INVESTIGATIONS IN QUANTUM INSPIRED NEURAL NETWORK LEARNING ALGORITHMS

A THESIS

submitted in partial fulfillment of the requirements for the award of the degree

of

DOCTOR OF PHILOSOPHY

by

OM PRAKASH PATEL

DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MARCH 2018

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **INVESTIGATIONS IN QUANTUM INSPIRED NEURAL NETWORK LEARNING ALGORITHMS** in the partial fulfillment of the requirement for the award of the degree of **DOCTOR OF PHILOSOPHY** and submitted in the **DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING**, **Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from JANUARY, 2014 to MARCH, 2018 under the supervision of Dr. Aruna Tiwari, Associate Professor, Discipline of Computer Science & Engineering, Indian Institute of Technology Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the Student with date

(OM PRAKASH PATEL)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of the Thesis Supervisor with date

(Dr. ARUNA TIWARI)

OM PRAKASH PATEL has successfully given his Ph.D. Oral Examination held on 15/10/2018.

Signature of Chairperson (OEB)	Signature of External Examiner	Signature of Thesis Supervisor
Date:	Date:	Date:
Signature of PSPC Member #1	Signature of PSPC Member #2	Signature of Convener, DPGC
Date:	Date:	Date:

Signature of Head of Discipline

Date:

Acknowledgements

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisor Dr. Aruna Tiwari, who was a constant source of inspiration during my work, without her constant guidance and research directions this research work could not be completed. Her continuous support and encouragement has motivated me to remain streamline in my research work. I am grateful to HOD of Computer Science for all his extended help and support.

I am thankful to Dr. Surya Prakash and Dr. Vivek Kanhangad, my research committee member for taking out some valuable time to evaluate my progress all these years. Their good comments and suggestions helped me to improve my work at various stages.

I especially thanks to Dr. Kapil Ahuja for his continuous support and encouragement during my work. I wish to thank all my colleagues, and staff from the Discipline of Computer Science and Engineering for their suggestions and friendship. I extend my sincere thanks to a government body Madhya Pradesh Council of Science and Technology (MPCST) apart from IIT Indore to help me with financial support to attend international conferences, which gave me a right platform at the right time and helped a lot to groom my research work.

I would like to thank my family, especially my mother, father, and my wife for always believing in me, for their continuous source of inspiration and their support in my decisions. Without whom I could not have made it here. I also want to thank my in-laws for their support and blessings.

Dated: (Om Prakash Patel)

Dedicated to my parents

Abstract

Artificial Neural Network (ANN) is one of the most popular and promising areas of research in artificial intelligence. ANN has been widely used for the classification task due to its characteristic of massive parallelism, learning ability, generalization ability, and fault tolerance. For solving classification task, many models have been used like Backpropagation, Perceptron, Recurrent Neural Network and these models have been successfully applied in several fields like economics, defense, stock market, engineering, and medical. The neural network is formed using several learning parameters like connection weights, the threshold of the neuron, number of layers, number of hidden layer neurons. Finding optimal classification results need the optimal value of these learning parameters. Several training techniques have been proposed to find the optimal value of these parameters within different neural network architectures. However, it is still an open area of research to find the optimal value of neural network parameters. The evolutionary algorithms like genetic algorithm, particle swarm optimization, ant colony optimization are also used by many researchers to find the optimal value of neural network learning parameters. Recently the quantum evolutionary algorithm (QEA) has been applied to many classification problems and producing very promising results. QEA is a population-based probabilistic EA that integrates concepts from quantum computing for higher representation power and robust search. QEAs are characterized by population dynamics, individual representation, and evaluation function. Initially, QEA can represent diverse individuals probabilistically because a quantum bit (Q) individual is made up of several qubits (q) represents the linear superposition of all possible states with the same probability. The observation process used here gives a large search space to find the optimal value of required parameter. The quantum rotational gate provide exploitation to restrict the algorithm to stuck with

the problem of local minima and maxima.

We proposed neural network architecture using quantum computing concept. The connection weights are evolved first using evolutionary quantum computing concept and neural network is formed constructively by adding neurons in the hidden layer one by one. However, the threshold of neurons has been decided manually which may lead solution to local minima and maxima problem. Therefore the work has been extended by evolving threshold of neuron along with connection weights. Finding a range of search space is also an important issue. Therefore, to evolve threshold of neuron optimally, the existing work is enhanced and threshold boundary parameter is proposed. The algorithm has been used to classify offline signature dataset.

For complex datasets, having noisy samples there are chances of overlapping in samples from multiple classes. In order to handle such problem, a neural network architecture using quantum and fuzzy concept has been proposed for two-class dataset having overlapped samples. The fuzzy concept has been used to evolve connection weight or learning of neural network whereas the fuzzy algorithm is optimized using evolutionary quantum computing concept. The fuzzifier parameter which decided overlapping between clusters has been evolved using evolutionary quantum computing concept. The fuzzy concept has one more parameter that is cluster centroids, which is generally initialized randomly. This work has been extended by proposing a quantum-inspired fuzzy based neural network learning algorithm for multi-class dataset having overlapped samples. In this, along with fuzzifier parameter, the cluster centroids which act as connection weights in the neural network are being evolved using quantum computing concept.

To deal with complex dataset like image dataset, web dataset, face reorganization object identification, and speech reorganization, deep neural network are proposed. However, the learning algorithm of deep neural network has some parameter like learning rate parameter which is initialized manually. We proposed quantum inspired deep neural network using stacked auto-encoder to solve the problem of classification of complex dataset. The learning algorithm of stacked auto-encoder has been optimized using evolutionary quantum computing concept.

The proposed algorithms are tested on benchmark datasets along with one

real life dataset that is offline signature dataset which is prepared manually. The proposed algorithms are compared with other state-of-the-art approaches and it is found that, proposed algorithms perform well in comparison to other state-of-the-art approaches. The proposed algorithms perform better due to optimizing their learning parameter using the evolutionary quantum computing concept. The quantum computing concept is characterized by population dynamics, individual representation, evaluation function. It provides a large search space to find the optimal value of a parameter using an observation process thus, exploration is achieved. On the other hand, the quantum rotational gate provides exploitation to evolve the optimal value of neural network parameters.

List of Publications

(A) International Journal

(i) Published

[1] O. P. Patel, A. Tiwari, R. Chaudhary, S. V. Nuthalapati, N. Bharill, M. Prasad, F. K. Hussain, and O. K. Hussain, Enhanced Quantum based Neural Network Learning and its Application to Signature Verification, Soft Computing, Springer, DOI: 10.1007/s00500-017-2954-3, ISSN: 1433-7479, pp. 1-14, 2017. (SCIE Index, Impact Factor: 2.47)

[2] O. P. Patel, A. Tiwari, and V. Bagade, Quantum inspired Stacked Auto-encoder based Deep Neural Network Algorithm (Q-DNN), Arabian Journal for Science and Engineering, Springer, DOI: 10.1007/s13369-017-2907-2, ISSN: 2193-567X, pp. 1-15, 2017. (SCIE Index, Impact Factor: 0.865)

[3] O. P. Patel and A. Tiwari, Novel Quantum inspired Binary Neural Network Algorithm, Sadhana - Academy Proceedings in Engineering Sciences, Springer, DOI: 10.1007/s12046-016-0561-0, ISSN: 0256-2499, vol. 41, no. 11, pp. 12991309, 2016. (SCIE Index, Impact Factor: 0.465)

(ii) Communicated (Under Review)

[1] O. P. Patel, N. Bharill, A. Tiwari, M. Prasad, M. Pratama, O. Kaiwartya, Y. Cao, and A. Saxena, A Novel Quantum inspired Fuzzy Based Neural Network for Data Classification, IEEE Transactions on Fuzzy Systems. (Second Revision Submitted 26 Nov., 2017)

[2] O. P. Patel, A. Tiwari, O. Gupta, V. Patel, M. Prasad, and F. K. Hussain Advanced Quantum based Neural Network Classifier and its Application for Firewall to Detect Malicious Web Request, Neurocomputing, Elsevier. (Submission Date - 14 Oct., 2017)

(B) Book Chapters

[1] O. P. Patel and A. Tiwari, Quantum based Learning with Binary Neural Network, e-book: Proceedings of the International Conference on Computational Intelligence in Data Mining (CIDM 2014), Springer-Verlag Berlin Heidelberg, DOI 10.1007/978-81-322-2208-8-43, ISBN 978-81-322-2208-8, vol. 2, pp. 473-482, 2014.

[2] O. P. Patel and A. Tiwari, Liver Disease Diagnosis using Quantum based Binary Neural Network Learning Algorithm, e-book: Proceedings of Fourth International Conference on Soft Computing for Problem Solving (SocProS 2014), Springer-Verlag Berlin Heidelberg, DOI 10.1007/978-81-322-2220-0-34, ISBN 978-81-322-2220-0, vol. 2, pp. 425-434, 2015.

(C) International Conferences

[1] O. P. Patel, A. Tiwari, O. Gupta, and V. Patel, A Quantum based Neural Network Classifier and its application for Firewall to Detect Malicious Web Request, IEEE Symposium Series on Computational Intelligence, Cape Town, South Africa, pp. 67 - 74, Dec. 2015. (Flagship Conference, Link: http://ieee-ssci.org.za:8080/)

[2] O. P. Patel and A. Tiwari, Advance Quantum inspired Binary Neural Network Algorithm, 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2015), Sponsored by the IEEE Computational Intelligence Society (CIS), Takamatsu, Japan, pp. 1-6, June 2015. (RANK C, Link: http://www.conferenceranks.com/)

[3] O. P. Patel and A. Tiwari, Quantum inspired Binary Neural Network Algorithm, 13th International Conference on Information Technology (ICIT), Bhubaneswar, India, IEEE, pp. 270-274, Dec. 2014. (Double-Blind Review Conference)

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ANN	Artificial Neural Network
EAs	Evolutionary Algorithms
RNN	Recurrent Neural Network
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
ABCA	Artificial Bee Colony Algorithm
CSA	Cuckoo Search Algorithm
QEAs	Quantum inspired Evolutionary Algorithms
Q-BNN	Quantum inspired Binary Neural Network Learning Algorithm
NQ-BNN	Novel Quantum inspired Binary Neural Network Learning Algorithm
Q-FNN	Quantum inspired Fuzzy based Neural Network Learning Algorithm
Q-FNNM	Quantum inspired Fuzzy based Neural Network Learning Algorithm for Multi-class
Q-DNN	Quantum inspired Stacked Auto-Encoder based Neural Network Learning Algorithm
CPON	Class Probability Output Networks
EQNN-S	Enhanced Quantum inspired Neural Network Learning Algorithm and its Application as Offline Signature Verification
RBF	Radial Basis Function
EC	Evolutionary Computation
EANN	Evolutionary Artificial Neural Networks
GNARL	GeNeralized Acquisition of Recurrent Links
FCM	Fuzzy C-Means

NFS	Neuro-fuzzy System
FNN	Fuzzy Neural Network
FRPCA	Fuzzy Robust Principal Component Analysis
MSE	Mean Squared Error
RGB	Red Green Blue
PPV	Positive Predicted Value
NPV	Negative Predicted Value
GB	GigaByte
CPU	Central Processing Unit
RAM	Random Access Memory
GHz	GigaHertz
UCI	University of California, Irvine
PID	PIMA Indian Diabetes
WBCD	Breast Cancer Wisconsin Original
BUPA	British United Provident Association
QNN	Quantum-based Algorithm for Optimizing Artificial Neural Network
EW	Equal-Width
EF	Equal-Frequency
IEM	Information Entropy Maximization
CAIM	Class-Attribute Interdependence Maximization
CACC	Class-Attribute Contingency Coefficient
CAIA	Class Attribute Interval Average
SVM	Support Vector Machine
QBNN-L	Liver Disease Diagnosis using Quantum based Binary Neural Network Learning Algorithm
NCLS	Number of Correctly Learned Samples
SOM-INN	Self-Organizing Maps and Informative Nearest Neighbor
SCSR	Simultaneous Clustering and Classification Over Cluster Structure Representation
GaX	Crossover based on the Genetic Algorithm
SaX	Crossover based on Simulated Annealing
GP	Genetic Programming
GONN	Genetically Optimized Neural Network

MNIST	Modified National Institute of Standards and Technology Database
FC-WTA	Fully-Connected Winner-Take-All
AQ-BNN	Advance Quantum based Binary Neural Network Learning Algorithm
MLP	Multi-layer Perceptron
PMT	Pixel Matching Technique
OSVNN	Online Signature Verification Neural Network

Chapter 1

Introduction

Classification is an ordered set of related categories used to group data according to its similarities. There are several real-life situations where we can see that the concept of classification is being utilized such as classification of medical data for disease diagnosis, classification of biometric identity, vehicle recognition, and face recognition [1, 2]. In some areas like disease diagnosis, defense, and automated airline operations the high accuracy and reliability is required which cannot be achieved by simple programming based on conditions or human interaction. For solving such classification problems, several soft computing techniques like ANN, fuzzy clustering, deep neural network learning algorithms, and EAs inspired by nature have been proposed in the last decades [3–9].

The ANN gained popularity in recent decades and successfully applied to various problems such as pattern classification, pattern matching, associative memories, optimization, and function approximation [10, 11]. Several architectures have been proposed like Perceptron, Backpropagation, and RNN to solve the problem from various fields like mathematics, medicine, economics, computer science, image classification like face recognitions, and many more. The performance of the neural network in the mentioned areas depends upon several parameters such as network architecture, input data, the number of neurons in the hidden layer, the number of hidden layers, activation function, and the connection weights [12–15]. Finding an optimal neural network structure is an open area of research. Since the last decade nature inspired EAs like GA, PSO, ACO, ABCA, CSA, and QEA have been successfully applied to optimize neural network parameters [16–23]. EAs are principally a stochastic search and

optimization algorithm based on the principles of natural biological evolution. EAs are robust, global, and may be applied without resource to domain-specific heuristics. EAs operate on a population of potential solutions, using the principle of survival of the fittest to produce successively better approximations to a solution. At each generation of the EA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and reproducing them using variation operators. This process may lead to the evolution of populations of individuals that are better suited to their environment than the individuals from which they were created, just as in natural adaptation. EAs are characterized by the representation of the individual, the evaluation function representing the fitness level of the individuals, and the population dynamics. These components should be appropriately designed to have a proper balance between exploration and exploitation.

Recently the QEA has been applied to many classification problems and producing very promising results [11, 23]. QEA is a population-based probabilistic EA that integrates concepts from quantum computing for higher representation power and robust search. QEA are characterized by population dynamics, individual representation, evaluation function. Initially, QEA can represent diverse individuals probabilistically because a quantum bit (Q) individual is made up of several qubits (q) represents the linear superposition of all possible states with the same probability. As the probability of each qubit approaches either 1 or 0 by the quantum rotational gate, the qubit individual converges to a single state, and the diversity property disappears gradually. By this inherent mechanism, QEA can treat the balance between exploration and exploitation. In this research work, we have proposed neural network learning algorithms for solving classification problems of two class and multi-class by evolving learning parameters using the quantum computing concept. The proposed learning algorithms are tested on benchmark datasets along with one real life dataset. The proposed algorithms perform better when compared to other state-of-the-art approaches in terms of classification accuracy.

1.1 Motivation and Scope

This thesis is a study of neural network learning algorithms using the quantum computing concept. Learning of neural network for solving various classification problems is an open area of research since last decades. In other words, it is found that the, performance of the neural network learning algorithms depends on many parameters like the selection of connection weights, threshold, number of hidden layer neurons, the architecture of neural network, learning rate parameters, etc. The classification accuracy is an important factor in some cases like disease diagnosis where wrong classification results may lead to incorrect treatment. It is also observed that, the performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes. This causes due to occurrence of overlapped samples in multiple classes of dataset. In such cases, forming a neural network structure required different learning strategies. Now a days the complexity of datasets is increasing day by day. The accurate classification of such dataset is not possible through traditional neural network learning algorithm. Thus, in this thesis novel neural network learning algorithms are proposed.

Recently QEA gained attraction of many researchers. QEA is based on the concept and principles of quantum computing, such as a quantum bit and superposition of states. Like other EAs, QEA is also characterized by the representation of the individual, the evaluation function, and the population dynamics. However, instead of binary, numeric, or symbolic representation, QEA uses a quantum bit (Q) which is made of several qubits (q) and shows probabilistic representation. A quantum bit (Q) individual is defined by a string of qubits (q). The quantum bit (Q) individual has the advantage that it can represent a linear superposition of states (binary solutions) in a search space probabilistically. Thus, the quantum bit (Q) representation has a better characteristic of population diversity than other representations. A quantum bit (Q) is also defined as a variation operator of QEA to drive the individuals toward better solutions and eventually to a single state. Initially, QEA can represent diverse individuals probabilistically because a quantum bit (Q) individual represents the linear superposition of all possible states with the same probability. As the probability of each qubit (q) approaches either 1 or 0 by the quantum rota-

tiongate, the qubit (q) individual converges to a single state and the diversity property disappears gradually. By this inherent mechanism, QEA can treat the balance between exploration and exploitation. In this thesis, the QEA is used in optimizing the different type of neural network learning parameters which helps to get better classification accuracy. Thus, based on quantum computing concept, various novel neural network learning algorithm are proposed to tackle two class, multi-class as well as complex dataset.

1.2 Objectives

The research work is addressed to design neural network learning algorithms for solving classification problem which make use of quantum computing concept to tackle issue of parameters optimization. It involves finding optimal connection weights and threshold of the neurons in the neural network formed. The performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes [24, 25]. This causes due to occurrence of overlapped samples in multiple classes of dataset. Thus, there is a need to apply fuzzy concept for handling such multiple class dataset which are having overlapped samples. Further, algorithms are proposed herein to handle the problem of overlapped samples in classification problems. For the classification of complex and large dataset like images, signal, voice, deep neural network architectures have been proposed [7, 26–28]. However, there are some learning parameters in deep neural network learning algorithm which are initialized manually. One of the deep neural network architecture based on stacked auto-encoders is enhanced by proposing optimization of its learning parameters using the quantum computing concept. Related literature has been investigated and based on it; following objectives were identified in our research:

1. To form a neural network with optimal connection weights, we propose a quantum inspired neural network learning algorithm. In this, connection weights of the neural network are evolved using the quantum computing concept.

2. To form a neural network with the optimal value of threshold along with optimal connection weights, we propose another novel quantum inspired

neural network learning algorithm.

3. To propose and develop a quantum inspired fuzzy based neural network learning algorithm for solving two class classification problem. The fuzzy concept is utilized to handle the samples which are overlapped to different class regions. In this, the fuzzifier parameter is evolved using the quantum computing concept.

4. To propose and develop a quantum inspired fuzzy based neural network learning algorithm for solving multi-class problem which can also handle the issue of overlapped samples. The problem of overlapping of samples in multi-class dataset is solved using the fuzzy concept. In this, neurons are visualized as fuzzy clusters of data samples. The centroids of such clusters are optimized using the quantum computing concept.

5. To propose and develop a quantum inspired stacked auto-encoder based deep neural network learning algorithm for handling complex dataset. It involves the optimization of learning rate parameter using the quantum computing concept.

6. To propose and develop an enhanced quantum inspired neural network learning algorithm with the concept of threshold boundary parameter for optimizing the threshold of neurons in the neural network. The proposed algorithm is applied for classification of the offline signature dataset.

1.3 Contributions

In this research work, we address the number of issues related to the optimization of neural network parameters in various learning algorithms which are based on the quantum computing concept. The details are presented as follows:

1. Proposed Quantum inspired Binary Neural Network Learning Algorithm (Q-BNN) for solving two class classification problem. This algorithm uses the quantum computing concept to evolve the connection weights from a large search space. The number of hidden layer neurons is decided constructively by adding neurons in the hidden layer as per the requirement.

2. Proposed a Novel Quantum inspired Binary Neural Network Algorithm (NQ-BNN) for solving two class classification problem. In this work, we have enhanced our proposed Q-BNN by evolving threshold of neurons along with connection weights of the neural network using the quantum computing concept.

3. Proposed Quantum inspired Fuzzy based Neural Network Learning Algorithm (Q-FNN) for solving two class classification problem. The proposed algorithm forms three layer neural network structure and uses the fuzzy concept to handle the samples which are overlapped to different class regions. The fuzzy concept helps to the classify such samples by assigning membership degree according to the class from which these samples belong. However, there is parameter in the fuzzy concept like fuzzifier parameter which is initialized randomly. Hence, fuzzifier parameter is evolved using the quantum computing concept.

4. Proposed Quantum inspired Fuzzy based Neural Network Learning Algorithm for Multi-class classification problem (Q-FNNM). In this, Q-FNN algorithm is further enhanced here for solving multi-class problem which can also handle the issue of overlapped samples. The problem of overlapping of samples in multi-class dataset is solved using the fuzzy concept by assigning membership degree to the samples according to class from which the samples belong. In the proposed Q-FNNM, both connection weights which are taken as cluster centroids and fuzzifier parameter (m) have been optimized using the quantum computing concept.

5. Proposed a Quantum inspired Stacked Auto-Encoder based Neural Network Learning Algorithm (Q-DNN), to classify the complex dataset. The learning algorithm of stacked auto-encoder is optimized by evolving the learning rate parameter using the quantum computing concept.

6. Proposed an Enhanced Quantum inspired Neural Network Learning Algorithm (EQNN-S). In this work, we have enhanced our proposed NQ-BNN algorithm by proposing threshold boundary parameter for evolving threshold of the neurons optimally. The proposed EQNN-S has been used to classify offline signature dataset.

1.4 Organization of the Thesis

In this thesis, we proposed several novel neural network learning algorithm by making use of the quantum computing concept to optimize the learning process. This thesis is carried out an extensive literature review.

Chapter 2 addresses the background study of previous and related work in the field of optimization of parameters in the neural network learning algorithms. This chapter starts with neural network architectures and its parameters. After that, an overview is given of the recent and existing methodologies for optimization of neural network parameters. A survey of quantum-inspired evolutionary algorithm is discussed. This chapter also introduced the discussion of the fuzzy based neural network and similar methodologies and deep neural network algorithms. A survey related to offline signature verification methods is also discussed.

Chapter 3, incorporates the discussion of proposed Q-BNN algorithm. The proposed quantum inspired learning algorithm helps to find optimal connection weights. This chapter also discusses the issue of selection of the optimal threshold. The enhanced Q-BNN algorithm and proposed NQ-BNN to finds the threshold of neurons along with connection weights. At the end of this chapter, the performance of the proposed work is compared with other state-of-the-art approaches.

In chapter 4, we have proposed one more novel neural network learning algorithm named as Q-FNN for two-class classification problems. The fuzzy concept is used to form neural network architecture. The fuzzy algorithm uses a fuzzifier parameter which is initialized manually. The random initialization of fuzzifier parameter may leads to solution in local maxima problem. Therefore, the quantum computing concept has been utilized to evolved fuzzifier parameter. The chapter, finally reports the experimental evaluation, which compares the classification accuracy with other state-of-the-art approaches.

In chapter 5, we have proposed the Q-FNNM for multi-class classification problem which can also handle the issue of overlapped samples. The problem of overlapping of samples in multi-class dataset is solved using the fuzzy concept. In this, neurons are visualized as fuzzy clusters of data samples. The cluster centroids are evolved using the quantum computing concept rather than initial-

izing randomly. At the end of the chapter, the result of the proposed algorithm is compared with other related state-of-the-art approaches.

In chapter 6, we have proposed a Q-DNN algorithm. We form a deep neural network architecture using auto-encoders. The learning of deep neural network is done in two phases, i.e., the first phase is for the pre-training of each auto-encoder and the second phase is for the fine-tuning process. The learning algorithm used here is a gradient descent algorithm. In this, learning rate parameter of the gradient descent algorithm has been optimized using the quantum computing concept. The comparative evaluation of the proposed algorithm is done with some other deep neural network models and shallow architecture neural network models on benchmark dataset along with complex image dataset.

In chapter 7, we have proposed a enhanced quantum inspired neural network learning algorithm which is used for offline signature verification. In this proposed algorithm a threshold boundary parameter is introduced to get optimal value of threshold. The proposed algorithm is used for offline signature verification. In this, we have extracted unique features of an offline signature like pixel density, number of loops, and angle with horizontal. The classification of these features has been done using the enhanced quantum inspired neural network learning algorithm. The result of proposed algorithm is compared with other state-of-the-art approaches.

In chapter 8, conclusions are drawn regarding the research results of each of the addressed problems and the overall work in the thesis. The contributions to the state-of-the-art of the tackled subjects are outlined, and some future work directions are also highlighted.

Chapter 8 is followed with the references, appendix A, and appendix B. Appendix A consist the discussion about the benchmark dataset along with one real life dataset that is offline signature dataset. Appendix B consist various validation methods.

Chapter 2

Literature Survey

In this chapter, the review has been carried out for the related work. This chapter starts with the discussion of some well-known models of the neural network. Then we discuss about the learning algorithms of the neural network which helps to decide the parameters of the neural network like connection weights, threshold, number of hidden layer neurons during learning. After that, we discuss how the EA optimizes neural network architecture and its parameters. Further, the quantum computing concept is discussed which is utilized in the quantum inspired evolutionary algorithm to form a neural network architecture. The performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes. This causes due to the occurrence of overlapped samples in multiple classes of the dataset. Thus, there is a need to apply the fuzzy concept for handling such multiple classes dataset which are having overlapped samples. Thus, our discussion proceeds with the basics of the fuzzy concept and issues related to its parameters are discussed further in detail. For the classification of complex and large dataset like images, signal, and voice, several deep neural network models have been proposed. One of the models is stacked auto-encoder based deep neural network which is enhanced by optimizing its learning algorithm using the quantum computing concept in our proposed work. Therefore, we discuss about stacked auto-encoder based deep neural network and issue related to its learning algorithms. The proposed algorithms are tested on various benchmark datasets and one of the algorithms is tested on real-life dataset that is an offline signature dataset. For the experimental evaluation of the proposed algorithms, the performance measures are

presented.

2.1 Neural Network Concepts and Learning Algorithms

Neural Networks models are inspired by the working of the brain, although they do not pretend to be the accurate models of the central nervous system. Even if they are biologically inspired systems, they are best regarded as primarily nonlinear statistical models [29]. The neural network can be considered as a combination of neurons and synaptic connections, which are capable of transmitting data through multiple layers. Several researchers have proposed neural network learning models in last eight decade. First work was carried out by Warren McCulloch and Walter Pitts in 1943 [30]. They proposed a basic neural network structure, which consists of three layers, i.e., an input layer, a hidden layer, and an output layer. For learning of this neural network in 1949 Donald Hebb demonstrate the rule for updating connection strengths (connection weights) between neurons [31]. His rule, now called Hebbian learning, remains an influential model to this day. Hebbian learning methods were enhanced by Bernie Widrow, named as networks adalines, and by Frank Rosenblatt (1962) with his Perceptrons [32, 33]. The connection weights are updated by mapping input samples and desired output. Perceptron performs well with linearly separable classes, but its performance decreases with non-linear separable classes. In the mid-1980s at least four different groups reinvented the Backpropagation learning algorithm first found in 1969 by Bryson and Ho [34]. Backpropagation is a method used in neural networks to calculate the error contribution of each neuron after a batch of data is processed. In the context of learning, Backpropagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function. This technique is also sometimes called backward propagation of errors because the error is calculated at the output and distributed back through the network layers. However, there are some parameters in the Backpropagation learning algorithm whose value is initialized manually. Even on increasing the number of hidden layers it faces vanishing gradient problem. Due to vanishing gradient

problem the learning performance decreases. In the Backpropagation learning algorithm finding the number of hidden layers, and the number of neurons to get optimal results is also an open issue. In 1988, Broomhead and Lowe developed Radial Basis Function (RBF) an ANN which uses radial basis functions as activation functions [35]. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. It typically has three layers: an input layer, a hidden layer with a non-linear RBF activation function, and a linear output layer. In this learning model, the crisp clustering is used to form hidden layer neurons. This causes inefficient learning with multiple class dataset in which samples belong to more than one classes [24, 25]. This causes due to the occurrence of overlapped samples in multiple classes of the dataset. This problem is resolved by researchers using the fuzzy concept to form hybrid neural networks based Neuro-fuzzy concept [36–38]. The fuzzy concept helps to the classify such samples by assigning membership degree according to the class from which these samples belong. However, there are some parameters in the fuzzy concept like fuzzifier parameter which controls the extent of overlapping among fuzzy clusters [39] and initial cluster centroids which are initialized manually. Initializations of these parameters causes the decrement in the performance of neuro-fuzzy based learning algorithms. The complexity of dataset increases by the time. The researchers have shown that multilayered architectures trained using unsuitable learning algorithms not performed well when dealing with complex datasets such as image, signals, speech, video, and Web data [7,40]. To deal with such complex problem deep neural network learning algorithms were proposed [7]. However, these learning algorithms initialize its learning parameter randomly which may or may not give optimal results.

With the development of neural network learning algorithms, the performance of these algorithms became the major issue. The performance of learning algorithms depends on several parameters like connection weight, the threshold of neurons, the number of hidden layer neurons, and neural network architecture. Several new techniques have been proposed to optimize these parameters. However, the most successful technique used to optimize these parameters is using EAs. EAs are inspired by the nature inspired Evolutionary Computation (EC), which is discussed in detail subsequently.

