# Implementation of Quantum Gates using MATLAB

**Nitesh Kumar, Neeraj Sahu**

***Abstract:** Now days we require low cost and high performance computational based applications. Quantum inspired computational device or circuit performs effective result compare to classical based devices. In the development of quantum-based devices and network needs number of quantum logic gates. Here we studied mathematical description of different types of single and multiple qubits-based quantum logic gates, the reversibility property of quantum gates also proved mathematically. We analyze the cost and effectiveness of each quantum gates has been implemented using neural network with the help of MATLAB. The cost and effectiveness of quantum gates has been analyzed with the comparison of different types of activation function.*

***Keywords:** Qubit, Quantum gates, Neural Networks, classical devices, MATLAB.*

## I. INTRODUCTION

Quantum Computer provides many facilities inbuilt to classical computer. Classical computer cannot manage wide band signals to simple signal. They also face many problems like absence of rule for optimal architecture, time consuming training, limited memory capacity. But quantum computation based on the quantum mechanical nature of physics [1], [2], [3]. The classical computation always performed Boolean logic, which uses variable 0 and 1 [7]. The physical interpretation of 0 and 1 is voltage on/off. But the quantum computation provides a new set of rules that go ahead of this classical model. The quantum computing basic variable define quantum bit [2], [3], [4]. Quantum bit defines as a vector in a two-dimensional complex hilbert space here we require combine both quantum computation and neural computation which is called quantum neural networks. In the present time quantum computing is latest field of algorithms cryptography and artificial intelligence [8], [9]. The basic properties of classical and quantum gate is logical operation [10].

Manuscript published on January 30, 2020.
* Correspondence Author

**Mr. Nitesh Kumar*,** Pursuing PhD, Department of Computer Science & Enginnering, Raffles University, Neemrana, Rajasthan, India.

**Dr Neeraj Sahu,** Assistant Professor, Raffles University, Neemrana, Rajasthan, India.

Quantum neural network is advanced artificial neural network. Quantum neural network and be developed using the principles of quantum computing. Quantum mechanics are more powerful computationally than a classic machine. The purpose of quantum computation is to reduce the number of elements of computer using the quantum laws.

In quantum computation two possible states for Qubit are |0> and |1>. The difference between classical bits and quantum bits are that a Qubit can be in state other than |0> or |1>. The superposition | φ > of Qubit is linear combination of these states.

$$\varphi = \alpha|0> + \beta|1>$$

Where α is the amplitude of measuring |0> and β is amplitude of measuring of value |1>, and α and β are the complex coefficients satisfy the normalization condition $\alpha^2 + \beta^2 = 1$.

$$|0> = \begin{bmatrix}1\\0\end{bmatrix} \quad , |1> = \begin{bmatrix}0\\1\end{bmatrix}$$

The quantum gate always representation in matrix form using $\sum_i |\text{input}i >< output\,i|$ . An artificial neuron network (ANN) is a computational model based on the structure and function of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes –or learns, in a sense-based on that input and output.

Quantum gates are two types single qubit gates or multiple qubit gate. We analyzed both single qubit and multiple qubit in this paper. NOT, Hadamard are single qubit gate and C-NOT gate are multiple qubit quantum gate. We also compare these gates result using MATLAB.

## II. QUANTUM ACTIVATION FUNCTION

Activation function used as threshold functions or also called transfer function. Transfer function is used to transform the activation level of neurons into output signals. In neural network, there are many transfer functions like Log-sigmoid, hyperbolic tangent, purelin etc.

Sigmoid function always generates a positive number between 0 and 1. Sigmoid function work like as $\Phi(z) = \frac{1}{1+e^{-x}}$. Hyperbolic tangent function also used symbol as tanh. The range of tanh are between -1 to 1 can represent as $f(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$. The linear function is purelin and which range is -∞ to +∞ . $\sum_i |\text{input}i >< output\,i|$

In quantum logic gate we need a linear output that's why purelin may be used to train a quantum logic gate.

