J. N., & Poutré, J. A. L. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1-4), 17-37.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. New York, NY, USA: Oxford University Press, Inc.

Bramer, M. A. (Ed.). (1999). *Knowledge discovery and data mining*. Stevenage, UK, UK: Institution of Electrical Engineers.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, *2007*, 349–398.

Buonomano, D. V., & Maass, W. (2009, January). State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, *10*(2), 113–125.

Carnevale, N. T., & Hines, M. L. (2006). *The NEURON book*. New York, NY, USA: Cambridge University Press.

Cawley, S., Morgan, F., McGinley, B., Pande, S., McDaid, L., Carrillo, S., et al. (2011). Hardware spiking neural network prototyping and application. *Genetic Programming and Evolvable Machines*, *12*, 257-280.

Chauvin, Y., & Rumelhart, D. E. (Eds.). (1995). *Backpropagation: theory, architectures, and applications*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

Cios, K. J., Pedrycz, W., Swiniarski, R. W., & Kurgan, L. A. (2007). *Data Mining: A Knowledge Discovery Approach* (1st ed.). Springer. Hardcover.

Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Congress on evolutionary computation* (Vol. 3, pp. 1951–1957).

Coello, C., Pulido, G., & Lechuga, M. (2004, june). Handling multiple objectives with particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, *8*(3), 256 - 279.

Defoin-Platel, M., Schliebs, S., & Kasabov, N. (2007). A versatile quantum-inspired evolutionary algorithm. In *Ieee congress on evolutionary computation* (p. 423-430).

Delorme, A., & Thorpe, S. J. (2003, November). SpikeNET: an event-driven simulation package for modelling large networks of spiking neurons. *Network (Bristol, England)*, *14*(4), 613–627.

Deng, D., & Kasabov, N. K. (2000). ESOM: An algorithm to evolve self-organizing maps from on-line data streams. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN'00* (Vol. 6, p. 3-8). Como, Italy: IEEE Press.

Dias, D. B., Madeo, R. C. B., Rocha, T., Bíscaro, H. H., & Peres, S. M. (2009). Hand movement recognition for brazilian sign language: a study using distance-based neural networks. In *Proceedings of the 2009 international joint conference on neural networks* (pp. 2355–2362). IEEE Press.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *26*(1), 29–41.

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS95 Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, *43*, 39–43.

Eberhart, R. C., Shi, Y., & Kennedy, J. (2001). *Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation)* (1st ed.). Morgan Kaufmann. Hardcover.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*(2), 179–211.

Estévez, P. A., Tesmer, M., Perez, C. A., & Zurada, J. M. (2009, February). Normalized mutual information feature selection. *Trans. Neur. Netw.*, *20*, 189–201.

Fan, K., Brabazon, A., O'Sullivan, C., & O'Neill, M. (2007). Option pricing model calibration using a real-valued quantum-inspired evolutionary algorithm. In *Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 1983–1990). New York, NY, USA: ACM.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (Eds.). (1996). *Advances in knowledge discovery and data mining*. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Feng, X., Wang, Y., Ge, H., Zhou, C., & Liang, Y. (2006). Quantum-inspired evolutionary algorithm for travelling salesman problem. In G. LIU, V. TAN, & X. HAN (Eds.), *Computational methods* (p. 1363-1367). Springer Netherlands.

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, *21*(6-7), 467–488.

Florian, R. V. (2005). A reinforcement learning algorithm for spiking neural networks. In *SYNASC '05: Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (p. 299). Washington, DC, USA: IEEE Computer Society.

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468-1502.

Fox, R. (1997). Stochastic versions of the hodgkin-huxley equations. *Biophysical Journal*, *72*(5), 2068 - 2074.

Fredkin, E., & Toffoli, T. (1982, April). Conservative logic. *International Journal of Theoretical Physics*, *21*(3), 219–253.

Gao, H., & Diao, M. (2009, nov.). Quantum particle swarm optimization for mc-cdma multiuser detection. In *Artificial intelligence and computational intelligence, 2009. aici '09. international conference on* (Vol. 2, p. 132 -136).

Gerstner, W. (2000). Population dynamics of spiking neurons: Fast transients, asynchronous states, and locking. *Neural Comput.*, *12*(1), 43–89.

Gerstner, W. (2001). What is Different with Spiking Neurons? In H. Mastebroek & J. E. Vos (Eds.), *Plausible neural networks for biological modelling.* Kluwer Academic.

Gerstner, W., & Hemmen, J. L. van. (1994). Coding and information processing in neural networks. In *Models of neural networks II* (pp. 1–93).

Gerstner, W., & Kistler, W. (2002a). *Spiking neuron models: An introduction*. New York, NY, USA: Cambridge University Press.

Gerstner, W., & Kistler, W. M. (2002b). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge, MA: Cambridge University Press.

