Enhanced quantum-based neural network learning and its application to signature verification

Article *in* Soft Computing · May 2019

DOI: 10.1007/s00500-017-2954-3

8 authors, including:

op Patel
Indian Institute of Technology Indore
28 PUBLICATIONS **878** CITATIONS
SEE PROFILE

Neha Bharill
Indian Institute of Information Technology Dharwad
35 PUBLICATIONS **1,105** CITATIONS
SEE PROFILE

Mukesh Prasad
University of Technology Sydney
155 PUBLICATIONS **3,636** CITATIONS
SEE PROFILE

Farookh Khadeer Hussain
University of Technology Sydney
397 PUBLICATIONS **5,945** CITATIONS
SEE PROFILE

Soft Computing
https://doi.org/10.1007/s00500-017-2954-3

METHODOLOGIES AND APPLICATION

Enhanced quantum-based neural network learning and its application to signature verification

Om Prakash Patel[1] · Aruna Tiwari[1] · Rishabh Chaudhary[1] · Sai Vidyaranya Nuthalapati[1] · Neha Bharill[1] · Mukesh Prasad[2] · Farookh Khadeer Hussain[2] · Omar Khadeer Hussain[3]

Abstract
In this paper, an enhanced quantum-based neural network learning algorithm (EQNN-S) which constructs a neural network architecture using the quantum computing concept is proposed for signature verification. The quantum computing concept is used to decide the connection weights and threshold of neurons. A boundary threshold parameter is introduced to optimally determine the neuron threshold. This parameter uses min, max function to decide threshold, which assists efficient learning. A manually prepared signature dataset is used to test the performance of the proposed algorithm. To uniquely identify the signature, several novel features are selected such as the number of loops present in the signature, the boundary calculation, the number of vertical and horizontal dense patches, and the angle measurement. A total of 45 features are extracted from each signature. The performance of the proposed algorithm is evaluated by rigorous training and testing with these signatures using partitions of 60–40 and 70–30%, and a tenfold cross-validation. To compare the results derived from the proposed quantum neural network, the same dataset is tested on support vector machine, multilayer perceptron, back propagation neural network, and Naive Bayes. The performance of the proposed algorithm is found better when compared with the above methods, and the results verify the effectiveness of the proposed algorithm.

Keywords Quantum computing · Neural network · Signatures · Feature extraction · Classification

1 Introduction

Artificial neural network has emerged as one of the most promising areas of research in recent years. It is used for pattern recognition, classification, mining, clustering and many more (Castellani 2017; Sartakhti et al. 2017; Wang and Wang 2010; Gupta and Singh 2011). Researchers have presented many learning algorithms, such as perceptron, back propagation and multilayer perceptron for solving both two-class and multi-class problems. The performance of a neural network system varies according to its architecture, hidden layers, the number of neurons in the hidden layer, connection weights, and threshold. It is also dependent on the features

Communicated by V. Loia.

✉ Om Prakash Patel
phd1301201003@iiti.ac.in

✉ Mukesh Prasad
mukesh.nctu@gmail.com

Aruna Tiwari
artiwari@iiti.ac.in

Rishabh Chaudhary
ee1200231@iiti.ac.in

Sai Vidyaranya Nuthalapati
ee1200221@iiti.ac.in

Neha Bharill
phd12120103@iiti.ac.in

Farookh Khadeer Hussain
Farookh.Hussain@uts.edu.au

Omar Khadeer Hussain
o.hussain@adfa.edu.au

[1] Department of Computer Science and Engineering, Indian Institute of Technology Indore, Simrol Indore, Madhya Pradesh, India

[2] Centre for Artificial Intelligence, School of Software, FEIT, University of Technology Sydney, Sydney, Australia

[3] School of Business, University of New South Wales, Canberra, Australia

Published online: 30 November 2017

of the input dataset (Chan et al. 2009; Sarikaya et al. 2014). One way to determine the neural network architecture is to form it constructively (Patel and Tiwari 2014). Deciding neural network architecture in this way helps to limit the number of neurons. The connection weights of neurons in the neural network can be decided in several ways such as by random initialization, genetic algorithm, and quantum computing. Many researchers use quantum computing to solve the problem in an evolutionary fashion (Han and Kim 2002; Lu et al. 2013; Qiu et al. 2017). Han and Kim (2002) have conducted significant work to address the knapsack issue using the quantum computing concepts. Based on this, a neural network algorithm has been proposed in which optimization of the learning parameters was carried out using quantum computing concept (Lu et al. 2013). Patel and Tiwari (2014) proposed a quantum-based algorithm for binary neural network learning algorithm in which the neural network architecture is formed constructively. In this algorithm, the connection weights are decided using the quantum computing concept. Further improvement is proposed in this paper by deciding the connection weights and threshold using the quantum computing concept. The neural network formed in this way is trained and tested on a signature dataset, which is manually prepared.

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 (Buryne and Forre 1986), synthetic discriminant functions (Wilkinson et al. 1991) and grid features (Qi and Hunt 1994). 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 (Bajaj and Chaudhury 1997; Dimauro et al. 1997). A number of researchers worked on finding appropriate features of a signature. Ferrer et al. (2012) proposed a technique in which grayscale features such as local binary pattern (LBP) and local directional pattern (LDP) are analyzed. Signature verification plays an important role not only in document authenticity but also in security of information and data. Many researchers worked in the filed of information and data security using signature verification (Sun et al. 2014; Kuzuno and Tonami 2015; Nakamura et al. 2016; Ren et al. 2016; Wang et al. 2016). These approaches use specific pattern of data or information as signature rather than human-made signatures.

Finding appropriate static features from a signature image is still an open area of research. With the advancement of technology, the extraction of features enters a whole new dimension. Moreover, the efficiency of matching two signature images is an important parameter in any signature verification technique. To make this task more efficient, an enhanced quantum-based neural network learning algorithm which is based on the quantum computing concept and its application to signature verification is proposed.

The remainder of this paper is organized as follows. Section 2 briefly describes the preliminaries. Section 3 describes the complete proposed algorithm. In this section, we also illustrate the proposed algorithm using a sample signature. The experiments and results are discussed in Sect. 4. Section 5, concludes the paper and details our proposed future work.

2 Preliminaries

In this paper, an enhanced quantum-based neural network learning algorithm is proposed. The proposed algorithm is trained and tested on a manually prepared signature dataset. The proposed algorithm is divided into two major sections: developing a quantum neural network and evaluating proposed algorithm performance using the signature database. The necessary prerequisites are briefly described. The quantum neural network preliminaries and related basics of extracting features from the signature dataset are explained in detail.