EC is the study of the computational system based on the idea of natu-

ral evolution and adaptation [29, 41–43]. Neo-Darwinian paradigm represents a widely accepted collection of evolutionary theories that define the evolutionary processes of reproduction, mutation, and selection as the main physical processes operating within individuals in a population [44]. As compared to conventional methods, the major advantages of EC are the computational simplicity, well applicability to broad classes of problems, the capability of self-optimization, and hybridization with other methods [45]. EAs are especially useful for optimization problems where the number of parameters is large, and the optimal solutions are difficult to obtain. At the algorithmic level, they differ mainly in their representations of potential solutions and their operators. EAs are the population based algorithm, it uses the collective learning process of a population of individuals. The strength of EAs is due to updating the whole population of possible solutions at each generation. This is equivalent to parallel explorations of the overall search space in a problem. Initial population may be either a random sample of the solution space or a specific value. The population evolves on the basis of the fitness function and its operator, and its update continues till the optimal solution is achieved. The application area of EAs is very wide, it is applied in the problem of neural network parameter optimization, economics, medical, and mathematics.

As discussed above, the EAs are used in many fields along with ANN to evolved neural network parameters optimally and called as Evolutionary Artificial Neural Networks (EANN) [46]. The search of the optimal network structure is known to be a complex, non-differentiable, and multi-modal optimization problem. In the structural design, EAs are employed in two ways: to evolve the structures only [47] and to evolve both the structures and the connection weights simultaneously [48]. In general, when EAs are used to determine the structures only, the training error is calculated by performing a random initialization of weights, which are then determined by a learning algorithm [49]. However, as indicated in [50], the training results depend on the random initial weights and the choice of the learning algorithm. Leung et al. presented an improved GA to tune the structure and parameters simultaneously [51]. Angeline et al. [48] suggested that genetic algorithms are not well suited for evolving networks and proposed an evolutionary program, called GeNeralized Acquisition of Recurrent Links (GNARL), to acquire both the structure and weights. Li et al. used an

improved PSO algorithm to learn ANNs parameters that are weights and bias, and it is used a binary PSO algorithm to evolve the architecture, simultaneously [52].

Although evolving both structures and weights cause the permutation problem [53–55]. Some researchers have avoided crossover and only adopted mutations in the evolution of structures [56, 57]. Also, in most EAs, the fitness value governs the evolutionary search [50, 58]. However, in the case of the simultaneous evolution of structures and connection weights, a good fitness value does not necessarily represent the quality of the structure. Often, the fitness of a neural network with a good structure and bad weights may be worse than the fitness of a network with a bad structure, and a good set of weights [59]. As a result, some potential structures may be discarded if only fitness is used.

In last decades, QEA are proposed as evolutionary algorithm to optimize neural network parameters. Similar to the general EAs, the QEA is characterized by the individuals, the evaluation function, and the population dynamics. However, instead of binary, numeric, or symbolic representation, it uses a probabilistic quantum bit representation. The QEA has distinct advantages over general EAs when it is used in evolving the neural networks. First, an individual in the QNN is composed of quantum bits that represent the probability of parameter subspace rather than a specific value. So, if the network has a bad fitness problem, QEA modifies the quantum states rather than discarding the network as occurs in other EAs. Thus, the risk of throwing away a potential structure or network parameters is mitigated. Second, instead of the crossover and mutation, the QEA uses observation to create a new network. Thus, the negative impact of the permutation problem is reduced. Third, a partitioning strategy is used to find the near-optimal network parameters. It explores each network parameter space region-by-region and rapidly finds the promising subspace for further exploitation. This is helpful to provide a set of appropriate network parameters when evolving the network structure. The detailed discussion about the quantum computing concept is discussed next.

2.2 Quantum Computing Concept

Inspired by the quantum physics concept, the quantum mechanical computers were proposed in the 1980s [60] and it is formalized in 1985 [61]. Many efforts on quantum computers have progressed actively since the early 1990s because these computers were shown to be more powerful than classical computers with various specialized problems. There are well-known quantum algorithms such as Shor's quantum factoring algorithm [62] and Grover's database search algorithm [63]. The quantum computing field can be classified into two fields. One concentrates on generating new quantum algorithms using the automatic programming techniques. Second one concentrates on QEA for a classical computer, a branch of study in evolutionary computing that is characterized by certain principles of quantum mechanics [23, 64]. QEA uses a quantum bit for probabilistic representation. A quantum bit individual is defined by a string of qubits. The quantum bit individual has the advantage that it can represent a linear superposition of states (binary solutions) in a search space probabilistically. Thus, the Q-bit representation has a better characteristic of population diversity than other representations.

The quantum bit (Q_i) can be represented by several qubits (q).

$$Q_i = (q_{i1}|q_{i2}|.....|q_{ik}) \tag{2.1}$$

Here, the k number of qubits which represent a quantum bit (Q). A single qubit (q_{ij}) where j=1,2.....k, is the smallest unit for representing information. A qubit is fundamentally different from the binary bit used in traditional classical/digital computers in the sense of representing data. A single binary bit can represent only two states, "0" and "1", whereas the qubit (q_{ij}) can represent the linear superposition of two states simultaneously, which is determined by probability model [23]. Thus, qubit q_{ij} can be represented as

$$q_{ij} = \alpha_{ij} \mid 0\rangle + \beta_{ij} \mid 1\rangle = \begin{bmatrix} \alpha_{ij} \\ \beta_{ij} \end{bmatrix} \tag{2.2}$$

where, α and β are the complex numbers representing the probability of a qubit in "0" state and "1" state, respectively. A probability model is applied which

shows the qubit in "0" state by α^2 and in "1" state by β^2, where

$$\alpha_{ij}^2 + \beta_{ij}^2 = 1;\ 0 \leq \alpha_{ij} \leq 1,\ 0 \leq \beta_{ij} \leq 1$$

As discussed above, a quantum bit (Q_i) formed using a single qubit (q_{i1}) where j=1,2,...,k, and (k=1) can represent two states, e.g., "0" state and "1" state. A quantum bit (Q_i) having two qubits ($q_{i1}|q_{i2}$), can represent four states, e.g., "00", "01", "10", and "11". In the same way, three-qubits ($q_{i1}|q_{i2}|q_{i3}$), can represent eight states, thus n qubits ($q_{i1}|q_{i2}|....|q_{in}$), can represent 2^n states. For example, an individual quantum bit Q_i has two qubits can be represented as follows:

$$Q_i = \left\langle \begin{array}{c} \alpha_{i1}|\alpha_{i2} \\ \beta_{i1}|\beta_{i2} \end{array} \right\rangle \tag{2.3}$$

The below example shows the representation of the four states of a quantum bit (Q_i) having two qubits ($q_{i1}|q_{i2}$).

$$Q_i = (\alpha_{i1} \times \alpha_{i2})\langle 00\rangle + (\alpha_{i1} \times \beta_{i2})\langle 01\rangle + (\beta_{i1} \times \alpha_{i2})\langle 10\rangle + (\beta_{i1} \times \beta_{i2})\langle 11\rangle \tag{2.4}$$

It is noted that qubit (q_{ij}) is made of two components α_{ij} and β_{ij}, where each component value lies between "0" and "1". The quantum bit (Q_i) having two qubits ($q_{i1}|q_{i2}$) can be initialized with a random value between 0 and 1 as discussed above. Here α_{i1}, α_{i2}, β_{i1}, and β_{i2} is initialized as follows:

$$Q_i = \left\langle \begin{array}{c} 1/\sqrt{2}|1/\sqrt{2}|1/\sqrt{2} \\ 1/\sqrt{2}|1/\sqrt{2}|1/\sqrt{2} \end{array} \right\rangle \tag{2.5}$$

With respect to Eq. (2.4) and Eq. (2.5), the state representation of a quantum bit (Q_i) is defined as follows:

$$\begin{aligned} Q_i = {} & (1/2\sqrt{2})\langle 000\rangle + (1/2\sqrt{2})\langle 001\rangle + (1/2\sqrt{2})\langle 010\rangle + (1/2\sqrt{2})\langle 011\rangle \\ & + (1/2\sqrt{2})\langle 100\rangle + (1/2\sqrt{2})\langle 101\rangle + (1/2\sqrt{2})\langle 110\rangle + (1/2\sqrt{2})\langle 111\rangle \end{aligned} \tag{2.6}$$

A Quantum bit a further converted into real coded value to get exploration using the observation process which is presented next.

2.2.1 Real Coded Value Generation: Observation Process

To get exploration and real coded value, there is a need to convert qubit (q) into a real coded value q^{real}. The conversion of a real coded value from a qubit has been done with the help of the observation process [23] as presented in Algorithm 2.1.

Algorithm 2.1 Algorithm for Observation Process.

begin
q_{ij}, link=0 and random number matrix r_{ij}.
for j=1:k
    q_{ij}=α_{ij}; $0 \leq \alpha_{ij} \leq 1$
    r_{ij}=rand();
    This rand function generates a uniform value between 0 and 1.
endfor
for j=1:k
    if (r_{ij} <(α_{ij}*α_{ij}))
        s_{ij}=1;
    else
        s_{ij}=0;
    endif
    if(link∼=0)
        $q_{ij}^{real} = N(\mu_{link}, \sigma_{link})$;
    endif
endfor

This process starts by taking a random number matrix R_i, where R_i= $[r_{i1}r_{i2}....r_{ik}]$, corresponding to $Q_i = (\alpha_{j1} \mid \alpha_{i2} \mid ..... \mid \alpha_{ik})$. The value of r_{ij} is selected with the help of a random function which generates uniform number between 0 to 1. Then, a further mapping is done by using a binary matrix S_i where $S_i = [s_{i1}s_{i2}....s_{ik}]$. The value of matrix S_i is generated as follows:

$$if(r_{ij} \leq (\alpha_{ij})^2) \ then \ s_{ij} = 1 \ else \ s_{ij} = 0. \tag{2.7}$$

To select weights from binary values, the Gaussian random generator has been used with mean value μ and variance σ, represented as $N(\mu, \sigma)$. The observation process shows the process of conversion of a single qubit (q_{ij}). As shown in Eq. (2.2) the qubit (q_{ij}) is formed by using two components α_{ij} and β_{ij}. For processing of qubits, the α_{ij} component is considered because the value of second component β_{ij} will be $\sqrt{1-\alpha_{ij}^2}$. This observation process is called

for the conversion of all the qubits of Q_i into real coded value. Thus, this Q_i is utilized for getting a real coded value of the learning parameters of neural network algorithms.

The observation process can be understood with the help of an example. Let a quantum bit of length two qubits is represented as $Q = \langle 0.707|.707\rangle$, therefore a random number matrix is generated using a random number R=[0.85 0.02]. Now using Eq. (2.7), the binary matrix is generated as S=[01]. Once the binary matrix is achieved, then the formula is used here to convert a binary number to a decimal value (bin2dec(S)+1). This returns a number between 1 to 4 and corresponding four Gaussian random values are also mentioned for example N(0.25, 0.03), N(0.40, 0.03), N(0.55, 0.03), and N(0.70, 0.03). As a binary value achieved here return two as the decimal value, therefore the real coded value Q^{real} corresponding to quantum bit Q is selected from N(0.40, 0.03).

2.2.2 Qubit Update Process

This observation process helps to achieve exploration, and it converts qubits into real coded values. The value of quantum bit gets evolved for achieving the best value of learning parameters of a neural network from a large search space in several generations. Thus during the generations, quantum bit value gets updated. The new value of a quantum bit is generated through the quantum update function [23]. To update Q_{g+1} from Q_g, quantum rotation gate is required, which is described as follows:

$$U(\Delta\theta) = \begin{vmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{vmatrix} \tag{2.8}$$

where $\Delta\theta$ is a rotation angle which is used to generate Q_{g+1} from Q_g.

$$\begin{vmatrix} \alpha_{ij}^{g+1} \\ \beta_{ij}^{g+1} \end{vmatrix} = \begin{vmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{vmatrix} * \begin{vmatrix} \alpha_{ij}^{g} \\ \beta_{ij}^{g} \end{vmatrix} \tag{2.9}$$

As presented in Lu et al. [11], $\Delta\theta$ is calculated on the basis of the objective function F^* and F_g. The objective function F^* represent the best or global objective function during all generations (g) and F_g is the objective function value of the current generation. As shown in the observation process, each qubit α_{ij} is associated with a binary value s_{ij}, therefore a mapping is done between F^*

and binary bit value s to update individual qubit. If the objective function values F_g is worse than that of objective function value in F^*, and state of s_{ij} is zero, and best objective function state of s^*_{ij} is one, then decrement in the probability of α_{ij} may produce the worst result. Therefore, to increase the probability of α_{ij} to one, $\Delta\theta$ made negative. While, if the objective function value F_g is better than the objective function in F^*, and state of s_{ij} is one, and the best objective function state s^*_{ij} is zero, then increasing the probability of α_{ij} to one, may produce the worst result. Therefore, to update α_{ij}, angular displacement made positive $\Delta\theta$. In other cases, angular displacement will remain zero. The value of angular displacement must be selected in such a way so that it can cover the maximum value of α in the range of (0 1) and also should not take many iterations to cover these values [11]. Therefore, $\Delta\theta$ must be initialized between $(0.01 \times \pi, 0.05 \times \pi)$. The quantum bit update process is explained in the tabular form in Table 2.1.

For preventing the quantum bit α^g_i from acquiring values 0 or 1, following constraints are applied:

$$\alpha_{ij} = \begin{cases} \sqrt{\epsilon}, & if \quad \alpha_{ij} < \sqrt{\epsilon} \\ \alpha_{ij} & if \quad \sqrt{\epsilon} \leq \alpha_{ij} \leq \sqrt{1-\epsilon} \\ \sqrt{1-\epsilon} & if \quad \alpha_{ij} > \sqrt{1-\epsilon} \end{cases} \tag{2.10}$$

where ϵ is assigned a very small value (approximately approaching to zero), so that it can cover maximum value in the range of (0, 1).

The performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes [24,25]. This causes due to the

Table 2.1: Qubits Update.

s_{ij}	s^*_{ij}	$F_g < F^*$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	$0.03 * \Pi$
1	*0*	*true*	*0*
1	*1*	*false*	*0*
1	*1*	*true*	*0*

occurrence of overlapped samples in multiple classes of dataset. Thus, there is a need to apply the fuzzy concept for handling such multiple class dataset which are having overlapped samples. The fuzzy concept helps to classify such samples by assigning membership degrees according to the class from which these samples belong. Based on the features of fuzzy concept and neural network many researchers has proposed neuro-fuzzy model to deal with real-life problems such as speed prediction [65], oil consumption estimation, policy making [66], forecasting electricity loads [67], and medical diagnosis. Jang et al. [68] proposed a Neuro-fuzzy System (NFS) by integrating neural networks and fuzzy systems to exploit best features of both the approaches and offer several advantages such as better intelligibility, adaptability, quick convergence, and higher accuracy. Kahramanli and Allahverdi [69], proposed a new hybrid neural network that includes ANN and Fuzzy Neural Network (FNN) and obtained good classification accuracy for PID dataset. Luukka [70], proposed a classification method in which data is first preprocessed using the Fuzzy Robust Principal Component Analysis (FRPCA) algorithms to obtain data in a more feasible form, and was then classified using a similarity classifier.

However, there are parameters in the fuzzy concept like fuzzifier parameter, cluster centroids which are initialized randomly but initialization in such a way does not guarantee to get optimal results. Therefore we proposed to optimize the fuzzy parameters which are based on Fuzzy C-Means of these parameters the quantum computing concept has been utilized. The detailed discussion of the fuzzy parameters along with the FCM algorithm is presented next.

2.3 Fuzzy Clustering

Fuzzy clustering is based on the fuzzy set theory that allows an object to have varying grades of membership in a set [71]. The fuzzy set theory is used in the clustering algorithm so that a sample can belong to the multiple clusters [72]. A fuzzy clustering produces a fuzzy partition that can be expressed by the membership matrix, U. The degree of membership of sample i in cluster j is represented by u_{ij}. This is subjected to the following constraints [39, 73]:

$$u_{ij} \in [0, 1], 1 \leq i \leq n, 1 \leq j \leq c \tag{2.11}$$

$$\sum_{j=1}^{c} u_{ij} = 1, 1 \leq i \leq n \tag{2.12}$$

$$\sum_{i=1}^{n} u_{ij} > 0, 1 \leq j \leq c \tag{2.13}$$

where c is the number of clusters, n is the total number of samples in a dataset, and u_{ij} denotes the membership of the i^{th} sample belonging to the j^{th} cluster such that $i = 1, 2, .., n$ and $j = 1, 2, .., c$.

Fuzzy cluster analysis gives the flexibility to express that the samples in a dataset can belong to more than one cluster at the same time. In addition to this, the membership degrees can also express how ambiguously or definitely a sample should belong to a cluster. The concept of fuzzy analysis has been successfully integrated into many clustering algorithms [74–77]. One of the most widely used fuzzy clustering algorithms is discussed next.

FCM algorithm

The FCM algorithm was initially presented by Dunn [78] and completed by Bezdek [39]. The FCM partitions the collection of n sample x_i, $i = 1, ..., n$ into c fuzzy clusters and finds a set of cluster centers. The FCM executes iteratively until the difference in the cluster centers of previous and the current iteration is less than the defined termination criteria ($\epsilon^{'}$) [73]. The aim is to minimize an objective function of dissimilarity measure. In FCM, the membership degree u_{ij} can take any values between 0 and 1. The objective function of FCM can be formulated as follows:

$$J_m(U, V^{'}) = \sum_{i=1}^{n} \sum_{j=1}^{c} (u_{ij})^m \|x_i - v_j^{'}\|^2 \tag{2.14}$$

The FCM algorithm is based on the iterative optimization of objective function given in Eq. (2.14) by updating the membership matrix U and cluster centers to get a set of final cluster centers $V^{'}$. The parameters related to FCM are presented in Table 2.2. Algorithm 2.2, presents the steps of FCM algorithm.

Algorithm 2.2 Algorithm for FCM to Iteratively Minimize $J_m(U, V^{'})$

1: **Input:** X, V, c, m, $\epsilon^{'}$
2: **Output:** $U, V^{'}$
3: **Begin**

Algorithm 2.2 (Continued)

4: **Randomly initialize cluster centers** $V = \{v_1, v_2.., v_c\}$.
5: **Compute the cluster membership matrix** U.

$$u_{ij} = \frac{\parallel x_i - v_j \parallel^{\frac{-2}{m-1}}}{\sum_{k=1}^{c} \parallel x_i - v_k \parallel^{\frac{-2}{m-1}}}, \forall i, j \tag{2.15}$$

6: **Check the constraint**

$$\sum_{j=1}^{c} u_{ij} = 1 \tag{2.16}$$

7: **Compute the cluster centers** $v_j^{'}$ for $j = 1, 2, .., c$.

$$v_j^{'} = \frac{\sum_{i=1}^{n} [u_{ij}]^m x_i}{\sum_{i=1}^{n} [u_{ij}]^m}, \forall j \in [1, c] \tag{2.17}$$

8: **If improvement in** $J_m(U, V^{'})$ **is less than** ϵ or **if** $\parallel V^{'} - V \parallel < \epsilon^{'}$, then stop; **Else**: $V = V'$ and go to Line 5.
9: **Return** $U, V^{'}$
10: **End**

The fuzzifier parameter (m) controls the degree of fuzziness of every sample between the fuzzy clusters and reflects the partition percentages of every sample in each cluster, and it drastically affects the clustering results. However, there are some parameters in the FCM which needs to be initialized properly because it may affect the performance of the algorithm.

2.3.1 About Fuzzifier Parameter

One of the most important parameters in FCM is the fuzzifier denoted by m. It is also called as weighting exponent and has a significant impact on the performance of FCM. The FCM is sensitive to the initialization of m and usually gets trapped in local optima due to improper selection of m. When m is se-

Table 2.2: Parameters Specification for FCM.

Parameters	Description	Values
ϵ	Termination criteria	0.001 [39]
m	Weighting exponent	$m \in [1.5, 2.5]$ [79]
$c_{\min}$	Minimum number of clusters	2
$c_{\max}$	Maximum number of clusters	$\sqrt{n}$ [80]
n	Number of data samples	Size of dataset
c	Number of clusters	$c \in [c_{\min}, c_{\max}]$ [81]

lected close to one, the FCM approaches the crisp clustering algorithm and if m approach infinity, then the only solution for the FCM is the mass center of the dataset. Hence, choosing a suitable weighting exponent is a very challenging task when implementing FCM. However, different methods have been adopted by the researchers to select an appropriate value of fuzzifier for the execution of FCM.

Pal and Bezdek [79] have given heuristic guideline regarding best choice of m, they consider various cluster validity index to analyze the best choice of the fuzzifier. On the basis of experimental evaluation, they found that suitable value for m is selected in the range of [1.5, 2.5]. Similar recommendations have been given by other researchers [82, 83], but these recommendations are based on empirical studies and may not be appropriate in general for all the datasets.

Yu et al. [84] developed a new theoretical and practical approach for selecting the fuzzifier parameter in FCM. In this approach, a new local optimality test of solutions for the FCM is proposed. Based on this test, one obtains theoretical rules for selecting the fuzzifier parameter in FCM, which show that a proper m depends on the dataset itself. However, theoretical analysis and numerical experimental results show that $m = 2$ is not a reasonable heuristic guideline.

On the basis of robust analysis of FCM, Wu [85] proposed a new guideline for selecting parameter m. This guideline suggested that a large value of m makes FCM more robust to noise and outliers. However, considerably large m values that are greater than theoretical upper bound makes sample mean a unique optimizer. In the case of large theoretical upper bound, when a dataset contains noise and outliers, the fuzzifier $m = 4$ is recommended for FCM.

2.3.2 Deciding Initial Cluster Centroids

In FCM, one more parameter needs to be initialized, i.e., cluster centroids in addition to m, which significantly affects clustering results. Choosing the initial cluster centroids is extremely important as it has a direct impact on the formation of final clusters. The major problem with FCM is that it is sensitive to initialization cluster centroids. The random selection of initial cluster centroids does not guarantee unique clustering results. This makes FCM algorithm to converge at local optimal solutions. Many methods have been proposed by

the researchers [86–89] that automatically determine the number of clusters and locations of cluster centroids. However, they do not provide good generalization capabilities in obtaining appropriate centroids [86].

Thus, there is no exact method available for the selection of the optimal global solution of fuzzifier and initial cluster centers. Based on the concept of fuzzy clustering we proposed two algorithms, i.e., Q-FNN for solving two class classification problem and the Q-FNNM for solving multi-class classification problem. In these algorithms, the neural network is formed using the fuzzy clustering in which parameters are optimized using the quantum computing concept.

2.4 Deep Neural Network

Neural network learning techniques have been widely applied in a variety of areas such as pattern recognition, natural language processing, and computational learning. During the past decades, machine learning brings enormous influence on our daily life. Nevertheless, when it comes to the human information processing mechanisms (e.g., speech and vision), the performance of traditional machine learning techniques is far from satisfactory. For example, gradient descent algorithm which has played an important role in ANNs since last 3-4 decade. However, due to improper initialization of its parameters it does not perform well and get trapped in the problem of local optima

Although the training accuracy is high of this, but the performance of the gradient descent algorithm when applied to the testing data might not be satisfactory because, with random initialization of its parameters, the algorithm often gets trapped in local optima.

These learning algorithms are used to form neural network architectures with one hidden layer as well as with multiple hidden layers. The scientific research has shown that multilayered architectures trained using unsuitable learning algorithms performed poorly [7, 90]. The single layer neural network architecture, unable to solve such problem. Also increasing the number of layers in multilayered architecture degrade the performance.

Inspired by deep hierarchical structures of human speech perception and production systems, the concept of deep learning algorithms was introduced [7].

Deep neural networks do not necessarily have multiple hidden layers; here deep word shows that this concept is introduced by deeply analyzing several existing neural network models [91]. Larochelle et al. (2007), Vincent et al. (2008), and Kim et al. (2015), have proposed deep neural network with few numbers of hidden layers [26–28]. Breakthroughs in deep learning have been achieved since 2006 when Hinton proposed a novel deep structured learning architecture [92]. In this, he gives a basic idea of the layer-wise-greedy-learning is that unsupervised learning should be performed for network pre-training before the subsequent layer-by-layer training. By extracting features from the inputs, the data dimension is reduced, and a compact representation is hence obtained. Then, export the features to the next layer, after that all the samples will be labeled, and the network will be fine-tuned with the labeled data. However, to deal with a complex dataset like image dataset, the dimensions of a dataset is an important issue. The dimension of a input data set plays an important role in the classification.

To deal with such dataset auto-encoder is proposed which can efficiently handle the datasets which requires dimensional reduction for better classification accuracy [7, 40, 93, 94].

The deep neural network using stacked auto-encoder performs well, but proper selection of learning parameters is required for good performance. The deep neural network using stacked auto-encoder uses gradient descent algorithm as learning algorithm [28, 40]. In the gradient descent algorithm, parameters are like learning rate parameter is initialized randomly. If the appropriate selection of this parameter is not done, then the problem of over-fitting or under-fitting may occurs and sometime with no convergence . It may also happen that algorithm may provide good training accuracy, but bad testing accuracy. Without proper selection of this parameter, the machine learning algorithms give the result with poor generalization accuracy [95, 96]. In this thesis the auto-encoder is enhanced by evolving its learning rate parameter using the quantum computing concept. Hence, the framework of auto-encoder is discussed below:

The auto-encoder neural network is an unsupervised learning algorithm [97, 98]. In this algorithm, dimensions of the target output are equal to the dimensions of the input dataset. Assuming Y is output, then it must be equal to the input X ($Y_i = X_i$). The number of hidden layer neurons is lesser than the

number of input layer neurons. Therefore, auto-encoders are theorized to learn a better feature representation of data. The auto-encoder consists of two parts, i.e., the encoder part and the decoder part.

The encoder part maps Z from X as follows:

$$\phi : X \mapsto Z \tag{2.18}$$

The decoder part maps Y from Z as follows:

$$\psi : Z \mapsto Y \tag{2.19}$$

To minimize the objective function $(||X - (\phi o \psi)Y||^2)$

$$arg\ min||X - (\phi o \psi)Y||^2) \tag{2.20}$$

Mathematically, a n-p-n auto-encoder is given by the encoder; which takes a $X \in \mathbf{R^n}$ as input and encodes it into a $Z \in \mathbf{R^p}$, where n is the number of neurons in the input and output layer and p is the numbers of neurons in the hidden layer.

$$Z = \sigma_1(WX + b) \tag{2.21}$$

Whereas the decoder part expands the compressed representation back into a $Y \in \mathbf{R^n}$ and can be written as follows.

$$Y = \sigma_2(W'Z + b') \tag{2.22}$$

The auto-encoder tries to minimize the Mean Squared Error (MSE) of reconstruction of the input given by Eq. (2.20) and can be written as follows.

$$\mathcal{L}(\mathcal{X}, \mathcal{Y}) = ||X - Y||^2 = ||X - \sigma_2(W'(\sigma_1(WX + b) + b'))||^2 \tag{2.23}$$

There are various ways to train the auto-encoder, one of such algorithms is discussed below.

Training of an Auto-encoder

A simple n-p-n auto-encoder can be thought of as a multilayer Perceptron with a single hidden layer and therefore can be trained using gradient descent/Backpropagation learning algorithm. Mathematically, weights connecting neuron i to j are updated using the following equation.

$$W_{ij}(t+1) = W_{ij}(t) - \eta * \frac{\partial E}{\partial W_{ij}} \tag{2.24}$$

where $W_{ij}(t)$ is the connection weight between the i^{th} node to the j^{th} hidden layer neuron at iteration t. Here, E shows the error or difference between the expected output and original output. The learning rate parameter is represented as η which is chosen on a trial basis between 0 and 1 for the learning of neural network. In general, the value of the learning rate parameter chosen once at the beginning of execution is fixed throughout the execution of the process. Choosing the learning rate parameter in this way may lead to the local minima. Thus, there is a need to design some mechanism for assuming the appropriate setting of these parameters.

Pre-Training

To avoid local optimal solutions in deep neural networks, layer-wise pre-training is done [99]. The unsupervised pre-training helps to overcome the challenges of deep learning. The unsupervised pre-training gives a region of parameter space in which not only training error is better, but also testing error is reduced. It involves the training of each auto-encoder individually. Deep architectures contain similar units stacked on top of each other, this is known as vertical composition. A stacked auto-encoder consists of auto-encoders stacked on top of each other and trained in a greedy layer-wise manner. Then it is fine-tuned for classification which is given next [97].

Fine-Tuning

The fine-tuning process is also called as supervised learning in the deep neural network. As discussed above, the fine-tuning process is done when all the hidden layers are finalized using the unsupervised (pre-training) process. These auto-

encoders are connected in the bottom-up fashion, and the top hidden layer is connected to an output layer which represents class information. Thus, fixing the weights (evolved during pre-training) of all layers by adding auto-encoders in the bottom-up fashion and then initialize connection weights from the last hidden layer to the output layer [100]. This brings neural network architecture in the vicinity of global optima, but before finally adding a classifier layer on the top which acts as the output layer, fine-tuning of the entire network is required using the Backpropagation. In this thesis the auto-encoder is enhanced by evolving its learning rate parameter using the quantum computing concept and it is named as Q-DNN. The Q-DNN proposed here follows the steps of pre-training and fine-tuning. Also, along with this, the optimized selection of learning rate parameter is done using the quantum computing concept.

Till now the reviewed literature is presenting the concepts and preliminaries which are used to design novel quantum inspired learning algorithms. One of such novel learning algorithm NQ-BNN is further enhanced to apply on a real life problem of signature verification. Hence, a primer on signature verification is discussed next.

