IMPLEMENTATION OF QUANTUM GATE OPERATIONS USING NEURAL NETWORKS

A Thesis by

Amir Zabihi Shahri

Bachelor of Science, Khavaran University, 2012

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2017

IMPLEMENTATION OF QUANTUM GATE OPERATIONS USING NEURAL NETWORKS

The following faculty members have examined the final copy of this thesis for form and the content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with major in Electrical Engineering.

_____
Preethika Kumar, Committee Chair

_____
Ali Eslami, Committee Member

_____
Hamid Lankarani, Committee Member

DEDICATION

To

My parents and my brother

ACKNOWLEDGEMENTS

I would like to extend my sincere and heartfelt obligation towards all the personages who have helped me in this endeavor. Without their active guidance, help, cooperation and encouragement, I would not have made headway in this thesis.

First and foremost, I would like to thank Almighty God for giving me the strength to complete my research in time.

I am very thankful and pay my gratitude to my adviser Dr. Preethika Kumar for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my M.Sc. degree.

I extend my gratitude to Wichita State University for giving me this opportunity to be part of their valuable academic community. I also would like to thank my committee members, Dr. Hamid Lankarani and Dr. Ali Eslami for their precious support, help and encouragement in channeling my hard work in the right path to accomplish this research.

Last, but not the least, I acknowledge with a deep sense of reverence my appreciation towards my parents and my brother, who have always supported me morally as well as economically thorough this journey.

ABSTRACT

All quantum circuits are designed using different quantum gates, which can be decomposed into elementary gates. The number of elementary gates in a quantum circuit is called the gate count. Typically, there is a direct correspondence between the gate count and the complexity of a quantum circuit. Quantum systems are generally very fragile, and a quantum bit (qubit) can lose its super-position state very easily. This process is called decoherence. As such, when implementing a quantum operation, it becomes necessary to minimize the gate count as much as possible.

This thesis focuses on two important applications where reducing the gate count is very significant. The first is quantum error correction (QEC). In a quantum error correction code (QECC), one information qubit is encoded with two or more auxiliary qubits to form a logical/encoded qubit. Oftentimes, when performing gate operations on logical qubits, decoding is required, which opens the system to decoherence. The aim here is to design encoded quantum gates in such a way that the decoding process is no longer needed for implementing gate operations on QEC circuits. The second focus is a controlled-NOT (CNOT) gate operation between uncoupled (remote) qubits. In order to implement a gate operation in a linear nearest neighbor (LNN) architecture, the qubits that are not neighbors need to be brought adjacent to each other before a gate operation can be performed between them. The LNN architecture is significant because most physical implementations of a practical quantum computer use this layout. Here, the aim is to implement a CNOT gate between remote qubits in an LNN architecture without bringing the qubits adjacent to each other, thereby tremendously reducing the gate count. This research employed a neural network approach based on gradient descent technique to reduce the gate count.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure Page

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF ABBREVIATIONS

CNOT	Controlled Not
LNN	Linear Nearest Neighbor
NN	Nearest Neighbor
ODE	Ordinary Differential Equation
QEC	Quantum Error Correction
QECC	Quantum Error Correction Code
RMS	Root Mean Square

CHAPTER 1

MOTIVATION

1.1 Introduction

If today computer theories are based on conventional physics, quantum mechanics has all the fundamentals for making quantum computers. In 1985, Deutsch developed the theory of quantum computers [1], which was based on the work of Feynman [2] and other scientists. Unlike conventional (digital) computers, where data is encoded into binary codes with either 0 or 1 value, quantum computers use qubits. Qubits or quantum bits are fundamental structures which include all quantum features. Unlike bits, qubits possess the superposition property that allow them to store both 0 and 1 values at the same time. However, upon measurement, a qubit behaves just like a classical bit and collapses to either 1 or 0. Thus, measurement alters the state of a qubit. As such, the general state of a qubit is modelled by a unit vector in a 2-dimensional complex vector space:

$$|\varphi> = \alpha|0> + \beta|1> \tag{1.1}$$

where

$$|0> = \begin{bmatrix}1\\0\end{bmatrix}; |1> = \begin{bmatrix}0\\1\end{bmatrix} \tag{1.2}$$

The terms “" α " and " β " are called probability amplitudes, and $|\alpha|^2$ and $|\beta|^2$ correspond to probabilities finding a qubit to be in the |0> and |1> state, respectively. Thus, we have

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.3}$$

In general, a quantum computer with N qubits can simultaneously exist in a superposition of 2^N states. For instance a system with only two qubits can hold four values (00, 01, 10 and 11) simultaneously.

This ability to exist in superposition states gives quantum computers the advantage over digital computers (classical computers), since it allows them to efficiently solve certain problems like factoring, which are very hard to solve on a classical computer [3].

1.2 Quantum Gate Operations

The circuit model of a classical computer is very useful for computing processes and is usually used to design and construct computing hardware. In the circuit model, computer scientists use different types of Boolean logic gates acting on some binary input in order to solve various problems. Analogous to the circuit model of logic gates that are used in classical computers, in quantum computers, we have a quantum circuit model comprising quantum gates [4].

In quantum computers, there are two different types of gate operations: single qubit and multi-qubit gate operations. Single qubit operations operate only on one qubit. Multi-qubit gate operations operate on two or more than two qubits. Section 1.3 and 1.4 presents some examples of single and multi-qubit gates.

1.3 Single Qubit Gate Operations

NOT Gate or X Gate (Pauli-X)

A NOT gate in quantum computing operates very similar to the NOT gate in classical computers. The quantum NOT gate flips the state of a qubit from $|0>$ to $|1>$, and vice versa. The matrix representation for the NOT gate is:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{1.4}$$

Alternatively, a NOT gate operation in a qubit state, |q>, can be described as:

$$X|q> = |q'> \tag{1.5}$$

Figure 1.1 shows the circuit diagram for the NOT gate, which changes the state of the qubit from $\alpha|0> + \beta|1>$ to $\alpha|1> + \beta|0>$.

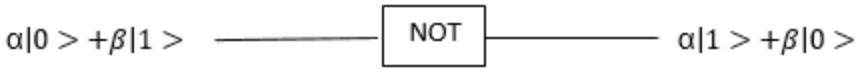


Figure 1.1. NOT gate operation on a qubit.

Phase-Flip or Z Gate (Pauli-Z)

A Z-gate in quantum computing has no classical analogue. This gate applies a phase of +180^o when the qubit is in the |1> state. When the qubit is in the |0> state, then it does not change its phase. The matrix representation of the Z gate is:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.6}$$

Figure 1.2 shows the circuit diagram for the Z gate, which changes the state of the qubit from $\alpha|0> +\beta|1>$ to $\alpha|0> -\beta|1>$.

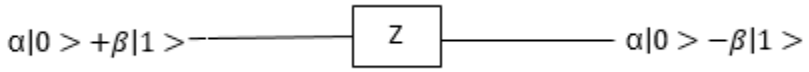


Figure 1.2. Z gate operation on a qubit.

Y Gate (Pauli-Y)

The Y gate has no classical analogue. This gate acts on a single qubit, and maps the |0> state to i|1> and |1> to - i|0>. The matrix representation for the Y gate is:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{1.7}$$

Figure 1.3 shows that the circuit diagram for the Y gate, which changes the state of the qubit from $\alpha|0> +\beta|1>$ to $\alpha i|1> -\beta i|0>$.

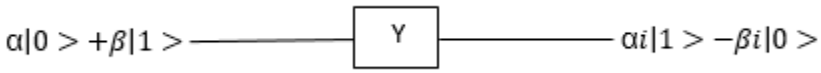


Figure 1.3.Y gate operation on a qubit.

Hadamard or H Gate

The Hadamard gate is one of the most important gates in quantum computing. This gate has no classical analogue, and is used for creating superposition states.

The gate operation can be described as:

$$H|q>=\begin{cases}\frac{|0>+|1>}{\sqrt{2}}, & \text{if } |q>=|0> \\ \frac{|0>-|1>}{\sqrt{2}}, & \text{if } |q>=|1>\end{cases} \tag{1.8}$$

Figure 1.4. H gate operation on a qubit.

The matrix representation for Hadamard gate is:

$$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \tag{1.9}$$

1.4 Multi Qubit Gate Operations

Controlled-NOT (CNOT) Gate

The controlled-NOT (CNOT) gate is an example of a two-qubit controlled gate where one qubit acts as the control and the other qubit acts as the target. Figure 1.5 shows the circuit representation of the CNOT gate. Here $|q_1>$ is the control, and $|q_2>$ is the target. The basic operation of a CNOT gate is to flip the state of the target qubit, when the control is |1>. If the state of the control is |0>, no operation is performed on the target.

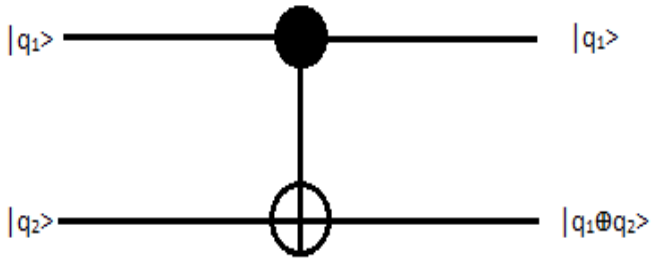


Figure 1.5. Controlled-NOT (CNOT) gate on a two-qubit gate operation.

The CNOT gate operation can be described as:

$$q_1 q_2 > \rightarrow \begin{cases} |q_1 q_2 >, & \text{if } |q_1 > = |0 > \\ |q_1 q_2{}' >, & \text{if } |q_1 > = |1 > \end{cases} \tag{1.10}$$

The matrix representation of the CNOT gate is:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.11}$$

Table 1.1 shows the state table for the CNOT gate which is identical to performing a classical exclusive-OR (XOR) gate on the target qubit. The order of the qubit states are $|q_1 q_2 >$.

TABLE 1.1

STATE TABLE FOR CNOT GATE OPERATION

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

Controlled-Z gate

A controlled-Z gate is a two qubit gate that applies a Z gate on the target qubit only when the control qubit is in the |1> state. Figure 1.6 shows the circuit diagram where $|q_1 >$ is the control and $|q_2 >$ is the target.

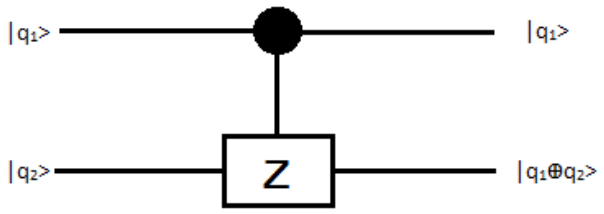


Figure1.6. Controlled-Z gate on a two-qubit gate operation.

Table 1.2 shows the state table. As can be seen from Table 1.2, under the gate operation a 180^o phase is applied when both qubits are in the |1> state.