This method forms a neural network architecture, which consists of four layers: an input layer, two hidden layers, and the output layer. The number of input nodes is equal to the number of attributes of the signature dataset. Let $P_1 = (X_1^1, X_1^2, X_1^3, \ldots, X_1^{c_1})$ denote the input samples, where c_1 is the number of input samples and $X_l^i = (x_i^l, x_i^2, x_i^3, \ldots, x_i^e)$ where e is the number of attributes in one instance of the input sample. The number of input layer nodes is equal to e. The number of neurons in the hidden layer is decided constructively. For ith hidden layer neuron, connection weights are denoted as follows:

$$W_i^{real} = (w_{i1}, w_{i2}, w_{i3}, \ldots, w_{ie}) \tag{1}$$

In the proposed algorithm, these connection weights are decided using the quantum computing concept. Its representation in terms of quantum bits is defined as follows:

$$W_i^{quant} = (Q_{i1}, Q_{i2}, Q_{i3}, \ldots, Q_{ie}) \tag{2}$$

where each w_{ij} is denoted by Q_{ij}, thus W_i^{real} can be represented as W_i^{quant}.

The activation function for ith neuron is taken as a step function and represented as:

$$net_i = \sum_{j=1}^{e} w_{ij} \times x_j \tag{3}$$

$$f(net_i) = \begin{cases} 1 & \text{if } net_i \leq \text{threshold} \\ 0 & \text{if } net_i > \text{threshold} \end{cases} \tag{4}$$

Here, the threshold of the neuron is also decided using the quantum computing concept. The representation of the quantum threshold for ith neuron is as follows:

$$Th_i = \left(Q_i^{Th}\right) \tag{5}$$

To process these quantum weights and quantum threshold, quantum computing-related preliminaries are discussed next (Han and Kim 2002).

2.1 Quantum computing concept

The quantum computing concept is utilized to select the learning parameters. The parameters are represented in terms of a quantum bit (Q_i). These quantum bits (Q_i) are made up of several qubits q_{ij} (where $j = 1, 2, \ldots, k$) which can be represented as follows:

$$Q_i = (q_{i1}|q_{i2}|\ldots|q_{ik}) \tag{6}$$

Here, k is the number of qubits which represents the quantum bit (Q_i). A single qubit (q_{ij}) is the smallest unit for representing information. Thus, qubit q_{ij} can be represented as:

$$q_{ij} = \alpha_{ij} \mid 0\rangle + \beta_{ij} \mid 1\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix} \tag{7}$$

where α_{ij} and β_{ij} are complex numbers representing the probability of a qubit in "0" state and in "1" state. A probability model is applied here which represents "0" state by α_{ij}^2 and "1" state by β_{ij}^2, represented as:

$$\alpha_{ij}^2 + \beta_{ij}^2 = 1; \quad 0 \le \alpha \le 1, \quad 0 \le \beta \le 1 \tag{8}$$

As discussed above, a quantum bit (Q_i) formed using qubits. Each qubit represents two states, i.e., "0" state or "1" state. Collectively, these qubits form a single quantum bit (Q_i). Thus, a quantum bit $(Q_i$ having two qubits $(q_{i1}, q_{i2}, (j = 1, 2))$ in it, represents four states, i.e., "00," "01," "10" and "11." In the same way, a three-qubit system $(q_{i1}, q_{i2}, q_{i3}, (j = 1, 2, 3))$ represents eight states, and n qubits $(q_{in}, (j = 1, \ldots, n))$ will have 2^n states. An individual quantum bit having two qubits can be represented as follows:

$$Q_i = \left\langle \begin{matrix}\alpha_{i1}|\alpha_{i2} \\ \beta_{i1}|\beta_{i2}\end{matrix} \right\rangle \tag{9}$$

Thus, quantum bit (Q_i) with its four states on two qubits $(q_{ij}, (j = 1, 2))$ is represented as follows:

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

Normally, a quantum bit (Q_i) having two qubits (q_{ij}) with its component $\alpha_{i1}, \alpha_{i2}, \beta_{i1}$ and β_{i2} is represented and initialized as follows:

$$Q_i = \left\langle \begin{matrix}1/\sqrt{2}|1/\sqrt{2} \\ 1/\sqrt{2}|1/\sqrt{2}\end{matrix} \right\rangle \tag{11}$$

With respect to Eqs. (10) and (11), the state representation of quantum bits Q_i is as follows:

$$\begin{aligned} Q_i = {} & (1/\sqrt{2} \times 1/\sqrt{2})\langle 00\rangle + (1/\sqrt{2} \times 1/\sqrt{2})\langle 01\rangle \\ & + (1/\sqrt{2} \times 1/\sqrt{2})\langle 10\rangle + (1/\sqrt{2} \times 1/\sqrt{2})\langle 11\rangle \end{aligned} \tag{12}$$

Following this representation of the parameters in quantum bits, we next discuss conversion process. The conversion process converts quantum bits to real coded values.

2.1.1 Conversion from quantum bits to real values

In the proposed algorithm, quantum weights and threshold are required to convert into real coded value. The exploration is achieved using conversion process. The weight matrix in terms of quantum bits W_i^{quant} is converted into a real value weight matrix W_i^{real}. Similarly, the threshold value in terms of quantum bits, Th_i is converted into real value Th_i^{real}. This conversion process starts by taking random number matrix R, where $R_i = [r_{i1} r_{i2} \ldots r_{ik}]$, corresponding to a quantum bit $Q_i = (q_{i1}|q_{i2}|\ldots|q_{ik})$. Then, further mapping is done by using binary matrix S_j where $S_i = [s_{i1} s_{i2} \ldots s_{ik}]$ and Gaussian random number generator. The Gaussian random number generator is represented as $N(\mu, \sigma)$ with the help of variable mean value μ and variance σ. The binary to decimal formulation is used which map binary matrix S_j to Gaussian number generator. The value of matrix S_i is passed into $bin2dec(S_i)$ formula to select a value from a Gaussian random generator (Lu et al. 2013). The value of matrix S_i is generated using r_{ij} and α_{ij} of q_{ij} as follows:

$$if\,(r_{ij} \le (\alpha_{ij})^2) \quad \text{then} \quad s_{ij} = 1 \quad \text{else} \quad s_{ij} = 0.$$

Thus, using this for all qubits, $Q_i = (q_{i1}|q_{i2}|\ldots|q_{ik})$ is converted into real values. In this way, all Q_i components of quantum weight (W_i^{quant}) and quantum threshold (Th_i) are evaluated to get real coded value of connections weights (W_i^{real}) and real coded value of threshold (Th_i^{real}). The real value generated using conversion process in the first generation may or may not be optimal. Therefore, to get optimal value of quantum weight (W_i^{quant}) and quantum threshold (Th_i) are evolved using quantum update process which is discussed in detail in Sect. 3. The proposed algorithm is used to classify offline signature; therefore, preliminaries related to the signature are discussed next:

2.2 Preliminaries of signature image

Firstly, original and forged signatures of a few people have been collected in the paper, and then, these hard copies are scanned to get signatures in the form of the image. Due to variation in the pen, discrepancies, and background noise in images, we first make these images on same scale (Sanmorino and Yazid 2012). To get images on the same scale, we preprocess all images. The following steps are followed for the preprocessing and extracting the features from the signature image.