2.5 Offline Signature Verification

The offline signature has been an ancient biometric hallmark for authenticating individuals and documents. Scientists and researchers have worked for many years in the field of offline signature verification. Several methods and technologies have been proposed in this area, and some of them include elastic matching [101], synthetic discriminant functions [102], and grid features [103]. Other new methods have been proposed, such as geometric measure-based approaches, grid-based approaches, techniques based on granulometric size distributions, neural classifiers, and dynamic time warping [104,105]. Some researchers worked on finding appropriate features of a signature. Ferrer et al. [106] proposed a technique in which grayscale features such as local binary pattern and local directional pattern are analyzed.

Finding appropriate static features from a signature image is still an open area of research. With the advancement of technology, the extraction of features opens a new dimension of research. Finding appropriate static features from

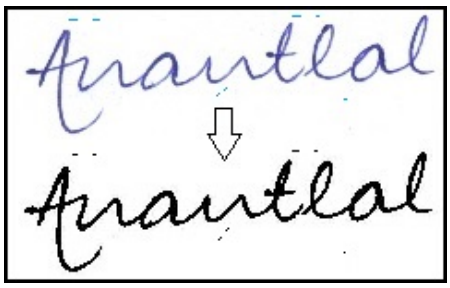

Figure 2.1: Conversion from RGB to Black and White.

a signature image is still an open area of research. Moreover, the efficiency of matching two signature images is an important parameter in any signature verification technique.

To extract features of any offline signature, there is a requirement of bringing all on the same scale. The document consists of the signatures may be of different color, noise, variation in the signatures width due to variation in the pen. Therefore, to get images on the same scale, we pre-process all images. The following steps are followed for the pre-processing of the signature image.

Conversion from RGB to Black and White

To pre-process the signature image first its color is made uniform. Therefore, each signature image of the different color is firstly converted into a black and white image as shown in Figure 2.1.

Noise Removal

The noise removal process is performed after converting all images uniformly, i.e., in the black and white color. The signature images have noise due to the two main sources: first, the background of paper on which the signature is taken and Second, the noise arises while scanning the paper having signatures as shown in Figure 2.2. This noise will hinder the training and testing of signatures and hence must be removed.

Thinning

A signature impression may be made with pens of varying tips. However, the difference in tip size shouldn't be a factor to distinguish signatures. The thickness of every stroke in a signature is reduced to a width of a single pixel as shown in Figure 2.3.

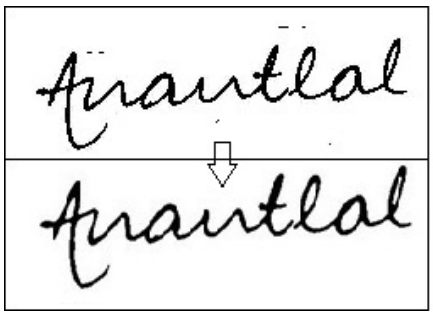

Figure 2.2: Noise Removal of Signature.

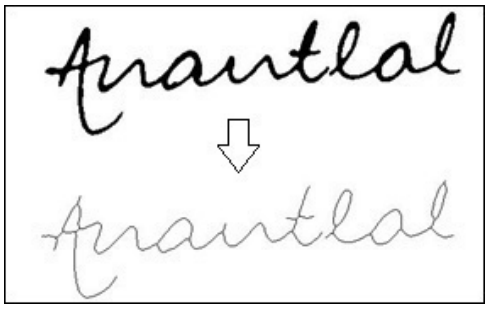

Figure 2.3: Thinning of Signature.

2.6 Performance Evaluation

The proposed quantum inspired neural network learning algorithms is tested on several benchmarks datasets which are presented in detail in appendix A. To judge its performance proposed algorithms, these are compared with several state-of-the-arts approaches on different measures which is discussed next:

2.6.1 Two Class Dataset Evaluation

To compare and evaluate the performance of the proposed system, classification accuracy alone is not an adequate measure, some more appropriate performance measures are used here that compares predicted value to true labels. We consider here classification accuracy, sensitivity, specificity, and confusion matrix. The formulations of these measures are defined as follows:

Accuracy

It measures the proportion of correctly classified samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \tag{2.25}$$

Sensitivity

It measures the fraction of positive data that are classified as positive.

$$Sensitivity = \frac{TP}{TP + FN} \times 100 \tag{2.26}$$

Specificity

It measures the fraction of negative data classified as negative.

$$Specificity = \frac{TN}{TN + FP} \times 100 \tag{2.27}$$

where, TP, TN, FP and FN denote True positive, True negative, False positive and False negative.

- True Positive (TP): It denotes the correct classification of positive data.
- True Negative (TN): It denotes the correct classification of negative data.
- False Positive (FP): It denotes the incorrect classification of positive data that are classified as negative.
- False Negative (FN): It denotes the incorrect classification of negative data that are classified as positive.

Confusion Matrix

A confusion matrix is a table describing the performance of a classifier on a set of test data in terms of actual and predicted classification. The performance of a classifier is commonly evaluated using the data in the matrix. The confusion matrix for the two class problem is given in Table 2.3.

Table 2.3: Confusion Matrix for Two Class Dataset.

	Predicted Negative	Predicted Positive
Actual negative	True Negative (TN)	False Positive (FP)
Actual positive	False Negative (FN)	True Positive (TP)

2.6.2 Multi-class Dataset Evaluation

The parameters used for evaluating the performance of a algorithm are computed with the help of the confusion matrix. This matrix contains information about actual and predicted classification performed by the classifier [107].

The confusion matrix evolves around four simple concepts. The confusion matrix for the multi-class problem is given in Table 2.4.

1. True Positive(tp): It denotes the correct classification of positive data of the class. In the case of multi-class classification problem, the diagonal element of the matrix represents the true positive for each class.

2. True Negative(tn): It denotes the correct classification of negative data of the class. In the case of multi-class classification problem, true negative for a particular A^{th} class is evaluated as follows:

$$tn_A = \varepsilon_{BC} + \varepsilon_{CB} + tp_B + tp_C$$

3. False Positive(fp): It denotes the incorrect classification of negative data of class that is classified as positive. The parameter in case of multi-class classification problem is evaluated for a particular A^{th} class is defined as follows:

$$fp_A = \varepsilon_{BA} + \varepsilon_{CA}$$

4. False Negative(fn): It denotes the incorrect classification of positive data of class that is classified as negative. In case of multi-class classification problem, false negative for a particular A^{th} class is evaluated as follows:

$$fn_A = \varepsilon_{AB} + \varepsilon_{AC}$$

Table 2.4: Confusion Matrix for Multi-Class Dataset.

		Predicted		
		A	B	C
Actual	A	tp_A	ε_{AB}	ε_{AC}
	B	ε_{BA}	tp_B	ε_{BC}
	C	ε_{CA}	ε_{CB}	tp_C

The parameters used to compare the performance of an algorithm are described henceforth with the use of following parameter:

Average Accuracy:

It is used to measure the ability of the classifier to produce an accurate diagnosis.

$$AverageAccuracy = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \tag{2.28}$$

Error Rate

It is defined as the average ratio of false classification of each class to the total number of data in each class.

$$ErrorRate = \frac{1}{n}\sum_{i=1}^{n}\frac{fp_i + fn_i}{tp_i + tn_i + fp_i + fn_i} \tag{2.29}$$

Precision/Positive Predicted Value(PPV):

It is the ratio of the times when a data sample is correctly predicted to the number of times a data sample is predicted to be true.

$$PPV_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i}{tp_i + fp_i} \tag{2.30}$$

$$PPV_\mu = \frac{\sum_{i=1}^{n} tp_i}{\sum_{i=1}^{n}(tp_i + fp_i)} \tag{2.31}$$

Recall/Sensitivity

It is the percentage of the time when a data sample is predicted to be true when a data sample is actually true.

$$Recall_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i}{tp_i + fn_i} \tag{2.32}$$

$$Recall_\mu = \frac{\sum_{i=1}^{n} tp_i}{\sum_{i=1}^{n}(tp_i + fn_i)} \tag{2.33}$$

Specificity

It specifies the number of times when the a data sample is predicted to be false and is actually false.

$$Specificity_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tn_i}{fp_i + tn_i} \tag{2.34}$$

$$Specificity_\mu = \frac{\sum_{i=1}^{n} tn_i}{\sum_{i=1}^{n}(fp_i + tn_i)} \tag{2.35}$$

Negative Predicted Value(NPV)

It is the ratio of the number of times a data sample is predicted to be false to the number of times a data sample is predicted to be false.

$$NPV_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tn_i}{tn_i + fn_i} \tag{2.36}$$

$$NPV_\mu = \frac{\sum_{i=1}^{n} tn_i}{\sum_{i=1}^{n}(tn_i + fn_i)} \tag{2.37}$$

F-score

It can be interpreted as a weighted mean of precision and sensitivity. It reaches its best value at 1 and worst at 0.

$$F - score_M = \frac{(\beta^2 + 1) \times PPV_M \times Recall_M}{\beta^2 \times (PPV_M + Recall_M)} \tag{2.38}$$

$$F - score_\mu = \frac{(\beta^2 + 1) \times PPV_\mu \times Recall_\mu}{\beta^2 \times (PPV_\mu + Recall_\mu)} \tag{2.39}$$

Chapter 3

Quantum inspired Binary Neural Network Learning Algorithm

3.1 Introduction

ANNs have been successfully applied to problems in pattern classification, pattern matching, associative memories, optimization, and function approximation [10, 11, 108]. To solve the problem from various fields like mathematics, economics, computer science, and many more, several architectures have been proposed like Perceptron, Backpropagation, and RNN. The performance of the neural network in the mentioned areas mainly depends upon parameters like network architecture, connection weights, and the threshold of neurons [12–15]. Deciding connection weights and threshold in an optimized manner is a challenging task.

The search of optimal connection weights is known to be a complex, non-differentiable, and multi-modal optimization problem. Therefore, in this chapter to solve the problem of finding the optimal connection weights of neural network we propose a algorithm which is named as Quantum inspired Binary Neural Network Learning Algorithm (Q-BNN). It forms three layer network structure and works for two class problem. The proposed method makes use of evolutionary quantum computing concept for evolving optimal connection weights and threshold of neurons is decided manually. Deciding threshold of neurons manually may leads an algorithm to the problem of local maxima. Therefore, to solve this problem, we enhanced our proposed Q-BNN algorithm and proposed

a Novel Quantum inspired Binary Neural Network Algorithm (NQ-BNN). It also forms a three layers neural network architecture and works for two class classification problem. The basic architecture of Q-BNN and NQ-BNN is shown in Figure 3.1. In the proposed approach for finding the optimal value of threshold a new parameter, i.e., a quantum separability parameter is introduced.

3.2 Proposed Quantum inspired Binary Neural Network Learning Algorithms

In the proposed Q-BNN we make use of the quantum computing concept to find the connection weights of the neural network and its architecture is formed constructively by adding the hidden layer neuron one by one. The quantum computing concept has been used in the form of the QEA. In the Q-BNN algorithm, we formed a three layers neural network for solving two class classification problems. The connection weights of the neurons are evolved using the quantum computing concept. First, we add one neuron in the hidden layer and initialize its connection weights W_i^{quant} in terms of a quantum bit. Then we apply the observation process as discussed in Sections 2.2, using which we get proper exploration and connection weight W_i^{real} in terms of real values. We have also used here the step activation function as the threshold. Here, sum^* and sum_g are the objective function value which depends on parameters $conut1$ and $count2$

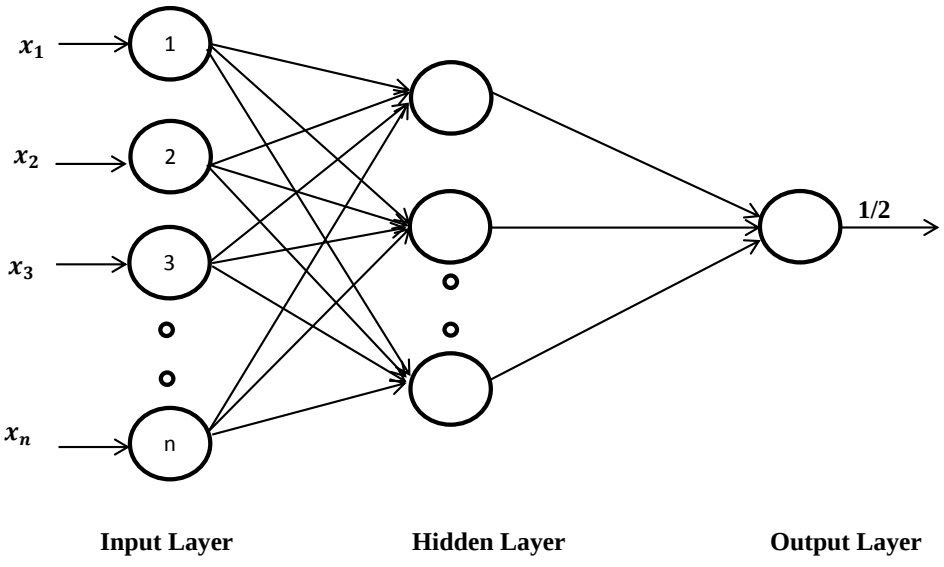

Figure 3.1: Architecture of Q-BNN and NQ-BNN.

showing the number of learned sample. To evolve optimal connection weights in serval generation, we use the quantum update function as discussed in Sections 2.2. If all samples do not learned using one hidden layer neuron, then we add another hidden layer neuron for the unlearned sample and apply the same process of learning. However, here for the learning of neuron, the threshold is decided manually using input sample and connection weights. Therefore, there is a chance to get trapped in the problem of local maxima.

To overcome this issue, we proposed an enhanced version of proposed the Q-BNN algorithm and named as Novel Quantum inspired Binary Neural Network Algorithm (NQ-BNN). In this algorithm, along with connection weights W_i^{quant} of the neural network the threshold λ_i^{quant} of neurons is also evolved using the quantum computing concept. The neural network architecture is formed constructively. To find the threshold of neurons optimally, we proposed a quantum separability parameter. In this algorithm, we use objective functions as sum^* and sum_g for connection weights, and sum_λ^* and sum^t are used for the threshold of the neurons. Here, g is the number of generations taken for evolving connection weights and t is the number of generations taken for evolving the threshold of neurons using the quantum separability parameter. In this algorithm, the connection weights are evolved in generations g=1:100. During each generation of evolution of connection weights, the threshold of neurons is evolved in generations t=1:100. Here, to get exploration for evolving connection weights and the threshold we use the observation process. To get exploitation, we use quantum rotation gate which updates the qubits of connection weights and the threshold using the fitness function sum^*, sum_g, sum_λ^*, and sum^t, respectively. The details of proposed algorithms with initialization of parameters are presented next.

3.2.1 Representation of Connection Weights and Threshold in Terms of qubits

In this section, we have used an evolutionary principle of quantum computing to represent connection weights and threshold of the neurons. The preliminaries related to quantum computing concept are already discussed in Sections 2.2. Here, $X=(X_1, X_2, X_3, ..., X_{c_1})$ and $Y=(Y_1, Y_2, Y_3, ..., Y_{c_2})$ denotes the instance

of input samples of two different classes. Let, X_i and Y_i are one of the instances of class A and class B, respectively. Each instance of X_i= $(x_1, x_2, x_3, ..., x_n)$ and $Y_i = (y_1, y_2, y_3, ..., y_n)$ have n attributes. As each instance has n attributes, therefore input nodes will be equal to n. For i^{th} hidden layer neuron, connection weights for generation g are denoted as follows:

$$(W_i^{real})^g = ((W_{i1}^{real})^g, (W_{i2}^{real})^g, (W_{i3}^{real})^g, ..., (W_{in}^{real})^g) \tag{3.1}$$

The weight matrix corresponding to Eq. (3.1) of i^{th} hidden layer neuron in the form of quantum bits Q^g can be represented as:

$$(W_i^{quant})^g = (Q_{i1}^g, Q_{i2}^g, Q_{i3}^g, ..., Q_{in}^g); \tag{3.2}$$

The quantum bit (Q_i^g) can be represented by several qubits (q_{ij}^g).

$$Q_i^g = (q_{i1}^g | q_{i2}^g | ..... | q_{ik}^g) \tag{3.3}$$

Here, the k number of qubits represents a quantum bit (Q_i^g). A single qubit (q_{ij}^g) where j=1,2.....k, is the smallest unit for representing information.

In the proposed learning algorithms, we make use of the step function as an activation function for forming the neuron which is given as follows.

$$net_i = \sum_{i=1}^{n} W_i^{real} \times X_i \tag{3.4}$$

$$f(net_i) = \begin{cases} 1 & if \quad net_j \leq \lambda_i^{real} \\ \\ 0 & if \quad net_i > \lambda_i^{real} \end{cases} \tag{3.5}$$

The threshold of the neurons is a crucial parameter for learning. The random selection of the value of this parameter may lead the solution to the problem of local maxima. Therefore, we evolved threshold of neurons using the quantum computing concept. The threshold of neurons is initialized in terms of qubits for generation t as follow:

$$(\lambda_i^{quant})^t = (\alpha_i^t \mid \alpha_{i+1}^t); \tag{3.6}$$

We only use quantum threshold in NQ-BNN algorithm $(\lambda_i^{quant})^t$, the threshold in Q-BNN is decided manually.

It is required that a quantum bit of $(W_i^{quant})^g$ and $(\lambda_i^{quant})^t$ has to be converted into a real coded value. This conversion is carried out with the help of observation process discussed in Section 2.2, that helps to achieving exploration [11].

The connection weights and the threshold of neurons with real coded value are represented as $(W_i^{real})^g$ and the threshold $(\lambda_i^{real})^t$, and their corresponding quantum weights and quantum thresholds are represented as $(W_i^{quant})^g$ and $(\lambda_i^{quant})^t$, respectively.

The real coded values of connection weights $(W_i^{real})^g$ are used in both the algorithms, i.e., Q-BNN and NQ-BNN. On the other hand, the real coded value of threshold $(\lambda_i^{real})^t$ is used only in NQ-BNN. However, the real coded value generated in the first generation may or may not be optimal. Therefore, we need to find the optimal value of it by updating qubits of connection weights and threshold of neurons in several generations. The qubits are updated using the quantum update process discussed in Section 2.2, which provides exploitation. It is used to generate $(W_i^{quant})^{g+1}$ from $(W_i^{quant})^g$ and $(\lambda_i^{quant})^{t+1}$ from $(\lambda_i^{quant})^t$. It forces qubit to update in such direction so that it leads to optimal results. The quantum update process required objective function therefore we are only presenting here quantum bit update Table 3.1, according to objective functions of Q-BNN and NQ-BNN.

3.2.2 Quantum inspired Binary Neural Network Learning Algorithm

In this section, Q-BNN algorithm is discussed. This algorithm makes use of the evolutionary quantum computing concept for constructing the neural network. It forms a three layers network structure consists of input, hidden, and an output layer. Here, X=$(X_1, X_2, X_3, ..., X_{c_1})$ and Y=$(Y_1, Y_2, Y_3, ..., Y_{c_2})$ denotes the instance of input samples of two different classes. Let, X_i and Y_i are one of the instances of class A and class B, respectively. Each instance of X_i= $(x_1, x_2, x_3, ..., x_n)$ and Y_i = $(y_1, y_2, y_3, ..., y_n)$ have n attributes. As each instance has n attributes, therefore input nodes will be equal to n. The number

Table 3.1: Qubits Update for Q-BNN and NQ-BNN.

s_{ij}^{g}	s_{ij}^{*}	$sum_g < sum^*$ $sum^t < sum_\lambda^*$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	*0*
1	*0*	*true*	$0.03 * \Pi$
1	*1*	*false*	*0*
1	*1*	*true*	*0*

of neurons in the hidden layer are decided constructively. The proposed system deals with two class problem therefore only one neuron required at the output layer.

Training starts by firstly taking a single neuron in the hidden layer. The weights of this neuron are initialized as $(W_i^{quant})^g = (Q_{i1}^g, Q_{i2}^g, Q_{i3}^g, ..., Q_{in}^g)$, where $Q_i^1 = (q_{i1}^1 | q_{i2}^1)$ with k=2 and g=1. Here k is a subspace selection of weights, and g (user-defined variable) is the maximum number of generations to update quantum weights to get an optimized result. After initialization of quantum bits, we apply the observation process discussed in Section 2.2, to get a real coded value of connection weights $(W_i^{real})^g$. Then whole samples of class A and class B is applied to the neuron along with weight $(W_i^{real})^g$. Thus, after finalizing a neuron, it is checked against samples of both the classes A and B. We make use of the step activation function where $f(net_i)$ is compared with threshold λ which is defined manually.

Here, two parameters $count1$ and $count2$ have been taken, which describe the number of learned samples of class A and B, respectively. The local objective function is taken as sum_g, and the global objective function is taken as sum^*. To update the qubits of connection weights we make use of these objective function and their binary bits as shown in Table 3.1. The best fitness is selected as follows.

$$sum^* = max(sum^*, sum_g) \tag{3.7}$$

The overall process of Q-BNN in the form of Algorithm 3.1 is presented next.

Algorithm 3.1 Algorithm for Q-BNN.

1: Take the input samples as $(X_1, X_2, X_3, ..., X_{c_1})$ and $(Y_1, Y_2, Y_3, ..., Y_{c_2})$.
2: Take the first neuron in the hidden layer and initialize it with the weights W_g^{quant} in terms of quantum bits as follows:

$$(W_i^{quant})^g = (Q_{i1}^g, Q_{i2}^g, Q_{i3}^g, ..., Q_{in}^g) \tag{3.8}$$

3: where $g = 1, ..., m$; m is the number of generations to update weights.
4: $sum^* = 0$
5: $S^* = 0$
6: **for g=1 to m**
7: **Call observation process** $(W_i^{quant})^g$
8: **for i=1 to** c_1
9: $net_A(i) = \sum (W_i^{real})^g \times X_i$
10: **if**$(net_A(i) > \lambda)$
11: increase count1 by 1;
12: **endif**
13: **endfor**
14: **for j=1 to** c_2
15: $net_B(j) = \sum (W_i^{quant})^g \times Y_j$
16: **if**$(net_B(j) \leq \lambda)$
17: increase count2 by 1;
18: **endif**
19: **endfor**
20: $(sum_g = count1 + count2)$;
21: **if**$(sum_g \geq (c_1 + c_2)$ **)**
22: Stop learning.
23: else
24: sum^*=max(sum^*, sum_g)
25: Evaluate sum_g, sum^*, binary bits and update quantum bits to evolved the quantum weight $(W_i^{quant})^{g+1}$ using the Table 3.1.
26: **endif**
27: g=g+1
28: **endfor**
29: **if ((**$(g == m) \wedge (sum_g \leq (c_1 + c_2)$ **))**
30: Add new neuron for unlearned samples $((c_1 + c_2)$-$sum^*)$ and finalize its weights by using the Step-1 to Step-28.
31: **endif**

In the above-discussed Q-BNN algorithm the threshold is decided manually. Finding the threshold by using such formulation does not guarantee for optimal result. Therefore to overcome the drawback of Q-BNN, an enhanced NQ-BNN algorithm is proposed which is discussed next.

3.2.3 Novel Quantum inspired Binary Neural Network Learning Algorithm

In the proposed NQ-BNN algorithm we apply the evolutionary quantum computing concept to evolve two parameters of the neural network, i.e., connection weights and threshold. In this algorithm, we introduce novel quantum separability parameter $(\lambda_i^{quant})^t$ for evolving optimal value of threshold of neurons. We initialize it in terms of qubits as represented in Eq. (3.6). After initialization of quantum bits for quantum separability parameter $(\lambda_i^{quant})^t$, the observation process is used as discussed in Section 2.2, to achieve exploration and the real coded value of separability parameter $(\lambda_i^{real})^t$. The real coded value of separability parameter $(\lambda_i^{real})^t$ is compared with net_A corresponding to class A and net_B corresponding to class B at $g = 1$ to find out the optimal separability plane between two classes. In this algorithm, the connection weights $(W_i^{quant})^g$ are evolved in generations g=1:100. During each generation of evolution of connection weights, the quantum separability parameter $(\lambda_i^{quant})^t$ of neurons is evolved in generations t=1:100. After completion all generations to update quantum separability parameter t, the best value of $(\lambda_i^{real})^t$ is selected corresponding to the best value of the objective function. Now, quantum weights are updated for generation $g = 2$ and again the same process is implemented to find out the best value of $(\lambda_i^{real})^t$ in all generations of t. The process will continue for the generations g to find out the best value of weights $(W_i^{real})^g$ and quantum separability parameter $(\lambda_i^{real})^t$. To update quantum weights, $(W_i^{quant})^g$ and quantum separability parameter $(\lambda_i^{quant})^t$ quantum update Table 3.1 is used.

The NQ-BNN is designed to solve two class classification problem using the quantum computing concept. In this algorithm, some necessary parameters have been used, which are X_i and Y_j as the input samples of class A and class B, respectively. Two values $count1$ and $count2$ have been taken, which describe the number of samples of class A and B that are learned. Here, to update connection weights, the objective functions are taken as sum_g and sum^*. The sum_g is the objective function corresponding to weight update process in each generation, whereas, sum^* the best objective function for all the generations (g=1 to the current generation). To update quantum separability parameter, the objective function are sum_λ^* and sum^t. The sum^t is the objective function

of the current generation (t) whereas, sum^*_λ is the best objective function to update the separability parameter in all the generations of t (generation t=1 to the current generation). The overall process of NQ-BNN in the form of Algorithm 3.2 is presented next.

Algorithm 3.2 Algorithm for NQ-BNN.

1: Take input samples as $(X_1, X_2, X_3, ..., X_{c_1})$ and $(Y_1, Y_2, Y_3, ..., Y_{c_2})$.
2: Take the first neuron at hidden layer and initialize it with the weights W_g^{quant} in terms of quantum bits as follows:

$$(W_i^{quant})^g = (Q_{i1}^g, Q_{i2}^g, Q_{i3}^g, ..., Q_{in}^g) \tag{3.9}$$

3: where $g = 1, ..., m$; m is the number of generations to update weights.
4: $sum^* = 0$
5: $S^* = 0$
6: **for g=1 to m**
7: **Call observation process** $(W_i^{quant})^g$**.**
8: **Call Quantum Separability Parameter** $(W_i^{real})^g$**.**
9: sum^*=max(sum^*, sum_g).
10: **if(**$sum^* \geq (c_1 + c_2)$ **)**
11: Stop learning.
12: else
13: Evaluate sum_g, sum^*, $s_k^{g,i}$, $s_k^{*,i}$ and update quantum bits to evolve the quantum weight $(W_i^{real})^{g+1}$ by using (sum_g, sum^*), and
14: Table 3.1 for the same neuron.
15: **endif**
16: g=g+1
17: **endfor**
18: **if (**$(g == m) \wedge (sum^* \leq (c_1 + c_2)$ **))**
19: Add new neuron for unlearned sample ($(c_1 + c_2)$-sum^*) and finalize its weight by using the Step-1 to Step-16.
20: **endif**

Quantum Separability Parameter $(W_i^{real})^g$**.**

Step-1 : Initialization of different parameters.

for t=1 to z

z is generation to update $(\lambda_i^{quant})^t$

$(\lambda_i^{quant})^t = (\alpha_i^t \mid \alpha_{i+1}^t)$;

count1=0;

count2=0;

$sum^*_\lambda = 0$;

Call observation process $(\lambda_i^{quant})^t$ to generate real coded value

$(\lambda_i^{real})^t$

for i=1 to c_1

$net_A(i) = \sum(W_i^{real})^g \times X_i$

if($net_A(i) > (\lambda_i^{real})^t$)

increase count1 by 1;

endif

endfor

for j=1 to c_2

$net_B(j) = \sum(W_i^{real})^g \times Y_j$

if($net_B(j)(\lambda_i^{real})^t$)

increase count2 by 1;

endif

endfor

$(sum^t = count1 + count2)$;

sum_λ^*=max(sum_λ^*, sum^t)

update quantum bits for $(\lambda_i^{quant})^{t+1}$ by using (sum^t, sum_λ^*),

quantum update Table 3.1, and observation discussed in

Section 2.2 to get real coded value it.

sum_g=sum_λ^*

$t = t + 1$

endfor

return sum_g;

3.3 Experimental Evaluation

The proposed algorithms Q-BNN and NQ-BNN are tested on benchmark dataset like Breast Cancer dataset, Heart disease dataset, PIMA Indian diabetes dataset, and BUPA liver dataset. To evaluate the performance of proposed Q-BNN and NQ-BNN the comparison is made with other state-of-the-art approaches [11, 109–111].

3.3.1 Datasets and Experimental Settings

The experiment is carried on Intel core, I-5 processor with 4 GB RAM on Windows-7 operating system. The qubits of connection weights $(W_i^{quant})^g$ and quantum separability parameter $(\lambda_i^{quant})^t$ are initialized as 0.707|0.707. The displacement angle $(\Delta\theta)$ has been used as $0.03 * \Pi$. During this update process to ensure that qubit must not converge to "0" and "1" the limiting parameter ϵ has been taken as 0.001. The maximum number of generations for g and t are taken as 100.

3.3.2 Experimental Results and Discussion

In this section, first, the result and comparative analysis of the Q-BNN algorithm is presented, and then we present the results and comparative analysis of the NQ-BNN algorithm.

Results of Q-BNN algorithm

In this section, the performance of the Q-BNN algorithm is discussed on three benchmark datasets like Breast Cancer dataset, PIMA Indian diabetes dataset, and Heart disease dataset. The experiments are conducted using the 10-fold cross-validation scheme.