## III. SINGLE QUBIT GATES

### A. Quantum NOT Gate

Not gate is the single qubit gate, which takes the qubit |0> as input and generate the output |1> or if takes input as |1> qubit then produce the result as |0>.
We can describe the quantum NOT gate by using following diagram:

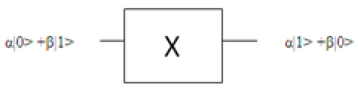


**Fig. 1. Quantum NOT gate**

Suppose we have two input as |0> qubit and |1> qubit then to calculate the resultant we know the matrix representation of above input qubit is $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$, $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$ .The total output of quantum NOT gate we will be calculate the value of input qubit into matrix form, then

= |0> <1| + |1> <0|

$$=\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} \qquad \left(|0\rangle = \begin{bmatrix}1\\0\end{bmatrix},\ \langle 0| = \begin{bmatrix}1 & 0\end{bmatrix}\right)$$

$$= \begin{bmatrix}0 & 1\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\1 & 0\end{bmatrix} = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

Step- 1: when we apply input as |0>

$$X[|0\rangle] = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}.|0\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}.\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

Step 2: When we apply input as |1>

$$X[|1\rangle] = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}.|1\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}.\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

Step 3: When we apply input as α|0> +β|1>

$$X[\alpha|0\rangle + \beta|1\rangle] = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} \quad = \begin{bmatrix}\alpha\\0\end{bmatrix} + \begin{bmatrix}\beta\\0\end{bmatrix}$$

$$= \begin{bmatrix}\alpha\\\beta\end{bmatrix} \quad \text{i.e. as input}$$

$$X\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \begin{bmatrix}\beta\\\alpha\end{bmatrix}$$

$$\begin{bmatrix}\beta\\\alpha\end{bmatrix} = \alpha\begin{bmatrix}0\\1\end{bmatrix} + \beta\begin{bmatrix}1\\0\end{bmatrix} = \alpha|1\rangle + \beta|0\rangle$$

Proof of reversible:

Step 1: When we apply input as |1>

$$X[|1\rangle] = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}|1\rangle = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$$

Step 2: When we apply input as |0>

$$X[|0\rangle] = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}|0\rangle$$

$$= \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$$

Step 3: When we apply input as α|1> +β|0>

$$X[\alpha|1\rangle + \beta|0\rangle] = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}.[\alpha|1\rangle + \beta|0\rangle]$$

$$=\alpha\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} + \beta\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \alpha|0\rangle + \beta|1\rangle$$

According to quantum computing we can also write in qubit form of above equation as α|1> +β|0>. So the truth table of quantum NOT gate will be like as:

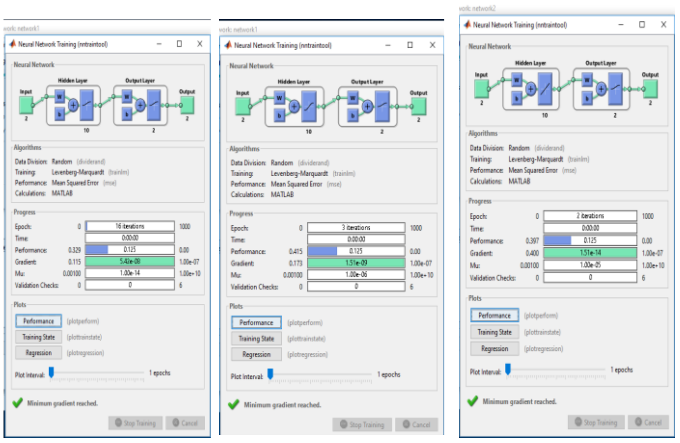
**Table I. Quantum NOT gate input and output.**

| Input | Output |
|---|---|
| \|0> | \|1> |
| \|1> | \|0> |
| α\|0> +β\|1> | α\|1> +β\|0> |

The basic must condition for the quantum gate that the result of input will be unitary to verify the condition of Unitary; we know that the resultant output of NOT gate output $= \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$ is unitary. When we train the neural network for quantum NOT gate using MATLAB we use the following training parameter to train the network and get below output, the output is compared using different type activation function like sigmoid, tangent, purelin. The input and target data are Input Data $=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$, Target Data = $\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$.