2.2.1 Conversion from RGB to black and white

To preprocess signature image first its color is made uniform. Therefore, each signature image of the different colors is converted into a black and white image.

2.2.2 Noise removal

The noise removal process is performed after converting all images in uniform, i.e., black and white color. The signature images have noise due to two main sources: first, the background paper on which the signature is taken which may not be uniform of the same color. Secondly, the noise arises while scanning the paper having signatures. This noise will hinder the training and testing of signatures and hence must be removed. Median filtering is used as a remedy here.

2.2.3 Thinning

A signature impression may be made with pens of varying tips. However, the difference in tip size should not be a factor to distinguish signatures. The thickness of every stroke in a signature is reduced to a width of a single pixel.

The steps discussed above help to standardize a given signature image. Once the preprocessing is done, the feature extraction process starts which is as follows (Kruthi and Shet 2014; Shanker and Rajagopalan 2007; Ferrer et al. 2012):

2.2.4 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. 1.

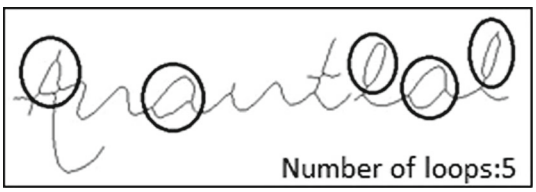

Fig. 1 Counting the number of loops

Fig. 2 The image is cropped to the signature's exact width and height

2.2.5 Dimensions

The exact height and width of the signature are calculated. The image is also cropped to this exact width and height for further use as shown in Fig. 2.

2.2.6 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, vertical stripes give the other ten feature values.

2.2.7 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 shown in Fig. 3. 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.

2.2.8 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 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 as shown in Fig. 4.

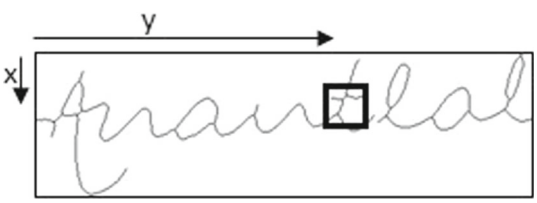

Fig. 3 Square dense patch

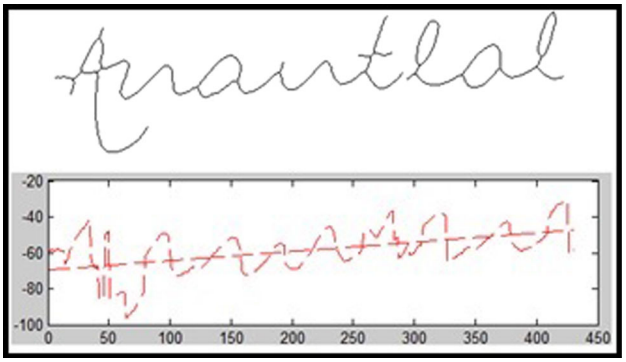

Fig. 4 Angle

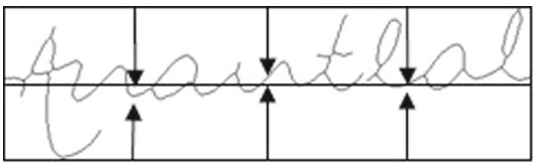

Fig. 5 Bounding caps

2.2.9 Bounding caps

The image is divided into four vertical patches of uniform width. On 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. Similar distances are calculated for the first pixel from the bottom of the signature as shown in Fig. 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 1, total 45 feature values serve to define a signature image. These values are used as the input layer of the neural network. In this way, the dataset of persons can be defined as $P = (P_1, P_2, P_3, \ldots, P_n)$ where each person has c_1 number of signatures represented as $P_l = (X_l^1, X_l^2, X_l^3, \ldots, X_l^{c_1})$. Here each signature has e number of attributes, which can be represented as $X_l^i = (x_i^l, x_i^2, x_i^3, \ldots, x_i^e)$.

3 Proposed approach

In this section, an enhanced quantum-based neural network learning algorithm is presented, and its performance is measured on the manually prepared signature database. In this algorithm to get optimal connection weights and threshold of neuron, the quantum computing concept is used. The basic architecture of the proposed system is shown in Fig. 6. The proposed system works in three phases; in first and second phases, preprocessing and feature extraction is done on signature data which is discussed above. In the third phase, the extracted features are used to train and test the proposed enhanced quantum-based neural network algorithm.

Table 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

3.1 Input dataset

The signature classification belongs to a multi-class problem, as each person is considered as individual class. This multi-class problem is solved using multiple two-class classification problem. The signatures corresponding to n persons represent the n number of classes. For solving a multi-class classification problem as multiple two-class classification problem, let us assume that samples $P = (P_1, P_2, P_3, \ldots, P_n)$ denote the categories of n person corresponding to their signature. Let a category of lth person denoted by (P_l) be a collection of variation of signature of signature of lth person. This is denoted by $P_l = (X_l^1, X_l^2, X_l^3, \ldots, X_l^{c_1})$ having c_1 number of signatures, where $X_l^i = (x_i^l, x_i^2, x_i^3, \ldots, x_i^e)$ where e is the number of attributes in one signature of input sample. To solve the multi-class problem as multiple two-class problem, one class has been considered as P_l, while another class is considered as $P - P_l$.