Table 3.2, shows the classification results on Breast Cancer dataset. It is observed that the best classification accuracy is achieved with set-4 and set-6 is 100%, and worst classification accuracy is achieved for set-7 is 99.254%.

Table 3.2, shows the classification results on PIMA Indian diabetes dataset. It is observed that the best classification accuracy is achieved with set-3 which is 97.91% and the worst classification accuracy is achieved with set-8 is 93.85%.

Table 3.2, shows the classification results on Heart disease dataset. It can be observed from Table 3.2, that the best classification accuracy is achieved for set-7, and set-9 is 92.64% and the worst classification accuracy achieved is 85.29% by set-1. In all the cases we see only few number of hidden layer neurons are required.

Comparative Result of Q-BNN Algorithm

The Q-BNN algorithm is compared with evolutionary quantum neural network learning algorithm [11]. The comparison is made on the parameters like the number of neurons in the hidden layer and testing accuracy. It can be observed from the Table 3.3, that the proposed Q-BNN algorithm perform better in terms of all parameters used for comparison. The constructive approach used here helps to get better accuracy with few numbers of hidden layer neurons. The number of neurons in the hidden layer to form a neural network architecture by the Q-BNN is 2, 2, and 2 whereas the number of neurons in the hidden layer required by the QNN algorithm is 12, 3, and 2 for the Breast Cancer, PIMA Indian Diabetes Dataset, and Heart disease dataset, respectively.

It is observed that in case of Breast Cancer dataset, number of hidden layer neurons are drastically reduced. The Q-BNN algorithm achieves better classification accuracy with respect to the QNN algorithm. The classification accuracy for the Breast Cancer dataset obtained by the Q-BNN algorithm and QNN algorithm is 99.63% and 99.59%, respectively. The classification accuracy for PIMA Indian Diabetes dataset obtained by the Q-BNN and the QNN algorithm is 85.6% and 76%, respectively. For the Heart disease dataset, the Q-BNN achieves 92.65 %, and the QNN achieves 79.41% classification accuracy. It can be observed that Q-BNN in all tested datasets achieves better classification accuracy with respect to QNN.

Table 3.2: Classification Accuracy and Number of Hidden Layer Neurons.

	Breast Cancer Dataset		PIMA Indian Diabetes Dataset		Heart Disease Dataset	
	Classification Accuracy (%)	Number of hidden layer neurons	Classification Accuracy (%)	Number of hidden layer neurons	Classification Accuracy (%)	Number of hidden layer neurons
set-1	99.325	2	96.354	2	85.294	2
set-2	99.895	2	96.547	2	89.705	2
set-3	99.473	2	97.916	2	91.176	2
set-4	100	2	96.352	2	88.235	2
set-5	99.985	2	96.625	2	89.705	2
set-6	100	2	95.312	2	88.235	2
set-7	99.254	2	96.875	2	92.647	2
set-8	99.524	2	93.857	2	89.705	2
set-9	99.521	2	95.521	2	92.647	2
set-10	98.375	2	94.257	2	91.176	2

Table 3.3: Comparison of Q-BNN with QNN in Terms of Classification Accuracy.

	Comparison for Breast Cancer Dataset		Comparison for PIMA Indian Diabetes Dataset		Comparison for Heart Disease Dataset	
Parameter	QNN	Q-BNN	QNN	Q-BNN	QNN	Q-BNN
Number of neurons at hidden layer	12	2	3	2	2	2
Classification Accuracy	99.59%	99.63%	76.00%	85.60%	79.41%	92.65%

Result of NQ-BNN Algorithm

In this section, we discussed the performance of NQ-BNN on three benchmark datasets, Breast Cancer dataset, PIMA Indian Diabetes dataset, and BUPA liver dataset. We also compare the proposed algorithm with the Q-BNN algorithm along with other state-of-the-art approaches in terms of classification accuracy.

Classification of Breast Cancer Dataset

In this experiment, the dataset has been split into 10-folds. The training of neural network with Breast Cancer dataset produces best results with two numbers of hidden layer neurons.

Table 3.4, shows the classification result in terms of various parameters as classification accuracy, and the number of neurons in the hidden layer. As shown in Table 3.4, the best classification accuracy is 99.95%, and worst classification accuracy is 98.65%. Table 3.5, shows the comparison of the proposed NQ-BNN with a novel discretization technique using the class attribute interval average

Table 3.4: Results of NQ-BNN for Breast Cancer Dataset.

	Hidden layer neuron	Classification Accuracy (%)
set-1	2	99.95
set-2	2	99.84
set-3	2	99.95
set-4	2	99.95
set-5	2	99.82
set-6	2	99.81
set-7	2	98.65
set-8	2	99.58
set-9	2	99.94
set-10	2	98.95
Average		99.65

and the Q-BNN in terms of classification accuracy [109, 112]. Table 3.5, shows that the proposed NQ-BNN achieves classification accuracy as 99.65%, which is far better than other state-of-the-art approaches.

Classification of PIMA Indian Diabetes Dataset

For the training and the testing purpose, the dataset has been split into 10 fold. Table 3.6, shows the classification results in terms of various parameters as classification accuracy, and the number of neurons in the hidden layer. The best results are achieved with three numbers of hidden layer neurons. Table 3.6, shows the best classification accuracy is 89.54%, and the worst classification accuracy is 87.21% and the average classification accuracy is 88.28%. It is observed that the proposed NQ-BNN produces good results in terms of training and classification accuracy. The proposed NQ-BNN is compared with MTiling-real algorithm [110, 112]. Table 3.7, shows the classification accuracy as compared to the MTiling-real algorithm. It is observed that it produces better results that is 88.29%.

Classification of BUPA Liver Dataset

For the training and the testing purpose, the dataset has been split into 10 fold. The training of neural network formed for BUPA liver dataset produces the best results with three numbers of hidden layer neurons. Table 3.8, shows the classification accuracy and number of hidden layer neurons of the proposed algorithm.

The best classification accuracy is achieved as 95.68%, and the worst classification accuracy is 93.47% and the average classification accuracy is 94.82%. Table 3.9, shows the comparison of classification accuracy with other state-of-the-art approaches [111, 113]. It is observed that the proposed NQ-BNN produces better results in terms of classification accuracy as 94.83% than other state-of-the-art approaches mentioned in Table 3.9.

Table 3.5: Comparison of Various Learning Algorithms with NQ-BNN on Breast Cancer Dataset in Terms of Classification Accuracy.

	Classification Accuracy (%)			
Methods	MLP Classifier	Naive Bayes Classifier	Decision Tree Classifier	Radial Basis Function Classifier
NQ-BNN*			**99.65**	
Q-BNN*			**99.63**	
EW	94.57	**97.28**	91.30	95.00
EF	94.71	96.28	90.80	95.00
ChiMerge	92.89	91.88	93.00	92.00
IEM	74.69	81.68	93.60	80.98
CAIM	93.56	93.99	93.80	93.42
CACC	95.14	95.28	94.10	94.85
CAIA	**95.42**	96.57	**96.57**	**95.99**

Table 3.6: Results of NQ-BNN for PIMA Indian Diabetes.

	Hidden layer neuron	Classification Accuracy (%)
set-1	3	88.58
set-2	3	88.73
set-3	3	88.56
set-4	3	89.34
set-5	3	89.54
set-6	3	88.21
set-7	3	87.54
set-8	3	87.65
set-9	3	87.25
set-10	3	87.41
Average		88.28

Table 3.7: Comparison of Various Learning Algorithms with NQ-BNN on PIMA Indian Diabetes Dataset in Terms of Classification Accuracy.

Methods	Classification Accuracy (%)
NQ-BNN	**88.29**
Q-BNN	85.6
MTiling-real	80.6
MPyramid-real	80.3
Perceptron	80.9

Table 3.8: Results of NQ-BNN for BUPA Liver Dataset.

	Hidden layer neuron	Classification Accuracy (%)
set-1	3	95.34
set-2	3	95.24
set-3	3	95.24
set-4	3	95.68
set-5	3	94.62
set-6	3	94.78
set-7	3	94.25
set-8	3	95.21
set-9	3	94.36
set-10	3	93.47
Average		94.82

Table 3.9: Comparison Of Various Learning Algorithms with NQ-BNN on BUPA Liver Dataset in Terms of Classification Accuracy.

Classification Algorithm	Accuracy (%)
NQ-BNN	**94.83**
QBNN-L	90.35
Logistic	67.39
Linear Logistic Regression	69.57
Gaussian Processes	73.91
Logistic Model Trees	68.12
Multilayer Perceptron	68.84
K-STAR	59.42
Rule Induction	64.49
SVM	69.23
Classification and Regression Trees	66.35

Discussion

Table 3.5, 3.7, and Table 3.9, shows the performance of the NQ-BNN algorithm. The results show that the performance of NQ-BNN is better as compared to the other state-of-the-art algorithms including the Q-BNN. The results are better due to the three main reasons. First, selection of connection weights and threshold using the evolutionary quantum computing concept. The quantum computing concept is characterized by population dynamics, individual representation, evaluation function. The observation process gives exploration by proving a large search space to find the optimal value of parameters. The quantum rotation gate provides exploitation to avoid the algorithm to stuck in the problem of local minima and maxima. Second, due to the constructive formation of network architecture. The constructive formation of neural network help to avoid adding of hidden layer neurons unnecessarily thus, good results are achieved with few numbers of neurons. There is also the possibility to get more neurons at the hidden layer which is not required actually to solve the problem.

3.4 Summary

In this chapter, first we have presented the Q-BNN algorithm. In the Q-BNN algorithm, we have used the evolutionary quantum computing concept to evolve connection weights of the neural network. The neural network is formed con-

structively by adding neuron one by one in the hidden layer. The proposed Q-BNN algorithm is trained and tested on benchmark datasets like Breast Cancer, PIMA Indian Diabetes dataset, and Heart disease dataset. The proposed algorithm performs well in comparison to other evolutionary algorithms as shown in the comparative study. In the Q-BNN, the threshold of the neuron is decided manually. Deciding threshold of the neuron does not guarantee for the optimal solution. To overcome this issue, the NQ-BNN algorithm is presented in this algorithm along with the connection weights of the neural network and the threshold of the neurons is decided using the quantum computing concept which is named as quantum separability parameter. Deciding separability parameter using the evolutionary quantum computing concept helps to achieve an optimal value of this parameter. The proposed NQ-BNN algorithm is trained and tested using benchmark dataset like Breast Cancer, PIMA Indian Diabetes dataset, and BUPA Liver dataset. The proposed performs better than the proposed Q-BNN algorithm as shown in comparative results. The proposed NQ-BNN algorithm is also compared with other state-of-the-art approaches, and it is found that the NQ-BNN performs better other state-of-the-art approaches.

Chapter 4

Quantum inspired Fuzzy based Neural Network Learning Algorithm for Two Class Classification Problem

4.1 Introduction

As discussed in the previous chapter, for forming a neural network architecture the quantum computing concept has been utilized which evolves the connection weights and threshold of the neurons. The performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes [24, 69, 114]. This causes due to the occurrence of overlapped samples in multiple classes of dataset. Thus, there is a need to apply fuzzy concept for handling such multiple class dataset which are having overlapped samples. The fuzzy concept helps to the classify such samples by assigning membership degrees according to the class from which these samples belong.

In the fuzzy clustering algorithm, there is an important parameter, that is fuzzifier (m) which is defined manually. The fuzzifier parameter (m) of fuzzy concept controls the extent of overlapping among fuzzy clusters [39]. The capability of handling overlapped samples using the fuzzy concept is dependent on value of m. Therefore, in this chapter, an algorithm is designed for the two class classification which makes use of the fuzzy concept by proposing a mech-

anism to decide fuzzifier value by using the quantum computing concept. The algorithm is named as Quantum inspired Fuzzy based Neural Network Learning Algorithm (Q-FNN). The Q-FNN forms three layers neural network architecture where neurons are considered as fuzzy neurons and the fuzzy clustering is utilized to decide the connection weights of the hidden layer neurons. The basic architecture of Q-FNN is shown in Figure 4.1.

4.2 Proposed Work

This section presents details of a proposed Q-FNN algorithm for solving two class classification problem. The proposed algorithm forms three layers neural network architecture using the fuzzy concept and the quantum computing concept. Here, the connection weights are considered as cluster centroids which are obtained using the fuzzy clustering and the fuzzifier (m) is evolved using the quantum computing concept. The Q-FNN uses concept of fuzzy clustering, which is discussed in brief as follows:

The fuzzy clustering attempts to partition a finite collection of n data samples $X = \{x_1, x_2, ..., x_n\}$ into c fuzzy clusters with $V = \{v_1, v_2, .., v_c\}$ cluster centroids. The inclusion of data samples in a cluster is described by a fuzzy partition matrix $U = [\mu_{ij}]_{n \times c}$ where μ_{ij} is the degree of membership at which data

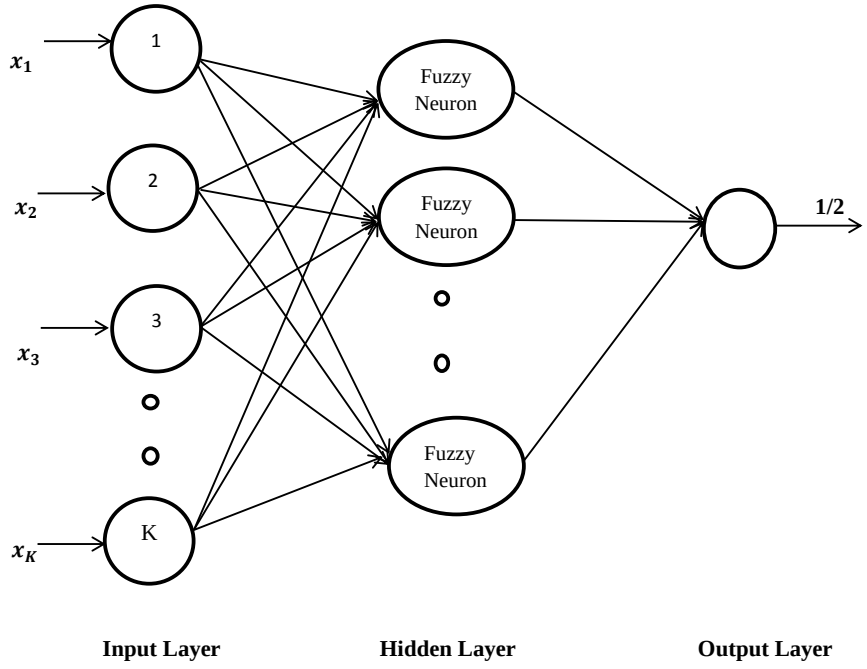

Figure 4.1: Architecture of Q-FNN.

sample x_i belongs to the cluster v_j. The membership degree is allowed to have any values between 0 and 1. The FCM algorithm is based on the minimization of following objective function, defined as follows:

$$J_m(U,V) = \sum_{i=1}^{c}\sum_{j=1}^{n}(\mu_{ij})^m \|x_j - v_i\|^2, m > 1 \tag{4.1}$$

where, m ($m > 1$) is a fuzzifier parameter controls the fuzziness of resulting clusters. The fuzzy clustering algorithm is based on the iterative optimization of objective function given in Eq. (4.1) by updating the membership matrix U and cluster centroid V. The fuzzy clustering algorithm is discussed in detail in the form of pseudo code in Section 2.3.

The Q-FNN architecture consists of three layers: an input layer, a hidden layer, and an output layer. Let $X = \{x_1, ...x_n\}$ is the training set consists of total n number of data samples, where each data sample $x_i \in R^K$ and K represents the dimensionality of the data sample. The training patterns x_i belongs to one of the l classes. The nodes in the input layer are equal to the dimensions (K) of the data samples. The learning of hidden layer and output layer is detailed in coming sections.

4.2.1 Learning of Hidden Layer

In the proposed Q-FNN architecture, hidden layer training is done by using the concept of fuzzy clustering. Since fuzzy clustering is used for the hidden layer learning, therefore at least two neurons (clusters) are required initially to start the learning of the hidden layer. Then, we add more neurons in the hidden layer if all samples of the training set are not learned with the existing neurons. The learning of a hidden layer comprised of the following steps: deciding weights of the neurons, weight update process, and handling unclassified samples. The detailed description of these steps is presented as follows.

Step a: Deciding weights of the neurons

Suppose, training set $X = \{x_1, ...x_n\}$ composed of unclassified data samples belongs to two classes, i.e., A and B. Let us denote class A which is made up of data samples (x_1^A,....,x_i^A) and class B is composed of data samples (x_{i+1}^B,....,x_n^B).

The initial connection weights for the hidden layer neurons composed of data samples belong to class A is computed as follows:

$$xmin_j^A = \min(x_{1j}^A, ..., x_{ij}^A) \tag{4.2}$$

$$xmax_j^A = \max(x_{1j}^A, ..., x_{ij}^A) \tag{4.3}$$

where $xmin_j^A$ and $xmax_j^A$ are the minimum and maximum values on a j^{th} dimension of all data samples belong to class A.

$$w_j^A = \frac{xmin_j^A + xmax_j^A}{2} \tag{4.4}$$

where w_j^A denote the connection weight in a j^{th} dimension of the neuron belongs to class A. The weight vector for a neuron associated with class A is represented as:

$$w^A = [w_1^A, ..., w_K^A] \tag{4.5}$$

Similarly, the initial value of connection weights for the hidden layer neuron belongs to class B is defined as:

$$xmin_j^B = \min(x_{i+1j}^B, ..., x_{nj}^B) \tag{4.6}$$

$$xmax_j^B = \max(x_{i+1j}^B, ..., x_{nj}^B) \tag{4.7}$$

where $xmin_j^B$ and $xmax_j^B$ denote the minimum and maximum values on a j^{th} dimension of all data samples belong to class B.

$$w_j^B = \frac{xmin_j^B + xmax_j^B}{2} \tag{4.8}$$

where w_j^B denotes the connection weight in a j^{th} dimension of the neuron belong to class B. Initially, we associate one neuron with class B and denote the weight vector for that neuron as:

$$w^B = [w_1^B, ..., w_K^B] \tag{4.9}$$

Finally, the set of weight vector W_h for the hidden layer is represented as

$$W_h = \begin{bmatrix} w_1^A & w_2^A & ... & w_K^A \\ w_1^B & w_2^B & ... & w_K^B \end{bmatrix}$$

Step b: Weight update process

In Q-FNN architecture, the learning of the hidden layer starts by using the concept of fuzzy clustering. In this architecture, as the fuzzy clustering concept is used for learning, therefore the neurons in the hidden layer represent the fuzzy clusters (c). The weight matrix (W_h) of the hidden layer neurons are taken equivalent to cluster centroids (V). The fuzzifier parameter (m) plays an important role in the performance of the fuzzy clustering algorithm. Therefore, the fuzzifier parameter is evolved using the quantum computing concept, and it is represented in terms of quantum bits as follows:

$$m^g = Q_i^g \tag{4.10}$$

where Q_i^g consists of two-qubits which are represented as follows:

$$Q_i^g = (q_{i1}^g | q_{i2}^g) \tag{4.11}$$

In Q-FNN, the two qubits are enough to represent the fuzzifier parameter (m) to find its value in the range of [1.5, 2.5] [79]. These two qubits divide search space into four subspaces and find the value of m from these subspaces probabilistically rather than sequentially. Using the observation process discussed in Section 2.2, we get a real coded value of fuzzifier parameter (m^g), and it is represented as (M_{real}^g).

After initialization of all the parameters, update the weights of the hidden layer by executing the fuzzy clustering stated in Section 2.3, with a real coded value of fuzzifier parameter evolved using the quantum computing concept. After updating the weights using the fuzzy clustering for a generation (g), the cluster membership matrix corresponding to the trained data samples is obtained. On the basis of obtained cluster membership matrix, the degree of overlap corresponding to each data sample is computed to determine the number of learned samples corresponding to the formed clusters or neurons. The

degree of overlap $\delta(x_i)$ corresponding to each data sample is computed using the inter-cluster overlap measure [115]. The overlap of each data sample x_i in the generation (g) between the two fuzzy clusters F_l and F_r is computed as follows:

$$Dom^g_{\min}(x_i) = \min(\mu^g_{F_l}(x_i), \mu^g_{F_r}(x_i)) \tag{4.12}$$

$$Dom^g_{\max}(x_i) = \max(\mu^g_{F_l}(x_i), \mu^g_{F_r}(x_i)) \tag{4.13}$$

where, $\mu^g_{F_l}(x_i)$ and $\mu^g_{F_r}(x_i)$ represents the membership degree corresponding to fuzzy clusters F_l and F_r, $Dom^g_{\max}(x_i)$ and $Dom^g_{\min}(x_i)$ is the maximum and minimum degree of membership of each data sample x_i. If a data sample is highly vague, i.e., $Dom^g_{\max}(x_i)$=0.5, it means that data sample belongs to both the clusters with an equal degree of membership, then a degree of overlap $\delta(x_i) = 1$ otherwise the degree of overlap $\delta(x_i)$ is considered as 0. If data sample has $\delta(x_i) = 1$, then the data sample is considered as unclassified. If $\delta(x_i) \neq 1$ then data sample is considered learned to the respective neuron if the data sample actual class is perfectly matched to the neuron class otherwise, it is considered as unclassified. Thus, on the basis of $\delta(x_i)$, we proposed the modified step activation function of the neuron by taking care of overlapped samples, which is defined as follows:

$$net^j_h = \delta(x_i) \tag{4.14}$$

$$f(net^j_h) = \begin{cases} (1 \; or \; 2) \; (\; class \; number) & if \;\; net^j_h \neq 1 \\ 0 \; (miss - classified) & if \;\; net^j_h = 1 \end{cases} \tag{4.15}$$

where, $f(net^j_h)$ is the activation function for the hidden layer, such that j=1, 2, 3, .., T, T is the number of neurons in the hidden layer.

Here, in the proposed algorithm the Number Of Correctly Learned Samples (NCLS) is calculated as the fitness function. It is most challenging, and yet an essential concept in learning algorithm is the fitness function. In Q-FNN architecture, the NCLS in each generation represents the local fitness function denoted by $F^g_{Lbest}(M^g_{real}, V^g)$ and is defined as:

$$F^g_{Lbest}(M^g_{real}, V^g) = NCLS \tag{4.16}$$

In Q-FNN, we compute a global fitness function which stores the maximum

Table 4.1: Qubits Update for Q-FNN.

s_i^g	s_i^{global}	$F_{Gbest}((M_{real})^{best}, V_{best}) > F^g_{Lbest}(M^g_{real}, V^g))$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	*0*
1	*0*	*true*	$0.03 * \Pi$
1	*1*	*false*	*0*
1	*1*	*true*	*0*

NCLS among all the generations represented by $F_{Gbest}((M_{real})^{best}, V_{best})$ and is defined as:

$$F_{Gbest}((M_{real})^{best}, V_{best}) = \max(F_{Gbest}((M_{real})^{best}, V_{best}), F^g_{Lbest}(M^g_{real}, V^g)) \tag{4.17}$$

where $(M_{real})^{best}$ and V_{best} denote the best value of fuzziness parameter and cluster centroids generated corresponding to the formed clusters or neurons.

The value of fitness functions depends on the optimal value of fuzzifier parameter which is evolved using the quantum computing concept. The real coded value of fuzzifier parameter in the first generation may or may not be optimal. Therefore, to get optimal value of it, we need to update qubits of fuzzifier parameter. The qubits are updated using quantum update process discussed in Section 2.2 and Table 4.1. The quantum update process provides proper exploitation to search a global optimal value of fuzzifier parameters in several generations.

Step c: Handling unclassified samples

Suppose, unclassified data samples left in the training set X that is not learned with the existing neurons. So, for the training of these data samples in the hidden layer, we are using again the fuzzy clustering. Therefore, at least two additional neurons (clusters) are required in the hidden layer. It might be possible that the unclassified data samples present in the training set X belongs to two classes or it may consist of data samples belong to only one class. Thus, on the basis of two possibilities, the weights of the neurons are decided as follows:

1. **Unclassified data samples belong to two classes**: If the unclassified data samples present in set X belongs to two classes, then the weights of the neurons for the training of these data samples is decided by using Step a, and the weights of the neurons are updated by using Step b.

2. **Unclassified data samples belong to single class**: If unclassified data samples present in set X belong to a single class. Then for the training of these data samples, the connection weights of neurons are computed as follows:

   (a) Compute the Euclidean distance of each data sample x_i from the origin (O), i.e., (0,0) which is computed as follows:

   $$d(x_i, O) = \|x_i - O\| \tag{4.18}$$

   (b) Find the nearest and farthest data samples from the origin on the basis of Euclidean distance and separate these data samples from other data samples. Let x_p and x_q be the near and far data sample from the origin.

   (c) Calculate the Euclidean distance of the rest of the data samples from x_p and x_q.

   (d) On the basis of Euclidean distance, separate the data samples which are closer to the x_p from the data samples closer to the x_q. Let $X_N = \{x_1, .., x_p\}$ and $X_F = \{x_q, .., x_s\}$ be the set of data samples closer to the near and far data sample.

   (e) Compute the average of all the data samples present in the set X_N.

   $$w^N = \frac{1}{p} \sum_{i=1}^{p} x_i \tag{4.19}$$

   (f) Compute the average of all the data samples present in the set X_F.

   $$w^F = \frac{1}{s} \sum_{i=q}^{s} x_i \tag{4.20}$$

(g) Finally, the weight vector w_h for the hidden layer is denoted by

$$W_h = \begin{bmatrix} w_1^N & w_2^N & ... & w_K^N \\ w_1^F & w_2^F & ... & w_K^F \end{bmatrix}$$

After deciding the weights of neurons for the unclassified data samples belong to only one class, the learning of these neurons and the weight update process is performed using Step b. The overall process of Q-FNN in the form of Algorithm 4.1, is presented next.

Algorithm 4.1 Algorithm for Q-FNN

Input Training data $X=\{x_1, ..x_n\}$.
Output Q-FNN architecture.
Begin
Initialize the nodes in the input layer equal to the dimensions (K) of data sample x_i where $i = 1, .., n$ and $x_i \in R^K$.
Initialize neurons in the hidden layer
$X^{previous}=\phi$
while ($X \neq \phi$) **do**
 Initialize the weights of hidden layer neurons.
 if data samples in X belong to two classes **then**
  Decide the weights of the neurons using **step a**
 else if data samples in X belong to only single class **then**
 Decide the weights of the neurons using point 2 of **step c**
 end if
 Start learning of hidden layer by initializing weight matrix (W_h) as cluster centroids for generations (g=1) denoted by V^g and fuzziness parameter using Eq. (4.10) and Eq. (4.11) denoted by Q_i^g.
 repeat
  Obtain real coded value M_{real}^g corresponding to the quantum value of fuzziness parameter m^g using an observation process.
  Update weights by executing FCM algorithm (Algorithm 2) using the parameters M_{real}^g and V^g.
  Compute the degree of overlap of each data samples using Eq. (4.12) and Eq. (4.13).
  Compute the number of correctly learned sample with the proposed step activation function discussed in Eq. (4.14) and Eq. (4.15).
  Fitness Evaluation Compute local and global fitness function using Eq. (4.16) and Eq. (4.17) to find the number of correctly learned sample in generation (g).
  Store the best cluster centroid and fuzziness parameter in each generation g.
  if $(F_{Lbest}^g(M_{real}^g, V^g) \geq F_{Gbest}((M_{real})^{best}, V_{best}))$ **then**
   $V^{g+1} = V^g$
   $V_{best} = V^g$
   $(M_{real})^{best} = M_{real}^g$
  else

Algorithm 4.1 (Continued)

V^{g+1}=V_{best}
V_{best}=V_{best}
$(M_{real})^{best} = (M_{real})^{best}$
end if
Update the quantum bit of fuzziness parameter using Table 4.1.
if $((F_{Gbest}((M_{real})^{best}, V_{best})==n)$ or $(g==g_{\max}))$ **then**
break
else
g=g+1
end if
until$((g > g_{\max})$ or $(F_{Gbest}((M_{real})^{best}, V_{best}) == n))$
Separate the learned data samples from set X and store the unclassified data samples in set $X^{previous}$.
if$(X == X^{Previous})$
break
end if
$X = X^{previous}$
Again add two neurons in the hidden layer for learning of remaining data samples present in set X.
end while
return
End

After learning of the hidden layer, the total number of clusters (c) represents the final number of hidden layer neurons, and the final cluster centroids (V) represents the final connection weights (W_h) from the input layer to the hidden layers. Thus, each hidden layer neuron will produce an output (O_h^j) (where $j = 1, 2, .., t$, t is the number of neurons in the hidden layer) in the following manner: if a data sample is classified on the basis of $f(net_h^j)$ to class 1, then hidden layer neuron (O_h^j) will produce output +1 else 0. Similarly, if a data sample is classified on the basis of $f(net_h^j)$ to class 2, then hidden layer neuron (O_h^j) gives an output -1 else 0.

4.2.2 Learning of Output Layer

The proposed Q-FNN classifier is designed to solve the two-class classification problems. So, only one output neuron is required in the output layer which will combine the outputs of all hidden layer neurons. The weights from each hidden neuron to the output neuron is set to 1 and denoted by w_O^j where $j = 1, 2, .., t^{'}$, $t^{'}$ is the number of neurons in the hidden layer. In output layer, we calculate the output value of neuron (net_O) by summing the product of (O_h^j) and the weight

link w_O^j and applying the step activation function. If the output of (net_O) is greater than 0, the value of $f(net_O)$ becomes 1 and data samples belongs to class A otherwise it becomes 2 and data samples belong to class B.