**Table II. NOT gate performance using different types of activation function.**

| Activation Function | Iteration | Performance | Gradient |
|---|---|---|---|
| LOGSIG | 16 | 0.329 | 5.42e-08/ (0.115) |
| PURELIN | 2 | 0.397 | 1.51e-14/ (0.400) |
| TRANSIG | 3 | 0.415 | 1.51e-09/ (0.173) |



Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 16 iterations | 1000
Time: | 0:00:00 |
Performance: 0.329 | 0.125 | 0.00
Gradient: 0.115 | 5.42e-08 | 1.00e-07
Mu: 0.00100 | 1.00e-14 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Minimum gradient reached.

work: network1
Neural Network Training (nntraintool)
Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 3 iterations | 1000
Time: | 0:00:00 |
Performance: 0.415 | 0.125 | 0.00
Gradient: 0.173 | 1.51e-09 | 1.00e-07
Mu: 0.00100 | 1.00e-06 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Minimum gradient reached.

work: network2
Neural Network Training (nntraintool)
Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 2 iterations | 1000
Time: | 0:00:00 |
Performance: 0.397 | 0.125 | 0.00
Gradient: 0.400 | 1.51e-14 | 1.00e-07
Mu: 0.00100 | 1.00e-05 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Minimum gradient reached.

**Fig 2. Implementation quantum NOT gate using neural network**

Quantum NOT gate is performing better performance by using transig activation function but to reduce the cost and time of any quantum computer we can use purelin function as activation because it takes number of low iterations compare to transig function.

## B Quantum Z Gate

Z gate is the single qubit gate, which takes the qubit |0> as input and generate the output |0> or if takes input as |1> qubit then produce the result as -|0>. We can describe the quantum Z gate by using following diagram:

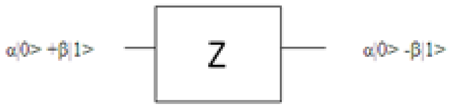


**Fig. 3. Quantum Z gate**

The total output of quantum Z gate we will be calculate the value of input qubit into matrix form, then

= |0> <0| + |1> <-1|

$$=\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix}+\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0 & -1\end{bmatrix}$$

$$=\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}+\begin{bmatrix}0 & 0\\0 & -1\end{bmatrix}$$

$$=\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$$

Step 1. When the Input as |0>

$$Z|0> = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$$

$$= \begin{bmatrix}1\\0\end{bmatrix} = |0>$$

Step 2. When the Input as |1>

$$Z|0> = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\-1\end{bmatrix} = -|1>$$

Step 3. When the Input as α|0> +β|1>

$$Z\ [\alpha|0> + \beta|1>] = \alpha\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\-1\end{bmatrix} = \alpha|0> -\beta|1>$$

Proof of reversible:

Step 1. When the output as |0>

$$Z|0> = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0>$$

Step 2. When the Input as -|1>

$$Z|-1> = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1>$$

Step 3. When the Input as α|0> -β|1>

$$Z\ [\alpha|0> -\beta|1>] = \alpha\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} - \beta\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \alpha\begin{bmatrix}1\\0\end{bmatrix} - \beta\begin{bmatrix}0\\-1\end{bmatrix} = \alpha|0> +\beta|1>$$

So, it is mathematically proved that quantum Z gate is also reversible. According to quantum computing we can also write in qubit form of above equation as α|1> +β|0>. So, the truth table of quantum Z gate will be like as:

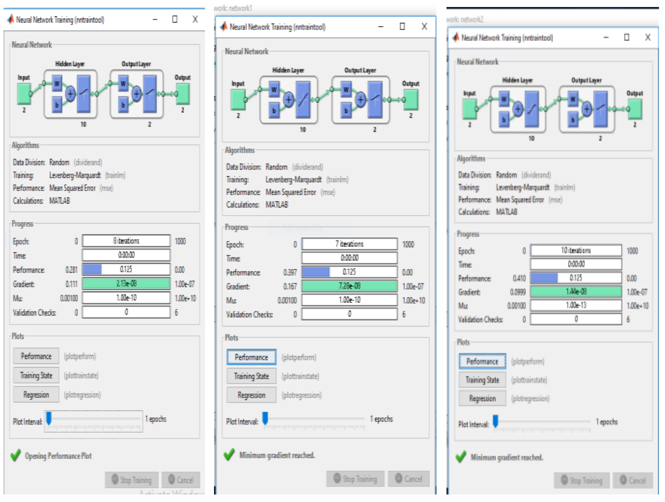
**Table III. Quantum Z Gate Input and Output.**