3.2 Learning of hidden layer

Hidden layer learning of multiple two-class classification problem is initiated by considering the individual n number of two-class classification problem. It means when learning

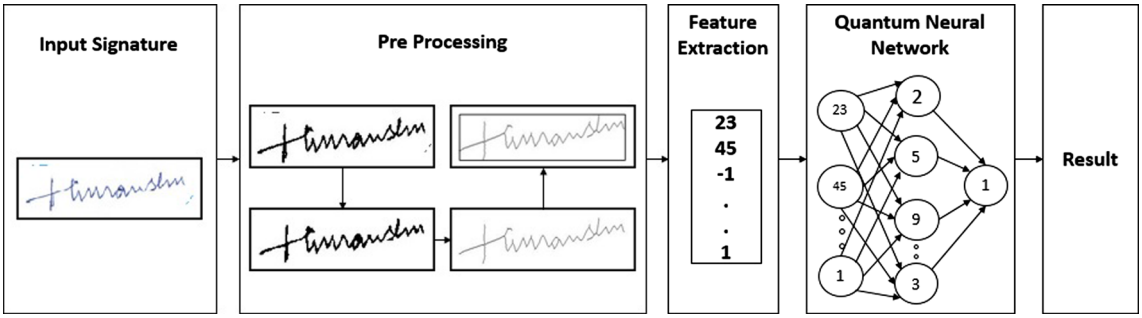

Fig. 6 Basic architecture

for the signature corresponding to the first person P_1, its samples $(X_1^1, X_1^2, X_1^3, \ldots, X_1^{c_1})$, where $X_1^i = (x_i^1, x_i^2, x_i^3, \ldots, x_i^e)$ are taken as instances for say class A. The remaining samples corresponding to signatures of persons $(P - P_1)$ $=(Y_1^1, Y_1^2, Y_1^3, \ldots, Y_1^{c_2})$, where $Y_1^i = (y_i^1, y_i^2, y_i^3, \ldots, y_i^e)$, are taken as for the class B. Thus, total samples belonging to categories of P are divided into two classes A and B. Now this two-class problem is solved further with the following steps.

3.2.1 Initialization and representation of parameters

Firstly, a neuron is selected at hidden layer and its quantum weights (W_1^{quant}) $g = 1$ are initialized in terms of quantum bits. Along with initialization of quantum weights, the parameters F^*, S^*, 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 proper threshold of neuron.

3.2.2 Conversion from quantum bits to real values

Now, the quantum weights (W_1^{quant}) are converted into a real weight matrix (W_1^{real}) using conversion process. Now, the real weight matrix (W_1^{real}) is applied to input dataset to calculate the boundary parameters for finding the threshold.

3.2.3 Boundary parameters calculation

In the proposed algorithm, the threshold Th_i^{real} 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_1^{real}). These values help to find out actual distribution of projection of input dataset with connection weights. Thus, its gives more diversity to search appropriate value of threshold and avoid local minima and 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 threshold in the defined range, which solves the problem of selecting a random threshold value and also saves the time of the learning process. The following formulations show the calculation of boundary parameter.

$$(\text{TBP})_t^{max} = \max(max\_net_A, max\_net_B); \tag{13}$$

$$(\text{TBP})_t^{min} = \min(min\_net_A, min\_net_B); \tag{14}$$

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

$$Th_t^{real} = \begin{cases} (\text{TBP})_t^{min} & \text{if } Th_t^{real} < (\text{TBP})_t^{min} \\ (\text{TBP})_t & \text{if } (\text{TBP})_t^{min} \leq Th_t^{real} \leq (\text{TBP})_t^{max} \\ (\text{TBP})_t^{max} & \text{if } Th_t^{real} > (\text{TBP})_t^{max} \end{cases} \tag{15}$$

The TBP_1^{min} and TBP_1^{max} parameters are calculated corresponding to the weights and input dataset. Now, the quantum threshold $(Th_1)_1$ $(t = 1)$ is initialized in terms of qubits for $g = 1$ and $t = 1$. Along with these parameters, some more parameters count1 = 0, count2 = 0, and F_{Th}^* are also initialized. Using the boundary parameter and conversion process, the real coded value of threshold $(Th_1^{real})_1$ is achieved. However, this is not final value of the threshold; the threshold is finalized using Eq. (15). After getting the final value of the threshold, the real value weight matrix is applied to input dataset and net_A and net_B values are calculated. These net_A and net_B values are compared with $(Th_1^{real})_1$ for learning of the system, and the parameters count1, count2, and F_1 $(t = 1)$ are obtained. The values of fitness $F_1(t = 1)$ and F_{Th}^* are compared, and maximum value is assigned to F_1 $(g = 1)$. Now qubits of the threshold are updated according to the value of F_1 $(t = 1)$ and F_{Th}^*. Similarly F_1 $(g = 1)$ and F^* are compared, and if these satisfy stopping criteria, then we stop learning here, else the quantum bits of weights are updated using qubit updation process.

3.2.4 Qubit updation

To evolve the new value of weight matrix, W_i^{real} and threshold Th_i^{real}, the quantum weights W_i^{quant} and quantum threshold Th_i are updated using fitness value and binary bits corresponding to qubits. Let the quantum weights be evolved in g number of iterations and let quantum threshold be evolved in t number of iterations. To update quantum weight $(W_i^{quant})_{g+1}$ from $(W_i^{quant})_g$ and quantum threshold $(Th_i)_{t+1}$ from $(Th_i)_t$, quantum rotation gates (Han and Kim 2002) are required which is shown as follows:

$$U(\Delta\theta) = \begin{vmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{vmatrix} \tag{16}$$

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

$$q_{ij} = \begin{vmatrix} \alpha_{ij}^{g+1} \\ \beta_{ij}^{g+1} \end{vmatrix} = \begin{vmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{vmatrix} * \begin{vmatrix} \alpha_{ij}^{g} \\ \beta_{ij}^{g} \end{vmatrix} \tag{17}$$

3.2.5 Selection of $\Delta\theta$

To update quantum bit (Q_i^g), each single qubit (q_{ij}^g) is required to update. To update each qubit (q_{ij}^g), the current fitness value F_g, best fitness value F^*, current binary bit s_{ij}^g and binary bit corresponding to best fitness s_{ij}^* are considered (Han and Kim 2002). Initially at iteration $g = 1$ and $t = 1$, the best fitness value F^* can be initialized by the user. As shown in the conversion process, component of qubits α_{ij}^g is associated with binary value s_{ij}^g; therefore, mapping is done between F^* and binary string s_{ij}^* to update individual qubit (Han and Kim 2002). Selection of $\Delta\theta$ is done as follows:

- If the fitness value in the current generation is worse than that of best fitness value in last iterations, and state of s_{ij}^g is zero and state of s_{ij}^* is one, then decrement in probability of the qubit, α_{ij}^{g+1}, may produce worst result. Therefore, to increase the probability of α_{ij}^g to one, $\Delta\theta$ is made negative.
- If the fitness value in the current generation is better than the best fitness value in last iterations, and state of s_{ij}^g is one and s_{ij}^* state is zero, then increasing probability of the qubit, α_{ij}^{g+1} to one, may produce worst result. Therefore, to update α_{ij}^g, angular displacement is made positive by $\Delta\theta$.
- In all other cases, angular displacement will remain zero.