TABLE 1.2

STATE TABLE FOR CONTROLLED-Z GATE OPERATION

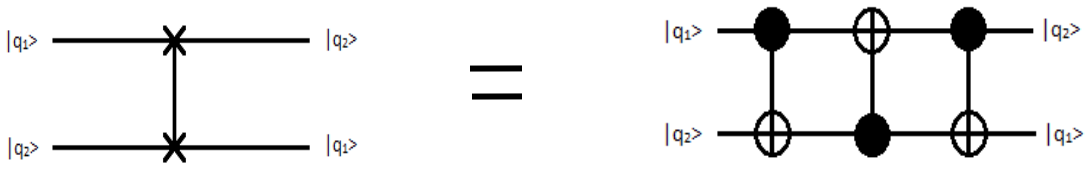
Input	Output
|00>	|00>
|01>	|01>
|10>	|10>
|11>	– |11>

Swap Gate

The Swap gate is a two-qubit gate operation, which swaps the states of the two-qubits involved in the gate operation. The operation can be expressed as:

$$|q_1 q_2> \rightarrow |q_2 q_1> \tag{1.12}$$

Figure 1.7 shows the circuit diagram of a Swap gate applied on a two-qubit system. A Swap gate can be decomposed into three CNOT gates as is shown in Figure 1.7. Table 1.3 shows the state table for a Swap gate.



(a) **(b)**

Figure 1.7. Swap gate applied on two-qubit system: (a) symbol for Swap gate; (b) three CNOT gates that comprise a Swap gate.

The matrix representation of the Swap gate is:

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.13}$$

TABLE 1.3

STATE TABLE OF SWAP GATE OPERATION ON A TWO QUBIT SYSTEM

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

Toffoli Gate

Like the CNOT, Controlled-Z, and Swap gates, a Toffoli gate is also a multi-qubit gate. However, it operates on a system of three qubits. The Toffoli gate uses two qubits as controls and one qubit as the target. Figure 1.8 shows the circuit representation of a Toffoli gate where $|q_1\rangle$ and $|q_2\rangle$ are the controls, and $|q_3\rangle$ is the target. Under this gate, the state of qubit $|q_3>$ is flipped only when both controls are in the $|1\rangle$ state. Table 1.4 shows the state table for a Toffoli gate.

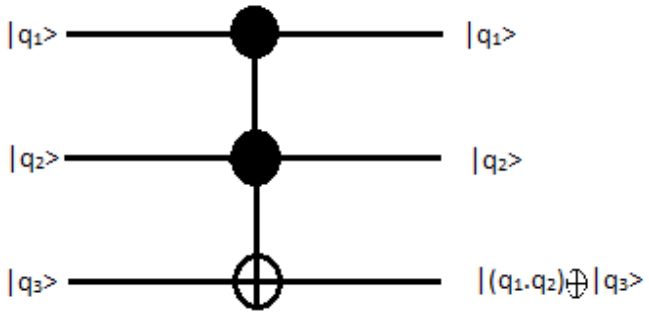


Figure 1.8. Toffoli gate operation on a multi-qubit quantum system.

The matrix representation of the Toffoli gate is described as:

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.14}$$

Fredkin Gate (Controlled-Swap gate)

The Fredkin gate is a multi-qubit gate that operates on a system of three qubits. It is a controlled-Swap gate. One of the qubits is a control. Figure 1.9 also shows the circuit representation of a Fredkin gate. When the control qubit ($|q_1\rangle$) is $|1\rangle$, the states of the other two qubits ($|q_2\rangle$ and $|q_3\rangle$) are swapped. Table 1.5 shows the state exchange on the Fredkin gate.

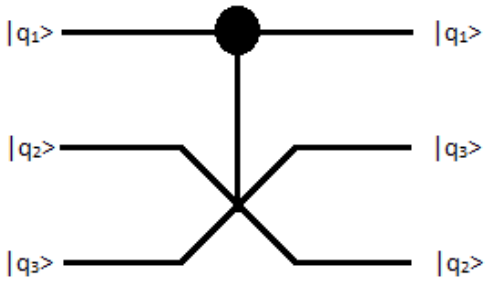


Figure 1.9. The Fredkin (controlled-Swap) gate operation on a multi-qubit quantum system.

The matrix representation of the Fredkin gate is described as:

$$\text{Fredkin} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.15}$$

TABLE 1.4

STATE TABLE OF A TOFFOLI GATE OPERATION ON A THREE QUBIT SYSTEM

Input	Output
|000>	|000>
|001>	|001>
|010>	|010>
|011>	|011>
|100>	|100>
|101>	|101>
|110>	|111>
|111>	|110>

TABLE 1.5

STATE TABLE OF A FREDKIN GATE OPERATION ON A THREE QUBIT SYSTEM

Input	Output
|000>	|000>
|001>	|001>
|010>	|010>
|011>	|011>
|100>	|100>
|101>	|110>
|110>	|101>
|111>	|111>

Universal Gates

A set of gates is called universal if any unitary operator can be approximated by a circuit using these gates to an arbitrary accuracy [5]**.** In other words, any multi-qubit arbitrary gate

operation can be constructed from a set of gates which is called a universal gate set. For example in [6], Barenco et al. showed that any two-qubit controlled-unitary operation can be achieved using a combination of single qubit gates and CNOT gates. In [7], Klappenecker showed that a controlled unitary operation can be realized with at most six elementary gates. Figure 1.10 shows a unitary operation constructed by single gates (E, A, B, C) and two CNOT gates.

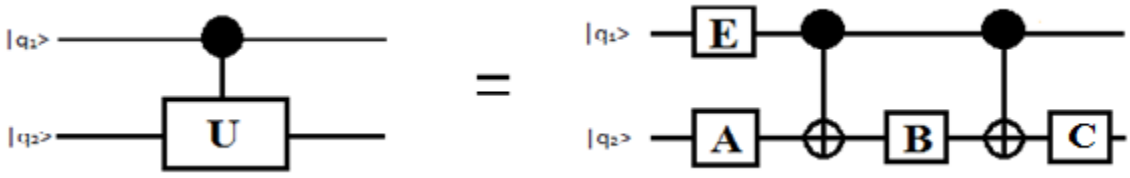


Figure 1.10. Unitary gate operation with controlled-NOT gate and four single gates.

Quantum gates S, H and CNOT are examples of a universal gate sets, the proof of which is described in [8] by Nielsen. Till date, much research has been done in order to decompose complex qubit operations into universal gate sets [9-16].

1.5 Quantum Error Correction

Quantum error correction codes (QECC) are used to protect quantum information against decoherence. Reducing corruption of quantum states by decoherence is one of the main challenges in the design of a practical quantum system. In circuits with large gate counts, qubits can lose their superposition states if gates are not implemented within certain time scales. For instance, a practical implementation of Shor's algorithm for factorization involves several gate operations. As such, all operations need to be performed within certain time limits in order to avoid decoherence [4]. In addition, the quantum system may get entangled with its environment and get corrupted. As such, quantum errors are continuous in nature, which makes them hard to protect information against. The most basic error correction circuit is the three-qubit code error correction code. The three qubit code encodes a single logical qubit into three physical qubit that can correct single σ_x bit-flip error [17]. Logical basis states can be described as:

$$|0>_L \rightarrow |000>, |1>_L \rightarrow |111> \quad (1.16)$$

As such, an arbitrary single qubit state is encoded as:

$$|\varphi> = \alpha|0> + \beta|1> \rightarrow |\varphi> L = \alpha|0>_L + \beta|1>_L = \alpha|000> + \beta|111> \quad (1.17)$$

In this code, an information qubit shows in (1.1), is encoded using two ancilla (qubits in |0> state) to create a logical qubit state $\alpha|000> + \beta|111>$. Here, the encoded states |000> and |111> are referred to as the "logical 0" or $|0>_L$, and "logical 1" or $|1>_L$ states, respectively. The encoding circuit to create these encoded states is shown in Figure 1.11.

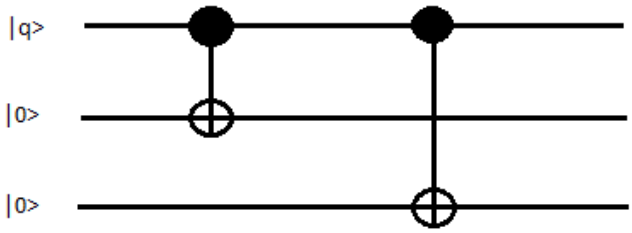


Figure 1.11. Encoded circuit diagram for three qubit quantum system.

Figure 1.12 shows the error correction circuit for this code. The circuit can correct only one single qubit bit-flip error. The two ancilla qubits check the parity between the 3 qubits forming the codeword, and in doing so they can detect the one bit-flip error. The error is then corrected by operators (NOT-gates) that are applied on the three-qubit error. Table 1.6 shows possible bit-flip errors that may occur, including the no error possibility. It also shows the state of the ancillas based on the parity of the three qubits (the "M" represents the measurement of ancilla qubits).

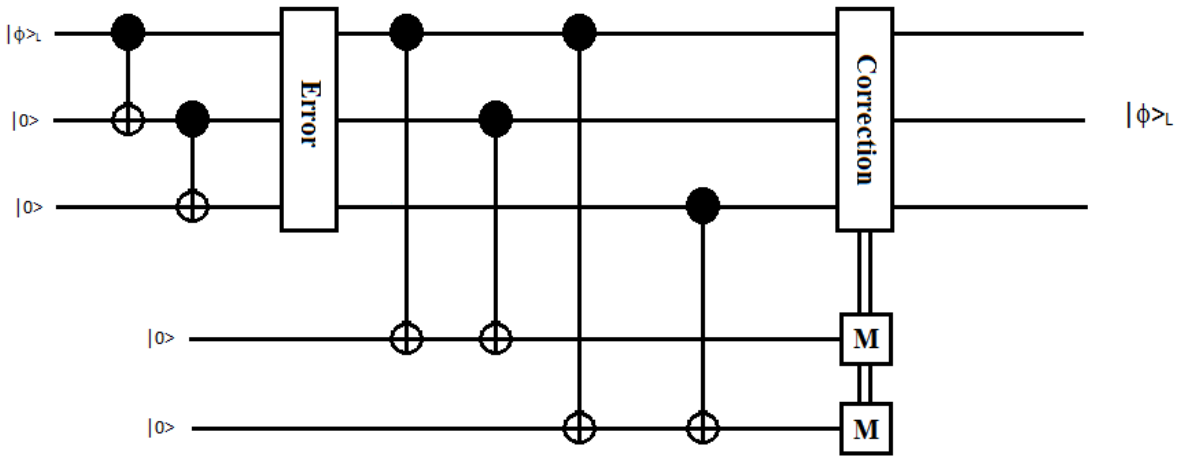


Figure 1.12. Three-qubit error code which is able to correct one single X error.