| Input | Output |
|---|---|
| \|0> | \|0> |
| \|1> | -\|1> |
| α\|0> +β\|1> | α\|0> -β\|1> |

The basic must condition for the quantum gate that the result of input will be unitary to verify the condition of Unitary; we know that the resultant output of Z gate output $= \begin{bmatrix}1 & 1\\0 & -1\end{bmatrix}$ is unitary. When we train the neural network for quantum Z gate using MATLAB we use the following training parameter to train the network and get below output, the output is compared using different type activation function like sigmoid, tangent, purelin . The input and target data are Input Data $=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$, Target Data $= \begin{bmatrix}1 & 1\\0 & -1\end{bmatrix}$

**Table IV. Quantum Z gate performance using different types of activation function.**

| Activation Function | Iteration | Performance | Gradient |
|---|---|---|---|
| LOGSIG | 8 | 0.281 | 2.13e-08/ (0.111) |
| PURELIN | 7 | 0.397 | 7.28e-08/ (0.167) |
| TRANSIG | 10 | 0.410 | 1.44e-08/ (0.0999) |



Neural Network Training (nntraintool)

Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 8 iterations | 1000
Time: 0:00:00
Performance: 0.281 | 0.125 | 0.00
Gradient: 0.111 | 2.13e-08 | 1.00e-07
Mu: 0.00100 | 1.00e-10 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Opening Performance Plot

work: network1
Neural Network Training (nntraintool)

Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 7 iterations | 1000
Time: 0:00:00
Performance: 0.397 | 0.125 | 0.00
Gradient: 0.167 | 7.28e-08 | 1.00e-07
Mu: 0.00100 | 1.00e-10 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Minimum gradient reached.

work: network2
Neural Network Training (nntraintool)

Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress
Epoch: 0 | 10 iterations | 1000
Time: 0:00:00
Performance: 0.410 | 0.125 | 0.00
Gradient: 0.0999 | 1.44e-08 | 1.00e-07
Mu: 0.00100 | 1.00e-13 | 1.00e+10
Validation Checks: 0 | 0 | 6

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs

Minimum gradient reached.

**Fig4.Implementation quantum Z gate using neural network**

Quantum Z gate is performing better performance by using transig activation function but the to reduce the cost and time of any quantum computer we can use purelin function as activation because it takes number of low iterations compare to transig function.

## C. Hadamard Gate

Hadamard gate is the single qubit gate, which takes the qubit |0> as input and generate the output $\frac{|0>+|1>}{\sqrt{2}}$ or if takes input as |1> qubit then produce the result as $\frac{|0>-|1>}{\sqrt{2}}$. We can describe the quantum Hadamard gate by using following diagram:

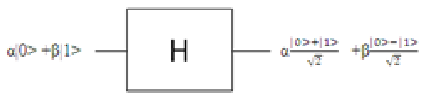


**Fig. 5: Hadamard gate**

The total output of quantum Hadamard gate we will be calculating the value of input qubit into matrix form, then

$$= |0> \frac{|0>+|1>}{\sqrt{2}} + |1> \frac{|0>-|1>}{\sqrt{2}}$$

$$= \frac{1}{\sqrt{2}} \left[ |0><0| + |0><1| + |1><0| - |1><1| \right]$$

$$= \frac{1}{\sqrt{2}} \left[ \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} - \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} \right]$$

$$= \frac{1}{\sqrt{2}} \left[ \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 1\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\1 & 0\end{bmatrix} - \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} \right] = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