In case of threshold updation, the binary bit value will be s_{ji}^t in comparison with $(s_{ji}^*)_\lambda$ and fitness value will be F_t and F_λ^*. The value of angular displacement must be selected in such a way so that it can cover the maximum value of α_{ij}^g in the range of (0 1) and also should not take many iterations to cover these values. Therefore, $\Delta\theta$ must be initialized between $(0.01 \times \pi, 0.05 \times \pi)$ (Lu et al. 2013). Table 2 shows the tabular representation of the updation process. The overall process is explained in the form of an algorithm which is presented next: **Algorithm**: *EQNN-S Algorithm*

Step-1: Take Input sample as $(X_1^1, X_1^2, X_1^3, \ldots, X_1^{c_1})$ and $(Y_1^1, Y_1^2, Y_1^3, \ldots, Y_1^{c_2})$ corresponding to each person
Take first neuron with the weights W_g^{quant}
$W_g^{quant} = (Q_{w1}, Q_{w2}, Q_{w3}, \ldots, Q_{we})$
where $g = 1, \ldots, m$; m is the number of iterations to update weights

Step-2: **for** g=1 to m
Initialization of more parameters
$F^* = 0$
$S^* = 0$;
$max\_net_A = -\infty$;
$min\_net_A = \infty$;
$max\_net_B = -\infty$;
$min\_net_B = \infty$;
Call conversion process(W_g^{quant})
for i=1 to c_1
$net_A(i) = \sum W_g^{real} \times X_1^i$
$max\_net_A = \max(max\_net_A, net\_A(i))$
$min\_net_A = \min(min\_net_A, net\_A(i))$
endfor
for i=1 to c_2
$net_B(i) = \sum W_g^{real} \times Y_1^i$
$max\_net_B = \max(max\_net_B, net\_B(i))$
$min\_net_B = \min(min\_net_B, net\_B(i))$
endfor
Call Quantum Threshold Parameter(W_g^{real})
if$(F_g \geq (c_1 + c_2))$
Stop learning
Assigned new dataset to class A as P_{l+1} and class B as $(P - P_{l+1})$
Repeat step-1 to step-2 for learning of class P_{l+1} and $(P - P_{l+1})$
else
Evaluate F_g, F^*, s_i^g, s_i^* and update quantum bits of weights by using Table 2, Eq. (16) and Eq. (17).
$F^* = \max(F^*, F_g)$
endif
if $((g == m) \wedge (F^* \leq (c_1 + c_2)))$
Add new neuron for unlearnt sample $((c_1 + c_2)\text{-}F^*)$ and finalize its weight by using Step-1 and Step-2.

For second neuron number of samples will be $((c_1 + c_2)\text{-}F^*)$ not $(c_1 + c_2)$.
endif
g=g+1
endfor
Repeat the process for each person from step-1 to step-2

Quantum Threshold function()

Step-1 : Initialization of different parameters
for t=1 to z
z is user defined variable to update Th_t
$Th_t = (Q_i^{Th})$;
count1=0;
count2=0;
$F_{Th}^* = 0$;
Call conversion process(Th_t) to generate real value Th_t^{real} using Eq. (13), Eq. (14) and Eq. (15)
for i=1 to c_1
$net_A(i) = \sum W_g \times X_1^i$
if$(net_A(i) \leq Th_t^{real})$
increase count1 by 1;
endif
endfor
for i=1 to c_2
$net_B(i) = \sum W_g \times Y_1^i$
if$(net_B(j) > Th_t^{real})$
increase count2 by 1;
endif
endfor
$F_t = count1 + count2$
$F_{Th}^* = \max(F_{Th}^*, F_t)$
update quantum bits for Th_{t+1} by using Table 2, Eqs. (16) and (17).
Generate updated real coded value of Th_{t+1}^{real} corresponding to Th_{t+1} by using conversion process and Eqs. (13), (14) and (15).
t=t+1
$F_g = F_{Th}^*$
endfor
return F_g;

3.3 Output layer learning

Once the all class samples are learned to form the hidden layer for data P_1 to P_n, the learning of output layer starts. The overall neural network works as multi-class classification; therefore, one more hidden layer is required, which maps each multiple two-class neural network and output layer neuron. The second hidden layer contains one neuron each corresponding to each sub-neural network. If there are x sub-neural networks, then there are x number of neurons in second hidden layer. As the proposed algorithm classifies the signature as original or forge, only one neuron is required at the output layer. The same process is applied here for deciding weights of second hidden layer neuron as applied to first hidden layer learning. Once the learning for dataset P_1 and $P - P_1$ with two number of hidden layer neuron is over, we add new neuron for learning of dataset P_2 and $P - P_2$ in the existing architecture. The output of each second hidden layer number will be a unique number which is according to the person for which particular sub-neural network is trained. As if sub-neural network is trained for P_1, then output from second hidden layer neuron is 1 if signature matches, or 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, or else 0. The connection weights between second hidden layer and output layer neuron are unity. 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.

Table 2 Qubits updation

s_{ij}^g	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

3.4 Illustration of EQNN-S

3.4.1 Preprocessing and feature extraction

In the first phase, signature images are scanned, and the preprocessing process is conducted. The image is converted to a standard form using preprocessing techniques, as discussed in Sect. 2. In this process firstly, the image is converted to a black and white image. Secondly, noise removal technique is implemented so that proper features can be extracted from the image. Thirdly, after successfully removing the noise, the image is thinned, which finally renders a clean image. The above discussed steps help in processing the image for proper feature extraction.

In the second phase, features are extracted from images. At first, the image is cropped to its exact dimension, i.e., any extra white space along the boundary of the image is removed. From this, two feature values are extracted: the exact height and width of the signature pattern. Having done this, the next feature is extracted, namely bounding caps, which gives 6 feature values. The 5 most dense horizontal and vertical patches are recorded. From these, twenty feature values are extracted: the first five describe the densest horizontal patches. The next five describe their location. The next five features describe the densest vertical patches with their locations forming the final five features. The next fifteen feature values represent the 5 most dense square patches and their location. Finally, the angle of the signature is calculated. These sum up to 45 feature values which are given as an input to the neural network.