4.3 Experimental Evaluation

In this section, we evaluate the performance of the proposed Q-FNN algorithm on two class datasets. The performance of the proposed algorithm is also compared with other state-of-the-art approaches [116–120].

4.3.1 Datasets and Experimental Settings

The performance of the proposed algorithm is tested on WBCD, Sonar, Hepatitis, Ionosphere, and Heart dataset. The experiments implemented in MATLAB R2014a were conducted on Intel core, I-5 processor with 4 GB RAM on Windows-7 operating system. To compute the generalizability of the proposed approach in comparison with existing work in literature, we divided the training and testing data into two different partitions, i.e., 60-40 training-testing partition and 10-fold cross validation. In 10-fold cross validation, the entire dataset is divided into ten blocks of approximately equal size. During the implementation of our algorithm, 90% of the data is used to train our model while the rest 10% data is used for testing. The same process is repeated 10 times and each time a different set of data is used for training and testing. The maximum number of generations for g is taken as 100.

4.3.2 Performance on Benchmark Datasets

We evaluate the classification capability of the Q-FNN classifier on benchmark datasets. The performance of Q-FNN is evaluated and reported on these datasets for different training and testing partitions in terms of classification accuracy, sensitivity, and specificity in Table 4.2. It can be seen from the results that Q-FNN achieves more than 90% accuracy for Sonar, Hepatitis, WBCD, and Heart dataset. The classification accuracy achieved for WBCD, Sonar, Hepatitis, and Heart is 99.85%, 94.74%, 97.75%, and 94.44%, respectively. For other datasets like Ionosphere, the classification accuracy achieved more as 86.21%.

Table 4.2: Classification Results of Q-FNN on Different Training-Testing partitions.

Datasets (# samples x # features x # class)	Partitions	Classification Accuracy (%)	Sensitivity (%)	Specificity (%)
WBCD (683x9x2)	60-40	96.94	98.15	94.01
	10-fold	**99.85**	99.91	98.89
Sonar (208x60x2)	60-40	94.05	95.56	92.31
	10-fold	**94.74**	90.00	100
Hepatitis (80x19x2)	60-40	86.67	92.59	81.82
	10-fold	**97.75**	100	87.50
Ionosphere (351x33x2)	60-40	85.00	96.00	78.89
	10-fold	**86.21**	92.31	81.25
Heart (270x13x2)	60-40	**94.44**	95.83	93.33
	10-fold	85.19	91.67	80.00

Table 4.3: Comparison of Various Learning Algorithms with Q-FNN on Datasets in Terms of Classification Accuracy.

Classifier	Dataset				
	WBCD	Sonar	Hepatitis	Ionosphere	Heart
Q-FNN	99.85	94.74	97.75	86.21	94.44
NQ-BNN	99.65	79.81	76.55	80.32	92.95
SOM-INN	95	86	89	-	79
SCSR	97	78	-	93.2	81.7
GaX	99.54	81.12	90.7	89.89	87.7
SaX	97.52	81.15	90.61	89.66	87.25
Multitree GP	97.95	79.81	-	91.74	82.96

Thus, it can be inferred from the results that Q-FNN achieves remarkable results on almost all the datasets. Also, Q-FNN achieved good sensitivity and specificity for all the datasets.

Furthermore, the classification performance of Q-FNN is compared with several classifiers on 5 benchmark datasets and reported in Table 4.3. It can be seen from the reported results that performance of proposed Q-FNN model is compared with the variety of classifiers such as hybrid NFS, evolutionary approach, i.e., GA, PSO, QNN, and many more. The results show that Q-FNN achieved remarkable results, this improvement is mainly because the proposed method finds the optimal weights of the network by performing learning using the fuzzy clustering with an evolution of fuzzifier parameter using the quantum computing concept.

4.3.3 Discussion

This can be easily observed that Q-FNN can produce a compact neural network structure with good accuracy on most of the datasets. This is because of four major features of Q-FNN. Firstly, the basic idea of forming a neural network architecture using the fuzzy concept. The fuzzy concept helps to classify such samples by assigning membership degrees according to the class from which these samples belong. Thus, the neural network formed using this concept learned for almost all training samples, which helps to get better generalization accuracy.

Secondly, the selection of fuzzifier parameter using the quantum computing

concept. The observation process provides exploration by providing region-by-region promising subspaces. On the other hand, the quantum rotation gate helps to achieve exploitation. The optimal value of fuzziness parameter controls the overlapping among the formed clusters or neurons. Thus, it proves the high generalization ability. Thirdly, the initialization and selection of connection weights. Here, connection weights have been taken as the centroid of clusters of two classes using the fuzzy concept. The random initialization of cluster centroids increases the number of iterations to find optimal centroids. Therefore, in this algorithm, the initialization of cluster centroids or connection weights by finding the minimum and maximum point helps to reduce unnecessary iterations to reach stable or optimal centroids or connection weights. Fourthly, is selection of the number of hidden layer neurons constructively to neural network architecture. In addition to this, the proposed modified step activation function is composed of membership degree and fuzzifier parameter that helps in the formation of hidden layer neurons.

4.3.4 Summary

In this chapter, we have presented Q-FNN algorithm to solve the two-class classification problem. In the Q-FNN algorithm, the neural network architecture is formed constructively by integrating the concept of fuzzy clustering and quantum computing. The fuzzy concept is used to find the connection weights and architecture of the neural network. In addition to this, the evolutionary quantum computing is used for evolving the fuzzifier parameter m in several generations. A series of empirical studies on 5 benchmark datasets has been conducted to evaluate the performance of the Q-FNN. Different experimental configurations have been adopted to provide a fair comparison with the variety of state-of-the-art approaches. The empirical findings reveal that Q-FNN yields better performance in comparison with reported approaches in terms of classification accuracy. The proposed model achieves remarkable classification accuracy, using 10-fold cross validation scheme. From the empirical evaluation, we conclude that our proposed Q-FNN obtains very high accuracy for most of the datasets.

Chapter 5

Quantum inspired Fuzzy Based Neural Network Learning Algorithm for Multi-class Classification Problem

5.1 Introduction

In the real-life application, there are several examples of multi-class problems like disease diagnosis, document classification, handwriting recognition, human face recognition, etc. In multi-class problems, there is more probability that dataset samples belong to more than one classes. This causes due to the occurrence of overlapped samples in multiple classes of the dataset. The Q-FNN algorithm presented in the previous chapter is unable to handle the multi-class dataset. In Q-FNN, connection weights/cluster centroids have been evolved using the fuzzy concept, which was initialized manually by finding midpoints of data samples and the fuzzifier parameter m has been optimized using the quantum computing concept. The major problem with the fuzzy based algorithms is that the algorithms are sensitive to the initialization of parameters. Choosing the initial cluster centroids is extremely important as it has a direct impact on the formation of final clusters which are considered as neurons of the hidden layer. The random selection of these parameters does not guarantee unique clustering results.

Here, in this chapter, to overcome both the issues, i.e., classification of the multi-class dataset and optimal selection of cluster centroids, we enhanced our Q-FNN algorithm and proposed the Quantum inspired Fuzzy based Neural Network Learning Algorithm for Multi-class classification problems (Q-FNNM). In this algorithm, the cluster centroids/connection weights are evolved using the quantum computing concept along with fuzzifier parameter. The proposed algorithm is discussed in detail next.

5.2 Proposed Work

In the proposed Q-FNNM algorithm the fuzzy clustering algorithm has been used for learning of neural network. The quantum computing concept has been used to optimize the parameters of fuzzy clustering. The quantum computing concept helps to find the optimal value of the cluster centroids V_g^{real} and fuzzifier parameter M_{real}^g. These cluster centroids V_g^{real} are considered as connection weights $(W_g^{real})^h$ of the hidden layer neurons. The detail description of initialization of various parameters and learning of neural network learning algorithm is discussed next.

5.2.1 Preliminaries

The proposed algorithm is trained and tested for the diagnosis of the multi-class dataset. Let $X = \{x_1, ...x_n\}$ is the training set consists of total n number of data samples, where each data sample $x_i \in R^K$ and K represents the dimensionality of the data sample. The training patterns x_i belongs to one of the classes.

5.2.2 Learning of Hidden Layer

The learning of neural network starts with the initialization of parameters using the quantum computing concept. Thus, quantum bits are used to represent these parameters. It is further converted to real coded value using the observation process discussed in Section 2.2. The real coded value is utilized in the learning of hidden layer neurons which are treated in the form of fuzzy clusters. Before discussing the learning algorithm, first initialization of parameters has been discussed in detail.

Initialization of Parameters

The Q-FNNM consists of three layers: an input layer, a hidden layer, and an output layer. Here input dataset consists of n number of data samples and each data sample x_i have K dimensionality. Thus, the number of input nodes are equal to the dimension K of the data samples. Here, the number of neurons in the hidden layer is decided constructively. Initially, in the hidden layer, we take the number of neurons equal to the number of classes of the input dataset. The basic architecture of Q-FNN is shown in Figure 5.1. More neurons are added constructively during learning if all data samples are not learned.

Let's denote the training set $X = \{x_1, ...x_n\}$ composed of data samples having three classes, i.e., A, B, and C. It is represented as (x_1^A,....,x_e^A), (x_{f+1}^B,....,x_f^B), and (x_{f+1}^C,....,x_n^C), respectively. In this algorithm, the centroids of each class are considered as the connection weights of the hidden layer. Each centroid or connection weight is represented in terms of quantum bits.

$$(W_g^{quant})^{cn} = (V_g^{quant})^{cn} \tag{5.1}$$

$$(V_g^{quant})^{cn} = (Q^{gw})^{cn} \tag{5.2}$$

Where g is the number of generations to evolve the connection weights and

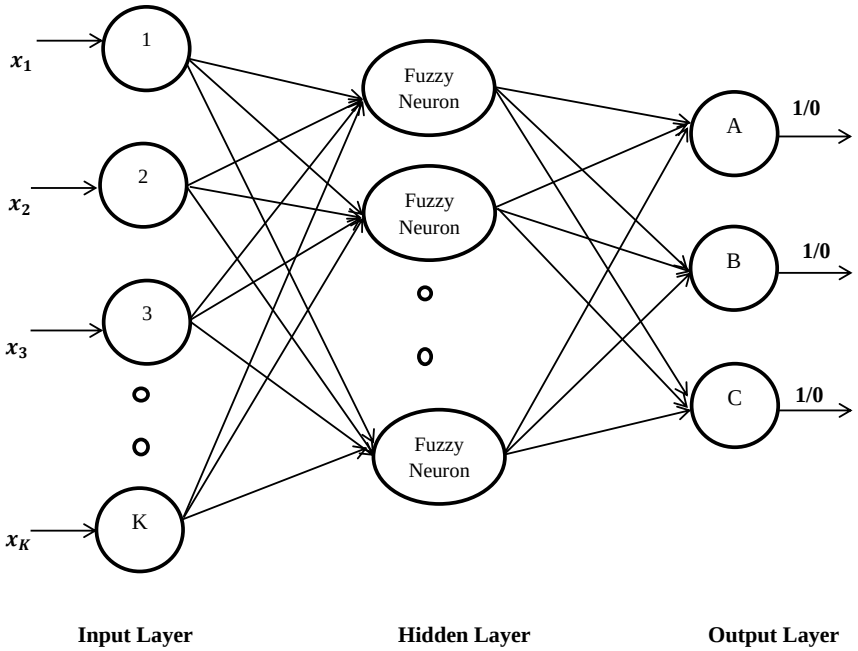

Figure 5.1: Architecture of Q-FNNM.

cn shows the class number. The quantum bit is made of qubits, so we have represented a quantum bit with the help of two qubits as follows:

$$(Q^{gw})^{cn} = ((q_1^{gw})^{cn}|(q_2^{gw})^{cn}) \tag{5.3}$$

The four search spaces are provided by a quantum bit to search the optimal value of centroids/connection weights. The search spaces are formed using input data samples. These search spaces are created with the help of the minimum sample value of the dataset and the maximum sample value of the dataset. For example, the search spaces to find centroid of class A, is generated by finding the minimum data sample value of class A as $xmin_j^A$ and maximum data sample value of class A as $xmax_j^A$. We divide the obtained space between these minimum and maximum data samples into four search spaces using the following formulations.

$$xmin_j^A = \min(x_{1j}^A, ..., x_{ij}^A) \tag{5.4}$$

$$xmax_j^A = \max(x_{1j}^A, ..., x_{ij}^A) \tag{5.5}$$

$$z = (xmax_j^A - xmin_j^A)/4 \tag{5.6}$$

Thus, the four search spaces are used here for finding the connection weights are 1 - ($xmin_j^A$, $xmin_j^A + t$), 2 - ($xmin_j^A + t$, $xmin_j^A + 2t$) 3 - ($xmin_j^A + 2t$, $xmin_j^A + 3t$), and 4 - ($xmin_j^A + 3t$, $xmax_i^A$). Similarly the search spaces are created for class B and class C, respectively.

These weight matrix $(W_g^{quant})^h$ for all hidden layer neurons are converted into real coded value using the observation process and represented as $(W_g^{real})^h$. Here, h shows the connection weights of all hidden layer neurons. The real value corresponding to quantum centroid $(V_g^{quant})^{cn}$ is $(V_g^{real})^{cn}$.

$$(W_g^{quant})^h = \begin{bmatrix} (V_1^{quant})^A & (V_2^{quant})^A & ... & (V_e^{quant})^A \\ (V_1^{quant})^B & (V_2^{quant})^B & ... & (V_f^{quant})^B \\ (V_1^{quant})^C & (V_2^{quant})^C & ... & (V_n^{quant})^C \end{bmatrix} = \begin{bmatrix} (Q_1^w)^A & (Q_2^w)^A & ... & (Q_e^w)^A \\ (Q_1^w)^B & (Q_2^w)^B & ... & (Q_f^w)^B \\ (Q_1^w)^C & (Q_2^w)^C & ... & (Q_n^w)^C \end{bmatrix}$$

Similarly, the real value matrix can be represented as follows:

$$(W_g^{real})^h = \begin{bmatrix} (V_1^{real})^A & (V_2^{real})^A & ... & (V_e^{real})^A \\ (V_1^{real})^B & (V_2^{real})^B & ... & (V_f^{real})^B \\ (V_1^{real})^C & (V_2^{real})^C & ... & (V_n^{real})^C \end{bmatrix}$$

Similarly, the fuzzifier parameter is also represented in terms of a quantum bit as (m^g). It plays an important role in fuzzy clustering, as it controls the degree of overlapping.

$$m^g = Q^{gm} \tag{5.7}$$

$$Q_{gm} = (q_1^{gm}|q_2^{gm}) \tag{5.8}$$

The real coded value corresponding to the quantum fuzzifier parameter is represented as M_{real}^g, which is evolved using the observation process. The search space for finding the optimal value of M_{real}^g is selected between [1.5,2.5] as suggested by Pal and Bezdek [79]. For finding the optimal value of M_{real}^g, the quantum bit is represented in terms of two qubits. It means that the value of M_{real}^g is found from the four search spaces.

Learning Process

Once, the real coded value of connection weight matrix $(W_g^{real})^h$ and fuzzifier parameter M_{real}^g is achieved, it is used in fuzzy clustering to perform the learning. Now, we execute the fuzzy clustering for evolving the parameters. The fuzzy clustering is executed and a membership matrix is achieved corresponding to each data sample of each class. On the basis of membership matrix, the degree of overlap is computed corresponding to each data sample. The degree of overlap shows the number of samples learned corresponding to the formed clusters or neurons. The degree of overlap is computed using the inter-cluster overlap measure as suggested by Bharill et al. [115]. The overlap of each data sample x_i between three fuzzy clusters F_1, F_2, and F_3 is computed as follows:

$$Dom_{\min}^g(x_i) = \min(\mu_{F_1}^g(x_i), \mu_{F_2}^g(x_i), \mu_{F_3}^g(x_i)) \tag{5.9}$$

$$Dom_{\max}^g(x_i) = \max(\mu_{F_1}^g(x_i), \mu_{F_2}^g(x_i), \mu_{F_3}^g(x_i)) \tag{5.10}$$

where, $Dom^g_{\max}(x_i)$ and $Dom^g_{\min}(x_i)$ represent the minimum and maximum degree of membership of data sample x_i and $\mu^g_{F_j}(x_i)$ represents the membership degree corresponding to fuzzy clusters F_j where j=1,..,3. If $Dom^g_{\max}(x_i) \geq 0.5$, means there is no overlapping and a data sample has been learned for a cluster or neuron whose membership degree is $\mu^g_{F_j}(x_i) \geq 0.5$ else a data sample is not learned for any cluster or neuron. Here, $f(net^{cn}_i)$ is a parameter which acts as an output of the hidden layer neuron, and it shows the number of data samples learned. It can be represented as a step activation function of neurons as follows:

$$f(net^{cn}_i) = \begin{cases} (j \;\; class\; number) & if \;\; \mu^g_{F_j}(X_i) \geq 0.5 \\ 0 \; (miss - classified) & if \;\; \mu^g_{F_j}(X_i) < 0.5 \end{cases} \tag{5.11}$$

Here, value of $f(net^{cn}_i)$ shows the number of learned samples, but it does not show the number of correctly learned samples. For example, if a sample x_i belongs to class A, then it belongs to a neuron 1 or cluster 1. If it is learned in class B, then $f(net^B_i)$ will show value 2 using Eq. (5.11) which shows sample learned in class B. It means that it is not correctly learned. To solve this issue, here a fitness function is introduced which shows the NCLS corresponding to the correct neuron. The local fitness function corresponding to each generation is denoted as F^g_{Lbest}.

$$F^g_{Lbest} = NCLS \tag{5.12}$$

As discussed in Section 2.2, that the real value of $(W^{real}_g)^h$ and M^g_{real} obtained in the first generation (g=1) may or may not be optimal. Therefore, there is a need to evolve both the parameters in several generations for better learning of neural network. To evolve the new value of these parameter we require to update the qubit of centroids or weights and the fuzzifier parameter. Here, we use a quantum update process as discussed in Section 2.2 and Table 5.1 which is required in current fitness value F^g_{Lbest}, and the best fitness value F^g_{Gbest} that shows the best fitness among all the previous generations. The best fitness value which stores the maximum number of learned samples among all the previous generations.

$$F^g_{Gbest} = \max(F^g_{Gbest}, F^g_{Lbest}) \tag{5.13}$$

Here we also store the quantum value and real coded of connection weights

Table 5.1: Qubits Update for Q-FNNM.

s_w^g/s_m^g	$s_w^{global}/s_m^{global}$	$F_{Gbest}^g > F_{Lbest}^g$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	*0*
1	*0*	*true*	$0.03 * \Pi$
1	*1*	*false*	*0*
1	*1*	*true*	*0*

parameter as W_{global}^{quant}, W_{global}^{real} and fuzzifier parameter as m_{global}^{quant}, M_{global}^{real} corresponding to the best fitness value. These values are required to evolve the qubits and real coded value of connection weights and fuzzifier parameter using the quantum update process.

The F_{Lbest}^g shows the number of samples learned in the current generation. If F_{Lbest}^g shows that all samples that are learned with the number of neurons taken initially in the hidden layer, then we stop learning else, we compare the current fitness value (F_{Lbest}^g) with the global fitness value (F_{Gbest}^g) and accordingly we update the connection weights and fuzzifier parameter using the qubit update process.

Still if we find the number of samples unlearned after completion of all generations which evolve W_g^{quant} and m^g then in such condition, we check the number of samples remained unlearned and their corresponding class. If samples of all the three classes are unlearned, then we add three more hidden layer neurons. After this, we follow the same process for the learning of unlearned samples as discussed above.

The overall process of Q-FNNM in the form of Algorithm 5.1, is presented next.

Algorithm 5.1 Algorithm for Q-FNNM.

1: **Input** Training data $X=\{x_1, x_2, ..., x_n\}$
2: **Output** Q-FNNM architecture.
3: **Begin**
4: **Initialize** the nodes in the input layer equal to the dimensions (K) of data sample X_i where $i = 1, .., n$ and $x_i \in R^K$.

Algorithm 5.1 (Continued)

Initialize neurons in the hidden layer equals to the number of classes of the dataset.
(F^{g}_{Gbest})=0
$(W^{quant}_{global})^{h}$ =0, $(W^{real}_{global})^{h}$=0
$X^{unlearned}$=ϕ
Initialize the weights $(W^{quant}_{1})^{h}$ of hidden layer neurons in terms of quantum bits.
Initialize the (m^{g}) in terms of a quantum bit.
While g=1:y
  Repeat
    Apply the observation process to get a real coded value of connection weight $(W^{real}_{g})^{h}$ and fuzzifier parameter (m^{g}).
    Executing fuzzy clustering using the parameters (M^{g}_{real}) and $(W^{real}_{g})^{h}$ as cluster centroid.
    Compute the degree of overlap or membership of each sample using Eq. (5.9) and Eq. (5.10).
    Compute the NCLS with the step activation function discussed in Eq. (5.11).
    Fitness Evaluation Compute local and global fitness function using Eq. (5.12) and Eq. (5.13) to find the NCLS in generation (g).
    Store the best connection weight $(W^{quant}_{global})^{h}$, $(W^{real}_{global})^{h}$ and fuzzifier parameter m^{quant}_{global}, M^{real}_{global} in each generation g.
    if $((F^{g}_{Gbest}==n)$ or $(g==\text{y}))$ **then**
      break
    else
      if $(F^{g}_{Lbest} \geq F^{g}_{Gbest})$ **then**
        $(W^{quant}_{global})^{h}$=$(W^{quant}_{g})^{h}$
        $(W^{real}_{global})^{h}$=$(W^{real}_{g})^{h}$
        m^{quant}_{global}=m^{g}
        M^{real}_{global}=M^{g}_{real}
        F^{g}_{Gbest}=F^{g}_{Lbest}
      else
        $(W^{quant}_{global})^{h}$=$(W^{quant}_{global})^{h}$
        $(W^{real}_{global})^{h}$=$(W^{real}_{global})^{h}$
        m^{quant}_{global}=m^{quant}_{global}
        M^{real}_{global}=M^{real}_{global}
      end if
      g=g+1
    Update the quantum bit of connection weights and fuzzifier parameter using quantum update process.
    end if
  until$((g > \text{y})$ or $(F^{g}_{Gbest} == n))$
  Separate the learned samples from set X and store the unclassified samples in set $X^{unlearned}$.
  $X = X^{unlearned}$

Algorithm 5.1 (Continued)

40: **Again** add new neurons in the hidden layer according to unlearned samples classes for learning of remaining samples present in set X.
41: **end while**
42: Apply the same process from step-11 to step-41 for unlearned samples.
43: **return**
44: **End**

The number of neurons in the hidden layer shows the possible number of clusters of the dataset. The final cluster centroids show the connection weights $(W^{real}_{global})^h$ between the hidden layer and the input layer. After completion of hidden layer learning, we start the learning of the output layer which is discussed next.

5.2.3 Learning of Output Layer

In the output layer, we keep the number of neurons equals to the number of classes of the dataset. The connection weights between the hidden layer and the output layer are initialized as unity. Here, each neuron in the output layer gives the output as 1 or 0. The output of the output layer neuron will depends upon the output from the hidden layer neurons. The correctly learned output from the hidden layer neurons is considered as (O^i_h). This gives output as the class number of correctly classified samples from the hidden layer. For example, let the total six hidden layer neurons (two for each class) are formed, for the three class dataset. Then at the time of testing only one neuron will give output as class number and other will give output 0. The output of hidden layer neurons (O^i_h) is multiplied with connection weights W_{output} of the output layer.

$$net_{output} = \sum (O^i_h) * W_{output} \tag{5.14}$$

The output layer neuron which is taken equal to the number of classes of the dataset will receive net_{output} as input. If it is equal to the class number for which the output neuron is formed, then it gives output 1 else 0. For example, if class-1 or A dataset has been given as input and it is correctly classified, then the value of (O^i_h) and net_{output} will be 1. In this case, the neuron of class 1 at output layer gives output as 1 else 0. Thus, the output layer gives output 100 which shows that the sample belongs to class-1 or A. Similarly, if output layer

gives the output as 010, and 001 it means that the sample belongs to class 2 or B, and class 3 or C, respectively.

5.3 Experimental Evaluation

In the experiments, we evaluate the performance of proposed Q-FNNM framework on various benchmark datasets are IRIS, WINE, Glass, and Dermatology using several measures like accuracy, sensitivity, specificity, precision, and F-measure. In addition to this, the performance of proposed Q-FNNM is compared with the other state-of-the-art approaches.

5.3.1 Experimental Setup

The proposed Q-FNNM classifier is implemented in Matlab and on an Intel core, I-5 processor with 4 GB RAM on the Windows-7 operating system. To evaluate the performance of the proposed approach on different validation methods, it is very common to partition the dataset into two separate sets: a training set and a testing set. These training and testing sets are divided according to the three different validation methods. We divide the dataset into 60-40 and 70-30 training-testing ratios where 60% and 70% part of the dataset has been taken for the training purpose, whereas 40%, and 30% part of the dataset has been considered for testing purpose. Here, a 10-fold cross validation technique is also used to evaluate the performance of the proposed classifier. While training our proposed Q-FNNM approach, we use 90% of data for the training of the algorithm and the rest 10% data has been taken for the testing purpose.

5.3.2 Parameters Specification

To evolve the connection weights $(W_g^{real})^h$ and fuzzifier parameter M_{real}^g, we fix the maximum number of generations $g_{\max}$ to 100 in the proposed Q-FNNM algorithm. The connection weights are initialized in the range of the minimum and maximum value of samples of the dataset. The value of fuzzifier parameter has been selected in the range of 1.5 to 2.5 according to the study suggested by Pal and Bezdek [79]. We have fixed the value of termination criteria $T = 0.001$ for all the datasets which are proven to be work well for most of the datasets [121].

The value of $\Delta\theta$ is taken as $0.03 \times \pi$.

5.3.3 Performance on Benchmark Datasets

We evaluated the performance of proposed Q-FNNM classifier using the four benchmark datasets. Table 5.2 shows the performance of the Q-FNNM on the various parameters using the three different dataset partitions. The best mean accuracy of IRIS dataset is achieved as 98.36% for data partitions 70-30%. For the WINE and Dermatology datasets, the best mean accuracy is achieved in 10-fold cross validation scheme as 98.35%, and 99.504%, respectively. The Glass dataset achieves the best mean accuracy as 89.814% for the dataset partitions 70-30%. The best value of sensitivity and specificity for the datasets IRIS, WINE, Glass, and Dermatology are 97.631% 98.88%, 97.253% 98.621%, 64.964% 92.702%, and 98.388% 99.707%, respectively. Similarly, for the datasets IRIS, WINE, Glass, and Dermatology the best value of Precision, and F-measure are 98.111% 98.71%, 97.542% 96.7%, 67.912% 66.3%, and 98.609% 98.4%, respectively. From the results, it can be observed that the proposed algorithm Q-FNNM perform well for the above-mentioned datasets.

The proposed algorithm is compared with the state-of-the-art approaches and reported in Table 5.3. It can be observed that the proposed algorithm performs well for most of the datasets. The mean classification accuracy of the proposed algorithm on datasets like WINE, Glass, and Dermatology are far better than other state-of-the-art methods.

Table 5.2: Results of Q-FNNM on Various Parameters.

		Classification Accuracy of Q-FNNM.							
Performance Measures	Training-testing partition	Dataset							
		IRIS		WINE		Glass		Dermatology	
		Mean	Std dev.	Mean	Std dev.	Mean	Std dev.	Mean	Std dev.
Classification accuracy	60-40	97.14	1.6334	97.716	1.789	88.362	0.351	99.027	0.313
	70-30	98.36	1.4111	97.979	1.467	89.814	1.125	99.235	0.396
	10-fold cross validation	98.12	1.212	98.35	1.816	88.72	1.034	99.504	0.525
Sensitivity	60-40	95.662	1.8421	97.253	1.188	58.174	6.198	96.552	1.019
	70-30	97.631	2.147	97.014	2.339	64.964	5.001	97.528	1.254
	10-fold cross validation	96.359	1.6289	94.731	2.409	63.231	6.033	98.388	1.75
Specificity	60-40	97.809	1.029	98.44	0.728	91.735	0.458	99.435	0.181
	70-30	98.68	1.1248	98.621	0.9864	92.702	0.619	99.549	0.227
	10-fold cross validation	98.88	1.205	98.047	1.871	92.023	0.632	99.707	0.31
Precision	60-40	95.427	1.9469	95.917	2.116	63.346	7.418	96.587	1.126
	70-30	97.988	1.7281	96.547	2.462	67.912	8.413	97.523	1.485
	10-fold cross validation	98.111	2.151	97.542	2.148	66.562	7.3423	98.609	1.512
F-measure	60-40	95.56	1.932	96.5	1.6	60.0	1.7	96.5	1.0
	70-30	97.8	1.94	96.7	2.3	66.3	6.5	97.5	1.3
	10-fold cross validation	98.71	1.72	96.01	2.09	65.1	5.3	98.4	1.5

Table 5.3: Comparison of Q-FNNM with State-of-the-Art Approaches in Terms of Classification Accuracy.