Step 1. When we have input: |0>

$$H|0> = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix}1\\1\end{bmatrix} = \frac{1}{\sqrt{2}} \left[ \begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} \right]$$

$$= \frac{1}{\sqrt{2}} (|0> + |1>)$$

Step 2. When we have input: |1>

$$H|1> = \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix}1\\-1\end{bmatrix} = \frac{1}{\sqrt{2}} \left[ \begin{bmatrix}1\\0\end{bmatrix} - \begin{bmatrix}0\\1\end{bmatrix} \right] = \frac{1}{\sqrt{2}} (|0> - |1>)$$

Step 3. When we have input: α|0> +β|1>

$$= H (\alpha|0> + \beta|1>)$$

$$= \alpha H|0> + \beta H|1> = \alpha \frac{|0>+|1>}{\sqrt{2}} + \beta \frac{|0>-|1>}{\sqrt{2}}$$

Proof of reversible:

Step 1. When we have input: $\frac{1}{\sqrt{2}}(|0> + |1>)$

$$= H\left[\frac{1}{\sqrt{2}}(|0> + |1>)\right]$$

$$= \frac{1}{\sqrt{2}} \left[ \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \right]$$

$$= \frac{1}{\sqrt{2}} \left[ \frac{1}{\sqrt{2}} \begin{bmatrix}1\\1\end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix}1\\-1\end{bmatrix} \right]$$

$$= \frac{1}{2} \left[ \begin{bmatrix}2\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0>$$

Step 2. When we have input: $\frac{1}{\sqrt{2}}(|0> - |1>)$

$$H\left[\frac{1}{\sqrt{2}}(|0> - |1>)\right]$$

$$= \frac{1}{\sqrt{2}} \left[ \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} - \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \right]$$

$$= \frac{1}{\sqrt{2}} \left[ \frac{1}{\sqrt{2}} \begin{bmatrix}1\\1\end{bmatrix} - \frac{1}{\sqrt{2}} \begin{bmatrix}1\\-1\end{bmatrix} \right]$$

$$= \frac{1}{2} \left[ \begin{bmatrix}0\\2\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1>$$

Step 3. When we have input: $\alpha \frac{|0>+|1>}{\sqrt{2}} + \beta \frac{|0>-|1>}{\sqrt{2}}$

$$= H \left(\alpha \frac{|0>+|1>}{\sqrt{2}} + \beta \frac{|0>-|1>}{\sqrt{2}}\right)$$

$$= H \left(\alpha \frac{|0>+|1>}{\sqrt{2}}\right) + H \left(\beta \frac{|0>-|1>}{\sqrt{2}}\right)$$

$$= \alpha \frac{|0>+|1>}{\sqrt{2}} + \beta \frac{|0>-|1>}{\sqrt{2}}$$

= α|0> +β|1> (From the Step 1 and Step 2)

According to quantum computing we can also write in qubit form of above equation as α|0> +β|1>. So, the truth table of quantum Hadamard gate will be like as:

**Table V. Hadamard gate input and output.**

| Input | Output |
|---|---|
| \|0> | $\frac{\|0>+\|1>}{\sqrt{2}}$ |
| \|1> | $\frac{\|0>-\|1>}{\sqrt{2}}$ |
| α\|0>+β\|1> | $\alpha \frac{\|0>+\|1>}{\sqrt{2}} + \beta \frac{\|0>-\|1>}{\sqrt{2}}$ |

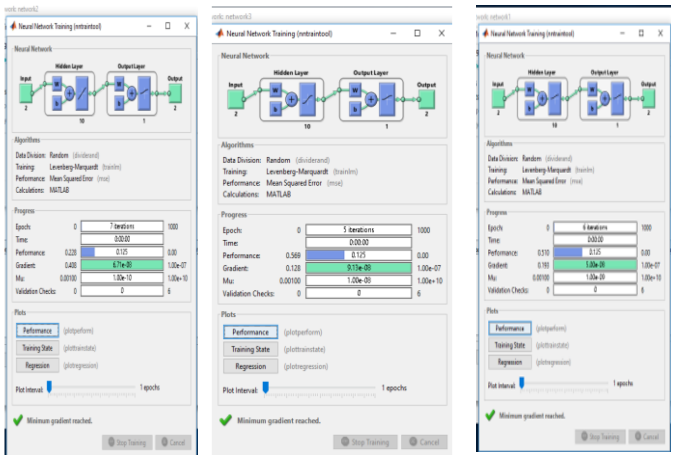
The basic must condition for the quantum gate that the result of input will be unitary to verify the condition of Unitary; we know that the resultant output of Hadamard gate outputt $\frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$ is unitary.