TABLE 1.6

STATE TABLE FOR THREE QUBIT ERROR CORRECTION

Error	**Final State, Data, Ancilla(Garbage)**
No Error	$\alpha\|000\rangle\|00\rangle+\beta\|111\rangle\|00\rangle$
Qubit 1	$\alpha\|100\rangle\|11\rangle+\beta\|011\rangle\|11\rangle$
Qubit 2	$\alpha\|010\rangle\|10\rangle+\beta\|101\rangle\|10\rangle$
Qubit 3	$\alpha\|001\rangle\|01\rangle+\beta\|110\rangle\|01\rangle$

As can be seen from Table 1.6, for each possible situation, either no error or single bit-flip error, the ancilla qubits are flipped to a unique state. These qubits are then measured to detect on which qubit the error occurs. The results of the measurement will then dictate if a correction gate needs to be applied, and if so, to which qubit. This is shown in Table 1.7. The three-qubit code can only correct single bit flip errors. To correct any random single qubit error, we need larger redundancy, that is, larger logical qubits, for instance, five-qubit, seventh-qubit Steane code [18] and 9-qubit Shor codes [19].

TABLE 1.7

ANCILLA MEASURMENT FOR BIT-FLIP ERROR CORRECTION

Ancilla Measurement	**Final State**	**Result**
00	$\alpha\|000\rangle + \beta\|111\rangle$	No Error
01	$\alpha\|001\rangle + \beta\|110\rangle$	X-Gate on Qubit 3
10	$\alpha\|010\rangle + \beta\|101\rangle$	X-gate on Qubit 2
11	$\alpha\|001\rangle + \beta\|110\rangle$	X-gate on Qubit 1

So far among all single qubit error correcting codes, the five-qubit code is the most efficient. This is because this code uses the entire 2^5-dimensional space used for encoding, and there is no wastage as will be explained.

We will now consider an N-qubit system where only single qubit errors (X_i, Y_i and Z_i) occur at a time. It has been shown that any random quantum error can be broken up into a linear combination of X, Y or Z errors). Since there are two codewords, $|0\rangle_L$ and $|1\rangle_L$, the encoding subspace is 2-dimensional (total dimension of vector space is 2^n). As there are"3n + 1" error conditions (3n for each of the X_i, Y_i and Z_i errors, plus one for no error (I)), and since each error subspace is 2-dimensional, for an error correcting code to be able to correct single-0qubit errors,

$$2^n \geq 2(1 + 3n) \tag{1.18}$$

For example, for a seven-qubit error correction code, the equation is $128 \geq 44$, meaning that although it will correct a single-qubit error, too much space (two dimensional vectors) will remain unused. However, for a five-qubit error correction code, which is the most efficient among all of them, the total number of available vectors is equal to the number of vectors needed to correct a single-qubit error ($32 \geq 32$).

Figure 1.13 shows the encoding circuit for five-qubit code which uses Hadamard, CNOT and multi-control Z gates [20].

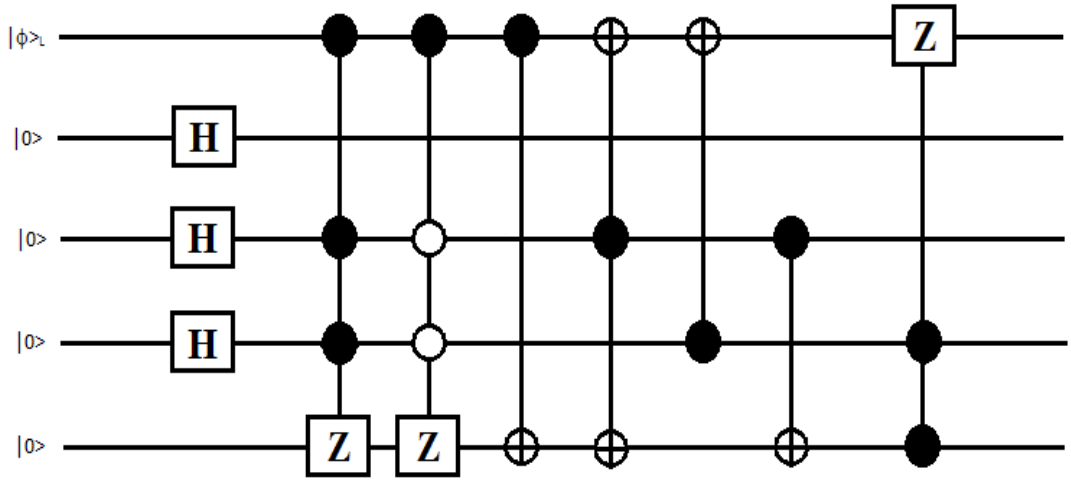


Figure 1.13. Five-qubit code error correction encoding circuit diagram.

Encoding in Five-qubit Code Error Correction

In order to perform encoding for a five-qubit error correction code, the system needs some measurement operators to identify and detect particular errors that occur in the system. Vectors corresponding to a single qubit error are $|0\rangle_L$, $|1\rangle_L$, $Xi|0\rangle_L$, $Xi|1\rangle_L$, $Zi|0\rangle_L$, $Zi|1\rangle_L$, $Yi|0\rangle_L$, $Yi|1\rangle_L$, where as before, $|0\rangle_L$ and $|1\rangle_L$ are codewords, and i = 1 through 5. Measurement operators are chosen such that these vectors are eigenvectors for the operators with eigenvalues ±1. For a five-qubit error correction code, the quantum system only needs four operators (stabilizers), as shown in equation (1.19).

$$M_0 = Z_1X_2X_3Z_4,\ M_1 = Z_2X_3X_4Z_0$$
$$M_2 = Z_3X_4X_0Z_1,\ M_3 = Z_4X_0X_1Z_2 \qquad (1.19)$$

where the X_i and Z_i are X and Z gate operators, respectively, on qubit "i".

An input state $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ is encoded into the five-qubit state $|\varphi\rangle=\alpha|0\rangle_L+\beta|1\rangle_L$. Here, the code words for the logical state basis $\{|0\rangle_L, |1\rangle_L\}$ are as follows [21]:

$$|0>_L = \frac{1}{(\sqrt{4})^2}(I+M_0)(I+M_1)(I+M_2)(I+M_3)|00000> = \frac{1}{4}[|00000> +|11000> +|01100> +|00110> +|00011> +|10001> -|10100> -|01010> -|00101> -|10010> -|01001> -|11110> -|01111> -|10111> -|11011> -|11101>] \quad (1.20)$$

and

$$|1>_L = \frac{1}{(\sqrt{4})^2}(I+M_0)(I+M_1)(I+M_2)(I+M_3)|11111> = \frac{1}{4}[|11111> +|00111> +|10011> +|11001> +|11100> +|01110> -|01011> -|10101> -|11010> -|01101> -|10110> -|00001> -|10000> -|01000> -|00100> -|00010>] \quad (1.21)$$

It is easy to see that these vectors are orthonormal; also they are simultaneous eigenvectors of $\{M_i\}$ with all the eigenvalues +1; $M_i|0>_L = |0>_L$ and $M_i|1>_L = |1>_L$ [22]. The measurement operators are also used for error detection. This is because each of the error states $Xi|0>_L$, $Xi|1>_L$, $Zi|0>_L$, $Zi|1>_L$, $Yi|0>_L$, $Yi|1>_L$, are either +1 or -1 eigen vectors of $\{M_i\}$. As such, each type of error corresponds to a unique combination or +1's and -1's, which can be used to detect the type of error, and also the qubit on which it occurs. This unique combination is called the syndrome, which is shown in Table 1.8. As can be seen, each error has its own unique pattern, so the system can recognize and correct all possible single-qubit errors. Figure 1.14 shows the five-qubit code's syndrome circuit (error detection).

TABLE 1.8

SYNDROME CODES FOR FIVE-QUBIT ERROR CORRECTION CODE

	X_0 Y_0 Z_0	X_1 Y_1 Z_1	X_2 Y_2 Z_2	X_3 Y_3 Z_3	X_4 Y_4 Z_4	I
$M_0=Z_1 X_2 X_3 Z_4$	+1 +1 +1	–1 –1 +1	+1 –1 –1	–1 –1 +1	–1 –1 +1	+1
$M_1=Z_2 X_3 X_4 Z_0$	–1 –1 +1	+1 +1 +1	–1 –1 +1	+1 –1 –1	+1 –1 –1	+1
$M_2=Z_3 X_4 X_0 Z_1$	+1 –1 –1	–1 –1 +1	+1 +1 +1	+1 –1 –1	+1 –1 –1	+1
$M_3=Z_4 X_0 X_1 Z_3$	+1 –1 –1	+1 –1 –1	–1 –1 +1	–1 –1 +1	–1 –1 +1	+1

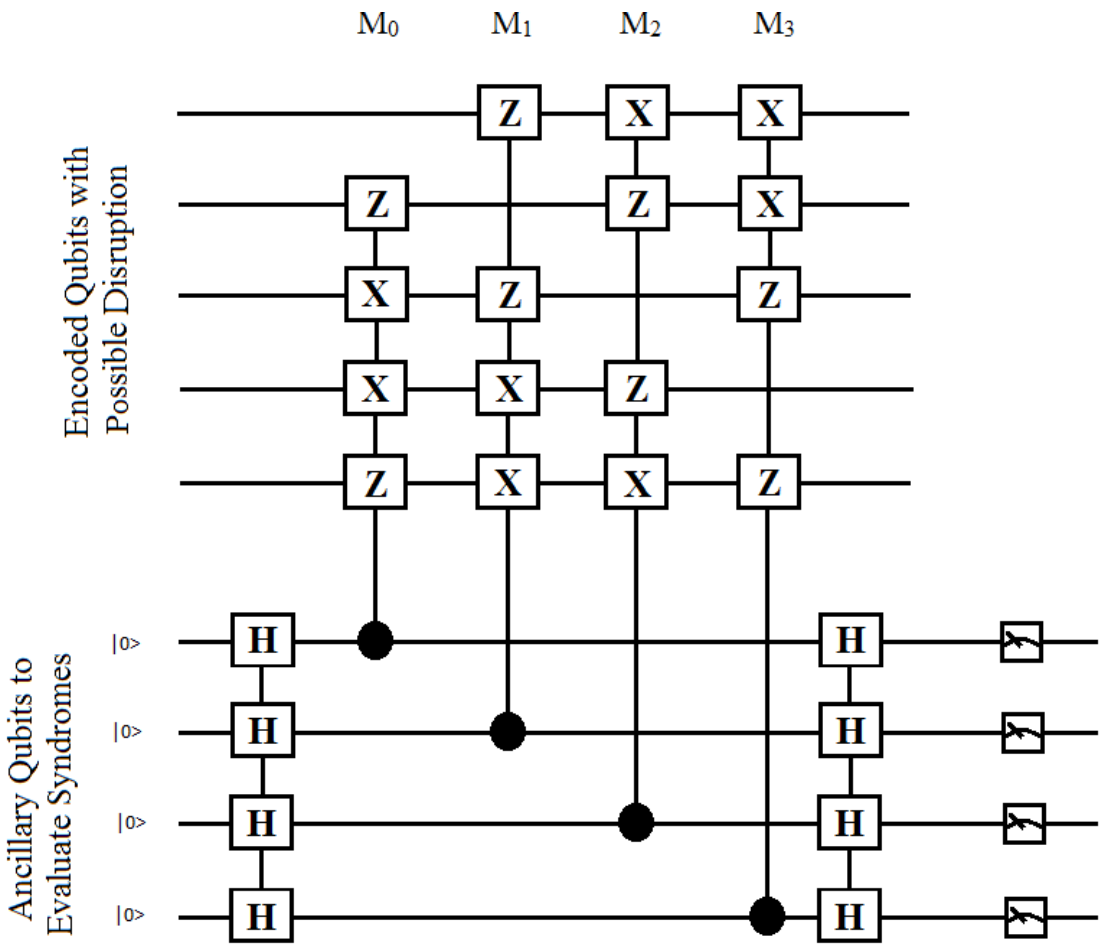


Figure 1.14. Five-qubit code error correction syndrome circuit diagram.