Approaches	Dataset			
	IRIS	WINE	Glass	Dermatology
Q-FNNM	98.12	98.35	88.72	99.504
Backpropagation	95.3	-	61	93.04
Binary coded GA	96	-	61.8	90
Real coded GA	97	-	64.5	92.5
Two Stage GP	96	85	64	-
Hybrid Decision tree classifier	98.66	90.17	76.27	-
Hybrid Naive Bayes classifier	98	86.41	52.33	-
SVM	97.27	-	72.24	96.04
GONN	99.2	94.95	86.25	97.98

5.3.4 Discussion

It can be observed from Table 5.3, that the proposed Q-FNNM perform well in comparison to other state-of-the-art approaches. The improvement in the results is due to the proper selection of connection weights. Here, the connection weights are taken as cluster centroids, which is generally selected randomly in the fuzzy clustering algorithm. The selection of cluster centroids in this way may lead the solution in local maxima. It is also important to select an optimal value of fuzzifier parameter for better performance of the fuzzy clustering. The selection of both the parameters using the quantum computing concept gives the optimal value from a large search space provided by this evolutionary method. The quantum gate provides exploitation, due to which the optimal values can be found in few generations only.

5.3.5 Summary

In this chapter, we have presented a Q-FNNM algorithm to solve multi-class classification problem. In the Q-FNNM algorithm, we have used the fuzzy clustering concept to form the neural network architecture, whereas the parameter of

fuzzy clustering, i.e., cluster centroids V and fuzzifier parameter m is optimized using the evolutionary quantum computing concept. In the proposed algorithm first, we take the number of hidden layer neurons equals to the number of classes of the dataset. Further, the number of hidden layer neurons are taken on the basis of classes of the unlearned samples. The connection weights are considered as cluster centroids taken in the fuzzy clustering, which is evolved using the evolutionary quantum computing concept. The performance of the proposed algorithm is tested on four multi-class datasets, e.g., IRIS, WINE, Glass, and Dermatology. The performance of the dataset is evaluated on the parameters that are accuracy, sensitivity, specificity, precision, and F-measure, respectively. The proposed algorithm is also compared with other state-of-the-art approaches, and it is found that the proposed algorithm outperforms in comparison to the other state-of-the-art approaches.

Chapter 6

Quantum inspired Stacked Auto-encoder based Deep Neural Network Algorithm

6.1 Introduction

Neural network learning techniques have been widely applied in a variety of areas such as pattern recognition, natural language processing, and computational learning. Nowadays, the real-life datasets (e.g., image, signal, Web data) are very complex, the classification of such dataset is very difficult with the conventional neural network. The conventional neural network does not perform well for complex datasets even on increasing number of hidden layers [7,90]. For complex datasets such as image, signals, speech, video, Web data, the learning of finer details can be done in more appropriate manner using the deep learning concept [7]. Several researchers have proposed deep neural network with few numbers of hidden layers [26–28]. Breakthroughs in deep learning have been achieved since 2006 when Hinton proposed a novel deep structured learning architecture [92]. Thus, only with few numbers of hidden layers, the complex data can be classified efficiently. For classification of the complex dataset, dimensions of the dataset is an important issue. To deal with such dataset stacked auto-encoder based deep neural network is proposed. It can efficiently deal with the dataset which requires dimensional reduction for better classification accuracy [7,40,93,94]. However, deep neural network based on stacked auto-encoder

perform well on complex datasets, but proper selection of its learning parameters is required to improve the performance. The deep neural structure designed with the stacked auto-encoder uses gradient descent as the learning algorithm, which need a proper selection of its learning rate parameter η to achieve good performance [28, 40, 95, 96, 122]. In General the value of learning rate parameter η is selected randomly between 0 and 1. Selection of the learning rate parameter in this way may cause the problem. If the appropriate selection of this parameter is not done, then the problem of over-fitting or under-fitting with no convergence may arise.

Therefore, to deal with this problem, this chapter presents a deep neural network learning algorithm using stacked auto-encoder is further enhanced by applying the quantum computing concept for evolving the learning rate parameter and it is named as Quantum inspired Stacked Auto-encoder based Deep Neural Network Algorithm (Q-DNN). The Q-DNN formed using stacked auto-encoder, in which gradient descent learning algorithm is being applied along with the quantum computing concept to optimize its learning process. The deep neural network formed using the stacked auto-encoder require three steps. In the first step, unsupervised (pre-training) of each auto-encoder is done. The pre-training process ensures lower training error as compared to the network trained without pre-training process. Now, after training of each auto-encoder, they are stacked on top of each other, and the output layer is added to the last hidden layer of the stacked auto-encoder. Once, this network is formed, the supervised learning (fine-tuning) of the whole network using the Backpropagation algorithm is done to optimize learning parameters [28, 98, 99]. The proposed algorithm is discussed in detail in subsequent sections.

6.2 Proposed Algorithm

Here, Q-DNN learning algorithm is proposed. The proposed algorithm forms deep neural network architecture using the stacked auto-encoders. The training of stacked auto-encoders is done using the gradient descent/Backpropagation learning algorithm which is optimized using the quantum computing concept. The evolutionary quantum computing concept is used to evolve the learning rate parameter which is represented as η_t^{quant}. The proposed Q-DNN is used to

solve the two class problems as well as the multi-class problem. First of all, few notations are discussed then the learning of proposed Q-DNN is discussed which involves initialization of connection weights and learning rate parameter η_t^{quant}.

6.2.1 Preliminaries

As first step is to do pre-training process of each stacked auto-encoder. Therefore, during the pre-training process, the input dataset in the first layer is represented as X=(X^1, X^2, X^3,....., X^l), where each X^i has n attributes represented as X^i= (x_1^i, x_2^i,....., x_n^i). The basic architecture of pre-training of Q-DNN is shown in Figure 6.1. However, the stacked auto-encoder consists of auto-encoders stacked on top of each other where each encoder is trained separately. During the fine-tuning process, the input layer is only one, therefore, the input dataset is X=(X^1, X^2, X^3,....., X^l), where each X^i has n attributes represented as X^i= (x_1^i, x_2^i, ..., x_n^i). The basic architecture of fine-tuning of Q-DNN is shown in Figure 6.2.

6.2.2 Q-DNN Learning

The proposed algorithm involves the number of steps in forming the quantum inspired stacked auto-encoder based deep neural network like, initialization of

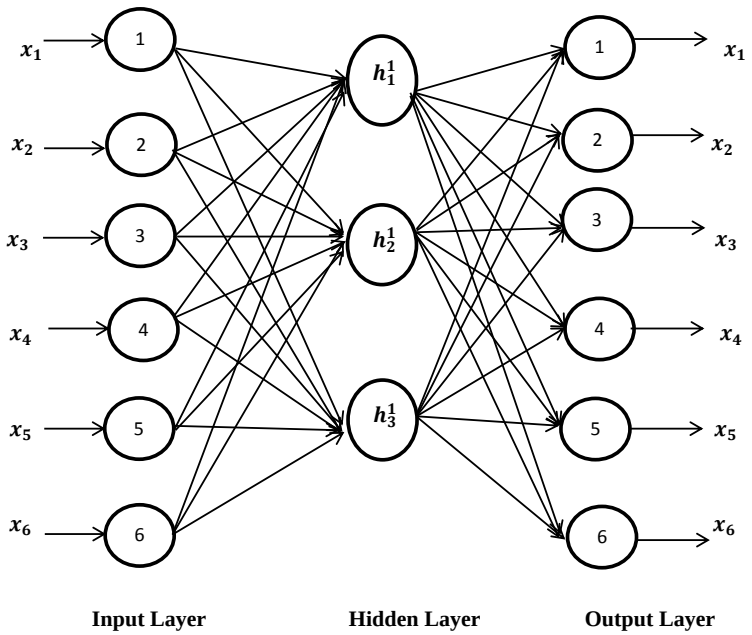

Figure 6.1: Pre-training of Q-DNN.

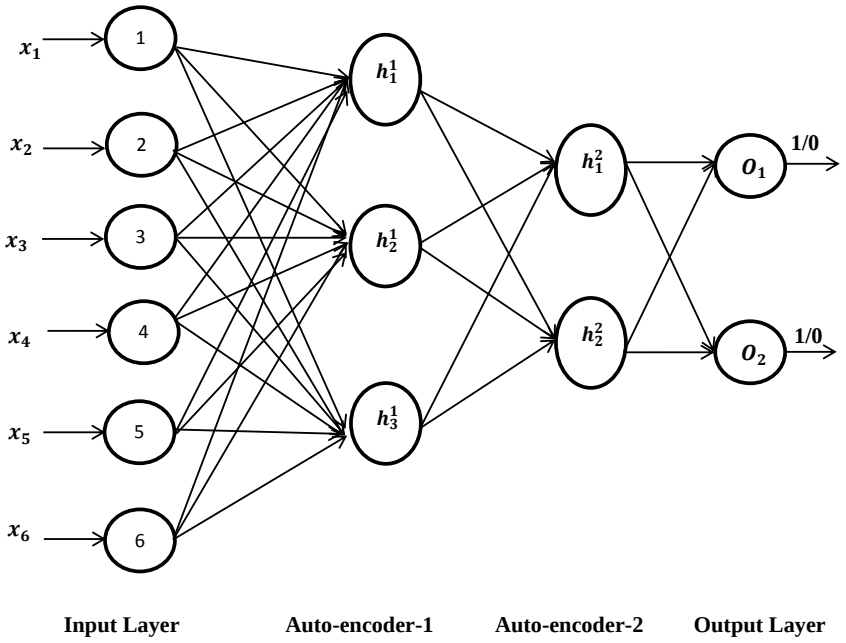

Figure 6.2: Fine-tuning of Q-DNN.

parameters, pre-training process, and fine-tuning process which are discussed next.

Initialization of Parameters

Before start training, we initialize connection weights, and learning rate parameter η_t^{quant} for the pre-training of each auto-encoder using the following formulation.

$$W_{tj} = rand(); \tag{6.1}$$

The W_{tj} is the connection weights between the input layer and hidden layer. The connection weights $(W_{jt})'$ between the hidden layer and output layer can be represented as follows [97]:

$$(W_{jt})' = (W_{tj})^{Transpose} \tag{6.2}$$

Once the connection weights (W_{tj}) of hidden layer and $(W_{jt})'$ of output layer is decided, the learning rate parameter η_t^{quant} can be represented in terms of the qubit as follows:

$$\eta_t^{quant} = (q_t^1 | q_t^2) \tag{6.3}$$

To evolve the value of the learning rate parameter η_t^{quant}, here two qubits have been used. As the real coded value of the learning rate parameter η_t^{real} must be between (0 1). Thus, using two qubits the search space between 0 and 1 is divided between four subspaces to find the real coded value of the learning rate parameter η_t^{real} using the observation process which is discussed in Section 2.2. The real coded value of the learning rate parameter η_t^{real} may or may not be optimal in the first generation. Therefore, we need to update qubit of learning rate parameter η_t^{quant}. The qubit of η_t^{quant} is updated using the error value of each generation E_t, global error value E_{global} and corresponding binary values s_t and s_t^{global} of their qubit. The qubits of η_t^{real} are updated using the quantum update process discussed in Section 2.2 and with the help of Table 6.1.

Along with initialization of connection weights and learning rate parameter in terms of qubits η_t^{quant}, some more parameters are initialized. The parameter like the error is initialized as $E_{global}= \infty$ of pre-training, binary bits value matrix of the quantum bit corresponding to the value of error $s_t^{global} = 0$, the connection weight matrix $W_{global}^h[\,]= [0]$, and learning rate parameter $\eta_{global}^{real} = 0$ are also initialized. Here global word in parameters represent the best value of these parameters in all generations. Using all above parameters, the pre-training algorithm is discussed next:

Pre-training Algorithm for Auto-encoder

As discussed earlier a deep neural network architecture using the stacked auto-encoder is made of auto-encoders stacked on top of each other. Therefore, training of each auto-encoder is required which is called pre-training. Here,

Table 6.1: Qubits Update for Q-DNN.

s_t	s_t^{global}	$(E_t > E_{global})$ or $(E_t^{fine-tune} > E_{global}^{fine-tune})$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	$0.03 * \Pi$
1	*0*	*true*	*0*
1	*1*	*false*	*0*
1	*1*	*true*	*0*

the following algorithm shows the pre-training process of an auto-encoder with evolutionary quantum computing concept. The overall process of pre-training of Q-DNN in the form of Algorithm 6.1 is presented next.

Algorithm 6.1 Algorithm for Pre-training.

1: Take Input sample as X=(X^1, X^2, X^3,....., X^l) where X^i= (x_1^i, x_2^i,....., x_n^i).
2: Select number of nodes in input layer equals to number of features (n) of the input dataset.
3: Select number of neurons in an auto-encoder (hidden layer) equals to half of its number of input nodes.
4: Select number of output layer neurons equals to numbers of nodes in input layer.
5: Initialization of parameters.
6: $E_{global}= \infty$
7: $S_t^{global} = 0$;
8: $W_{global}^h[\,]= [0]$
9: h is index for hidden layer.
10: $\eta_{global}^{real} = 0$
11: Initialize the connection weights, learning rate parameters η_t^{quant} using qubits.
12: $W_{tj}[\,]=$ rand();
13: $(W_{jt}[\,])' = (W_{tj}[\,])^{Transpose}$
14: **for** t= 1 to z
15: Apply observation process as discussed in Section 2.2, to get real coded value of η_t^{quant}.
16: Execute the gradient descent algorithm to evaluate connection weights and fitness function (E_t).
17: **If** ($E_{global} \leq E_t$)
18: $W_{global}^h[\,]= W_t[\,]$
19: $\eta_{global}^{real} = \eta_t^{real}$
20: $E_{global} = E_t$
21: Update qubit for η_{t+1}^{quant} using qubit update process by selecting proper value of $\Delta\theta$ using Table 6.1
22: **else**
23: update qubit for η_{t+1}^{quant} using qubit update process by selecting proper value of $\Delta\theta$ and Table 6.1
24: **endif**
25: **endfor**
26: Repeat the Step-11 to Step-25 until no change is found in E_{global} in consecutive iterations or till user define generations (z).
27: return(E_{global}, $W_{global}[\,]$, η_{global}^{real})

The above algorithm shows the pre-training process of an auto-encoder, the same process is followed for each auto-encoder. Using this, the connection weights for the first hidden layer of deep neural network architecture is obtained. Now, we apply the same algorithm for the second auto-encoder by

assuming the hidden layer of the previous auto-encoder as the input layer of the new auto-encoder. The number of hidden layer neurons in auto-encoder is selected equal to the half of the input layer nodes. The number of output layer neurons are equal to the nodes in the input layer. In the proposed Q-DNN we have trained three auto-encoders. Thus, we have formed three deep neural networks having a single layer, two layers, and three layers.

Once, the pre-training process of each auto-encoder is done, we apply the fine-tuning process for the final training of Q-DNN. The hidden layer and connection weights evolved during pre-training of each auto-encoder are fixed in the bottom-up fashion. The top hidden layer is connected to the output layer in which the number of neurons will be equal to the number of classes of input dataset [100]. Initialization of random weights for output layer is also done. Once, all these layers and connection weights are fixed, the fine-tuning process starts. The fine-tuning process starts with the initialization of some parameters and which is discussed next:

Fine-tuning Process

The fine-tuning process starts with initialization of some parameters as error $(E_{global})^{Fine-tune}= \infty$ of fine-tuning process, the binary bits matrix corresponding to quantum bit of error value $(S_t^{global})^{Fine-tune} = 0$, the connection weights $(W^h_{global}[\,])^{Fine-tune}= [0]$ of fine-tuning process, learning parameter $(\eta_{global})^{Fine-tune} = 0$ of fine-tuning. The overall process of fine-tuning of Q-DNN in the form of Algorithm 6.2 is presented next.

Algorithm 6.2 Algorithm for Fine-tuning.

1: Take Input sample as X=(X^1, X^2, X^3,....., X^l) where X^i= (x_1^i, x_2^i,....., x_n^i).
2: Connect last hidden layer to the output layer.
3: Initialization of parameters.
4: $(E_{global})^{Fine-tune}= \infty$
5: $(S_t^{global})^{Fine-tune} = 0$;
6: $(W^h_{global}[\,])^{Fine-tune} = [0]$
7: $(\eta_{global})^{Fine-tune} = 0$
8: Initialize connection weights $(W^h[\,])^{Fine-tune}= (W^h_{global}[\,])$ (final weights for particular hidden layer achieved during pre-training process of auto-encoders along output layer weights which is initialized randomly.
9: **for** t= 1 to z
10: Apply observation process discussed in Section 2.2, to get real coded value of $(\eta_t^{quant})^{Fine-tune}$.

Algorithm 6.2 (Continued)

11: Execute the Backpropagation algorithm to evaluate connection weights and fitness function $(E_t)^{Fine-tune}$.
12: **If**$((E_{global})^{Fine-tune} \leq (E_t)^{Fine-tune})$
13: $(W^h_{global}[\,])^{Fine-tune} = (W_t[\,])^{Fine-tune}$
14: $(W_t[\,])^{Fine-tune}$ is weight matrix achieved after completion of Back-propagation.
15: $(\eta^{Fine-tune}_{global})^{real} = (\eta_t)^{Fine-tune})^{real}$
16: $(E_{global})^{Fine-tune} = (E_t)^{Fine-tune}$
17: Update qubit for $(\eta_{t+1})^{Fine-tune}$ using qubit update process as discussed in Section 2.2 and Table 6.1.
18: **else**
19: Update qubit for $(\eta_{t+1})^{Fine-tune}$ using qubit update process as discussed in Section 2.2 and Table 6.1.
20: **endif**
21: **endfor**
22: Repeat the step-10 to step-21 until no change found in $(E_{global})^{Fine-tune}$ value or till user defined generations.
23: return $((E_{global})^{Fine-tune}$, $(W^h_{global}[\,])^{Fine-tune}$, and $(\eta^{Fine-tune}_{global})^{real})$.

In this way, the fine-tuning process of the stacked auto-encoder based deep neural network is done. After completion of the fine-tuning process, the weights and learning rate parameter are evolved as $(W^h_{global}[\,])^{Fine-tune}$ and $(\eta^{Fine-tune}_{global})^{real}$, respectively. Thus, using these parameters, the Q-DNN architecture is formed. Now, the performance of proposed Q-DNN is evaluated on benchmark datasets which are discussed in detailed next.

6.3 Experiment Evaluation

6.3.1 Experiment Setup

The proposed Q-DNN is implemented in Matlab and on an Intel core, I-5 processor with 8 GB RAM on the Windows-7 operating system. The performance of proposed Q-DNN is evaluated on, the BUPA Liver Disorder dataset, Ionosphere dataset, and the PIMA Indians Diabetes dataset [123]. Along with these datasets, the MNIST dataset is also used to evaluate the performance of proposed Q-DNN approach.

To evaluate the performance of proposed Q-DNN, we have divided BUPA Liver Disorder dataset, Ionosphere dataset, and PIMA Indians Diabetes dataset, into 70-30 training-testing ratio. The training set of MNIST dataset is composed

of 6000 out of 60000 samples (600 samples from each class randomly). Similarly, testing data is composed of 1000 samples out of 10000 samples (100 sample from each class randomly).

6.3.2 Results

The performance of the proposed Q-DNN has been evaluated for three different neural network architectures. Firstly, with a single hidden layer architecture and then with two hidden layers architecture and then with three hidden layers architecture. The performance of the proposed Q-DNN in terms of accuracy with average and maximum value is presented in Table 6.2. It can be observed from the results that mean best results for the BUPA Liver Disorder dataset is achieved as 95.66% with neural network architecture having two numbers of hidden layers. In addition to this, the mean worst result for BUPA Liver Disorder dataset obtained as 91.58 % with neural network architecture having three numbers of hidden layers. Similarly, for the Ionosphere dataset, and PIMA Indians Diabetes dataset the mean best results obtained are 97.53 % and 92.15 %, respectively with neural network architecture having two numbers of hidden layers. The mean worst results achieved for Ionosphere dataset and PIMA Indians Diabetes dataset are 91.22 % and 86.47 %, respectively with neural network architecture having three numbers of hidden layers. The results for the BUPA Liver Disorder dataset, Ionosphere dataset, and PIMA Indians Diabetes dataset in terms of sensitivity and specificity are shown in Table 6.3. The results for the BUPA Liver Disorder dataset in terms of the maximum value of sensitivity and specificity are 95.16 % and 97.56 % with two numbers of hidden layers.

Table 6.2: Classification Accuracy of Q-DNN.

		BUPA Liver Disorder	Ionosphere dataset	PIMA Indians Diabetes
One Layer	Max	95.14	93.26	90.4
	Mean	94.25	92.58	88.99
Two Layers	Max	96.11	98.07	93.4
	Mean	95.66	97.53	92.15
Three Layers	Max	92.33	92.3	87.8
	Mean	91.58	91.22	86.47

The results in terms of the minimum value of sensitivity and specificity with neural network architecture having three number of hidden layers are 90.98 % and 92.68 %. The maximum value of sensitivity and specificity for Ionosphere dataset and PIMA Indians Diabetes dataset are 98 %, 98 % and 95 %, 92.3 %, respectively with neural network architecture having two numbers of hidden layers. The minimum value for the same dataset achieved as 90.3 %, 94.23 % and 92 %, 84.6 %, respectively with neural network architecture having three numbers of hidden layers. It shows that the performance of the proposed Q-DNN may decreases when number of hidden layers for small datasets increases.

In the same way, the proposed Q-DNN is tested on MNIST dataset. We have already discussed that only 6000 samples have been taken for the training of Q-DNN and 1000 samples have been taken for the testing purpose. The results for MNIST dataset is presented in terms of multi-class parameters which are presented in Table 6.4. It can be observed from the results of the MNIST dataset that, the best value of average accuracy is 99.04 % with three numbers of hidden layers and worst value of average accuracy is 96.8 % with one hidden layer. The other parameters like the error, PPV, NPV, sensitivity, specificity, and $F - score$ [107] are also reported for the MNIST dataset in Table 6.4. It can be easily observed from the results that the proposed Q-DNN perform better on MNIST dataset if the number of hidden layers increases. It means the performance of the proposed Q-DNN increases by increasing the number of hidden layers for large datasets.

6.3.3 Comparison with State-of-the-Art Approaches

The performance of the proposed Q-DNN in terms of average accuracy is compared with some existing approaches. Table 6.5, shows a comparison of the proposed Q-DNN in terms of average accuracy with various algorithms as Naive Bayes, SVM, Deep belief network, SVM with CPON, Q-BNN, AQ-BNN, Backpropagation algorithm, and stacked auto-encoder using Backpropagation algorithm [27, 124].

Table 6.3: Sensitivity and Specificity of Q-DNN.

No of Hidden Layer	Parameters		BUPA Liver Disorder	Ionosphere dataset	PIMA Indians Diabetes
One Layer	Sensitivity	Max	95.16	92.3	90
		Mean	94.255	91.55	89.11
	Specificity	Max	95.12	94.2	90.7
		Mean	94.652	92.89	88.862
Two Layers	Sensitivity	Max	95.16	98	95
		Mean	93.99	97.154	94.51
	Specificity	Max	97.56	98	92.3
		Mean	96.48	97.256	91.26
Three Layers	Sensitivity	Max	91.9	90.3	92
		Mean	90.98	89.46	91.456
	Specificity	Max	92.68	94.23	84.6
		Mean	91.226	93.295	83.24

Table 6.4: MNIST Dataset Results of Q-DNN.

	One Layer	Two Layers	Three Layers
Accuracy	96.8	97.6	99.04
Error Rate	0.0318	0.234	0.0096
PPV_μ	84.1	88.3	95.2
NPV_μ	98.23	98.7	99.46
$Sensitivity_\mu$	84.1	88.3	95.2
$Specificity_\mu$	98.23	98.7	99.46
$F-score_\mu$	84.1	88.3	95.2
PPV_M	84.53	88.74	95.31
NPV_M	98.23	98.7	99.46
$Sensitivity_M$	84.1	88.3	95.2
$Specificity_M$	98.23	98.7	99.46
$F-score_M$	84.13	88.52	95.25

There are three different Q-DNN architectures are formed by varying the number of hidden layers as one, two, and three. Therefore, the comparison is made between all Q-DNN architecture with stacked auto-encoder using back-propagation with one, two, and three hidden layers as shown in Table 6.5. The proposed Q-DNN with two hidden layers gives the best results in terms of average accuracy for datasets like BUPA Liver Disorder, and PIMA Indians Diabetes in comparison to other methods. In the case of Ionosphere dataset, the best result in terms of the average accuracy is achieved by Deep belief networks while the result of Q-DNN is close to the best result. On the other hand, for MNIST dataset, the proposed Q-DNN with three hidden layers gives best results in terms of average accuracy in comparison with other methods. It is also observed that the Backpropagation algorithm with the random value of the learning parameter gets trapped in the problem of local minima which leads to unsatisfactory results. It is also observed from the results that the proposed Q-DNN perform better for all datasets in comparison to stacked auto-encoder for all three architectures having one, two and three hidden layers.

Table 6.5: Comparison of Proposed Algorithm with other Classifiers in Terms of Classification Accuracy.

Algorithm	PIMA Indians Diabetes	BUPA Liver Disorder	Ionosphere dataset	MNIST dataset
Naive Bayes	76.62	57.14	77.78	-
Single-layer SVM	74.03	69.6	88.89	95.3
Deep belief networks	70.7	58.77	**98.45**	96.54
SVM with CPON	83.11	77.14	97.22	98.84
Q-BNN	86.2	90.83	93.15	-
AQ-BNN	88.89	95.08	93.81	-
Backpropagation algorithm	88.61	83.54	92.2	93.2
Stacked Auto-encoder using Backpropagation	85.34[a]	92.13[a]	87.15[a]	91.32[a]
	87.13[b]	93.22[b]	89.74[b]	92.58[b]
	86.51[c]	91.72[c]	87.22[c]	94.33[c]
Proposed Algorithm (Q-DNN)	88.99[a]	94.25[a]	92.58[a]	96.8[a]
	92.15[b]	**95.66[b]**	97.53[b]	97.6[b]
	86.47[c]	91.58[c]	91.22[c]	**99.04[c]**

[a] Results of Stacked Auto-encoder using Backpropagation and Q-DNN having single layer.
[b] Results of Stacked Auto-encoder using Backpropagation and Q-DNN having two hidden layers.
[c] Results of Stacked Auto-encoder using Backpropagation and Q-DNN having three hidden layers.

The results show that the proposed Q-DNN performs far better in terms of accuracy with respect to other state-of-the-art approaches. The proposed algorithm forms, three deep neural network architectures with a single layer, two layers, and three layers. Here, deep neural network with two hidden layers gives good results for small size dataset while the deep neural network with three hidden layers gives good results for MNIST dataset. The small dataset faces the problem of overfitting as the number of hidden layer increases in the deep neural network [125, 126]. Therefore the performance decreases on increasing more number of hidden layers for classification of the small dataset. Here, the comparison is also made on classification accuracy of proposed Q-DNN with other deep learning algorithms [27, 127–131] on the MNIST dataset as shown in Table 6.6. It can be observed from Table 6.6, that there is a difference between classification accuracy of the stacked auto-encoder using Backpropagation tested in our approach and stacked denoising auto-encoder tested by [127]. The difference between the accuracy of both stacked auto-encoders is due to the different value of the learning rate parameter (η). Evolving the (η) using the quantum computing concept helps to get optimal results. Thus, the proposed Q-DNN achieves better classification accuracy as compared to the classification accuracy achieved by both the stacked auto-encoders with few numbers of hid-

Table 6.6: Comparison of Q-DNN with Deep Learning Algorithm on MNIST Dataset.

Method	Classification Accuracy (%)
SVM with CPON (4 layers)	98.84
Stacked Denoising Auto-encoder (3 layers)	98.72
Deep Boltzmann Machines	99.05
k-Sparse Auto-encoder	98.65
Shallow Denoising/Dropout Auto-encoder	98.4
Stacked FC-WTA Auto-encoder	98.89
Restricted Boltzmann Machines	98.4
Winner-Take-All Restricted Boltzmann Machines	98.02
Transformation	97.56
Stacked Auto-encoder using Backpropagation (3 layers)	94.33
Q-DNN (3 layers)	99.04

den layers. The layer-wise unsupervised learning (per-training) and supervised learning (fine-tuning) help to get optimal results with few numbers of hidden layers [99]. It can also be observed from Table 6.6, that the classification accuracy of proposed Q-DNN and Deep Boltzmann Machines [128] are very similar. However, the proposed Q-DNN achieves this classification accuracy with few numbers of hidden layer neurons as compared to Deep Boltzmann Machines.

6.4 Summary

In this chapter, we have enhanced deep neural network using stacked auto-encoder by applying the quantum computing concept for evolving the learning rate parameter which is named as Q-DNN. The proposed Q-DNN use to solve the two class problem as well as the multi-class problem. In the proposed algorithm first, we learn each auto-encoder using the pre-training process. Then after learning of each auto-encoder, we stacked it, top of each other and connect with output layer. Then we perform the fine-tuning process with optimized learning algorithm using the quantum computing concept. Using the proposed algorithm three deep neural network architectures are formed having a single layer, two layers, and three layers. The Q-DNN forms deep neural network architecture with few numbers of hidden layers using the layer-wise unsupervised learning (pre-training) and supervised learning (fine-tuning). The performance of proposed Q-DNN is evaluated on benchmark two class datasets like PIMA Indians Diabetes, BUPA Liver Disorder, and Ionosphere dataset along with multi-class MNIST dataset. The proposed Q-DNN gives good results for small size datasets with two numbers of hidden layers and for MNIST dataset with three numbers of hidden layers. The performance of proposed Q-DNN is compared with other state-of-the-art approaches, and it is found that proposed Q-DNN outperform in comparison of other state-of-the-art approaches.