When we train the neural network for quantum Hadamard gate using MATLAB we use the following training parameter to train the network and get below output, the output is compared using different type activation function like sigmoid, tangent, purelin.
The input and target data are:

Input Data $= \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$, Target Data $= \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$

**Table VI. Hadamard gate performance using different types of activation function.**

| Activation Function | Iteration | Performance | Gradient |
|---|---|---|---|
| LOGSIG | 6 | 0.510 | 5.00e-08 / (0.193) |
| PURELIN | 7 | 0.228 | 6.71e-08 / (0.408) |
| TRANSIG | 5 | 0.569 | 9.13e-08/ (0.128) |


```
work: network2
Neural Network Training (nntraintool)
Neural Network
Algorithms
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MATLAB
Progress
Epoch:            0      7 iterations   1000
Time:                    0:00:00
Performance:      0.228  0.125          0.00
Gradient:         0.408  6.71e-08       1.00e-07
Mu:               0.00100  1.00e-10     1.00e+10
Validation Checks: 0     0              6
Plot Interval:    1 epochs
Minimum gradient reached.

work: network3
Neural Network Training (nntraintool)
Algorithms
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MATLAB
Progress
Epoch:            0      5 iterations   1000
Time:                    0:00:00
Performance:      0.569  0.125          0.00
Gradient:         0.128  9.13e-08       1.00e-07
Mu:               0.00100  1.00e-08     1.00e+10
Validation Checks: 0     0              6
Plot Interval:    1 epochs
Minimum gradient reached.

work: network1
Neural Network Training (nntraintool)
Algorithms
Data Division:  Random  (dividerand)
Training:       Levenberg-Marquardt  (trainlm)
Performance:    Mean Squared Error  (mse)
Calculations:   MATLAB
Progress
Epoch:            0      6 iterations   1000
Time:                    0:00:00
Performance:      0.510  0.125          0.00
Gradient:         0.193  5.00e-08       1.00e-07
Mu:               0.00100  1.00e-09     1.00e+10
Validation Checks: 0     0              6
Plot Interval:    1 epochs
Minimum gradient reached.
```
Input
Hidden Layer
Output Layer
Output
W
b
10
1
2
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Stop Training
Cancel

**Fig 6. Implementation Hadamard gate using neural network**

After analyzing the above table, we find the quantum Hadamard gate is performing better performance by using transig activation function.

## IV. MULTI QUBIT GATES

### A. CNOT Gate

CNOT gate is the multi qubit gate, which takes the qubit |00> as input and generate the output |00> or if takes input as |01> qubit then produce the result as |01> or if takes input as |10> qubit then produce the result as |11> or if takes input as |11> qubit then produce the result as |10>. We can describe the quantum CNOT gate by using following diagram:

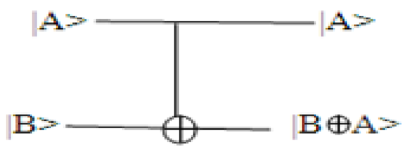


Fig.7. CNOT gate

The total output of quantum CNOT gate we will be calculating the value of input qubit into matrix form, then

$$|00> = |0> |0> = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} |01> = |0> |1> = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} |10> = |1> |0> = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} |11> = |1> |1> = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

$$\text{Because } |11> = \begin{bmatrix}0\\1\end{bmatrix} \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\begin{bmatrix}0\\1\end{bmatrix}\\1\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