1.6 Goal of This Thesis

Since 1982 when Feynman [23] suggested the idea of quantum computers and their advantages over classical or universal computers, scientists in physics, computer science and engineering have been putting a great deal of effort into building a quantum computer. However, building a quantum computer is not easy, and like any other scientific work, scientists and engineers have to go through a lot of challenges to bring quantum computers from theory to reality. One of the most challenging areas which quantum computer researchers face is noise and decoherence. As explained in [24], decoherence is an important concern since in practicality a quantum computer can never be isolated from its environment. In other words, it will, in general,

become entangled with a large number of environmental degrees of freedom. This opens a quantum system up to noise, and in time, a quantum system can lose its superposition state. The problem of decoherence made researchers believe that building a large scale quantum computer was only a theoretical concept. However, in 1995, Shor showed that if physical qubits were encoded as logical qubit using a quantum error correction algorithm, quantum states protected against decoherence. A quantum error correction code uses extra ancillary/auxillary qubits (called ancillas) to form larger encoded qubits, which protect against certain types of quantum errors. However, even though quantum error correction codes offer the possibility to protect the fragility of quantum states, they come with their own set of challenges. The greater the number of qubits added to the system, the greater is the chance of the system interacting with its environment. Also, if an error occurs during the encoding process itself, and is not corrected, it will effects further steps of quantum computation. Moreover, often times, when performing gate operations on encoded qubits, they need decoding, which once again makes them susceptible to errors. In this research, a new method to implement gates without decoding is explored. We call such gates "encoded" gates because they operate directly on the encoded qubit. For instance, suppose we want to perform an H gate operation on a physical qubit, which has been encoded using the three-qubit code. Currently, to do the operation, we need to decode the qubit, perform the Hadamard gate, and encode the qubit back again. However, if a means exits to perform as "encoded-Hadamard gate" on the 3 qubit code, decoding is not required.

The encoded-Hadamard gate on the three-qubit system thus performs an operation that is equivalent to a Hadamard gate on the physical (decoded) qubit. Encoded gates are therefore helpful for creating fault-tolerant quantum circuits as they remove the decoding process. A fault-tolerant gate design can help us to reduce the number of qubits used in the gate implementation

process, as sometimes additional ancillary qubits are needed to implement an operation on a logical qubit. Also, by using encoded gates the necessity of decoding and re-encoding can be completely avoided. The approach of building fault-tolerant gates in this research is based on using a Dynamic Learning Algorithm. Here, we used back propagation for the learning algorithm, which is one of the most popular techniques employed in training neural network. A neural network is an input-output map, which adjusts its interconnections or weights using different learning algorithms that leads a system's input state to its desired output state. When a neural network is able to do this with almost zero error, we said it is "trained". For a quantum system, the objective of training comprises of tuning the parameter values in the system's Hamiltonian matrix. (Every quantum system is modelled by a matrix of parameters called the Hamiltonian, which represents the total energy of the system). Therefore, the more the number of qubits in a system, the more complicated is the Hamiltonian matrix that needs to be trained.

Through this research we trained for system parameters for an encoded-Hadamard gate in the three-qubit code. We also investigated an encoded- Hadamard gate for the five-qubit code, for which we have not obtained results as yet, due to the difficulty in training. However, since we were able to train for the three-qubit code, we propose to study three-qubit system in detail to draw analogies for solving for the five-qubit code by using properties like symmetry. Garigipati et al. [3], found a method to find analytical solutions for some quantum operations in an "N" qubit system ($N>8$), by using neural network training applied to smaller systems ($N<8$). They used to train a neural network to find system parameters for $N<8$, and then derived equations for each parameter value that predicted the values of the parameters for $N>8$ to achieve the same quantum operation.

As future work, we will be looking into deriving similar analytical solutions for higher-dimensional codes. As such, we studied the three-qubit encoded-Hadamard gate in detail in this thesis. As one part of the research, we split the input state and desired output state matrix vectors into individual pairs, and trained each pair separately in order to find a universal training pattern for constructing an encoded-Hadamard gate. In doing so we were able to overcome the lengthy and time consuming trainings which can be one of the main disadvantages of neural network learning techniques.

This thesis also proposes a new method to implement a CNOT gate operation between uncoupled "remote" qubits in an N qubit linear nearest neighbor architecture (LNNA) without using Swap gates to bring the qubits adjacent to each other, as is currently practiced. In an LNNA, qubits are arranged along an array, and only the adjacent qubits interact with each other [25-35]. Such NN restrictions present different challenges in implementing quantum algorithms, which are usually designed without taking the NN restriction into consideration. For instance, in NN architectures, we cannot directly perform multi-qubit gate operations on non-NN qubits; which also known as remote qubits, as there is no interaction between them. To perform an operation, the remote qubits need to be brought adjacent to each other by using Swap gates before performing the desired gate operation. However, this method increases the computational overhead, because as the separation between remote qubits in an NN layout increases, the number of elementary gates (gate count) required to perform an operation between them also increases.

Here, we use a dynamic learning algorithm that solves for the system parameters of an LNN array such that a CNOT gate can be implemented directly between remote qubits. We present our results for two-, three- and four-qubit systems. Once again, as future work, our aim is

to train for smaller LNN chains, and then derive equations that allow us to calculate the parameters directly for chains of longer length (N>8).

CHAPTER 2

DYNAMIC LEARNING ALGORITHM

In this research, we used a neural network approach to find system parameters for quantum systems to implement different operations. We trained the parameters of different quantum systems using a dynamic learning algorithm based on the gradient descent technique, which is an iterative minimization method. Here, the objective is to find the minimum error, and training the neural network involves finding the minimum of a complicated nonlinear function (called "error function") [25]**.** Since the gradient of the error function always points in the direction of steepest descent of the error function, the objective is to find the global minimum. The minimum is computed with respect of the system parameters or “weights” being trained. Initially, we start with a random weight vector, and keep upgrading weights in the direction of gradient descent. Throughout our training, we use the back propagation technique. Figure 2.1 shows the error surface with respect to weights.

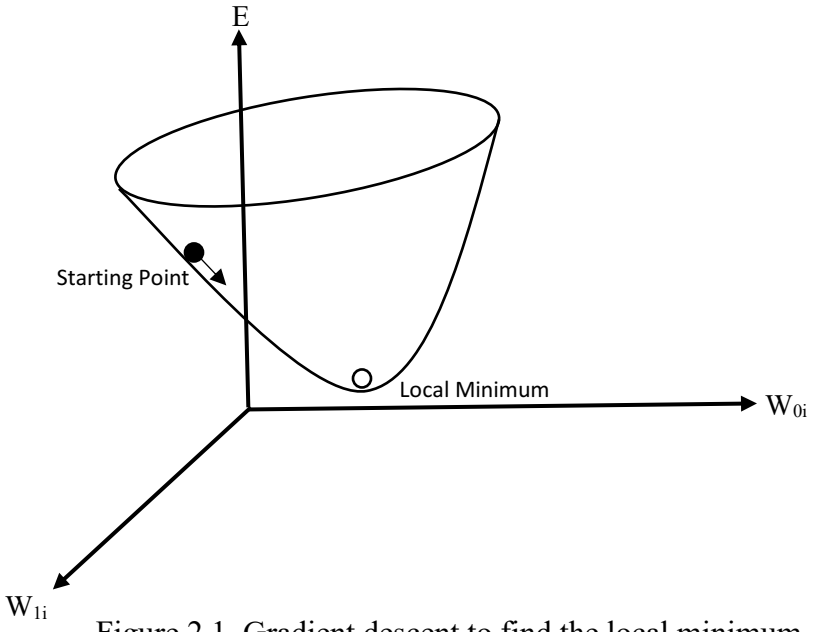


Figure 2.1. Gradient descent to find the local minimum.

2.1 Training Methodology

In this section, the training process for finding parameters values to get the desired gate operations will be discussed. Our methodology described in four major steps as follows:

Step 1: Choose a Hamiltonian for an N-qubit system. Since there can be different types of interactions between qubits, the training can be done over a wide range of quantum systems.

Step 2: Choose a learning rule that will be used to train the quantum system. Here, the dynamic learning algorithm based on the back propagation technique is used. The method is similar to the dynamic learning paradigm proposed by Behrman, et al., in [26], which was used to train a two-qubit Ising coupled system to implement a CNOT gate. However, while Behrman, et al., used measurement operators for training, we use quantum fidelity condition. This makes our scheme more general as we will later discuss. The training method uses a weight update rule based on gradient descent to adjust the device parameters, which are the weights of quantum system. The time evolution of the quantum system can be found by integrating the Schrödinger equation in MATLAB Simulink. For this, the ODE4 (ordinary differential equation) fixed size step solver will be used. (Simulink provides a set of explicit fixed-step continuous solvers. Explicit solvers compute the value of a state at the next time step as an explicit function of the current values of both the state and the state derivative. The solvers differ in the specific numerical integration technique that they use to compute the state derivatives of the model. The ODE4 solver uses the Fourth-Order Runge-Kuta (RK4) formula).

Step 3: Form training pairs for a specific gate operation, U here is the gate operation we are interested in finding system parameters for. A training pair will comprise of an initial state, $|\Psi\rangle_{in}$, and the corresponding output state, $|\Psi\rangle_{des}$, that would result when applying the gate operation, i.e., $|\Psi\rangle_{des}=U|\Psi\rangle_{in.}$ Thus, $|\Psi\rangle_{in}$ and $|\Psi\rangle_{des}$ comprise a training pair. The corresponding

density matrices for two states are $\rho=|\Psi>_{in}<\Psi|_{in}$ and $\rho_{des}= |\Psi>_{des}<\Psi|_{des} = U|\Psi>_{in}<\Psi|_{in}U^{-1}$. The density matrices will be used in training. A minimum of 2^N vector pairs will be 4 training pairs, one each corresponding to the initial states |00>, |01>, |10>, and |11> (in the computational basis).