Gerstner, W., Ritz, R., & Hemmen, J. L. v. van. (1993). Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. *Biol Cybern*, *69*(5-6), 503–515.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Goodman, E., & Ventura, D. (2006). Spatiotemporal Pattern Recognition via Liquid State Machines. In *The 2006 ieee international joint conference on neural network proceedings* (pp. 3848–3853). IEEE.

Gray, F. (1953). *Pulse code communication.* U.S. Patent 2 632 058.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (pp. 212–219). New York, NY, USA: ACM.

Grzyb, B. J., Chinellato, E., Wojcik, G. M., & Kaminski, W. A. (2009). Which model to use for the liquid state machine? In *Proceedings of the 2009 international joint conference on neural networks* (pp. 1692–1698). Piscataway, NJ, USA: IEEE Press.

Gu, J., Gu, M., Cao, C., & Gu, X. (2010). A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem. *Computers and Operations Research*, *37*(5), 927 - 937.

Gu, J., Gu, X., & Gu, M. (2009). A novel parallel quantum genetic algorithm for stochastic job shop scheduling. *Journal of Mathematical Analysis and Applications*, *355*(1), 63 - 81.

Guckenheimer, & Labouriau, J. (1993, January). Bifurcation of the hodgkin and huxley equations: A new twist. *Bulletin of Mathematical Biology*, 937 952.

Gudise, V. G., & Venayagamoorthy, G. K. (2003). *Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks.*

Han, K.-H., & Kim, J.-H. (2000). Genetic quantum algorithm and its application to combinatorial optimization problem. In *Evolutionary computation, 2000. proceedings of the 2000 congress on* (Vol. 2, p. 1354 -1360 vol.2).

Han, K.-H., & Kim, J.-H. (2002, December). Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Transactions on Evolutionary Computation*, *6*(6), 580–593.

Han, K.-H., & Kim, J.-H. (2003). On setting the parameters of quantum-inspired evolutionary algorithm for practical application. In *Congress on Evolutionary Computation. CEC '03* (Vol. 1, p. 178-194). Canberra, Australia: IEEE Press.

Han, K.-H., & Kim, J.-H. (2004, april). Quantum-inspired evolutionary algorithms with a new termination criterion, h epsi; gate, and two-phase scheme. *Evolutionary Computation, IEEE Transactions on*, *8*(2), 156 - 169.

Han, K.-H., Park, K.-H., Lee, C.-H., & Kim, J.-H. (2001). Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In *Evolutionary computation, 2001. proceedings of the 2001 congress on*

(Vol. 2, p. 1422 -1429 vol. 2).

Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Hebb, D. O. (Ed.). (1949). *The organization of behavior*. New York: Wiley.

Hey, T. (1999, 06). Quantum computing: an introduction. *Computing & Control Engineering Journal*, *10*, 105-112.

Hirvensalo, M. (2001). *Quantum computing*. New York, NY, USA: Springer-Verlag New York, Inc.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, *117*(4), 500–544.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press.

Hu, X., & Eberhart, R. (2002). Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Evolutionary computation, 2002. cec '02. proceedings of the 2002 congress on* (Vol. 2, p. 1677 -1681).

Huguenard, J. (2000). Reliability of axonal propagation: The spike doesn't stop here. *PNAS*, *97*(17), 9349-9350.

Iglesias, J., Eriksson, J., Grize, F., Tomassini, M., & Villa, A. E. (2005). Dynamics of pruning in simulated large-scale spiking neural networks. *Biosystems*, *79*(13), 11 - 20.

Ikegaya, Y., Matsumoto, W., Chiou, H.-Y., Yuste, R., & Aaron, G. (2008, 12). Statistical significance of precisely repeated intracellular synaptic patterns. *PLoS ONE*, *3*(12), e3983.

Indiveri, G., Chicca, E., & Douglas, R. (2009). Artificial cognitive systems: From VLSI networks of spiking neurons to neuromorphic cognition. *Cognitive Computation*, *1*, 119–127.

Indiveri, G., Stefanini, F., & Chicca, E. (2010). Spike-based learning with a generalized integrate and fire silicon neuron. In *International symposium on circuits and systems, ISCAS'10* (pp. 1951–1954).

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. on Neural Networks*, *14*(6), 1569-1572.

Izhikevich, E. M. (2004, September). Which model to use for cortical spiking neurons? *Neural Networks, IEEE Transactions on*, *15*(5), 1063–1070.

Izhikevich, E. M. (2006). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting (Computational Neuroscience)* (1st ed.). The MIT Press. Hardcover.

Izhikevich, E. M., & Edelman, G. M. (2008, March 4). Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences*, *105*(9), 3593–3598.

Jang, J. su. (2004). Face detection using quantum-inspired evolutionary algorithm. In *in proc. ieee congress on evolutionary computation* (pp. 2100–2106).