The same process is implemented for all the valid and forgery signatures, and data are divided into two parts. The first part is training data, and the other is testing data. For example, if we have three valid signatures and one forged signature of a person, then we use 2 valid signature for training purpose and 2 signature (1 valid + 1 forgery) for testing purpose. Thus, if 3 persons are there, then the dataset for training, represented by X, contains 6 signatures (2 of each class or person) and the dataset for testing consists of 6 signatures (2 of each class or person). Now, according to incremental learning of neural network, the dataset is divided into two virtual classes. Class A represents the data set of first person's signature, and class B represents the sum of datasets of remaining person's signature. For the ease of illustration, only two features are being considered. Let us say class A data are $P_1 = ((0.11, 0.15), (0.14, 0.17))$ and class B data are $P - P_1$ $((0.25, 0.35)(0.27, 0.36)(0.41, 0.42)(0.47, 0.39))$.

3.4.2 Classification using quantum neural network

Once, the signature database is prepared, the working of the quantum neural network starts. Initially, a single neuron is taken at the hidden layer, and connection weights are initialized. For two features the number of nodes at input layer will be two. The quantum weights are initialized for this connection as $(W_1^{quant})_1 = (0.578|0.546, 0.852|0.2)$. Next, the conversion process starts by generating some random number matrices as R=(0.2|0.8,0.7|0.6). Following step-1 in the conversion process, the binary matrix $S_1 = (10, 11)$ is obtained by comparing quantum bits and the random number. The binary matrix S_1 and Gaussian number generator are used to obtain the real coded value, which is achieved as (0.4, 0.5). After conversion of quantum weights into real coded form, the boundary parameter is calculated for threshold calculation. Once the boundary is decided and the quantum threshold is initialized as $(Th_1^{quant})_1 = (0.525|0.468)$, the same conversion process is used to get the real coded value of the quantum threshold. Conversion process and threshold boundary parameter generate the real coded value as $(Th_1^{real})_1 = 0.227$. Now, net_A and net_B are calculated with the help of input sample and real coded weight values. It is then compared with real coded value of threshold to compute fitness. With the above dataset the value of net_A is computed as (0.097, 0.113) and net_B is computed as (0.225, 0.234, 0.278, 0.279). Now, this value is compared with the obtained real coded value of the threshold. It can be clearly seen that value of $count1$ is 2 and $count2$ is 3 as mentioned in algorithm; thus, F_1 is 5 ($F_t = count1 + count2$) at $g = 1$ and $t = 1$. Now, with the same weight, the quantum threshold is updated to get a more accurate value of the quantum threshold. Let us say, in second iteration $t = 2$ at $g = 1$, the $(Th_1^{real})_2$ is 0.225. Now, the same value of $count1$, $count2$ and F_2 is achieved as in the first iteration by the same method. The same process will continue till $t = 100$ at $g = 1$, and the $(Th_i^{real})_t$ will be selected on the basis of best value of F_{Th}^*. In order to get a more appropriate value of weights, the quantum weights are updated using quantum update function. The same process will continue until neurons learn for the entire sample or $g = 100$, and $t = 100$ for each generation of weight updation g. If unlearnt sample remains, then new neuron is added, and the same process of learning is repeated. Once learning of class A and class B data is done, the dataset of class A and B is modified. Now, class A contains data of the second person's signature and class B contains the signature of the first and third person. The same training process is implemented with these new classes. After completion of hidden neuron training, the training of the output layer is done.

4 Experimental results

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 neural network and then 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 of RAM. The algorithm is tested on a manually prepared database of signatures.

To train and test the proposed algorithm, we have taken signature of 250 people. Ten signatures (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 then preprocessed, and the required features are extracted. Table 1 shows the number of extracted features, which is given as input to the proposed neural network learning algorithm.

Table 3 Confusion matrix

	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

4.1 Experimental parameters

The parameters used for evaluating the performance of the given algorithm are computed with the help of the confusion matrix which is given in Table 3. This matrix contains information about actual and predicted classification performed by the classifier.

The confusion matrix evolves around four simple concepts:

1. *True Positive (TP)* It is the number of times the input is true and is detected correctly. No further computation has to be done in the confusion matrix. The values are directly retrieved.
2. *True Negative (TN)* It is the number of times the input is false and is predicted correctly.

$$\mathrm{TN}_A = \varepsilon_{BC} + \varepsilon_{CB} + \mathrm{TP}_B + \mathrm{TP}_C$$

3. *False positive (FP)* It is the number of instances when the input is false but is predicted to be true.

$$\mathrm{FP}_A = \varepsilon_{BA} + \varepsilon_{CA}$$

4. *False Negative (FN)* It is the number of times when the input is true but is detected to be false.

$$\mathrm{FN}_A = \varepsilon_{AB} + \varepsilon_{AC}$$

The parameters used to compare the performance of an algorithm are described henceforth with the use of these concepts.

4.1.1 Average accuracy

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

$$\text{Average Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \tag{18}$$

4.1.2 Error rate

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

$$\text{Error Rate} = \frac{1}{n}\sum_{i=1}^{n}\frac{fp_i + fn_i}{tp_i + tn_i + fp_i + fn_i} \tag{19}$$

4.1.3 Precision/positive predicted value (PPV)

It is the ratio of the times when an authentic signature is correctly predicted to the number of times a signature is predicted to be true.

$$\mathrm{PPV}_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i}{tp_i + fp_i} \tag{20}$$

$$\mathrm{PPV}_\mu = \frac{\sum_{i=1}^{n} tp_i}{\sum_{i=1}^{n}(tp_i + fp_i)} \tag{21}$$

4.1.4 Recall/sensitivity

It is the percentage of the time when an authentic signature is predicted to be true when the signature is actually true.

$$Recall_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tp_i}{tp_i + fn_i} \tag{22}$$

$$Recall_\mu = \frac{\sum_{i=1}^{n} tp_i}{\sum_{i=1}^{n}(tp_i + fn_i)} \tag{23}$$

4.1.5 Specificity

It specifies the number of times when the forged signature is predicted to be forged and is actually forged.