CNOT= |00> <00| + |01> <01| + |10> <11| + |11> <10|

$$= \begin{bmatrix}1\\0\\0\\0\end{bmatrix}\begin{bmatrix}1&0&0&0\end{bmatrix} + \begin{bmatrix}0\\1\\0\\0\end{bmatrix}\begin{bmatrix}0&1&0&0\end{bmatrix} + \begin{bmatrix}0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0&0&1&0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\begin{bmatrix}0&0&0&1\end{bmatrix}$$

$$= \begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&1&0\end{bmatrix}$$

$$= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

Step 1. When the input is |00>

$$C|00> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = |00>$$

Step 2. When the input is |01>

$$C|01> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |01>$$

Step 3. When the input is |10>

$$C|10> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = |11>$$

Step 4. When the input is |11>

$$C|11> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} = |10>$$

Proof of reversible:

Step1. When the input is |00>

$$C|00> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

$$= \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = |00>$$

Step 2. When the input is |01>

$$C|01> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} = |01>$$

Step 3. When the input is |11>

$$C|10> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} = |10>$$

Step 4. When the input is |10>

$$C|11> = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = |11>$$

**Table VII. Quantum CNOT gate input and output.**

| Input | Output |
|---|---|
| \|00> | \|00> |
| \|01> | \|01> |
| \|10> | \|11> |
| \|11> | \|10> |

The basic must condition for the quantum gate that the result of input will be unitary to verify the condition of Unitary; we know that the resultant output of CNOT gate output

$$= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix} \text{ is unitary.}$$

When we train the neural network for quantum CNOT gate using MATLAB we use the following training parameter to train the network and get below output, the output is compared using different type activation function like sigmoid, tangent, purelin.

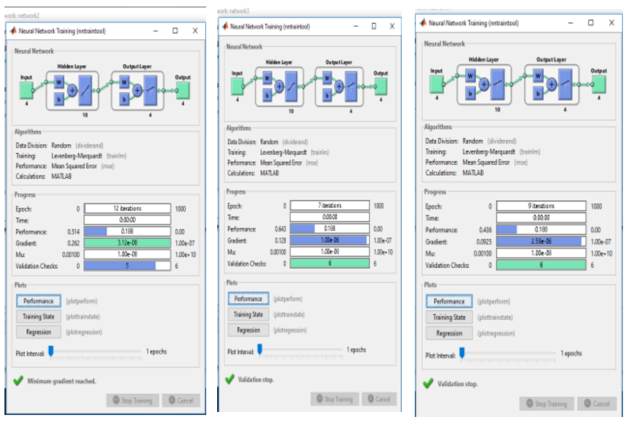
The input and target data are Input Data $= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$,

Target Data $= \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$

**Table VIII. CNOT gate performance using different types of activation function.**

| Activation Function | Iteration | Performance | Gradient |
|---|---|---|---|
| LOGSIG | 9 | 0.436 | 2.58e-06/ (0.0925) |
| PURELIN | 12 | 0.514 | 3.12e-08/ (0.262) |
| TRANSIG | 7 | 0.643 | 1.00e-06/ (0.00100) |

Quantum CNOT gate is performing better performance by using transig activation function.



Neural Network Training (nntraintool)

Algorithms
Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MATLAB

Progress

| | | | |
|---|---|---|---|
| Epoch: | 0 | 12 iterations | 1000 |
| Time: | | 0:00:00 | |
| Performance: | 0.514 | 0.188 | 0.00 |
| Gradient: | 0.262 | 3.12e-08 | 1.00e-07 |
| Mu: | 0.00100 | 1.00e-08 | 1.00e+10 |
| Validation Checks: | 0 | 5 | 6 |

Plots: Performance (plotperform), Training State (plottrainstate), Regression (plotregression)
Plot Interval: 1 epochs
Minimum gradient reached.

Progress

| | | | |
|---|---|---|---|
| Epoch: | 0 | 7 iterations | 1000 |
| Time: | | 0:00:00 | |
| Performance: | 0.643 | 0.188 | 0.00 |
| Gradient: | 0.128 | 1.00e-06 | 1.00e-07 |
| Mu: | 0.00100 | 1.00e-09 | 1.00e+10 |
| Validation Checks: | 0 | 6 | 6 |

Plot Interval: 1 epochs
Validation stop.