Jeong, Y.-W., Park, J.-B., Jang, S.-H., & Lee, K. (2009, nov.). A new quantum-inspired binary pso for thermal unit commitment problems. In *Intelligent system applications to power systems, 2009. isap '09. 15th international conference on* (p. 1 -6).

Jian, G., Li-juan, S., Ru-chuan, W., & Zhong-gen, Y. (2009, oct.). An improved quantum genetic algorithm. In *Genetic and evolutionary computing, 2009. wgec '09. 3rd international conference on* (p. 14 -18).

Jin, D. Z. (2004, Feb). Spiking neural network for recognizing spatiotemporal sequences of spikes. *Phys. Rev. E*, *69*, 021905.

John, G. H., Kohavi, R., & Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In *International conference on machine learning* (pp. 121–129).

Johnson, C., Venayagamoorthy, G. K., & Mitra, P. (2009). Comparison of a spiking neural network and an mlp for robust identification of generator dynamics in a multimachine power system. *Neural Networks*, *22*(5-6), 833-841.

Jones, M. (2005). *Ai application programming* (2nd ed.). Charles River Media.

Kampen, N. V. (2007). *Stochastic processes in physics and chemistry*. North Holland.

Kandel, E. R. (2000). *Principles of neural science*. McGraw-Hill Education. Paperback.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. *Techn Rep TR06 Erciyes Univ Press Erciyes*, *129*(2), 28-65.

Kasabov, N. (1996). *Foundations of neural networks, fuzzy systems, and knowledge engineering* (1st ed.). Cambridge, MA, USA: MIT Press.

Kasabov, N. (1998a). ECOS: Evolving connectionist systems and the ECO learning paradigm. In S. Usui & T. Omori (Eds.), *The Fifth International Conference on Neural Information Processing, ICONIP'98* (p. 1232-1235). Kitakyushu, Japan: IOA Press.

Kasabov, N. (1998b). Evolving fuzzy neural networks-algorithms, applications and biological motivation. In T. Yamakawa & G. Matsumoto (Eds.), *Methodologies for the Conception Design and Application of Soft Computing* (p. 271-274). Singapore: World Scientific.

Kasabov, N. (2007). *Evolving connectionist systems: The knowledge engineering approach* (Second ed.). London: Springer-Verlag.

Kasabov, N. (2008). Evolving intelligence in humans and machines: Integrative connectionist systems approach (feature article). *IEEE Computational Intelligence Magazine*, *3*(3), 23-37.

Kasabov, N. (2009). Integrative connectionist learning systems inspired by nature: current models, future trends and challenges. *Natural Computing*, *8*(2), 199-218.

Kasabov, N. (2010). To spike or not to spike: A probabilistic spiking neuron model. *Neural Networks*, *23*(1), 16-19.

Kasabov, N., Benuskova, L., & Wysoski, S. G. (2004). Computational neurogenetic modelling: gene networks within neural networks. In *Proc. ieee intl. joint conf. neural networks* (pp. 1203–1208). IEEE Press.

Kasabov, N., & Song, Q. (2002, Apr). DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems*, *10*(2), 144-154.

Kasinski, A. J., & Ponulak, F. (2006). Comparison of supervised learning methods for spike time coding in spiking neural networks. *Int. J.of Applied Mathematics and Computer Science*, *16*, 101-113.

Kempter, R., Gerstner, W., & van Hemmen, J. L. (1999, Apr). Hebbian learning and spiking neurons. *Phys. Rev. E*, *59*(4), 4498–4514.

Kennedy, J., & Eberhart, R. C. (1997). *A discrete binary version of the particle swarm algorithm* (Vol. 5).

Khan, M., Lester, D., Plana, L., Rast, A., Jin, X., Painkras, E., et al. (2008, june). Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *Neural networks, 2008. ijcnn 2008. (ieee world congress on computational intelligence). ieee international joint conference on* (p. 2849 -2856).

Khanesar, M., Teshnehlab, M., & Shoorehdeli, M. (2007, june). A novel binary particle swarm optimization. In *Control automation, 2007. med '07. mediterranean conference on* (p. 1 -6).

Khodier, M., & Christod, C. (2005, aug.). Linear array geometry synthesis with minimum sidelobe level and null control using particle swarm optimization. *Antennas and Propagation, IEEE Transactions on*, *53*(8), 2674 - 2679.

Kim, Y., Kim, J.-H., & Han, K.-H. (2006). Quantum-inspired multiobjective evolutionary algorithm for multiobjective 0/1 knapsack problems. In *Evolutionary computation, 2006. cec 2006. ieee congress on* (p. 2601 -2606).

Kistler, W. M., Seitz, R., & Hemmen, J. L. van. (1998). Modelling Collective Excitations in Cortical Tissue. *Physica D*, *114*(3/4), 273-295.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*(1-2), 273 - 324.

Kohavi, R., & Sommerfield, D. (1995). Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)* (p. 192-197). Montreal, Canada: AAAI Press.