$$Specificity_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tn_i}{fp_i + tn_i} \tag{24}$$

$$Specificity_\mu = \frac{\sum_{i=1}^{n} tn_i}{\sum_{i=1}^{n}(fp_i + tn_i)} \tag{25}$$

$$NPV_M = \frac{1}{n}\sum_{i=1}^{n}\frac{tn_i}{tn_i + fn_i} \tag{26}$$

$$NPV_\mu = \frac{\sum_{i=1}^{n} tn_i}{\sum_{i=1}^{n}(tn_i + fn_i)} \tag{27}$$

Table 4 Distribution of signature dataset

Data partition	Training	Testing
60–40	1400	1000
70–30	1700	700
Tenfold cross-validation	2160	240

4.1.6 FScore

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

$$FScore_M = \frac{\left(\beta^2+1\right) \times PPV_M \times Recall_M}{\beta^2 \times \left(PPV_M + Recall_M\right)} \tag{28}$$

$$FScore_\mu = \frac{\left(\beta^2+1\right) \times PPV_\mu \times Recall_\mu}{\beta^2 \times \left(PPV_\mu + Recall_\mu\right)} \tag{29}$$

4.1.7 Negative predicted value (NPV)

It is the ratio of the number of times a forged signature is predicted to be forged to the number of times a signature is predicted to be forged.

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. The third is a tenfold cross-validation scheme in which 90% of data have been used for training purpose and remaining 10% are used for testing purposes. This process is repeated ten times. The partitioning detail of the dataset is given in Table 4. Furthermore, to compare the results, the same dataset is classified on support vector machine(SVM), multilayer perceptron (MLP), backpropagation neural network (BPNN), and Naive Bayes.

It is clear from the results presented in Table 5 that EQNN-S outperforms the established SVM, MLP, BPNN 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 and secondly, the classification of signature dataset using evolutionary quantum-based neural network classifier. The quantum computing concept provides exploration due to which the large search space is achieved to get optimal value. The quantum rotation gate helps to achieve exploitation which saves the algorithm from getting trapped in the local minima and maxima problem. The classification accuracy for 60–40% partition is 0.9555. The classification accuracy of EQNN-S for 70–30% partition is 0.9583, which is higher than 60–40%, and hence reflects on the necessity of a proper training set. In the tenfold cross-validation test, the accuracy reaches 0.9623 which

Table 5 Performance of EQNN-S with other classifiers on different parameters

Parameters	60–40					70–30					Tenfold 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	0.9555	0.7111	0.8524	0.8856	0.822	0.9583	0.7261	0.8673	0.8948	0.826	0.9623	0.7291	0.8752	0.8991	0.7361
Error rate	0.0445	0.2889	0.0857	0.1583	0.177	0.0417	0.2709	0.758	0.1285	0.173	0.0377	0.277	0.687	0.1157	0.268
PPV_μ	0.931	0.7889	0.8515	0.7825	0.9531	0.9231	0.7870	0.8452	0.7855	0.8181	0.9231	0.5581	0.8411	0.7854	0.5847
NPV_μ	0.9612	0.5925	0.6587	0.5897	0.8840	0.9656	0.6941	0.7352	0.6358	0.8344	0.9721	0.8154	0.7824	0.6981	0.6627
$Sensitivity_\mu$	0.9325	0.5926	0.8468	0.7542	0.8243	0.9412	0.6389	0.8761	0.7851	0.8513	0.9501	0.6667	0.8946	0.8075	0.7648
$Specificity_\mu$	0.8889	0.7889	0.8425	0.8145	0.8378	0.9167	0.8194	0.8591	0.8246	0.8108	0.9167	0.7361	0.8752	0.8316	0.7283
$FScore_\mu$	0.9317	0.7111	0.7825	0.7526	0.8840	0.9321	0.70229	0.7981	0.7716	0.8344	0.9364	0.6076	0.8054	0.7928	0.6627
PPV_M	0.9444	0.7805	0.8546	0.8315	0.9583	0.9444	0.8009	0.8819	0.8416	0.8472	0.9444	0.8333	0.8955	0.8554	0.7593
NPV_M	0.9727	0.7106	0.9146	0.8561	0.5765	0.9730	0.7361	0.9258	0.8816	0.8333	0.9733	0.7973	0.9346	0.9015	0.7271
$Sensitivity_M$	0.9350	0.5926	0.6685	0.6375	0.8194	0.9431	0.6389	0.6824	0.6421	0.8472	0.9546	0.6667	0.7015	0.6782	0.7583
$Specificity_M$	0.8889	0.7889	0.8554	0.8125	0.8333	0.9167	0.8194	0.8675	0.8265	0.8055	0.9167	0.7361	0.8721	0.8335	0.725
$FScore_M$	0.9397	0.7241	0.8546	0.8146	0.8834	0.9437	0.7108	0.8726	0.8246	0.8472	0.9495	0.7322	0.8853	0.8356	0.7588

Table 6 Performance of EQNN-S with other classifiers

Data partition	EQNN-S			SVM			MLP			BPNN			Naive Bayes		
	Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max
60–40	0.9363	0.9555	0.9745	0.6936	0.7111	0.7354	0.8315	0.8515	0.8815	0.8254	0.8372	0.8654	0.8015	0.8222	0.8327
70–30	0.9401	0.9583	0.9798	0.7089	0.7261	0.7444	0.8438	0.8641	0.8946	0.8344	0.8465	0.8667	0.8149	0.8263	0.8334
Tenfold cross-validation	0.9410	0.9623	0.9801	0.7103	0.7291	0.7312	0.8627	0.8722	0.9014	0.8457	0.8674	0.8935	0.7251	0.7361	0.7462

shows improvement with the increase in the training set. SVM, MLP, BPNN 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. The values mentioned in Table 6 shows minimum, average and maximum values of accuracy for EQNN-S, SVM MLP, BPNN and Naive Bayes classifier across differently sized training datasets presented. It is clearly seen that EQNN-S beats the results of SVM, MLP, BPNN and Naive Bayes classifier. The result, hence, confirms that the proposed algorithm can be very useful to assist in verification of offline signatures.

4.2 Comparison with other methods

To judge the performance of the proposed algorithm with respect to other available classifier for signature verification, it is compared with some other recent approaches presented in the literature. Table 7 shows the results of the proposed algorithm and other approaches Q-BNN (Patel and Tiwari 2014), PMT (Bhattacharya et al. 2013), OSVNN (Pansare and Bhatia 2012) in terms of minimum, maximum and average accuracy. The proposed algorithm achieves maximum accuracy with respect to 60–40, 70–30% and tenfold cross-validation scheme are 0.9745, 0.9798 and 0.9801, respectively. The results show that with respect to all three data partitions the EQNN-S outperforms as compared to existing methods (Patel and Tiwari 2014; Bhattacharya et al. 2013; Pansare and Bhatia 2012).