Progress

| | | | |
|---|---|---|---|
| Epoch: | 0 | 9 iterations | 1000 |
| Time: | | 0:00:00 | |
| Performance: | 0.436 | 0.188 | 0.00 |
| Gradient: | 0.0925 | 2.58e-06 | 1.00e-07 |
| Mu: | 0.00100 | 1.00e-08 | 1.00e+10 |
| Validation Checks: | 0 | 6 | 6 |

Plot Interval: 1 epochs
Validation stop.

**Fig 8. Implementation CNOT gate using neural network**

## V. METHODOLOGY

To implement different types of single and multi-qubits gates using MATLAB we convert any qubit into matrix form. Each input and output qubits were changed into matrix form using 0 and 1. The selection of input & output we have select layer and iteration in to run neural network. Each of gate was implement using different types of activation function & iteration.

## VI. RESULT

After implementation of different types of single and multi-qubits gates, TRANSIG function giving effective performance. According to following result in the development of quantum-based devices may be choose.

**Table IX. Result of different gates using TRANSIG function**

| Gate | Performance | Gradient |
|---|---|---|
| NOT | 0.415 | 1.51e-09/ (0.173) |
| Z | 0.41 | 1.44e-08/ (0.0999) |
| HADAMARD | 0.569 | 9.13e-08 / (0.128) |
| CNOT | 0.643 | 1.00e-06/ (0.00100) |

## VII. CONCLUSIONS

In this paper we describe the mathematically of different types quantum gates and mathematically proved their reversibility property of quantum logic gates. The mathematically description are useful for new research on quantum computer or model.

In this paper also analyzed the performance of different quantum logic gates using different types of activation function in neural network. All basic qubit operation using matrix also described in this paper. This paper also describes the better performance of the function when we use in MATLAB.

## REFERENCES

1. P. A Benioff, "Quantum Mechanical Hamiltonian Model of Turing Machine", J Stat Phy, r., vol .29(3). pp 515-546, 1982.
2. R.P Feynman, "Simulating physics with computers", Inr. J. of Theo. Physics, vol. 21(6/7), pp- 467-488,1982.
3. D.Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", Pmc. of Roy.Soc of Londonser A, vol400, pp97-117, 1985.
4. Mikio Nakahara and Testsuo ohmi, "Quantum Computing", CRC Press, 2008.
5. Nielsen, M.A. amd Chuang, I.L., "Quantum Computation and quantum Information", UK: Cambridge University Press, 2000.
6. Sahni, V,"Quantum Computing", New Delhi:tata McGraw Hill,2007.
7. M.Morris Mano, Digital Design Prentice Hall, Third Edition, 2002.
8. Sitakanta Nayak, J.P Singh,"Quantum Comcepts in Neural Computation",Proceeding of the International Conference on soft Computing for Problem Solving, 2011, Advance in intelligent and soft Computing, Vol. 130, pp 395-400, Springer India.
9. Sitakanta Nayak, Shaktikanta Nayak,"A Study on Quantum Inspired Hybrid Neural Networks Model", International Journal of Advanced Research in Computer Science and Software Engineering, Vol-3, No-06, June-2013.
10. Sitakanta Nayak, Shaktikanta Nayak,"An Introduction to Basic Logic Gates for Quantum computer," International Journal of Advanced Research in Computer Science and Software Engineering, Vol-3, Issue 10, October-2013

## AUTHORS PROFILE

**Mr. Nitesh Kumar,** completed his M.Tech (Computer Science & Engineering ) from Rajasthan Technical Universsity, kota in 2017 . Currently, pursuing PhD in Computer Science & Enginnering from Raffles University, Neemrana . His reseacrh Interests are Quantum neural network and Digital Image Processing .

**Dr Neeraj Sahu,** obtained his M. Sc (1999), M. Phil (2008) from Bundelkhand University Jhansi (U.P) and Ph.D (2014) in Mathematics from Jiwaji University Gwalior (M.P), currently working as Assistant Professor in Raffles University. His research interests are Recurrent Neural Network, Quantum Neural Network and Neural Operation Research**.**