Kojima, H., & Katsumata, S. (2009). An analysis of synaptic transmission and its plasticity by glutamate receptor channel kinetics models and 2-photon laser photolysis. In *Proceedings of the 15th international conference on advances in neuro-information processing - volume part i* (pp. 88–94). Berlin, Heidelberg: Springer-Verlag.

Lang, K. J., & Witbrock, M. J. (1988). Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School San*

Mateo, Morgan Kauffmann (Ed.) (p. 52-59). San Mateo, USA: Morgen Kauffmann Press.

Layeb, A., & Saidouni, D.-E. (2007). Quantum genetic algorithm for binary decision diagram ordering problem. *International Journal of Computer Science and Network Security*, *7*(9), 130-135.

Lee, J.-C., Lin, W.-M., Liao, G.-C., & Tsao, T.-P. (2011). Quantum genetic algorithm for dynamic economic dispatch with valve-point effects and including wind power system. *International Journal of Electrical Power and Energy Systems*, *33*(2), 189 - 197.

Lestienne, R. (1995). Determination of the precision of spike timing in the visual cortex of anaesthetised cats. *Biological Cybernetics*, *74*(1), 55-61.

Li, Shao, Z., & Qian, J. (2002). An optimizing method based on autonomous animate: Fish swarm algorithm. In *Proceeding of system engineering theory and practice* (p. 32-38).

Li, B., & Zhuang, Z. (2002). Genetic algorithm based-on the quantum probability representation. In *Proceedings of the third international conference on intelligent data engineering and automated learning* (pp. 500–505). London, UK: Springer-Verlag.

Li, B.-B., & Wang, L. (2007, june). A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *37*(3), 576 -591.

Lin, Y., & Zhongjian, T. (2011, may). Prediction of grain yield based on spiking neural networks model. In *Communication software and networks (iccsn), 2011 ieee 3rd international conference on* (p. 171 -174).

Liu, H., & Setiono, R. (1996). A probabilistic approach to feature selection - a filter solution. In *Proceedings of the 13th international conference on machine learning* (p. 319-327).

Liu, J., Perez-Gonzalez, D., Rees, A., Erwin, H., & Wermter, S. (2010). A biologically inspired spiking neural network model of the auditory midbrain for sound source localisation. *Neurocomputing*, *74*(13), 129 - 139.

Liu, Y., & Ma, Y. (2008). A new parallel algorithm of adaptive qpso to solve constrained optimization problems. In *Proceedings of the 2008 second international conference on genetic and evolutionary computing*

(pp. 451–454). Washington, DC, USA: IEEE Computer Society.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002, March). Text classification using string kernels. *J. Mach. Learn. Res., 2*, 419–444.

Loiselle, S., Rouat, J., Pressnitzer, D., & Thorpe, S. (2005, Aug). Exploration of rank order coding with spiking neural networks for speech recognition. In *IEEE International Joint Conference on Neural Networks, IJCNN '05* (Vol. 4, p. 2076-2080).

Lu, Y., Liao, Z., & Chen, W. (2007, nov.). An automatic registration framework using quantum particle swarm optimization for remote sensing images. In *Wavelet analysis and pattern recognition, 2007. icwapr '07. international conference on* (Vol. 2, p. 484 -488).

Luger, G. F. (2004). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (5th Edition ed.). Addison Wesley.

Luitel, B., & Venayagamoorthy, G. K. (2010, June). Neural networks letter: Quantum inspired pso for the optimization of simultaneous recurrent neural networks as mimo learning systems. *Neural Networks, 23*(5), 583–586.

Ma, R., Liu, Y., & Lin, X. (2007, dec.). Hybrid qpso based wavelet neural networks for network anomaly detection. In *Digital media and its application in museum heritages, second workshop on* (p. 442 -447).

Maass, W. (1994). Lower bounds for the computational power of networks of spiking neurons. *Electronic Colloquium on Computational Complexity (ECCC), 1*(19).

Maass, W. (1996). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In M. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems 9, nips, denver, co, usa, december 2-5, 1996* (p. 211-217). MIT Press.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks, 10*(9), 1659 - 1671.

Maass, W. (1999). Computing with spiking neurons. In *Pulsed neural networks* (pp. 55–85). Cambridge, MA, USA: MIT Press.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on

perturbations. *Neural Computation*, *14*(11), 2531-2560.

Maguire, L., McGinnity, T., Glackin, B., Ghani, A., Belatreche, A., & Harkin, J. (2007). Challenges for large-scale implementations of spiking neural networks on fpgas. *Neurocomputing*, *71*(13), 13 - 29.

Markram, H. (2006). The blue brain project. *Nature Rev. Neurosci.*, *7*, 153-160.

Markram, H., Lubke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*(5297), 213-215.

Mcculloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysic*, *5*, 115–133.