Step 4: Train the network for 'm' iterations. Here, we are trying to find parameters of the system Hamiltonian that will implement the gate operation, U, within a time duration, t_f. Note that, t_f is the time within which we want to implement the gate operation, U, in a physical quantum system, as such, the duration of each iteration is equal to t_f. At the start of training, i.e., the first iteration, we randomly assign values to the parameters such that they are within experimental limits for a given quantum system. Next, the following procedure is followed during each iteration:

(a) Input on of the 2^N input states as the initial state, $\rho_{in} =|\Psi>_{in}<\Psi|_{in}$ corresponding to a training pair to the network. Allow the state to evolve under the Schrödinger equation for a time t_f. The density matrix for the "actual" output state, ρ_{out}, by propagation of the input state through the network, is calculated.

(b) Calculate the quantum fidelity between the "actual" output, ρ_{out}, and the desired output, ρ_{des} (from the training pair). Calculate the "error" for the training pair, E_p, by taking the difference between the desired quantum fidelity (which is '1' in all cases) and the calculated quantum fidelity

(c) Back-propagation the error (in time) through the network, and adjust the parameters. This can be done by integrating the Schrödinger equation from the final time, t_f, to 0 with the help of a change of variable, $t'= t_f – t$. The network can be set up such that different parameters are adjusted at different "learning" rates.

(d) Repeat (a), (b), (c) until all training pairs are exhausted for the iteration. At the end of each iteration, the root mean square (RMS) error will be calculated by using the following equation:

$$RMS = \frac{1}{2^{\frac{N}{2}}} \sqrt{\sum_{p=1}^{2^N} E_p^2} \tag{2.1}$$

where 2^N is the number of training pairs and E_p is the corresponding to each training pair. The training is stopped (after *m* iterations) when the RMS error falls below a certain threshold.

Step 5: A successfully trained network will have RMS error below a certain threshold, if the network successfully trains, we test the accuracy of the results by substituting these parameters in the Hamiltonian, and checking if the desired gate operation, U, is implemented for different input states. If not, we continue to train the network for more iterations, with the parameter values set at the values obtained from the previously trained network.

When training, as a first step, we will assume all the parameters of the system Hamiltonian to be variables. The network will then be trained to find the system parameters that realize U within time t_f. Once the network can be successfully trained for the case when all the parameters are variable, we will then train the network subject for more constraints with some of the parameters fixed. This will be practical since in any quantum system not all parameters can be tuned. Even if all parameters can be tuned, in a physical implementation, it is not desirable to treat all parameters as variables, as doing so can increase the complexity if the external control circuitry.

2.2 Learning Algorithm for Training

In this section, we will show how the dynamic learning algorithm is constructed, using the methodology described by Behrman et al. [26]. The main differences between their scheme and ours are:

(i) For training, Behrman et al., used a measurement operator. However, one limitation with this approach is the selection of the measurement operators. We might not be able to use the same measurement operator to train for different gate operations, and we have to define a suitable operator in each case. In our scheme, we use the quantum fidelity condition for training, where the error is calculated by finding the fidelity of the final state (corresponding to a given input state). As such, the scheme is more general.

(ii) In [26], when training, one of the parameters (bias) was treated as a variable (with time). As such, the gate operation was executed, in three time steps ($3t_s$) where the bias was changed at the start of each time step (t_s). In our scheme, none of the parameters are treated as variables (with time) while training. As such, the gate operation is implemented within a single time step (t_f).

To construct the dynamic learning algorithm, we use the time evolution Schrodinger equation which is given by:

$$\frac{\partial \rho}{\partial t} = -\frac{i}{\hbar}[H, \rho] \tag{2.2}$$

here, ρ is the density matrix, ħ is Plank's constant divided by 2π, and H is the Hamiltonian of the system. In the dynamic learning algorithm, the basic idea is to force the output state, $|\Psi\rangle_{out}$, of the system for a given input state, $|\Psi\rangle_{in}$, to the desired state, $|\Psi\rangle_{des}$, under a chosen gate operation, **U**, by controlling the parameters of the Hamiltonian (for instance, tunneling, bias and coupling).

Initially, the system parameters are chosen randomly (while confining to experiment limits). They are then trained by using the gradient descent learning rule,

$$W_{new} = W_{old} - \alpha \frac{dL}{dw} \tag{2.3}$$

where α is the learning rate, W is the parameter we are training, and L is the Lagrangian which is used to find the local minimum of the error function. Here, the density matrix ρ is constrained to satisfy the Schrodinger equation for the time interval of 0 to t_f. Hence, the Lagrangian can be defined as

$$L = \frac{1}{2} E_p^2 + \int_0^{t_f} \lambda(t)^\dagger \left(\frac{\partial \rho}{\partial t} + \frac{i}{\hbar}[H, \rho]\right) \gamma(t) dt \tag{2.4}$$

where E_p is the error corresponding to a training pair P, where P=1 to 2^N, $\lambda(t)^\dagger$ and $\gamma(t)$ are the Lagrange multipliers. Here, $\lambda(t)^\dagger$ is row vector and $\gamma(t)$ is column vector. By making the first variation of the Lagrangian multipliers zero, the resulting equation can be written as

$$\lambda_i \frac{\delta \gamma_j}{\delta t} + \frac{\delta \lambda_i}{\delta t} \gamma_j - \frac{i}{\hbar} \sum_k \lambda_k H_{ki} \gamma_j + \frac{i}{\hbar} \sum_k \lambda_i H_{jk} \gamma_k \tag{2.5}$$

with the boundary conditions at the final time, t_f:

$$\lambda_i(t_f) \gamma_j(t_f) = \left(E_p\right)(\rho_{des})_{ji} \tag{2.6}$$

here, $\lambda(t_f)$ and $\gamma(t_f)$ are the Lagrangian multipliers at the final time, $(\rho_{des})_{ji}$ is the ji^{th} element in the "desired" density matrix (defined in step 3 of the training process).

The error, E_p, is calculated as:

$$E_p = 1 - F(\rho_{des}, \rho_{out}) \tag{2.7}$$

where,

$$F(\rho_{des}, \rho_{out}) = \mathrm{Tr}\left(\sqrt{\sqrt{\rho_{des}}\, \rho_{out} \sqrt{\rho_{des}}}\right) \tag{2.8}$$

Here, $F(\rho_{des}, \rho_{out})$ is the quantum fidelity [8] [27], and ρ_{des} and ρ_{out} are as previously described. The quantum fidelity gives a measure of the two density matrices. Here, $F(\rho_{des}, \rho_{out})$ varies from 0 to 1 depending on how close ρ_{out} is to ρ_{des}. If the two matrices are identical, i.e.,

if $\rho_{out} = \rho_{des}$, then $F(\rho_{des}, \rho_{out}) = 1$. As can be seen from equation (2.7), the error, E_p, is calculated by taking the difference between the desired fidelity, 1, and fidelity of the trained network, for a given training pair.

In this thesis we use a dynamic learning algorithm based on a gradient descent technique to minimize the gate count in order to increase the quantum system stability against decoherence. We applied the dynamic learning algorithm to find solutions for two important applications where reducing the gate is very significant. First is Quantum Error Correction (QEC), and the second is implementation of CNOT gate between remote qubits in LNN architecture which we will describe in detail in the following chapters.

CHAPTER 3

IMPLEMENTATION OF HADAMARD FAULT TOLERANT QUANTUM GATE

As discussed in Chapter 1, quantum error correction is a major area in quantum computing, which tries to find solutions to overcome the fragility of quantum systems. Much research work has been done on developing quantum error correction codes to correct various types of errors, and a rigorous theoretical framework for the structure, properties and operation of different QEC codes [17] has been developed. However, not all codes allow implementation of encoded gates, that is, gates that can be directly implemented on a codeword (logical or encoded qubit) without decoding.

In this chapter, the dynamic learning algorithm is used to find optimized system parameters to implement an encoded-Hadamard (e-H) gate for the three-qubit code. Section 3.1 describes how the input and output states are categorized for training. Basically, the set of 8 basis vectors ($|000\rangle$ through $|111\rangle$) are each assigned to a codeword, $|0\rangle_L$ or $|1\rangle_L$. There are two different ways in which this assignment is done. Table 3.1 shows the encoded-Hadamard gate implementation based on majority number of zeros, where all states with majority of zeros are assigned to $|0\rangle_L$. Table 3.1 shows the encoded-Hadamard gate implementation based on majority number of zeros, where all states with majority of zeros are assigned to $|0\rangle_L$. Table 3.2 shows the corresponding output vectors. Table 3.3 shows the e-H gate implementation based on odd and even number of zeros, where all states with an odd number of zeros are assigned to $|0\rangle_L$. Table 3.4 shows the corresponding output vectors. Section 3.2 describes training process for two and four vector (partial e-H gate). Section 3.3 shows how different set of vector pairs are related to each other.

3.1 Organization of Eight Vector Matrix using Majority Number of Zeros for Encoded Hadamard Gate

Table 3.1 shows the e-H gate implementation based on majority number of zeros, where all states with majority of zeros are assigned to $|0\rangle_L$. Table 3.2 shows the corresponding output vectors. The matrix representation for the input and desired output vectors are:

$$\text{Input} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

TABLE 3.1

INPUT BASED ON MAJORITY NUMBER OF ZEROS

$\|0\rangle_L$	$\|1\rangle_L$
|000>	|011>
|001>	|101>
|010>	|110>
|100>	|111>

$$\text{Desired Output} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{3.2}$$

TABLE 3.2

DESIRED OUTPUT BASED ON MAJORITY NUMBER OF ZEROS IN DIRAC NOTATION AND VECTOR FORM

$\|000>\rightarrow \frac{\|000>+\|111>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$	$\|011>\rightarrow \frac{\|011>-\|100>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\0\\1\\-1\\0\\0\\0\end{bmatrix}$
$\|001>\rightarrow \frac{\|110>+\|001>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\0\\0\\0\\0\\1\\0\end{bmatrix}$	$\|101>\rightarrow \frac{\|101>-\|010>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\1\\0\\0\\-1\\0\\0\end{bmatrix}$
$\|010>\rightarrow \frac{\|101>+\|010>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\end{bmatrix}$	$\|110>\rightarrow \frac{\|110>-\|001>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\0\\0\\0\\0\\-1\\0\end{bmatrix}$
$\|100>\rightarrow \frac{\|011>+\|100>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\0\\1\\1\\0\\0\\0\end{bmatrix}$	$\|111>\rightarrow \frac{\|000>-\|111>}{\sqrt{2}} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\-1\end{bmatrix}$

3.2 Organization of Eight Vector Matrix using Odd and Even Number of Zeros for Encoded Hadamard Gate

Table 3.3 shows the e-H gate implementation based on odd and even number of zeros, where all states with an odd number of zeros are assigned to $|0\rangle_L$. Table 3.4 shows the corresponding output vectors. Table 3.3 shows the e-H gate implementation based on odd and even number of zeros, where all states with an odd number of zeros are assigned to $|0\rangle_L$. Table 3.4 shows the corresponding output vectors.