5 Conclusion

In this paper, an enhanced quantum-based neural network learning algorithm is proposed which is used to classify offline signatures. Each signature has been characterized by different 45 unique features. The features include the number of loops, dimensions, horizontal and vertical stripes, dense patches and a half distance. These features are given as input to the quantum-based neural network. The quantum-based neural network forms network structure constructively, which reduces unnecessary training of the system. The connection weights are decided using the quantum computing concept. To find the proper separation between input classes, a quantum threshold with boundary parameter is also proposed. The threshold boundary parameter helps to find the optimal value of threshold with the help of min-max function. The results show that EQNN-S performs better than SVM, MLP, BPNN and Naive Bayes classifier. The proposed EQNN-S is also compared with other signature verification

Table 7 Performance of EQNN-S with other methods

Methods	Training-testing partition (%)	Min	Avg	Max
EQNN-S	60–40	0.9363	0.9555	0.9745
	70-30	0.9401	0.9583	0.9798
	Tenfold cross-validation	0.941	0.9623	0.9801
Q-BNN	60–40	0.9125	0.9311	0.9412
	70–30	0.9254	0.9384	0.9554
	Tenfold cross-validation	0.9301	0.9455	0.9668
PMT	60–40	0.8812	0.8995	0.9014
	70–30	0.8945	0.9015	0.9212
	Tenfold cross-validation	0.9013	0.9142	0.9285
OSVNN	60–40	0.8567	0.8647	0.8951
	70–30	0.8821	0.8955	0.9311
	Tenfold cross-validation	0.9001	0.9254	0.9451

classifier, and the result shows that proposed algorithm outperforms with respect to other state of the art approaches.

As it is easy to extract more features like speed, pressure from the online signature. In the near future, our next proposal will be based on online signature verification with desired modification.

Compliance with ethical standards

Conflicts of interest The authors declare that they have no conflict of interest.

References

Bajaj R, Chaudhury S (1997) Signature verification using multiple neural classifiers. Pattern Recognit 30(1):1–7

Bhattacharya I, Ghosh P, Biswas S (2013) Offline signature verification using pixel matching technique. Procedia Technol 30:970–977

Buryne PD, Forre R (1986) Signature verification with elastic image matching. In: International Carnahan conference on security technology Gothenburg, Sweden, August. IEEE, pp 12–14

Castellani M (2017) Competitive co-evolution of multi-layer perceptron classifiers. Soft Comput: 1–16. https://doi.org/10.1007/s00500-017-2587-6

Chan L-H, Salleh S-H, Ting C-M (2009) PCA, LDA and neural network for face identification. In: 4th IEEE conference on industrial electronics and applications, Xian, China, IEEE, pp 1256–1259

Dimauro G, Impedovo S, Pirlo G, Salzo A (1997) A multi-expert signature verification system for bankcheck processing. IJPRAI 11(5):827–844

Ferrer M, Vargas J, Morales A, Ordonez A (2012) Robustness of offline signature verification based on gray level features. IEEE Trans Inf Forensics Secur 7(3):966–977

Gupta AK, Singh YP (2011) Analysis of bidirectional associative memory of neural network method in the string recognition. In: International conference on computational intelligence and communication networks (CICN), Gwalior, India, 2011. IEEE, pp 172–176

Han K-H, Kim J-H (2002) Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans Evol Comput 6(6):580–593

Kruthi C, Shet DC (2014) Offline signature verification using support vector machine. In: Fifth international conference on signal and image processing (ICSIP), Bangalore, India, 2014. IEEE, pp 3–8

Kuzuno H, Tonami S (2015) Detection of sensitive information leakage in Android applications using signature generation. Int J Space Based Situated Comput 5(1):53–62

Lu T-C, Yu G-R, Juang J-C (2013) Quantum-based algorithm for optimizing artificial neural networks. IEEE Trans Neural Netw Learn Syst 24(8):1266–1278

Nakamura S, Duolikun D, Enokido T, Takizawa M (2016) A read-write abortion protocol to prevent illegal information flow in role-based access control systems. Int J Space Based Situated Comput 6(1):43–53

Pansare A, Bhatia S (2012) Handwritten signature verification using neural network. Int J Appl Inf Syst 1(2):44–49

Patel OP, Tiwari A (2014) Quantum inspired binary neural network algorithm. In: International conference on information technology (ICIT), Bhubaneswar, India, 2014. IEEE, pp 270–274

Qi Y, Hunt BR (1994) Signature verification using global and grid features. Pattern Recognit 27(12):1621–1629

Qiu L, Sun X, Xu J (2017) Categorical quantum cryptography for access control in cloud computing. Soft Comput: 1–8

Ren Y, Wang H, Du J, Ma L (2016) Code-based authentication with designated verifier. Int J Grid Util Comput 7(1):61–67

Sanmorino A, Yazid S (2012) A survey for handwritten signature verification. In: 2nd International conference on uncertainty reasoning and knowledge engineering (URKE), Jakarta, Indonesia, 2012. IEEE, pp 54–57

Sarikaya R, Hinton GE, Deoras A (2014) Application of deep belief networks for natural language understanding. IEEE/ACM Trans Audio Speech Lang Process 22(4):778–784

Sartakhti JS, Afrabandpey H, Saraee M (2017) Simulated annealing least squares twin support vector machine (SA-LSTSVM) for pattern classification. Soft Comput 21(15):4361–4373

Shanker AP, Rajagopalan A (2007) Off-line signature verification using DTW. Pattern Recognit Lett 28(12):1407–1414

Sun X, Tian H, Wang Y, Ren Y (2014) Toward quantum-resistant strong designated verifier signature. Int J Grid Util Comput 5(2):80–86

Wang T, Wang Y (2010) Pattern classification with ordered features using mrmr and neural networks. In: International conference on information, networking and automation, ICINA 2010, Kunming, China, 2010, pp 2128–2131

Wang Y, Ma J, Lu X, Lu D, Zhang L (2016) Efficiency optimisation signature scheme for time-critical multicast data origin authentication. Int J Grid Util Comput 7(1):1–11

Wilkinson TS, Pender DA, Goodman JW (1991) Use of synthetic discriminant functions for handwritten-signature verification. Appl Opt 30(23):3345–3353