McGinley, B., OHalloran, M., Coneico, R. C., Morgan, F., Glavin, M., & Jones, E. (2010). Spiking neural networks for breast cancer classification using radar target signatures. *Progress in Electromagnetics Research M*, *17*, 79–94.

Meissner, M., Schmuker, M., & Schneider, G. (2006, March). Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics, 2006, 7, 125*, *7*(1), 125–136.

Mendes, R., Cortez, P., Rocha, M., & Neves, J. (2002). Particle swarms for feedforward neural network training. In *Neural networks, 2002. ijcnn '02. proceedings of the 2002 international joint conference on* (Vol. 2, p. 1895 -1899).

Meng, K., Wang, H. G., Dong, Z., & Wong, K. P. (2010, feb.). Quantum-inspired particle swarm optimization for valve-point economic load dispatch. *Power Systems, IEEE Transactions on*, *25*(1), 215 -222.

Mikki, S., & Kishk, A. (2005, july). Investigation of the quantum particle swarm optimization technique for electromagnetic applications. In *Antennas and propagation society international symposium, 2005 ieee* (Vol. 2A, p. 45 - 48 vol. 2A).

Narayanan, A. (1999). Quantum computing for beginners. In *Proceedings of Congress on Evolutionary Computation.*

Narayanan, A., & Moore, M. (1996). Quantum-inspired genetic algorithms. In *Evolutionary computation, 1996., proceedings of ieee international conference on* (p. 61-66).

Nielsen, M. A., & Chuang, I. L. (2000). *Quantum computation and quantum information*. Cambridge University Press.

Norton, D., & Ventura, D. (2006). Preparing more effective liquid state machines using hebbian learning. In *International joint conference on neural networks, ijcnn 2006* (p. 4243-4248). Vancouver, BC: IEEE.

Norton, D., & Ventura, D. (2010). Improving liquid state machines through iterative refinement of the reservoir. *Neurocomputing*, *73*(16-18), 2893 - 2904. (10th Brazilian Symposium on Neural Networks (SBRN2008))

Omran, M. G., Engelbrecht, A. P., & Salman, A. A. (2006). Particle swarm optimization for pattern recognition and image processing. In *Swarm intelligence in data mining* (p. 125-151).

Pant, M., Thangaraj, R., & Abraham, A. (2008). A new quantum behaved particle swarm optimization. In *Proceedings of the 10th annual conference on genetic and evolutionary computation* (pp. 87–94). New York, NY, USA: ACM.

Pearson, M., Melhuish, C., Pipe, A., Nibouche, M., Gilhesphy, L., Gurney, K., et al. (2005, aug.). Design and fpga implementation of an embedded real-time biologically plausible spiking neural network processor. In *Field programmable logic and applications, 2005. international conference on* (p. 582 - 585).

Pearson, M., Pipe, A., Mitchinson, B., Gurney, K., Melhuish, C., Gilhespy, I., et al. (2007, sept.). Implementing spiking neural networks for real-time signal-processing and control applications: A model-validated fpga approach. *Neural Networks, IEEE Transactions on*, *18*(5), 1472 -1487.

Peng-Yeng, Y. (2004). A discrete particle swarm algorithm for optimal polygonal approximation of digital curves. *Journal of Visual Communication and Image Representation*, *15*(2), 241 - 260.

Perrinet, L., & Samuelides, M. (2002). Sparse image coding using an asynchronous spiking neural network. In *In proceedings of esann* (pp. 313–8).

Ponulak, F. (2005). *ReSuMe – new supervised learning method for spiking neural networks* (Tech. Rep.). Poznań, Poland: Institute of Control and Information Engineering, Poznań University of Technology.

Rieke, F., Warland, D., Steveninck, R. de Ruyter van, & Bialek, W. (1999). *Spikes: exploring the neural code*. Cambridge, MA, USA: MIT Press.

Rocke, P., McGinley, B., Morgan, F., & Maher, J. (2007). Reconfigurable hardware evolution platform for a spiking neural network robotics controller. In P. Diniz, E. Marques, K. Bertels, M. Fernandes, & J. Cardoso (Eds.), *Reconfigurable computing: Architectures, tools and applications* (Vol. 4419, p. 373-378). Springer Berlin / Heidelberg.

Roggen, D., Hofmann, S., Thoma, Y., & Floreano, D. (2003, july). Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot. In *Evolvable hardware, 2003. proceedings. nasa/dod conference on* (p. 189 - 198).

Rumelhart, D., Hintont, G., & Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

SaÏghi, S., Buhry, L., Bornat, Y., N'Kaoua, G., Tomas, J., & Renaud, S. (2008, May). Adjusting the Neurons Models in Neuromimetic ICs using the Voltage-Clamp Technique. In *Proceeding of the InternationaI Symposium on Circuits And Systems 2008, IEEE* (p. 1564-1567). Seattle, United States. (pp. 1564-1567)

Saunders, C., Tschach, H., & Taylor, J. S. (2002). Syllables and other String Kernel Extensions. In *Proc. 19th international conference on machine learning (icml'02)* (pp. 530–537).