TABLE 3.3

INPUT BASED ON ODD AND EVEN NUMBER OF ZEROS

$\|0\rangle_L$	$\|1\rangle_L$
|000>	|001>
|011>	|010>
|101>	|100>
|110>	|111>

$$\text{Desired Output} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{3.3}$$

3.3 Implementation of Eight Vector Matrix for Encoded Hadamard Gate

By using the dynamic learning algorithm, we trained the system for the desired e-H gate operation until the RMS error for the gate reduced to 17%. As Figure 3.1 shows, the training procedure got exhausted, and RMS error saturated at 17%, and could not be reduced this value.

TABLE 3.4

DESIRED OUTPUT DISTINGUISHED BASED ON ODD AND EVEN NUMBER OF ZEROS IN DIRAC NOTATION AND VECTOR FORM

$\|000>\rightarrow \frac{\|000>+\|111>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$	$\|001>\rightarrow \frac{\|110>-\|001>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\-1\\0\\0\\0\\0\\1\\0\end{bmatrix}$
$\|011>\rightarrow \frac{\|011>+\|100>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\0\\1\\1\\0\\0\\0\end{bmatrix}$	$\|010>\rightarrow \frac{\|101>-\|010>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\-1\\0\\0\\1\\0\\0\end{bmatrix}$
$\|101>\rightarrow \frac{\|101>+\|010>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\end{bmatrix}$	$\|100>\rightarrow \frac{\|011>-\|100>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\0\\1\\-1\\0\\0\\0\end{bmatrix}$
$\|110>\rightarrow \frac{\|110>+\|001>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\\0\\0\\0\\0\\1\\0\end{bmatrix}$	$\|111>\rightarrow \frac{\|000>-\|111>}{\sqrt{2}}=\frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\-1\end{bmatrix}$

This was thus the best outcome so far achievable for an 8x8 (learning rate set as 0.1 for all parameters and total time, t_f, sets as 1ns) . Table 3.5 shows the parameters of the trained network for the e-H gate with RMS error of 17%. For both sets of input-output pairs (Sections 3.1 and 3.2), the RMS error was the same, and did not go below 17%. This was interesting, even because the $|0\rangle_L$ and $|1\rangle_L$ codeword subspaces were different as elaborated in Sections 3.1 and 3.2.

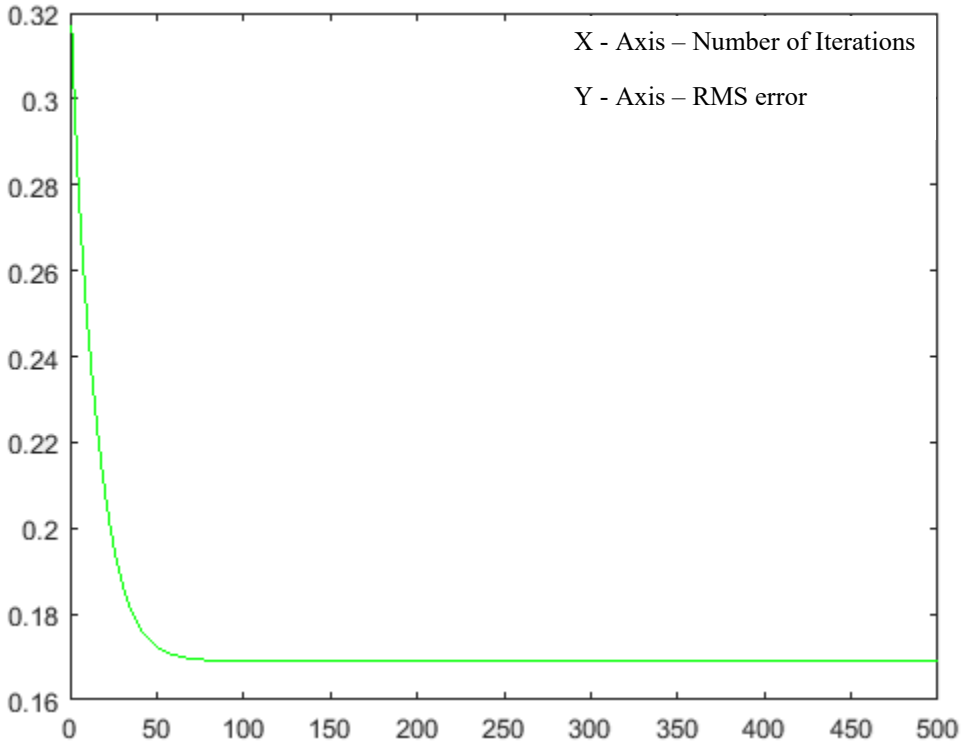


Figure 3.1. Three qubits system with 8x8 matrix inputs.

3.4 Implementation of Partial Encoded Hadamard Gate

Since the RMS error would not go below 17%, we decided to split the training input-output pairs into groups of 2 and 4, respectively, and study the convergence of the error to a minimum in each case.

TABLE 3.5

THREE QUBITS SYSTEM TRAINING PARAMETERS (GHZ)

Tunneling	**Bias**	**Coupling**
Δ_a =.3135	ε_a =.3135	$\xi_{a\text{-}b}$ = 0.3396
Δ_b =.0288	ε_b =.0288	$\xi_{b\text{-}c}$ = 0.3168
Δ_c =.2894	ε_c = .2894	$\xi_{a\text{-}c}$ = 0

We call these vectors "partial vectors", and the gate they implement a "partial encoded-Hadamard gate (partial e-H)". The goal was to see if any set of the input-output pairs were behaving as outliers, which were making it hard for the RMS error to descend to zero. First, we started to train the first and the eighth vectors together as is shown in Figure 3.2. The input-output training pairs are given by equation (3.4). Table 3.6 shows the training parameters for the first and the eighth vector pairs for the trained network. The network was trained for 6000 iterations in total, at the end of which the RMS error descended to 2.19% as shown in Figure 3.2. This was an improvement over the 17% error obtained when training all eight input-output pairs.

$$\text{Input} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{Desired Output} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & -1 \end{bmatrix} \tag{3.4}$$

To verify whether the trained network parameters in Table 3.6 computed a partial e-H or not, we substituted them in a three-qubit system Hamiltonian (here, N = 3):

$$H_N = \sum_{i=1}^{N} (\Delta_i \sigma_{Xi} + \varepsilon_i \sigma_{Zi}) + \sum_{i=1}^{N-1} (\xi_{i,i+1} \sigma_{Zi} \sigma_{Zi+1}) \tag{3.5}$$

where Δ_i, and ε_i for i=1,2,…,N are the tunneling and bias parameters, respectively for qubits Q_i, and $\xi_{i,i+1}$ is the coupling parameter between qubits $Q_i$and Q_{i+1}. Also, σ_{Xi} and σ_{Zi} are the Pauli matrices corresponding to Q_i.

TABLE 3.6

TRAINING PARAMETERS FOR FIRST AND EIGHTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = 0.3030$	$\varepsilon_a = -0.2047$	$\xi_{a\text{-}b} = 0.6069$
$\Delta_b = 0.3340$	$\varepsilon_b = 0.4076$	$\xi_{b\text{-}c} = 0.6069$
$\Delta_c = 0.3030$	$\varepsilon_c = -0.2047$	$\xi_{a\text{-}c} = -0.5133$

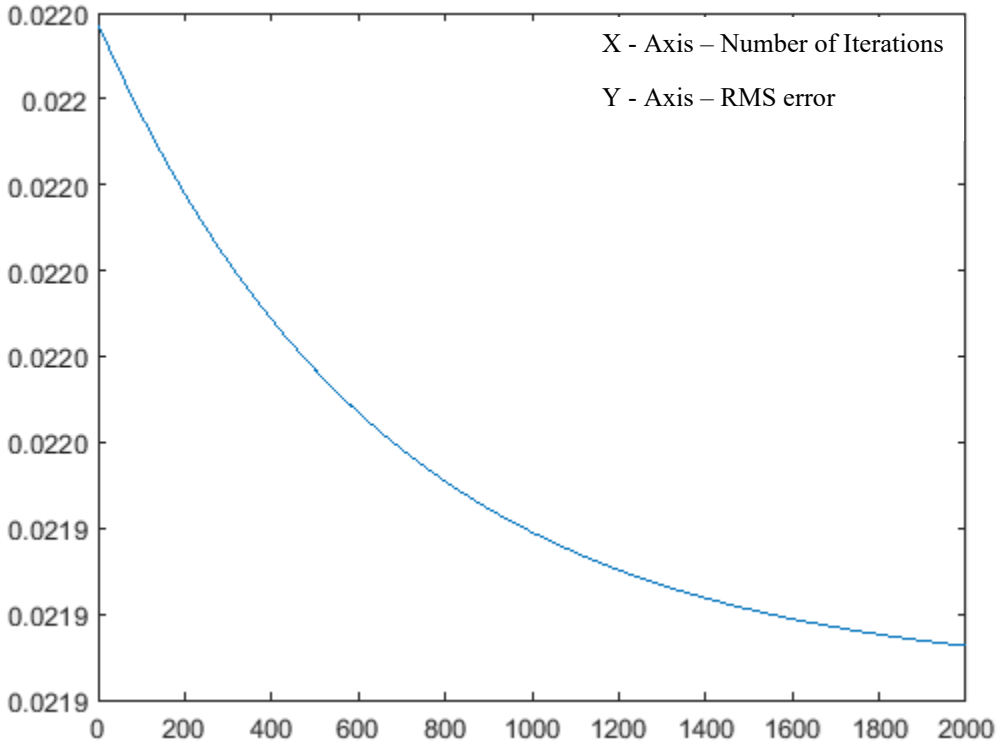


Figure 3.2. The least RMS error for the first and the eighth vector.