Chapter 7

Offline Signature Verification System using Enhanced Quantum inspired Neural Network

7.1 Introduction

In this chapter, we proposed an Enhanced Quantum inspired Neural Network Learning Algorithm (EQNN-S) which is used to develop offline signature verification system. This algorithm is enhanced version of proposed NQ-BNN algorithm by introducing a new threshold boundary parameter to get optimal value of the threshold. The proposed EQNN-S forms a neural network architecture constructively by adding neurons in the hidden layer. The proposed EQNN-S is four layer architecture with one input layer, two hidden layers and an output layer. The connection weights and threshold have been evolved using the quantum computing concept. The proposed EQNN-S algorithm is used to develop an offline signature verification system. However, to develop efficient offline signature verification system, extraction of unique features is required. In the proposed EQNN-S we have extracted unique features like the number of loops, dimensions, dense patches, angle, and bounding caps. The performance of proposed EQNN-S is compared with other state-of-the-art approaches, and it is found that proposed algorithm performs better in comparison with other approaches. The proposed EQNN-S algorithm with threshold boundary parameter is discussed next.

7.2 Proposed Approach

The proposed algorithm works in three phases, in first phase pre-processing of the signature is done which is discussed in Section 2.5. In second phase extraction of unique features is done and then in the third phase, the classification of these features is done using the proposed EQNN-S. The basic architecture of the proposed system is shown in Fig. 7.1.

7.2.1 Analysis of Features for Signature Images

Firstly, original and forged signatures of a few people have been collected in the form of paper, then these are scanned to get signatures in the form of an image. Due to variation in the pen, discrepancies, and background noise in the images we first make these images on the same scale [132]. To bring images on the same scale, we pre-process all the images as discussed in Section 2.5.

The pre-processing helps to standardize a given signature image. Once the pre-processing is done, the feature extraction process [106, 133, 134] is carried out. This process involves analysis of the number of loops, dimensions, dense patches, angle, and bounding caps of signature which is discussed next.

Number of Loops

The number of loops in a signature is counted, and this serves as the first feature of the signature as shown in Fig. 7.2.

Dimensions

The exact height and width of the signature are calculated. The image is also cropped according to the exact width and height for further use as shown in Fig. 7.3.

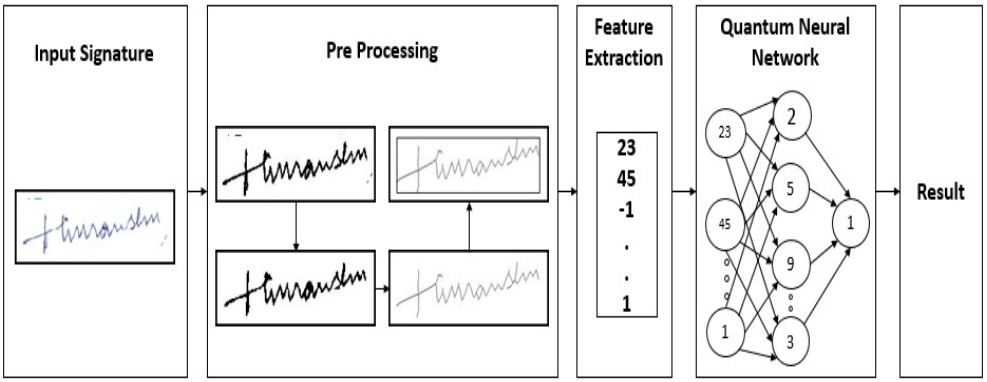

Figure 7.1: Basic Architecture of EQNN-S.

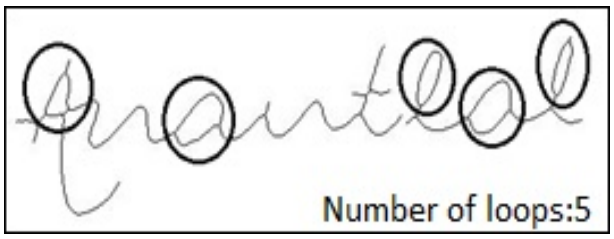

Fig. 7.2: Counting the Number of Loops.

Figure 7.3: Signature's exact Width and Height.

Vertical and Horizontal Dense Strips

This particular computation gives 20 feature values. Horizontal stripes running across the image are considered. Among these, the density of the five most dense strips is recorded. Furthermore, the distances of these strips from the origin are also noted. Similarly, the vertical stripes give the other ten feature values.

Dense Patch

Like the previous one, square patches of a defined size have been taken into consideration. The density of black pixels in these patches is calculated as shown in Fig. 7.4. The density of the most five dense regions serves as feature values along with their x and y coordinates. This process gives us 15 feature values.

Angle

On any given vertical line of pixels, the topmost and bottommost black pixels are considered. The average of their distance from the top is calculated. This gives the approximate middle point of the signature that lies on that vertical line. This computation is repeated for each vertical line. The computed center points

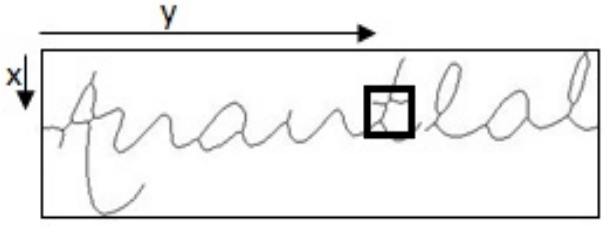

Fig. 7.4: Square Dense Patch of Signature.

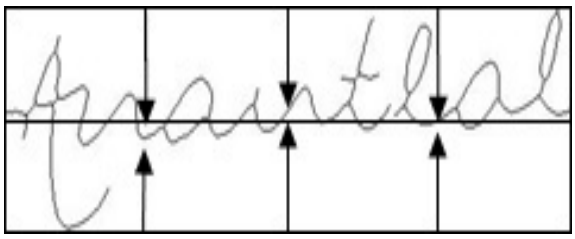

Fig. 7.5: Bounding Caps of Signature.

of the signature on each vertical line are interpolated to give a straight line. The angle that this line makes with the base gives the angle of the signature.

Bounding Caps

The image is divided into four vertical patches of uniform width. The three lines that help to make the strips, the distance of the first black pixel from the top is calculated. These three distances serve as feature values. Similarly, distances are calculated for the first pixel from the bottom of the signature as shown in Fig. 7.5. Here, the search for the first black pixel is done only till half the height of the image. In case, no such pixel is found, the corresponding value is set to a fixed value indicating no pixel has been found.

As presented in Table 7.1, total 45 feature values serve to define a signature image. These values are used as the input for the EQNN-S.

7.2.2 Preliminaries

The signature classification belongs to a multi-class problem, as each person is considered as an individual class. This multi-class problem is solved using the multiple two-class classification problems. The signatures corresponding to e persons represents the e number of classes. For solving a multi-class classification problem as multiple two-class classification problems, let us assume that samples P=($P_1, P_2, P_3, ..., P_e$) denotes the categories of e persons corresponding to their

Table 7.1: Description of Extracted Features.

Feature	Description	Count of features
Number of loops	The number of loops in the given signature image	1
Angle	The angle that the image makes with the base line	1
Dimensions	The exact width and height of the signature	2
Dense patch	Square patch with highest densities	15 (5 density and 5 of each coordinate)
Horizontal strips	The most dense horizontal strips	10 (5 density and 5 y-coordinates)
Vertical strips	The most dense vertical strips	10 (5 density and 5 x-coordinates)
Bounding Caps	The distances from fixed points on the top and bottom edges	6
	Total	45 values

signatures. Let a category of l^{th} person denoted by (P_l), is a collection of variation of signatures of l^{th} person. Let this is denoted by P_l=$(X_l^1, X_l^2, X_l^3, ..., X_l^{c_1})$ having c_1 number of signatures, where X_l^i=$(x_i^l, x_i^2, x_i^3, ..., x_i^n)$ where n is the number of attributes in one signature of input sample. The signature corresponding to the l^{th} person, (P_l) is considered as one class while signatures corresponding to P-P_l represent other class. The basic architecture neural network of E-QNNS is shown in Figure 7.6.

7.2.3 Learning of Hidden Layers

Hidden layer learning of multiple two class classification problems is initiated by considering the individual e number of two class classification problems. It means when learning for the signature corresponding to the first person P_1, its sample $(X_1^1, X_1^2, X_1^3, ..., X_1^{c_1})$, where X_1^i= $(x_i^1, x_i^2, x_i^3, ..., x_i^n)$ are taken as instances for say class A. The remaining samples corresponding to signatures of persons $(P - P_1)$ =$(Y_1^1, Y_1^2, Y_1^3, ..., Y_1^{c_2})$, where Y_1^i= $(y_i^1, y_i^2, y_i^3, ..., y_i^n)$, are taken as instances for the class B. Thus, the number of input layer nodes is equal to n. The total samples belonging to categories of P are divided into two classes A and B. The learning of hidden layer is done with initialization of parameters as follows:

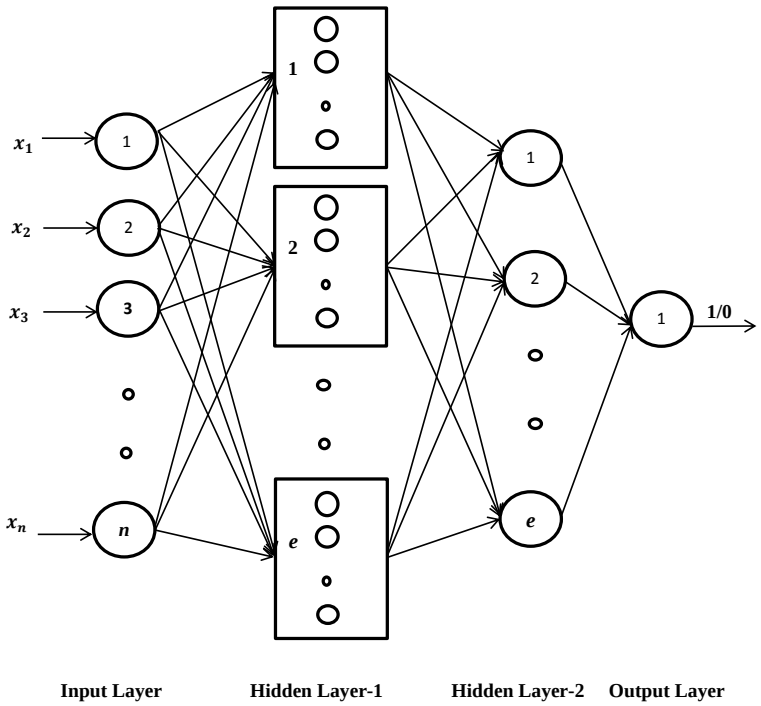

Figure 7.6: Architecture of EQNN-S Neural Network.

Initialization and Representation of Parameters

Firstly, a neuron is selected in the first hidden layer, and its quantum weights $(W_i^{quant})^g$ are initialized in terms of quantum bits. Along with initialization of quantum weights, the parameters sum^*, sum_g, max_net_A, min_net_A, max_net_B, and min_net_B are also initialized. The parameters max_net_A, min_net_A, max_net_B and min_net_B are used as boundary parameters for finding the proper threshold of neurons. Now, the quantum weights $(W_i^{quant})^g$ are converted into a real weights represented as $(W_i^{real})^g$ using the observation process discussed in Section 2.2. Now, the real weights $(W_i^{real})^g$ is applied to input dataset to calculate the boundary parameters for finding the threshold.

Boundary Parameters Calculation

The threshold $(\lambda_i^{real})^t$ of the neuron is evolved using the quantum computing concept and boundary parameters. Here, to select threshold, min_net and max_net parameters are introduced. These parameters are initialized as ∞ and $-\infty$, respectively. Now, the value of these parameters is found from the cartesian product of input sample of both classes and connection weights $(W_i^{real})^g$. These values help to find out actual distribution of projection of input dataset with connection weights. Thus, it gives more diversity to search appropriate value of threshold and avoid local maxima problem.

This parameter helps to find the boundary of the projection of input sample with connection weights and helps to select a threshold within this range. Thus, it helps to find the threshold in the defined range. The following formulations shows the calculation of boundary parameter.

$$(TBP)_g^{max} = max(max\_net_A, max\_net_B); \tag{7.1}$$

$$(TBP)_g^{min} = min(min\_net_A, min\_net_B); \tag{7.2}$$

Thus, the threshold in terms of these boundary parameters is formulated as follows:

$$(\lambda_i^{real})^t = \begin{cases} (TBP)_g^{min} \quad if \quad (\lambda_i^{real})^t < (TBP)_g^{min} \\ (TBP)_g \, if \, (TBP)_g^{min} \leq (\lambda_i^{real})^t \leq (TBP)_g^{max} \\ (TBP)_g^{max} \quad if \quad (\lambda_i^{real})^t > (TBP)_g^{max} \end{cases} \tag{7.3}$$

The TBP_g^{min}, and TBP_g^{max} parameters are calculated corresponding to the weights and input dataset. Now, the quantum threshold $(\lambda_i^{quant})^t$ is initialized in terms of qubits. Along with these parameters, some more parameters count1=0, count2=0, sum^t, and sum_λ^* are also initialized. Using the boundary parameter

Table 7.2: Qubits Update for EQNN-S.

s_{ij}^{g}	s_{ij}^{*}	$sum_g < sum^*$ $sum^t < sum_\lambda^*$	$\Delta\theta$
0	*0*	*false*	*0*
0	*0*	*true*	*0*
0	*1*	*false*	$-0.03 * \Pi$
0	*1*	*true*	*0*
1	*0*	*false*	*0*
1	*0*	*true*	$0.03 * \Pi$
1	*1*	*false*	*0*
1	*1*	*true*	*0*

and observation process, the real coded value of threshold $(\lambda_i^{real})^t$ is achieved. However, this is not final value of the threshold; the threshold is finalized using Eq.(7.1), Eq.(7.2), and Eq.(7.3). After getting the final value of the threshold, the real value weight matrix is applied to input dataset and then net_A and net_B values are calculated. This net_A and net_B values are compared with $(\lambda_i^{real})^t$ for learning of the system and the parameters count1, count2, and sum^t are obtained. Here we are using step activation function for this comparison. We assign the maximum value to sum_λ^* by comparing the value of sum^t and sum_λ^*. The qubits of the threshold are updated using the quantum update process as discussed in Section 2.2 and Table 7.2. The threshold function returns the sum_g as output by assigning the value of sum_λ^* to sum_g. The value of sum_g is compared with the total number of the input sample. If its learn we stop learning process else, we check the number of generations to update quantum weights using the quantum update process as discussed in Section 2.2 and Table 7.2. If all samples are not learned of class (P_1) and $(P-P_1)$, then we add more neurons for unlearned samples and follow the same process of initialization of connection weights and threshold using the quantum computing concept for subsequent neurons. Once the learning for dataset P_1 and $P - P_1$ is over, we add new neuron for the learning of dataset P_2 and $P - P_2$ in the existing architecture.

Learning of Second Hidden Layer

The second hidden layer contains one neuron corresponding to each sub-neural network. The sub-neural network is the number of neurons in first hidden layer corresponding to signature of each person. If there are x sub-neural networks, then there will be x number of neurons in the second hidden layer. The same process is applied here for deciding connection weights of second hidden layer neuron as applied to the first hidden layer learning. The output of each second hidden layer number will be a unique number which is according to the person for which a particular sub-neural network is trained. If the sub-neural network

trained for P_1, then output from the second hidden layer neuron is 1 if the signature matches, else it will be 0. In the same way, if the sub-neural network is trained for P_l person, then the output from the second hidden neuron is l if the signature matches, else 0.

The overall process is explained in the form of Algorithm 7.1 is presented next :

Algorithm 7.1 Algorithm for EQNN-S.

1: Take input sample as $(X_1^1, X_1^2, X_1^3, ..., X_1^{c_1})$ and $(Y_1^1, Y_1^2, Y_1^3, ..., Y_1^{c_2})$ corresponding to each person.
2: Take first neuron with the weights W_g^{quant} as $W_g^{quant} = (Q_{w1}, Q_{w2}, Q_{w3}, ..., Q_{we})$ where $g = 1, ..., m$; m is the number of iterations to update connection weights.
3: $sum^* = 0$.
4: **for** g=1 to m
5: Initialization of more parameters.
6: $max\_net_A = -\infty$;
7: $min\_net_A = \infty$;
8: $max\_net_B = -\infty$;
9: $min\_net_B = \infty$;
10: **Call observation process**$((W_i^{quant})^g)$
11: **for** i=1 to c_1
12: $net_A(i) = \sum (W_i^{real})^g \times X_1^i$
13: $max\_net_A = \max(max\_net_A, net_A(i))$
14: $min\_net_A = \min(min\_net_A, net_A(i))$
15: **endfor**
16: **for** i=1 to c_2
17: $net_B(i) = \sum (W_i^{real})^g \times Y_1^i$
18: max_net_B=max$(max_net_B, net_B(i))$
19: min_net_B=min$(min_net_B, net_B(i))$
20: **endfor**
21: **Call Quantum Separability Parameter**$((W_i^{real})^g)$
22: **if**$(sum_g \geq (c_1 + c_2))$
23: Stop learning.
24: Assigned new dataset to class A as P_{l+1} and class B as $(P - P_{l+1})$.
25: Repeat Step-1 to Step-23 for learning of class P_{l+1} and $(P - P_{l+1})$.
26: **else**
27: Evaluate sum_g, sum^* update quantum bits of weights by using Table 7.2.
28: $sum^* = \max(sum^*, sum_g)$
29: **endif**
30: **if** $((g == m) \wedge (sum^* \leq (c_1 + c_2)))$
31: Add new neuron for unlearned samples $((c_1 + c_2)$-$sum^*)$ and finalize its weights Step-2 to Step-29.
32: For second neuron the number of samples will be $((c_1 + c_2)$-$sum^*)$ not $(c_1 + c_2)$.
33: **endif**
34: g=g+1

Algorithm 7.1 (Continued)

35: **endfor**

36: Repeat the process of adding hidden layer neurons for each person from Step-1 to Step-35.

Quantum Separability Parameter()

Step-1: Initialization of different parameters

for t=1 to z

z is user defined variable to update $(\lambda_i^{quant})^t$

$(\lambda_i^{quant})^t = (\alpha_i^t \mid \alpha_{i+1}^t)$;

count1=0;

count2=0;

$sum_\lambda^* = 0$;

Call observation process(λ_i^{quant}) to generate real value $(\lambda_i^{real})^t$ using Eq.(7.1), Eq.(7.2), and Eq.(7.3)

for i=1 to c_1

$net_A(i) = \sum (W_i^{real})^g \times X_1^i$

if$(net_A(i) \leq (\lambda_i^{real})^t)$

increase count1 by 1;

endif

endfor

for i=1 to c_2

$net_B(i) = \sum (W_i^{real})^g \times Y_1^i$

if$(net_B(j) > (\lambda_i^{real})^t)$

increase count2 by 1;

endif

endfor

$sum^t = count1 + count2$

sum_λ^*=max(sum_λ^*, sum^t)

update quantum bits for $(\lambda_i^{quant})^{t+1}$ by using Table 7.2.

Generate updated real coded value of $(\lambda_i^{real})^{t+1}$ corresponding to $(\lambda_i^{quant})^{t+1}$ by using observation process and Eq.(7.1), Eq.(7.2), and Eq.(7.3).

sum_g=sum_λ^*

t=t+1

endfor

return sum_g;

7.2.4 Output Layer Learning

Once samples of all the classes are learned to form the hidden layer for data P_1 to P_e, then the learning of output layer starts. The proposed algorithm classify the signature as original or forge. Therefore only one neuron is required at the output layer. The connection weights between the second hidden layer and the output layer neuron are unity. The second hidden layer neurons give output as person number of class number. Hence, during testing, according to input signature, the output layer will get a unique number or 0, which specifies whether a signature is correctly classified or not.

7.3 Experimental Evaluation and Discussion

7.3.1 Experimental Setup

The proposed algorithm has been implemented in two parts. The first part includes processing of the signature image and extraction of features which are implemented in Matlab (Version 7.12.0.635 (R2011a)). The second part consists of training of the neural network and then its testing, which has been implemented in Java (Version 1.8.0_40). The two implementations are done on Intel i-5 4th generation processor with 6 GB RAM. The algorithm is tested on a manually prepared database of signatures.

To train and test the proposed algorithm, we have taken the signatures of 250 people. Ten signatures (seven original + three forged) of each people have been taken. Since all the signatures are collected in the copy form, these signatures are scanned to get in image form. These signatures are then preprocessed, and the required features are extracted. Table 7.1, shows the number of extracted features, which is given as input to the proposed neural network learning algorithm.

To judge the performance of the proposed algorithm, the dataset is divided into three different sets. In the first case, known as 60-40, 60% dataset is used for training and the rest 40% for testing. In the second case 70-30, 70% dataset serves as training data, and the remaining 30% serves as testing data. In the third case, 10-fold cross-validation scheme is applied in which 90% of data has been used for training purpose and remaining 10% is used for testing purposes. This process is repeated ten times. The partitioning detail of the dataset is given in Table 7.3.

Table 7.3: Distribution of Signature Dataset.

Data Partition	Training	Testing
60-40	1400	1000
70-30	1700	700
10-fold cross validation	2160	240

Furthermore, to compare the results, the same partitions of the dataset is used to investigate the classification performance using SVM, MLP, BPNN, and Naive Bayes classifier.

It is clear from the results presented in Table 7.4 that EQNN-S outperforms the established SVM, MLP, Backpropagation, and Naive Bayes classifier. This improvement in classification is mainly due to the two reasons. Firstly, the unique features have been selected to characterize signature. Secondly, the classification of the signature dataset using the evolutionary quantum-based neural network classifier. The quantum computing concept provides exploration due to which the large search space is achieved to get the optimal value. The quantum rotation gate helps to achieve exploitation which saves the algorithm from getting trapped in the local maxima problem. The classification accuracy for 60-40% partition is 95.55%.

The classification accuracy of EQNN-S for 70-30% partition is 95.83%, which is higher than 60-40% and hence reflects the necessity of a proper training set. In the 10-fold cross-validation test, the accuracy reaches to 96.23% which shows improvement with the increase in the training set. The SVM, MLP, Backpropagation, and Naive Bayes classifier also show an improvement in the classification accuracy when the training data set is increased. However, the results are still overshadowed by the performance of EQNN-S. This further supports our previous result that accuracy improves if the training set is increased in size.

Table 7.4: Performance of EQNN-S with Other Classifiers on Different Parameters.

Parameters	60-40					70-30					10-fold cross validation				
	EQNN-S	SVM	MLP	BPNN	Naive Bayes	**EQNN-S**	SVM	MLP	BPNN	Naive Bayes	**EQNN-S**	SVM	MLP	BPNN	Naive Bayes
Average Accuracy	95.55	71.11	85.24	88.56	82.2	95.83	72.61	86.73	89.48	82.6	96.23	72.91	87.52	89.91	73.61
PPV_μ	93.1	78.89	85.15	78.25	95.31	92.31	78.70	84.52	78.55	81.81	92.31	55.81	84.11	78.54	58.47
NPV_μ	96.12	59.25	65.87	58.97	88.40	96.56	69.41	73.52	63.58	83.44	97.21	81.54	78.24	69.81	66.27
$Sensitivity_\mu$	93.25	59.26	84.68	75.42	82.43	94.12	63.89	87.61	78.51	85.13	95.01	66.67	89.46	80.75	76.48
$Specificity_\mu$	88.89	78.89	84.25	81.45	83.78	91.67	81.94	85.91	82.46	81.08	91.67	73.61	87.52	83.16	72.83
$F-score_\mu$	93.17	71.11	78.25	75.26	88.40	93.21	70.229	79.81	77.16	83.44	93.64	60.76	80.54	79.28	66.27
PPV_M	94.44	78.05	85.46	83.15	95.83	94.44	80.09	88.19	84.16	84.72	94.44	83.33	89.55	85.54	75.93
NPV_M	97.27	71.06	91.46	85.61	57.65	97.30	73.61	92.58	88.16	83.33	97.33	79.73	93.46	90.15	72.71
$Sensitivity_M$	93.50	59.26	66.85	63.75	81.94	94.31	63.89	68.24	64.21	84.72	95.46	66.67	70.15	67.82	75.83
$Specificity_M$	88.89	78.89	85.54	81.25	83.33	91.67	81.94	86.75	82.65	80.55	91.67	73.61	87.21	83.35	72.5
$F-score_M$	93.97	72.41	85.46	81.46	88.34	94.37	71.08	87.26	82.46	84.72	94.95	73.22	88.53	83.56	75.88

Table 7.5: Performance of EQNN-S with Other Classifiers.

Data Partition	**EQNN-S**			SVM			MLP			Backpropagation			Naive Bayes		
	Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max
60-40	93.63	95.55	97.45	69.36	71.11	73.54	83.15	85.15	88.15	82.54	83.72	86.54	80.15	82.22	83.27
70-30	94.01	95.83	97.98	70.89	72.61	74.44	84.38	86.41	89.46	83.44	84.65	86.67	81.49	82.63	83.34
10-fold cross validation	94.10	96.23	98.01	71.03	72.91	73.12	86.27	87.22	90.14	84.57	86.74	89.35	72.51	73.61	74.62

Table 7.6: Performance of EQNN-S with other Approaches in Terms of Classification Accuracy.

Methods	Training-Testing Partition (%)	Classification Accuracy (%)
EQNN-S	60-40	95.55
	70-30	95.83
	10-fold cross validation	96.23
Q-BNN	60-40	93.11
	70-30	93.84
	10-fold cross validation	94.55
PMT	60-40	89.95
	70-30	90.15
	10-fold cross validation	91.42
OSVNN	60-40	86.47
	70-30	89.55
	10-fold cross validation	92.54

The values mentioned in Table 7.5, shows minimum, average, and maximum values of accuracy for EQNN-S, SVM MLP, Backpropagation and Naive Bayes classifier on different size of partition of training and testing set of the dataset is presented. It is seen that EQNN-S beats the results of SVM, MLP, Backpropagation, and Naive Bayes classifier. Hence, the results confirm that the proposed algorithm can be very useful to assist in verification of offline signatures.

7.3.2 Comparison with other Methods

To judge the performance of the proposed algorithm with respect to other classifier available for signature verification, it is compared with some other recent approaches presented in the literature. Table 7.6, shows the results of the proposed algorithm and other approaches Q-BNN [112], PMT [135], OSVNN [136] in terms of minimum, maximum, and average accuracy. The proposed algorithm achieves maximum accuracy with respect to 60-40%, 70-30%, and 10-fold-cross-validation scheme are 97.45%, 97.98%, and 98.01%, respectively. The results show that with respect to all three data partitions, the EQNN-S outperforms as compared to existing methods [112, 135, 136].

7.4 Summary

In this chapter, we have enhanced our proposed NQ-BNN learning algorithm by introducing a new threshold boundary parameter by proposing EQNN-S algo-

rithm. The threshold boundary parameter helps to get optimal value of threshold by providing optimal range to find threshold using the quantum computing concept. The proposed EQNN-S forms four layers neural network architecture with a input layer, two hidden layers, and an output layer. The proposed EQNN-S algorithm is used to develop a signature verification system. In this we have prepared signature database manually by scanning signatures of the persons on the paper. Then we applied pre-processing to bring all images on the same scale. Once pre-processing is done, then we extracted unique features of offline signature like the number of loops, angle, dense square patches, dimensions, horizontal and vertical dense strips. Once all features are extracted, then we used the EQNN-S for signature verification. The performance of the proposed algorithm is compared with other state-of-the-art approaches and found that the proposed EQNN-S outperforms over other approaches.

Chapter 8

Conclusion

8.1 Contributions

In this research work, we have proposed neural network learning algorithms using the quantum computing concept for solving classification problems of two class and multi-class. We have proposed a Q-BNN algorithm for solving two class classification problems by evolving connection weights using the quantum computing concept. The neural network learning algorithm is enhanced named as NQ-BNN by evolving threshold of neurons along with connection weights using the quantum computing concept for solving two class classification problem. The proposed Q-BNN and NQ-BNN algorithm are tested on benchmark dataset like Breast Cancer dataset, Heart disease dataset, PIMA Indian diabetes dataset, and BUPA liver dataset. The performance of proposed algorithm is also compared with other state-of-the-art approaches and it is found that proposed algorithm perform better in terms of classification accuracy with few numbers of hidden layer neurons.

The performance of neural network decreases for the multiple class dataset in which samples belong to more than one classes. This causes due to the occurrence of overlapped samples in multiple classes of the dataset. To solve such problems, the fuzzy concept has been utilized. Thus, we have proposed the Quantum inspired Fuzzy based Neural Network Learning Algorithm (Q-FNN) for solving two class classification problem. The proposed learning algorithm forms a three layer neural network structure and uses the fuzzy concept to handle the samples which are overlapped to different class regions. In this neural

network learning algorithm, neurons are treated as fuzzy neurons. The connection weights are updated using the fuzzy concept and the fuzzifier parameter used in the fuzzy concept is evolved using the quantum computing concept. The proposed algorithm is tested on benchmark dataset like WBCD, Sonar, Hepatitis, Ionosphere, and Heart dataset. The performance of proposed algorithm is compared with other evolutionary state-of-the-art approaches and it is found that proposed algorithm perform better than other evolutionary state-of-the-art approaches.