Schliebs, S., Defoin-Platel, M., & Kasabov, N. (2009a). Integrated feature and parameter optimization for an evolving spiking neural network. In M. Köppen, N. K. Kasabov, & G. G. Coghill (Eds.), *Advances in Neuro-Information Processing, 15th International Conference* (Vol. 5506, p. 1229-1236). Heidelberg, Germany: Springer.

Schliebs, S., Defoin-Platel, M., & Kasabov, N. (2009b). Integrated feature and parameter optimization for an evolving spiking neural network. In *Proceedings of the 15th international conference on advances in neuro-information processing - volume part i* (pp. 1229–1236). Berlin, Heidelberg: Springer-Verlag.

Schliebs, S., Defoin-Platel, M., & Kasabov, N. (2010, july). Analyzing the dynamics of the simultaneous feature and parameter optimization of an evolving spiking neural network. In *Neural networks (ijcnn), the 2010*

international joint conference on (p. 1-8).

Schliebs, S., Defoin-Platel, M., Worner, S., & Kasabov, N. (2009a). Integrated feature and parameter optimization for an evolving spiking neural network: Exploring heterogeneous probabilistic models. *Neural Networks*, *22*(5-6), 623 - 632.

Schliebs, S., Defoin-Platel, M., Worner, S., & Kasabov, N. (2009b). Quantum-inspired feature and parameter optimisation of evolving spiking neural networks with a case study from ecological modeling. In *International Joint Conference on Neural Networks, IEEE - INNS - ENNS* (Vol. 0, p. 2833-2840). Los Alamitos, CA, USA: IEEE Computer Society.

Schliebs, S., Nuntalid, N., & Kasabov, N. (2010). Towards spatio-temporal pattern recognition using evolving spiking neural networks. In K. Wong, B. Mendis, & A. Bouzerdoum (Eds.), *Neural information processing. theory and algorithms* (Vol. 6443, p. 163-170). Springer Berlin / Heidelberg.

Schoenauer, T., Atasoy, S., Mehrtash, N., & Klar, H. (2002, jan). Neuropipe-chip: A digital neuro-processor for spiking neural networks. *Neural Networks, IEEE Transactions on*, *13*(1), 205 -213.

Schrauwen, B., & Campenhout, J. M. V. (2006). Backpropagation for population-temporal coded spiking neural networks. In *Proceedings of the ijcnn* (p. 1797-1804).

Schrauwen, B., D'Haene, M., Verstraeten, D., & Campenhout, J. V. (2008). Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural Networks*, *21*(2-3), 511 - 523. (Advances in Neural Networks Research: IJCNN '07, 2007 International Joint Conference on Neural Networks IJCNN '07)

Schrauwen, B., & van Campenhout, J. (2004, 11). Improving SpikeProp: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC Workshop.*

Seung, H. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, *40*(6), 1063 - 1073.

Sguier, R., & Mercier, D. (2002). Audio-visual speech recognition one pass learning with spiking neurons. In J. R. Dorronsoro (Ed.), *Icann* (Vol. 2415, p. 1207-1212). Springer.

Sharma, V., & Srinivasan, D. (2010, july). A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets. In *Neural networks (ijcnn), the 2010 international joint conference on* (p. 1 -8).

Shawe-Taylor, J., & Cristianini, N. (2000). *Support vector machines and other kernel-based learning methods* (1st ed.). New York, NY, USA: Cambridge University Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. New York, NY, USA: Cambridge University Press.

Shepherd, D. J. (2006, June). On the role of hadamard gates in quantum circuits. *Quantum Information Processing*, *5*, 161–177.

Shi, Y., & Eberhart, R. (1998a). *A modified particle swarm optimizer.*

Shi, Y., & Eberhart, R. C. (1998b). Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, & A. E. Eiben (Eds.), *Evolutionary programming vii* (Vol. 160, pp. 591–600). Springer.

Shi, Y., & Eberhart, R. C. (1999). Empirical study of particle swarm optimization. In (Vol. 3, pp. 1945–1949).

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science* (p. 124-134).

Soltic, S., Wysoski, S., & Kasabov, N. (2008). Evolving spiking neural networks for taste recognition. In *IEEE World Congress on Computational Intelligence (WCCI), Hong Kong* (p. 2091-2097).

Sun, J., Feng, B., & Xu, W. (2004, june). Particle swarm optimization with particles having quantum behavior. In *Evolutionary computation, 2004. cec2004. congress on* (Vol. 1, p. 325 - 331 Vol.1).

Sun, J., Xu, W., & Fang, W. (2006). Quantum-behaved particle swarm optimization algorithm with controlled diversity. In V. Alexandrov, G. van Albada, P. Sloot, & J. Dongarra (Eds.), *Computational science iccs 2006* (Vol. 3993, p. 847-854). Springer Berlin / Heidelberg.