The Hamiltonian was then simulated, where the system was allowed to evolve for 1ns (t_f). The simulation confirmed a partial e-H gate operation, where an e-H gate was obtained only for the input-output pairs shown in equation (3.4). In each case, there was an overall global phase

in the corresponding output state (–155 and –154 degrees, respectively, for the first and eighth vectors), which can be ignored. Equations (3.6) and (3.7) show the output state from simulation, which confirm the partial e-H gate operation.

$$\begin{aligned} |000> \rightarrow .7045\, e^{-155.95°\, i}|000> + .0844\, e^{12.46°\, i}|001> + .0485\, e^{114.33°\, i}|010> \\ +.1262e^{149.51°i}|011> + .0844\, e^{12.46°i}|100> + .0114\, e^{-139.67°i}|101> + .1262\, e^{-} \\ {}^{149.51°i}|110> +.6747\, e^{-154.37°i}|111> \\ \cong (.7045|000>+.6747|111>)e^{-155°\, i} \end{aligned} \tag{3.6}$$

$$\begin{aligned} |111> \rightarrow .6747\, e^{-154.37°\, i}|000> + .0707\, e^{-149.92°\, i}|001> + .1191\, e^{-148.43°} \\ {}^{i}|010> +.0705e^{-148.64°i}|011> + .0707\, e^{-149.92°i}|100> + .0379\, e^{-126.48°i}|101> + \\ .0705\, e^{-148.64°i}|110> +.7136\, e^{27.48°i}|111> \\ \cong (.6747|000>+.7136|111>)e^{-154°\, i} \end{aligned} \tag{3.7}$$

The overall fidelity of the partial gate operation was 97.81%, which was expected (since the network trained to an RMS error of 2.19%). Next, we trained the system for the second and the seventh vectors (see equation (3.8)), the trained parameters for which are shown in Table 3.7. Comparing Tables 3.6 and 3.7 we can see that the training parameters are not similar for these two sets of vector pairs. Figure 3.3 shows the progress of the training after 1000 iterations, where the RMS error was saturated at 17%.

$$\text{Input} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \text{Desired Output} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.8}$$

Next, we trained for the third and sixth vectors (equation (3.9)). Figure 3.4 shows the progress of the training after 1000 iterations. Table 3.8 shows the training parameters for the third and sixth vectors.

TABLE 3.7

TRAINING PARAMETERS FOR SECOND AND SEVENTH VECTOR (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3373$	$\varepsilon_a = -.0245$	$\xi_{a-b} = .3521$
$\Delta_b = .1463$	$\varepsilon_b = .2851$	$\xi_{b-c} = .3522$
$\Delta_c = .3376$	$\varepsilon_c = .0246$	$\xi_{a-c} = 0$

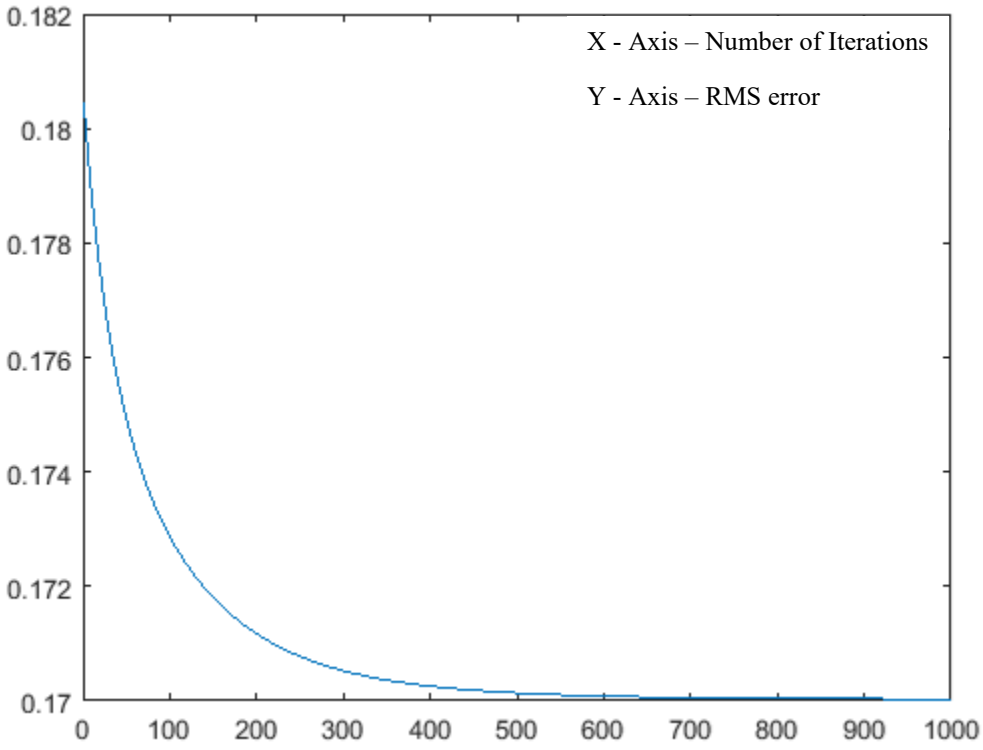


Figure 3.3. RMS error results after 1000 epochs for the second and the seventh vector.

The RMS error kept descending, and if further training was continued, we would achieve a lower RMS error. Here, we were studying the trend in the descent of the RMS error for

different training pairs. So we did not train further since it appeared that the RMS error would go down, and not saturate as for the second and seventh vector pairs.

$$\text{Input} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \text{Desired Output} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.9}$$

TABLE 3.8

TRAINING PARAMETERS FOR THIRD AND SIXTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3193$	$\varepsilon_a = .0016$	$\xi_{a\text{-}b} = .4272$
$\Delta_b = .2336$	$\varepsilon_b = .1863$	$\xi_{b\text{-}c} = .4272$
$\Delta_c = .3193$	$\varepsilon_c = .0016$	$\xi_{a\text{-}c} = -.1372$

Lastly, we trained for the fourth and fifth vectors (equation (3.10)). Figure 3.5 shows the progress of the training after 1000 iterations. Table 3.9 shows the training parameters for the third and sixth vectors. Once again the RMS error saturated at 17%.

$$\text{Input} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \text{Desired Output} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.10}$$

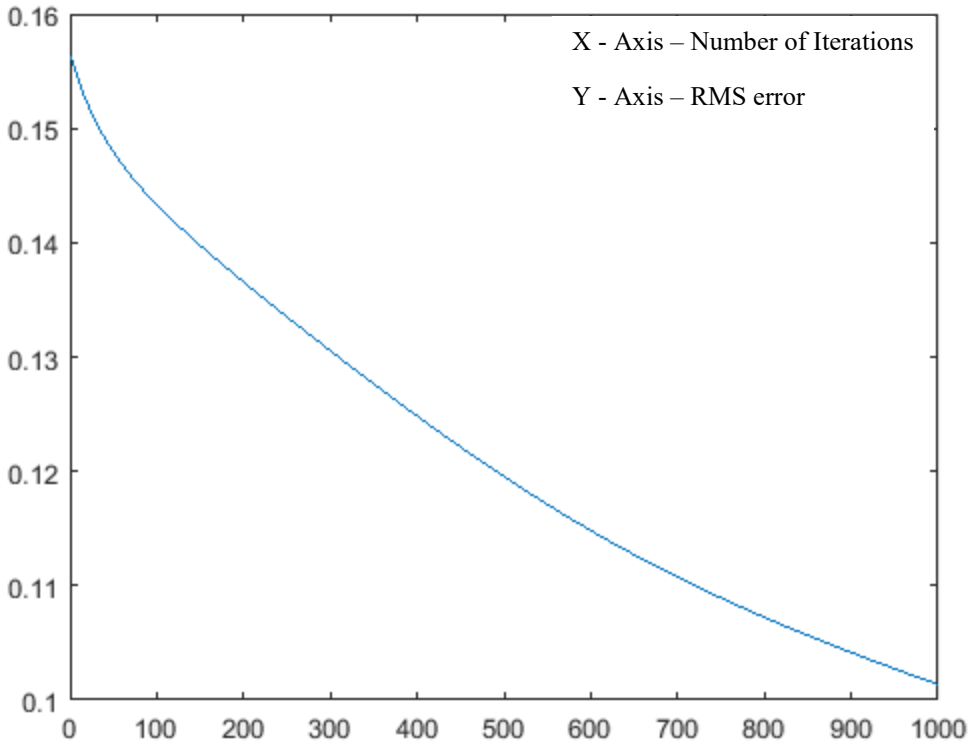


Figure 3.4. RMS error results for the third and the sixth vectors.

TABLE 3.9

TRAINING PARAMETERS FOR FOURTH AND FIFTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3373$	$\varepsilon_a = .0245$	$\xi_{a\text{-}b} = .3521$
$\Delta_b = .1463$	$\varepsilon_b = .2851$	$\xi_{b\text{-}c} = .3522$
$\Delta_c = .3376$	$\varepsilon_c = -.0246$	$\xi_{a\text{-}c} = 0$

By comparing the RMS error results for the four sets of paired vectors, it shows that some of them cause the training to saturate since the RMS error does not descend any further. For instance, Figure 3.5 shows the result for the fourth and fifth pairs for which the best RMS error we achieved was 17%.

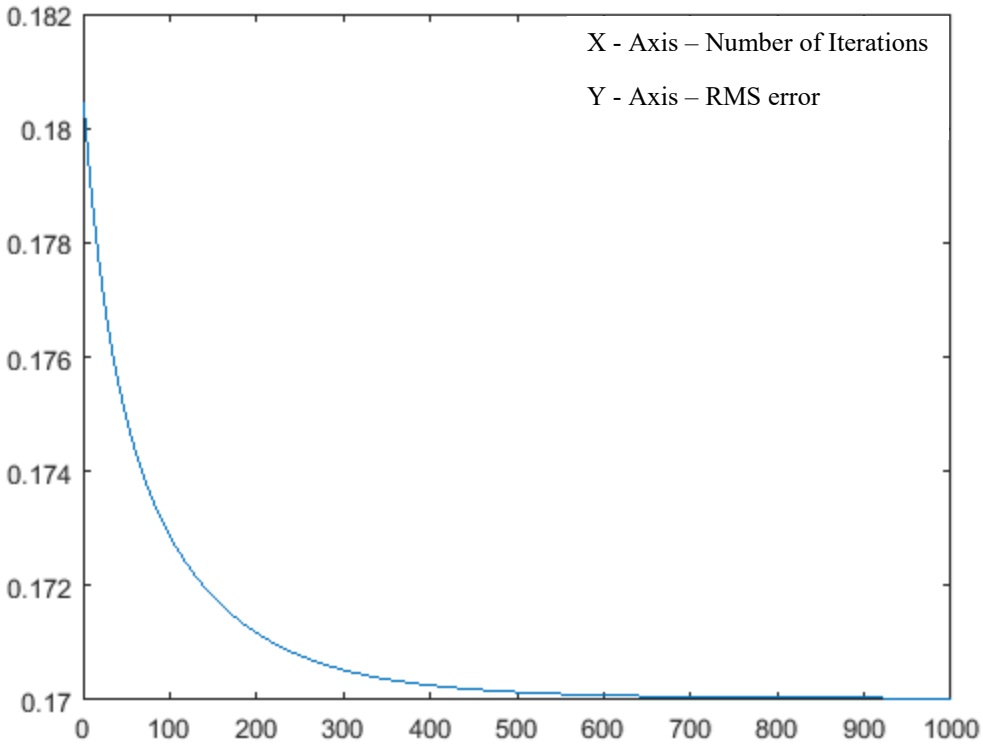


Figure 3.5. The RMS error results for the fourth and the fifth parameters.