The Q-FNN algorithm is further enhanced here for solving multi-class classification problems named as Q-FNNM algorithm. This can also handle the issue of overlapped samples. In this neural network learning algorithm, connection weights are taken as cluster centroids which is evolved using the evolutionary quantum computing concept along with fuzzifier parameter. The proposed algorithm is tested on benchmark dataset like IRIS, WINE, Glass, and Dermatology and compared with other state-of-the-art approaches. The proposed Q-FNNM outperform in comparison with other state-of-the-art algorithm.

We enhanced stacked auto-encoder based deep learning algorithm to solve the problem of classification of complex images, by proposing a Quantum inspired Stacked Auto-encoder based Deep Neural Network Learning Algorithm (Q-DNN). The stacked auto-encoder based deep neural network uses gradient descent algorithm for its learning. In gradient descent algorithm, the learning rate parameter is initialized randomly between 1 and 0. In the proposed algorithm, we evolve the learning rate parameter using the quantum computing concept. The deep learning architecture is formed with the help of unsupervised pre-training and supervised fine-tuning process. These help to form deep neural network architecture with only few numbers of hidden layers. Q-DNN forms three different neural network architecture with one hidden layer, two hidden layers, and three hidden layers. The proposed Q-DNN is tested on benchmark dataset BUPA Liver Disorder dataset, Ionosphere dataset, and the PIMA Indians Diabetes dataset along with MNIST dataset. The performance of proposed Q-DNN is compared with other state-of-the-art approaches along with deep neural network algorithms. The Q-DNN outperform in terms of classification accuracy in comparison to other approaches in few numbers of hidden layers.

We enchanted our proposed NQ-BNN by introducing a threshold boundary

parameter by proposing EQNN-S algorithm. The threshold bonary parameter helps to get optimal value of threshold by providing optimal range to find threshold using the quantum computing concept. The proposed EQNN-S forms four layers neural network architecture with a input layer, two hidden layers, and an output layer. The proposed EQNN-S algorithm is used to develop a signature verification system. We have extracted unique features of signatures as number of loops, angle, dense square patches, dimensions, horizontal and vertical dense strips. The performance of the proposed algorithm is compared with other state-of-the-art approaches and found that the proposed EQNN-S outperforms over other approaches.

The proposed algorithms perform better due to optimizing their learning parameter using the evolutionary quantum computing concept. The use of the quantum computing concept helps to find optimal value of the parameters. The quantum computing concept is characterized by population dynamics, individual representation, evaluation function. It provides a large search space to find the optimal value of a parameter using an observation process thus, exploration is achieved. On the other hand, the quantum rotational gate provides exploitation to evolve the optimal value of neural network parameters.

8.2 Future Work

There is number of possible extensions of proposed approaches presented in the previous chapters. One of the most obvious extensions is the hardware realization of the proposed algorithms. The quantum algorithm is achieving very promising results thus it can be used to make a dedicated hardware device which can work for the specific purpose. The hardware realization of the quantum algorithms and its application as offline signature verification can work for the many organizations to check forged signatures. Similarly, due to the high accuracy achieved using the quantum computing concept, it can be used to develop recommended systems, especially for the disease diagnosis. The proposed algorithms perform well for two class dataset, multi-class dataset along with complex dataset. Thus it can be extended up to solve the problem of big data classification too. We can also propose the quantum based soft computing algorithms which work on actual computers by executing them on a simulator which gives

environment of quantum computers.

Bibliography

[1] M. Aydin and E. Celik, "Assamese character recognition with artificial neural networks," in *Proc. of 21st IEEE International Conference on Signal Processing and Communications Applications Conference (SIU)*. IEEE, North Cyprus, Turkey, October, pp. 1–4, 2013.

[2] K. S. Hatamleh, M. Al-Shabi, A. Al-Ghasem, and A. A. Asad, "Unmanned aerial vehicles parameter estimation using artificial neural networks and iterative bi-section shooting method," *Applied Soft Computing*, vol. 36, pp. 457–467, 2015.

[3] J. R. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221–234, 1987.

[4] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.

[5] A. Sharma, R. Kumar, P. K. Varadwaj, A. Ahmad, and G. M. Ashraf, "A comparative study of support vector machine, artificial neural network and bayesian classifier for mutagenicity prediction," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 3, no. 3, pp. 232–239, 2011.

[6] J. P. Lacassie, J. R. Del Solar, B. Roser, and F. Herve, "Visualization of volcanic rock geochemical data and classification with artificial neural networks," *Mathematical Geology*, vol. 38, no. 6, pp. 697–710, 2006.

[7] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[8] A. F. Weller, A. J. Harris, and J. A. Ware, "Two supervised neural networks for classification of sedimentary organic matter images from palynological preparations," *Mathematical Geology*, vol. 39, no. 7, pp. 657–671, 2007.

[9] P. Tahmasebi and A. Hezarkhani, "A hybrid neural networks-fuzzy logic-genetic algorithm for grade estimation," *Computers & geosciences*, vol. 42, pp. 18–27, 2012.

[10] T. Wang and Y. Wang, "Pattern classification with ordered features using mrmr and neural networks," in *Proc. of IEEE International Conference on Information, Networking and Automation (ICINA)*. IEEE, Kunming, China, pp. 2128–2131, 2010.

[11] T. C. Lu, G. R. Yu, and J. C. Juang, "Quantum-based algorithm for optimizing artificial neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 8, pp. 1266–1278, 2013.

[12] L. H. Chan, S. H. Salleh, and C. M. Ting, "Pca, lda and neural network for face identification," in *Proc. of 4th IEEE International Conference on Industrial Electronics and Applications ICIEA*. IEEE, Xi'an, China, China, pp. 1256–1259, 2009.

[13] R. Sarikaya, G. E. Hinton, and A. Deoras, "Application of deep belief networks for natural language understanding," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 22, no. 4, pp. 778–784, 2014.

[14] G. Ou and Y. L. Murphey, "Multi-class pattern classification using neural networks," *Pattern Recognition*, vol. 40, no. 1, pp. 4–18, 2007.

[15] B. Sun, "Analysis and detection of nonlinear analogue based on variable threshold value neuron," in *Proc. of 5th IEEE International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. IEEE, Hong Kong, China, pp. 225–228, 2013.

[16] C. A. Pena Reyes and M. Sipper, "Evolutionary computation in medicine: an overview," *Artificial Intelligence in Medicine*, vol. 19, no. 1, pp. 1–23, 2000.

[17] M. Zhang, X. Gao, and W. Lou, "A new crossover operator in genetic programming for object classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 5, pp. 1332–1343, 2007.

[18] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary computation*, vol. 3, no. 2, pp. 82–102, 1999.

[19] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, vol. 1, 1992.

[20] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. of IEEE International Conference on Evolutionary Computation Proceedings*. IEEE, Anchorage, Alaska, USA, pp. 69–73, 1998.

[21] D. Teodorovic, P. Lucic, G. Markovic, and M. Dellorco, "Bee colony optimization: principles and applications," in *Proc. of 8th IEEE Seminar on Neural Network Applications in Electrical Engineering (NEUREL)*. IEEE, Belgrade, Serbia, pp. 151–156, 2006.

[22] X. S. Yang and S. Deb, "Cuckoo search via levy flights," in *Proc. of IEEE World Congress on Nature & Biologically Inspired Computing (NaBIC)*. IEEE, Coimbatore, India, pp. 210–214, 2009.

[23] K. H. Han and J. H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 580–593, 2002.

[24] S. Ghosh, S. Biswas, D. Sarkar, and P. P. Sarkar, "A novel neuro-fuzzy classification technique for data mining," *Egyptian Informatics Journal*, vol. 15, no. 3, pp. 129–147, 2014.

[25] C. S. Chen, "Robust self-organizing neural-fuzzy control with uncertainty observer for mimo nonlinear systems," *IEEE Transactions on Fuzzy Systems*, vol. 19, no. 4, pp. 694–706, 2011.

[26] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. of 24th International ACM Conference on Machine Learning*. ACM, Corvallis, USA, pp. 473–480, 2007.

[27] S. Kim, Z. Yu, R. M. Kil, and M. Lee, "Deep learning of support vector machines with class probability output networks," *Neural Networks*, vol. 64, pp. 19–28, 2015.

[28] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc.of the 25th International Conference on Machine learning*. ACM, Helsinki, Finland, pp. 1096–1103, 2008.

[29] A. G. Tettamanzi and M. Tomassini, *Soft computing: integrating evolutionary, neural, and fuzzy systems*. Springer Science & Business Media, 2013.

[30] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[31] D. O. Hebb, *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[32] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[33] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of," *Brain Mechanisms*, pp. 555–559, 1962.

[34] Y. Le Cun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for backpropagation," in *Proc. of Connectionist Models Summer School*. CMU, Pittsburgh, Pa: Morgan Kaufmann, USA, pp. 21–28, 1988.

[35] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," Royal Signals and Radar Establishment Malvern (United Kingdom), Tech. Rep., 1988.

[36] H. Berenji, "Neural networks and fuzzy logic in intelligent control," in *Proc. of 5th IEEE International Symposium on Intelligent Control*. IEEE, Philadelphia, Pennsylvania, pp. 916–920, 1990.

[37] A. Rocha and A. Serapiao, "Neurofuzzy symbolic systems and pattern recognition," in *Proc. of Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic*. IEEE, San Antonio, TX, USA, pp. 421–422, 1994.

[38] C. von Altrock and B. Krause, "Fuzzy logic and neurofuzzy technologies in embedded automotive applications," in *Proc. of 3rd International Conference on Industrial Fuzzy Control and Intelligent Systems*. IEEE, Houston, Texas, USA, pp. 55–59, 1993.

[39] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[40] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[41] T. Back, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2000, vol. 1.

[42] X. Yu and M. Gen, *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

[43] K. De Jong, "Evolutionary computation: a unified approach," in *Proc. of the 14th ACM Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, Philadelphia, PA, USA, pp. 737–750, 2012.

[44] K. D. Rose, "Arguments on evolution. a paleontologist's perspective," *BioScience*, vol. 40, no. 4, pp. 312–314, 1990.

[45] D. B. Fogel, "The advantages of evolutionary computation." in *BCEC*, pp. 1–11, 1997.

[46] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[47] N. Y. Nikolaev and H. Iba, "Learning polynomial feedforward neural networks by genetic programming and backpropagation," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 337–350, 2003.

[48] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[50] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions On Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.

[51] F. H. F. Leung, H. K. Lam, S.-H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural networks*, vol. 14, no. 1, pp. 79–88, 2003.

[52] L. Li and B. Niu, "A hybrid evolutionary system for designing artifical neural networks," in *Proc. of IEEE International Conference on Computer Science and Software Engineering*, vol. 4. IEEE, Wuhan, China, pp. 859–862, 2008.

[53] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," in *Artificial Life II*. Citeseer, 1990.

[54] P. J. Hancock, "Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification," in *Proc. of IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, Baltimore, MD, USA, 108–122, 1992.

[55] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. of IEEE International Workshop on Combinations of Genetic Algorithms and Neural Networks*, Baltimore, MD, USA, 1992.

[56] F. Jian and X. Yugeng, "Neural network design based on evolutionary programming," *Artificial Intelligence in Engineering*, vol. 11, no. 2, pp. 155–161, 1997.

[57] X. Yao and Y. Liu, "Towards designing artificial neural networks by evolution," *Applied Mathematics and Computation*, vol. 91, no. 1, pp. 83–90, 1998.

[58] T. Mu, J. Jiang, Y. Wang, and J. Y. Goulermas, "Adaptive data embedding framework for multiclass classification," *IEEE Transactions On Neural Networks And Learning Systems*, vol. 23, no. 8, pp. 1291–1303, 2012.

[59] J. C. F. Pujol and R. Poli, "Evolving the topology and the weights of neural networks using a dual representation," *Applied Intelligence*, vol. 8, no. 1, pp. 73–84, 1998.

[60] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, no. 5, pp. 563–591, 1980.

[61] D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer," in *Proc. of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[62] P. W. Shor, "Quantum computing," *Documenta Mathematica*, vol. 1, no. 1000, p. 1, 1998.

[63] L. K. Grover, "Quantum mechanical searching," in *Proc. of Congress on Evolutionary Computation*. IEEE, Washington, D.C. USA, vol. 3, pp. 2255–2261, 1999.

[64] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy, "Finding a better-than-classical quantum and/or algorithm using genetic programming," in *Proc. of Congress on Evolutionary Computation*. IEEE, Washington, D.C. USA, vol. 3, pp. 2255–2261, 1999.

[65] R. Ata and Y. Kocyigit, "An adaptive neuro-fuzzy inference system approach for prediction of tip speed ratio in wind turbines," *Expert Systems with Applications*, vol. 37, no. 7, pp. 5454–5460, 2010.

[66] A. Azadeh, M. Saberi, and S. Asadzadeh, "An adaptive network based fuzzy inference system–auto regression–analysis of variance algorithm for improvement of oil consumption estimation and policy making: The cases of canada, united kingdom, and south korea," *Applied Mathematical Modelling*, vol. 35, no. 2, pp. 581–593, 2011.

[67] C. H. Cheng and L. Y. Wei, "One step-ahead anfis time series model for forecasting electricity loads," *Optimization and Engineering*, vol. 11, no. 2, pp. 303–317, 2010.

[68] J. S. R. Jang, C. T. Sun, and E. Mizutani, "Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence," *IEEE Transactions on Automatic Control*, vol. 42, no. 10, pp. 1482–1484, 1997.

[69] H. Kahramanli and N. Allahverdi, "Design of a hybrid system for the diabetes and heart diseases," *Expert Systems with Applications*, vol. 35, no. 1, pp. 82–89, 2008.

[70] P. Luukka, "Classification based on fuzzy robust pca algorithms and similarity classifier," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7463–7468, 2009.

[71] A. Kandel and W. J. Byatt, "Fuzzy sets, fuzzy algebra, and fuzzy statistics," *Proceedings of the IEEE*, vol. 66, no. 12, pp. 1619–1639, 1978.

[72] J. K. Parker, L. O. Hall, and A. Kandel, "Scalable fuzzy neighborhood db-scan," in *Proc. of 2010 IEEE International Conference on Fuzzy Systems*. IEEE, Barcelona, Spain, July, pp. 1–8, 2010.

[73] J. K. Parker, L. O. Hall, and J. C. Bezdek, "Comparison of scalable fuzzy clustering methods," in *Proc. of 2012 IEEE International Conference on Fuzzy Systems*. IEEE, Brisbane, Australia, June, pp. 1–9, 2012.

[74] F. D. A. De Carvalho and C. P. Tenorio, "Fuzzy k-means clustering algorithms for interval-valued data based on adaptive quadratic distances," *Fuzzy Sets and Systems*, vol. 161, no. 23, pp. 2978–2999, 2010.

[75] M. Sato Ilic, "Symbolic clustering with interval-valued data," *Procedia Computer Science*, vol. 6, pp. 358–363, 2011.

[76] H. Zhao, Z. Xu, S. Liu, and Z. Wang, "Intuitionistic fuzzy mst clustering algorithms," *Computers and Industrial Engineering*, vol. 62, no. 4, pp. 1130–1140, 2012.

[77] C. W. De Almeida, R. M. De Souza, and A. L. Candeias, "Fuzzy kohonen clustering networks for interval data," *Neurocomputing*, vol. 99, pp. 65–75, 2013.

[78] J. C. Dunn, *A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters*. Taylor and Francis, 1973.

[79] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 370–379, 1995.

[80] J. Bezdek, *Pattern recognition in handbook of fuzzy computation*. Boston, NY, 1998.

[81] N. A. Erilli, U. Yolcu, E. Egrioglu, C. Aladag, and Y. oner, "Determining the most proper number of cluster in fuzzy clustering by using artificial neural networks," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2248–2252, 2011.

[82] L. O. Hall, A. M. Bensaid, L. P. Clarke, R. P. Velthuizen, M. S. Silbiger, and J. C. Bezdek, "A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 672–682, 1992.

[83] M. J. Fadili, S. Ruan, D. Bloyet, and B. Mazoyer, "On the number of clusters and the fuzziness index for unsupervised fca application to bold fmri time series," *Medical Image Analysis*, vol. 5, no. 1, pp. 55–67, 2001.

[84] J. Yu, Q. Cheng, and H. Huang, "Analysis of the weighting exponent in the fcm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 1, pp. 634–639, 2004.

[85] K. L. Wu, "Analysis of parameter selections for fuzzy c-means," *Pattern Recognition*, vol. 45, no. 1, pp. 407–415, 2012.

[86] S. S. Khan and A. Ahmad, "Cluster center initialization algorithm for k-means clustering," *Pattern Recognition Letters*, vol. 25, no. 11, pp. 1293–1302, 2004.

[87] H. Izakian and A. Abraham, "Fuzzy c-means and fuzzy swarm for fuzzy clustering problem," *Expert Systems with Applications*, vol. 38, no. 3, pp. 1835–1838, 2011.

[88] A. Bahrololoum, H. Nezamabadi pour, and S. Saryazdi, "A data clustering approach based on universal gravity rule," *Engineering Applications of Artificial Intelligence*, vol. 45, pp. 415–428, 2015.

[89] D. Binu, "Cluster analysis using optimization algorithms with newly designed objective functions," *Expert Systems with Applications*, vol. 42, no. 14, pp. 5848–5859, 2015.

[90] Z. Yu and M. Lee, "Real-time human action classification using a dynamic neural model," *Neural Networks*, vol. 69, pp. 29–43, 2015.

[91] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[92] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[93] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[94] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in Neural Information Processing Systems*, pp. 3–10, 1994.

[95] C. M. Lee, S. S. Yang, and C. L. Ho, "Modified backpropagation algorithm applied to decision-feedback equalisation," *IEEE Proceedings-Vision, Image and Signal Processing*, vol. 153, no. 6, pp. 805–809, 2006.

[96] Y. Fukuoka, H. Matsuki, H. Minamitani, and A. Ishida, "A modified back-propagation method to avoid false local minima," *Neural networks*, vol. 11, no. 6, pp. 1059–1072, 1998.

[97] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. of ICML Workshop on Unsupervised and Transfer Learning*, Washington, USA, pp. 37–49, 2012.

[98] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, 2007, pp. 153–160, 2007.

[99] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning," *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[100] J. Dolz, N. Betrouni, M. Quidet, D. Kharroubi, H. A. Leroy, N. Reyns, L. Massoptier, and M. Vermandel, "Stacking denoising auto-encoders in a deep network to segment the brainstem on mri in brain cancer patients: A clinical study," *Computerized Medical Imaging and Graphics*, vol. 52, pp. 8–18, 2016.

[101] P. De Bruyne and R. Forre, "Signature verification with elastic image matching," pp. 12–14, 1986.

[102] T. S. Wilkinson, D. A. Pender, and J. W. Goodman, "Use of synthetic discriminant functions for handwritten-signature verification," *Applied Optics*, vol. 30, no. 23, pp. 3345–3353, 1991.

[103] Y. Qi and B. R. Hunt, "Signature verification using global and grid features," *Pattern Recognition*, vol. 27, no. 12, pp. 1621–1629, 1994.

[104] R. Bajaj and S. Chaudhury, "Signature verification using multiple neural classifiers," *Pattern Recognition*, vol. 30, no. 1, pp. 1–7, 1997.

[105] G. Dimauro, S. Impedovo, G. Pirlo, and A. Salzo, "A multi-expert signature verification system for bankcheck processing," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 11, no. 05, pp. 827–844, 1997.

[106] M. A. Ferrer, J. F. Vargas, A. Morales, and A. Ordóñez, "Robustness of offline signature verification based on gray level features," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 966–977, 2012.

[107] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009.

[108] A. K. Gupta and Y. P. Singh, "Analysis of bidirectional associative memory of neural network method in the string recognition," in *Proc. of 2011 International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, Gwalior, India, pp. 172–176, 2010.

[109] A. Baka, W. Wettayaprasit, and S. Vanichayobon, "A novel discretization technique using class attribute interval average," in *Proc of 4th International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. IEEE, Bangkok, Thailand, pp. 95–100, 2014.

[110] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 436–451, 2000.

[111] S. Bahramirad, A. Mustapha, and M. Eshraghi, "Classification of liver disease diagnosis: a comparative study," in *Proc. of 2nd International Conference on Informatics and Applications (ICIA)*. IEEE, Dhaka, Bangladesh, pp. 42–46, 2013.

[112] O. P. Patel and A. Tiwari, "Quantum inspired binary neural network algorithm," in *Proc. of 13th International Conference on Information Technology (ICIT)*. IEEE, Bhubaneswar, India, pp. 270–274, 2014.

[113] O. P. Patel. and A. Tiwari, "Liver disease diagnosis using quantum-based binary neural network learning algorithm," in *Proceedings of 4th International Conference on Soft Computing for Problem Solving*. Springer, Silchar, India, pp. 425–434, 2015.

[114] R. Meyer and S. O'keefe, "A fuzzy binary neural network for interpretable classifications," *Neurocomputing*, vol. 121, pp. 401–415, 2013.

[115] N. Bharill and A. Tiwari, "Enhanced cluster validity index for the evaluation of optimal number of clusters for fuzzy c-means algorithm," in *Proc. of 2014 IEEE International Conference on Fuzzy Systems*. IEEE, Beijing, China, pp. 1526–1533, 2014.

[116] O. P. Patel and A. Tiwari, "Novel quantum inspired binary neural network algorithm," *Sadhana*, vol. 41, no. 11, pp. 1299–1309, 2016.

[117] L. J. Moreira and L. A. Silva, "Data classification combining self-organizing maps and informative nearest neighbor," in *Proc. of International Joint Conference on Neural Networks (IJCNN)*. IEEE, Anchorage, Alaska, pp. 706–713, 2016.

[118] Q. Qian, S. Chen, and W. Cai, "Simultaneous clustering and classification over cluster structure representation," *Pattern Recognition*, vol. 45, no. 6, pp. 2227–2236, 2012.

[119] N. Garcia Pedrajas, D. Ortiz Boyer, and C. Hervas Martinez, "An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization," *Neural Networks*, vol. 19, no. 4, pp. 514–528, 2006.

[120] J. H. Lee, J. R. Anaraki, C. W. Ahn, and J. An, "Efficient classification system based on fuzzy–rough feature selection and multitree genetic programming for intension pattern recognition using brain signal," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1644–1651, 2015.

[121] V. N. Vapnik and V. Vapnik, *Statistical learning theory.* Wiley New York, vol. 1, 1998.

[122] B. Gong, "A novel learning algorithm of backpropagation neural network," in *Proc. of IEEE International Conference on Control, Automation and Systems Engineering.* IEEE, Christchurch, New Zealand, pp. 411–414, 2009.

[123] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[124] O. P. Patel and A. Tiwari, "Advance quantum based binary neural network learning algorithm," in *Proc. of 2015 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).* IEEE, Takamatsu, Japan, pp. 1–6, 2015.

[125] T. Hastie, R. Tibshirani, and J. Friedman, "Overview of supervised learning," in *The elements of statistical learning*, pp. 9–41, 2009.

[126] A. Grubb and J. A. Bagnell, "Stacked training for overfitting avoidance in deep networks," in *Proc. of Workshop on Representation Learning Appearing at the ICML*, New York City, USA, pp. 1, 2013.

[127] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[128] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial Intelligence and Statistics*, 2009, pp. 448–455, 2009.

[129] A. Makhzani and B. Frey, "K-sparse autoencoders," *arXiv preprint arXiv:1312.5663*, 2013.

[130] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders," in *Advances in Neural Information Processing Systems*, 2015, pp. 2791–2799.

[131] T. Raiko, H. Valpola, and Y. LeCun, "Deep learning made easier by linear transformations in perceptrons," in *Artificial Intelligence and Statistics*, 2012, pp. 924–932.

[132] A. Sanmorino and S. Yazid, "A survey for handwritten signature verification," in *Proc. of 2nd IEEE International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE)*. IEEE, Jalarta, Indonesia, pp. 54–57, 2012.

[133] C. Kruthi and D. C. Shet, "Offline signature verification using support vector machine," in *Proc. of IEEE 5th International Conference on Signal and Image Processing (ICSIP)*. IEEE, Ghaziabad, India, pp. 3–8, , 2014.

[134] A. P. Shanker and A. Rajagopalan, "Off-line signature verification using dtw," *Pattern recognition letters*, vol. 28, no. 12, pp. 1407–1414, 2007.

[135] I. Bhattacharya, P. Ghosh, and S. Biswas, "Offline signature verification using pixel matching technique," *Procedia Technology*, vol. 10, pp. 970–977, 2013.

[136] A. Pansare and S. Bhatia, "Handwritten signature verification using neural network," *International Journal of Applied Information Systems*, vol. 1, no. 2, pp. 44–49, 2012.

[137] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," *Large Scale Kernel Machines*, pp. 301–320, 2007.

Appendix A

Datasets

In this section, we present datasets on which we have tested our algorithms and compared it with the state-of-the-art approches. In all, total thirteen dataset were used in the experiments. Eleven datasets are taken from the UCI Machine Learning Repository [123] and MNIST is taken from [137] and the remaining one is being prepared manually. The description of all datasets is given in Table A.1. These datasets containing small, medium, large, and complex data are required to show the generalization ability of our approaches. The brief discussion about these datasets are as follows:

Table A.1: Characteristics of the Datasets.

Datasets	# Samples	# Features	# Classes
Breast Cancer Wisconsin Original (WBCD)	699	9	2
PIMA Indian Diabetes (PID)	768	8	2
Sonar	208	60	2
BUPA Liver Disorders	345	6	2
Hepatitis	80	19	2
Ionosphere	351	33	2
Heart Disease	270	13	2
IRIS	270	13	3
WINE	178	13	3
Glass	214	9	7
Dermatology	366	34	6
MNIST	70000	784	10
Offline Signature	2500	45	250

Breast Cancer Wisconsin Original (WBCD)

The dataset consists of 699 samples with 9 features of each sample. It is a two class dataset. Each instance has one of the two possible classes that is benign or malignant. The percentage of the benign dataset 65.5% and the percentage of the malignant dataset is 34.5%.

PIMA Indian Diabetes

It consists of 768 data samples that belong to only female patients suffering from diabetes. It has 8 numerical features per sample. Each sample contains a label which indicates the class of the sample. The dataset has two classes where the first class is labeled as "negative to diabetes" and the second one is labeled as "positive to diabetes".

BUPA Liver Disorders Dataset

The dataset consist of 345 samples with 6 features are measured for each sample. The first 5 features are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption while the last attribute includes daily alcohol consumption. Each sample constitutes the record of a single male individual.

Heart Disease

This dataset has total 270 samples, and each sample has 13 attributes. The dataset has been taken of people belongs to an age group of 29 to 77. The dataset has been collected from male and female both.

Sonar

This dataset has total 208 samples, and each sample has 60 features. This dataset contains signals obtained from a variety of different aspect angles, spanning 90 degrees for mines and 180 degrees for rocks. Each pattern is a set of 60 numbers in the range 0.0 to 1.0, where each number represents the energy within a particular frequency band, integrated over a certain period of time. The two classes are R and M represents rock and mine (metal cylinder).

Hepatitis

This dataset has total 80 samples, and each sample has 19 features. Actually, this dataset has 155 samples, but non-missing attribute samples are only 80. This dataset contains a mixture of integer and real-valued attributes, with information about patients affected by the Hepatitis disease. The task is to predict if these patients will die (1) or survive (2).

Ionosphere

This dataset has total 351 samples, and each sample is characterized by 33 features. A system collected this radar data in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The two classes represent a good and a bad signal.

IRIS

The well-known Fishers IRIS dataset is a multivariate dataset for discriminant analysis. It consists of 150 samples belongs to 3 classes of IRIS flowers and four features were measured corresponding to each sample. The classes are IRIS Setosa, IRIS Versicolour, and IRIS Virginica. The features are sepal length, sepal width, petal length, and petal width. The data set contains 50 instances of each of the three classes.

WINE

This dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivators and consists of 178 samples. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

GLASS

The dataset consists of 214 samples, for each sample, 10 features are measured. All the samples are divided into 6 classes. The study of classification of types of glass was motivated by the criminological investigation. At the scene of the crime, the glass left can be used as evidence, if it is correctly identified.

Dermatology

This database contains 366 instances, for each instance 34 features are measured. This dataset is divided into 6 classes. The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, cronic dermatitis, and pityriasis rubra pilaris.

MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. It was created by re-mixing the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students. It is having total 70000 samples with 10 class where each sample has 784 features.

Offline Signature

This dataset is created by own which has a collection of 250 people signatures. Ten signature (seven original + three forged) of each people have been taken. Since all the signatures are collected on paper, these signatures are scanned to get in image form. These signatures are first pre-processed, and then total 45 features are extracted. The features are the number of loops, dense patches, angle, bounding caps, dimensions, etc.

Appendix B

Validation Methods

In machine learning field, it is common to partition the dataset into two separate sets: a training set and a testing set. To evaluate the generalizability of our approach and to compare our work with existing work in literature, we divided the training and testing data into four different partitions. The reason for dividing training and testing samples into different partitions is that we want to measure the quality of our proposed algorithms with different amount of training samples. The validation methods used for experimentation are discussed below.

50-50 training-testing partition

In standard 50-50 methodology, half of the samples are used for training the classifier and the rest for testing.

60-40 training-testing partition

In 60-40 training-testing methodology, 60% of the samples are used for training the classifier and the rest 40% for testing.

70-30 training-testing partition

In 70-30 training-testing methodology, 70% of the samples are used for training the classifier and the rest 30% for testing.

10-fold cross validation

In 10-fold cross validation technique entire dataset is divided into ten blocks of approximately equal size. The 90% of data is used to train the model and the

rest 10% for testing. This process is repeated 10 times, with a different data block left out for testing every time.