Talbi, H., Batouche, M., & Draa, A. (2004). A quantum-inspired genetic algorithm for multi-source affine image registration. In A. Campilho & M. Kamel (Eds.), *Image analysis and recognition* (Vol. 3211, p. 147-154). Springer Berlin / Heidelberg.

Talbi, H., Draa, A., & Batouche, M. (2006). A novel quantum-inspired evaluation algorithm for multi-source affine image registration. *Int. Arab J. Inf. Technol., 3*(1), 9-15.

Tang, L., & Xue, F.-Z. (2008). Using data to design fuzzy system based on quantum-behaved particle swarm optimization. In *Machine learning and cybernetics, 2008 international conference on* (Vol. 1, p. 624 -628).

Theodoridis, Y., & Nascimento, M. A. (2000, September). Generating spatiotemporal datasets on the www. *SIGMOD Rec., 29*, 39–43.

Theodoridis, Y., Sellis, T. K., Papadopoulos, A., & Manolopoulos, Y. (1998). Specifications for efficient indexing in spatiotemporal databases. In *Proceedings of the 10th international conference on scientific and statistical database management* (pp. 123–132). Washington, DC, USA: IEEE Computer Society.

Thorpe, S. J. (1997). How can the human visual system process a natural scene in under 150ms? On the role of asynchronous spike propagation. In *ESANN*. D-Facto public.

Thorpe, S. J., Delorme, A., & Rullen, R. van. (2001). Spike-based strategies for rapid processing. *Neural Networks, 14*(6-7), 715-725.

Thorpe, S. J., Fize, D., & Marlot, C. (1996, Jun). Speed of processing in the human visual system. *Nature, 381*, 520-522.

Thorpe, S. J., & Gautrais, J. (1998). Rank order coding. In *CNS '97: Proceedings of the 6th annual conference on Computational neuroscience: trends in research, 1998* (pp. 113–118). New York, NY, USA: Plenum Press.

Toffoli, T. (1980). Reversible computing. In *Proceedings of the 7th colloquium on automata, languages and programming* (pp. 632–644). London, UK: Springer-Verlag.

Valko, M., Marques, N. C., & Castelani, M. (2005). Evolutionary feature selection for spiking neural network pattern classifiers. In B. et al. (Ed.), *Proceedings of 2005 Portuguese Conference on Artificial Intelligence* (pp. 24–32). IEEE Press.

Valle, Y. del, Venayagamoorthy, G., Mohagheghi, S., Hernandez, J.-C., & Harley, R. (2008, april). Particle swarm optimization: Basic concepts, variants and applications in power systems. *Evolutionary Computation,*

IEEE Transactions on, *12*(2), 171 -195.

Verstraeten, D., Schrauwen, B., D'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*(3), 391 - 403.

Verstraeten, D., Schrauwen, B., & Stroobandt, D. (2005). Isolated word recognition using a liquid state machine. In *ESANN* (p. 435-440).

Vlachogiannis, J. G., & stergaard, J. (2009). Reactive power and voltage control based on general quantum genetic algorithms. *Expert Systems with Applications*, *36*(3, Part 2), 6118 - 6126.

Wachowiak, M., Smolikova, R., Zheng, Y., Zurada, J., & Elmaghraby, A. (2004, june). An approach to multimodal biomedical image registration utilizing particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, *8*(3), 289 - 301.

Waibel, A. (1989, March). Modular construction of time-delay neural networks for speech recognition. *Neural Comput.*, *1*, 39–46.

Wang, H., Yang, S., Xu, W., & Sun, J. (2007). Scalability of hybrid fuzzy c-means algorithm based on quantum-behaved pso. In *Fuzzy systems and knowledge discovery, 2007. fskd 2007. fourth international conference on* (Vol. 2, pp. 261–265).

Wang, J., & Zhou, Y. (2007). Hybrid quantum particle swarm optimization algorithm for combinatorial optimization problem. In *Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 185–185). New York, NY, USA: ACM.

Wang, L., Zhang, M., Niu, Q., & Yao, J. (2011). A modified quantum-inspired particle swarm optimization algorithm. In H. Deng, D. Miao, J. Lei, & F. Wang (Eds.), *Artificial intelligence and computational intelligence* (Vol. 7004, p. 412-419). Springer Berlin / Heidelberg.

Watts, M. (2009, May). A decade of Kasabov's evolving connectionist systems: A review. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *39*(3), 253-269.

Willms, A. R., Baro, D. J., Harris-Warrick, R. M., & Guckenheimer, J. (1999). An improved parameter estimation method for hodgkin-huxley models. *Journal of Computational Neuroscience*, *6*, 145-168.

Wootters, W. K., & Zurek, W. H. (1982). A single quantum cannot be cloned. *Nature*, *299*(5886), 802–803.

Wu, Q., McGinnity, T. M., Maguire, L., Valderrama-Gonzalez, G. D., & Dempster, P. (2010). Colour image segmentation based on a spiking neural network model inspired by the visual system. In *Proceedings of the 6th international conference on advanced intelligent computing theories and applications: intelligent computing* (pp. 49–57). Berlin, Heidelberg: Springer-Verlag.

Wysoski, S. G., Benuskova, L., & Kasabov, N. (2006b). On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition. In *Artificial Neural Networks ICANN 2006* (p. 61-70). Berlin / Heidelberg: Springer.

Wysoski, S. G., Benuskova, L., & Kasabov, N. (2007). Text-independent speaker authentication with spiking neural networks. In *ICANN (2)* (Vol. 4669/2007, p. 758-767). Berlin / Heidelberg: Springer.

Wysoski, S. G., Benuskova, L., & Kasabov, N. K. (2006a). Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. In *Advanced Concepts for Intelligent Vision Systems* (p. 1133-1142). Berlin / Heidelberg: Springer.

Xiao, J., Xu, J., Chen, Z., Zhang, K., & Pan, L. (2009). A hybrid quantum chaotic swarm evolutionary algorithm for dna encoding. *Computers and Mathematics with Applications*, *57*(11-12), 1949 - 1958. (¡ce:title¿Proceedings of the International Conference¡/ce:title¿ ¡xocs:full-name¿Bio-Inspired Computing-Theories and Applications BIC-TA 2007 Zhengzhou, China¡/xocs:full-name¿)

Xie, X., & Seung, H. S. (2004, Apr). Learning in neural networks by reinforcement of irregular spiking. *Phys. Rev. E*, *69*(4), 041909.

Xin, J., & Embrechts, M. (2001). Supervised learning with spiking neural networks. In *International Joint Conference on Neural Networks, IJCNN '01* (Vol. 3, p. 1772-1777). IEEE Press.

Xin-She, Y. (2009). Firefly algorithms for multimodal optimization. In O. Watanabe & T. Zeugmann (Eds.), *Stochastic algorithms: Foundations and applications, 5th international symposium, saga 2009, sapporo, japan, october 26-28, 2009. proceedings* (Vol. 5792, p. 169-178).

Springer.

Yang, J. an, Li, B., & Zhuang, Z. (2003, dec.). Multi-universe parallel quantum genetic algorithm its application to blind-source separation. In *Neural networks and signal processing, 2003. proceedings of the 2003 international conference on* (Vol. 1, p. 393 - 398 Vol.1).

Yang, S., Wang, M., & Jiao, L. (2004, june). A quantum particle swarm optimization. In *Evolutionary computation, 2004. cec2004. congress on* (Vol. 1, p. 320 - 324 Vol.1).

Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., & Nakanishi, Y. (2000, nov). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *Power Systems, IEEE Transactions on*, *15*(4), 1232 -1239.

You, X., Liu, S., & Shuai, D. (2006). On parallel immune quantum evolutionary algorithm based on learning mechanism and its convergence. In L. Jiao, L. Wang, X.-b. Gao, J. Liu, & F. Wu (Eds.), *Advances in natural computation* (Vol. 4221, p. 903-912). Springer Berlin / Heidelberg.

Yu, Y., Tian, Y., & Yin, Z. (2006, may). Hybrid quantum evolutionary algorithms based on particle swarm theory. In *Industrial electronics and applications, 2006 1st ieee conference on* (p. 1 -7).

Yuan, X., Nie, H., Su, A., Wang, L., & Yuan, Y. (2009). An improved binary particle swarm optimization for unit commitment problem. *Expert Systems with Applications*, *36*(4), 8049 - 8055.

Zainud-Deen, S., Hassen, W., Ali, E., Awadalla, K., & Sharshar, H. (2008, march). Breast cancer detection using a hybrid finite difference frequency domain and particle swarm optimization techniques. In *Radio science conference, 2008. nrsc 2008. national* (p. 1 -8).

Zhang, Jin, W., & Li, N. (2003). An improved quantum genetic algorithm and its application. In *Proceedings of the 9th international conference on rough sets, fuzzy sets, data mining, and granular computing* (pp. 449–452). Berlin, Heidelberg: Springer-Verlag.

Zhang, Li, N., Jin, W.-d., & Hu, L.-z. (2006). Novel quantum genetic algorithm and its applications. *Frontiers of Electrical and Electronic Engineering in China*, *1*, 31-36.

Zhang, Shao, H., & Li, Y. (2000). Particle swarm optimisation for evolving artificial neural network. *Systems, Man, and Cybernetics, 2000 IEEE International Conference on, 4*.

Zhang, H., Zhang, G., Rong, H., & Cheng, J. (2010, aug.). Comparisons of quantum rotation gates in quantum-inspired evolutionary algorithms. In *Natural computation (icnc), 2010 sixth international conference on* (Vol. 5, p. 2306 -2310).