Next, we studied the parameter values for the four different sets. On comparing the parameter values for the second (the second and the seventh vectors) and fourth (the fourth and the fifth vectors) pairs, we find that they are matched. (A similar matching is seen between the first pair (the first and the eighth vectors) and the third pair (the third and the sixth vectors)). We are comparing the second and fourth pairs here since the parameter values in each case are after 1000 iterations of training. By "matched" we mean that knowing the set of parameters for one pair, we can deduce the set of parameters for the second pair without training. This was because the parameters for two matched pairs differ only in three parameters (ε_a, ε_c and ξ_{ac}), all other parameters being the same. The three parameters of two matched pairs are simply negatives (see Tables 3.7 and 3.9) Observing this trend, we next re-trained for the third pair (third and sixth vectors). We used the trained parameters for the first pair (first and seventh vectors) for which the RMS error was 2.19%. However, we negated the values for ε_a, ε_c and ξ_{ac}. Table 3.10 shows

the updated training parameters for the third and sixth pairs, and Figure 3.7 shows the progress of the training. As can be seen the RMS error reduced to 2.19%.

TABLE 3.10

TRAINING PARAMETERS FOR THIRD AND SIXTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3030$	$\varepsilon_a = .2047$	$\xi_{a-b} = .6069$
$\Delta_b = .3340$	$\varepsilon_b = .4076$	$\xi_{b-c} = .6069$
$\Delta_c = .3030$	$\varepsilon_c = .2047$	$\xi_{a-c} = .5133$

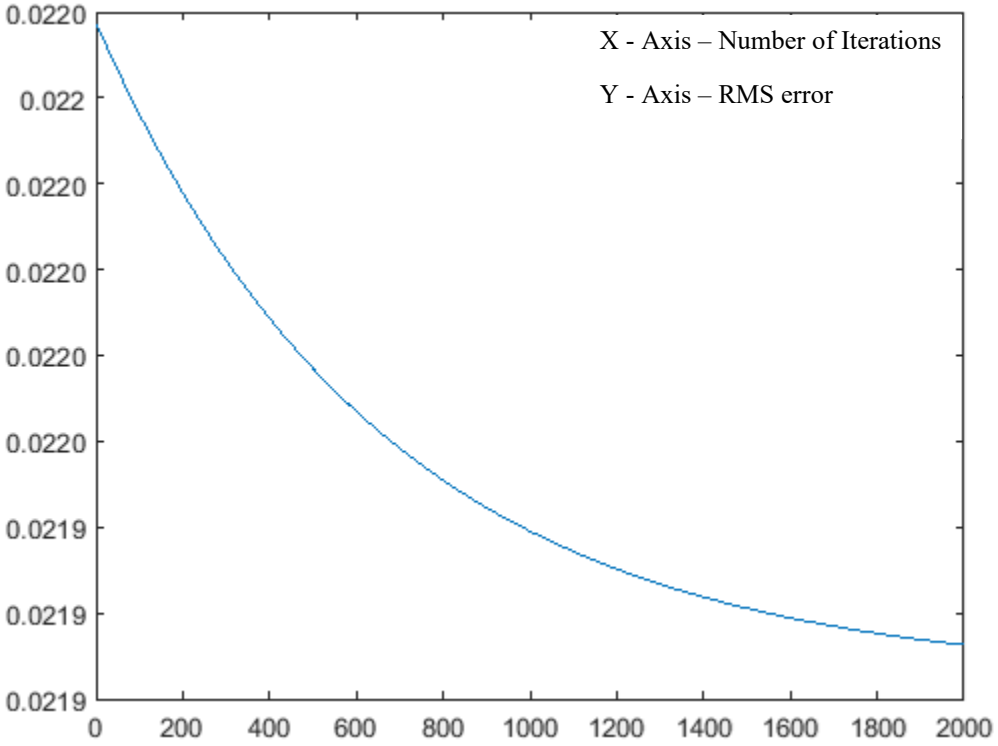


Figure 3.6. The RMS error results for the third and the sixth vectors.

Once again, we simulated the partial e-H gate by substituting these parameters in a three-qubit Hamiltonain as given by equatiuon. (3.5). Equations (3.11) and (3.12) show the final state for the third and the sixth vectors, which confirms the partial e-H gate operation.

$$|010> \rightarrow .0372\, e^{72.29^\circ i}|000> + .1301\, e^{149.29^\circ i}|001> + .7023\, e^{154.88^\circ i}|010> \qquad (3.11)$$

$$+.0842e^{167.72°i}|011> + .1301\ e^{149.37°i}|100> + .6761\ e^{-25.62°i}|101> + .0842\ e^{167.72°i}|110> +.0102\ e^{133.61°i}|111> \cong (.7045|010>+.6747|101>)e^{154° i}$$

$$|101> \rightarrow .0485\ e^{65.66° i}|000> + .1262\ e^{149.51° i}|001> + .7045\ e^{155.95° i}|010> +.0844e^{167.53°i}|011> + .1262\ e^{149.51°i}|100> + .6747\ e^{-25.62°i}|101> + .0844\ e^{167.53°i}|110> +.0114\ e^{139.67°i}|111> \cong (.7045|010>+.6747|101>)e^{154° i} \quad (3.12)$$

So far we have showed that the dynamic learning algorithm can train for a partial e-H gate on the first pair of the training set, and then by only changing the signs of three parameters, we can apply the results to third pair (matched pair) of the training set to implement a partial e-H gate. The next step was to investigate if this approach could be extended further. That is, we checked if parameters from the first pair of training could be used to train for the second and fourth pairs. Note that for both these pairs in the training set, the RMS error would not go below 17%. As a last case solution, we used parameters from the trained network of the first pair, and started training. The results were amazing. After only 430 iterations, the RMS error went down to 2.19%. Figure 3.7 shows the progress of training for the fourth pair (fourth and fifth vectors). Table 3.11 lists the values of the parameters for the trained network.

TABLE 3.11

TRAINING PARAMETER SETS FOR FOURTH PAIR WHICH INCLUDES FOURTH AND THE FIFTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .2977$	$\varepsilon_a = -.2041$	$\xi_{a-b}=.5249$
$\Delta_b = .3017$	$\varepsilon_b = .2082$	$\xi_{b-c}=.6120$
$\Delta_c = .3372$	$\varepsilon_c = -.4049$	$\xi_{a-c}=-.6067$

Next, the set of parameters in Table 3.11 were verified for a partial e-H operation by substituting them in the Hamiltonian given by equation (3.5), and the system was allowed to evolve for 1ns (t_f). Equations (3.13) and (3.14) show that the partial e-H gate was successfully implemented up to an overall global phase of approximately –155 that can be ignored.

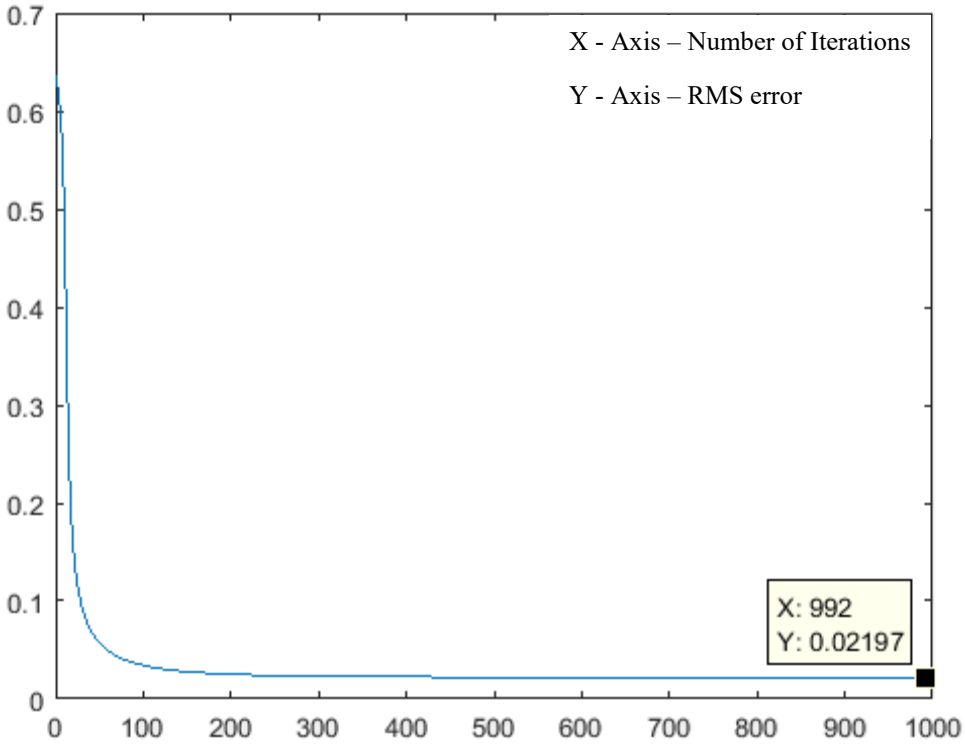


Figure 3.7. The RMS error for the fourth and the fifth vectors.

$$|100> \rightarrow .1145\, e^{-144.69°\, i}|000> + .0871\, e^{13.95°\, i}|001> + .0563\, e^{117.95°\, i}|010> +.7073e^{-155.79°i}|011> + .6732\, e^{-153.66°i}|100> + .0153\, e^{-156.93°i}|101> + .1233\, e^{-148.96°i}|110> +.0855\, e^{15.22°i}|111> \cong (.7073|011>+.6732|100>)e^{-155°\, i} \quad (3.13)$$

$$|011> \rightarrow .07\, e^{-174.65°\, i}|000> + .0785\, e^{-146.02°\, i}|001> + .1166\, e^{-149.05°\, i}|010> +.6732e^{-153.66°i}|011> + .7125\, e^{28.28°i}|100> + .0508\, e^{-107.56°i}|101> + .0761\, e^{-151.92°i}|110> +.0782\, e^{-149.31°i}|111> \cong (.6732|010>+.7125|101>)e^{-153°\, i} \quad (3.14)$$

Once we had the trained parameters for the fourth pair, we simply negated the three parameters, and obtained parameters for the second pair without training. Simulations once again

confirmed a partial e-H gate operation on the second pair of vectors. Equations (3.15) and (3.16) show the simulation results.

$$\begin{gathered}|001\rangle \rightarrow .0563\, e^{-62.04^\circ i}|000\rangle + .7073\, e^{155.79^\circ i}|001\rangle + .1145\, e^{144.69^\circ i}|010\rangle \\ +.0871 e^{166.04^\circ i}|011\rangle + .1233\, e^{148.96^\circ i}|100\rangle + .0855\, e^{-164.77^\circ i}|101\rangle + .6732\, e^{-26.33^\circ i}|110\rangle + .0153\, e^{156.93^\circ i}|111\rangle \\ \cong (.7073|001\rangle + .6732|110\rangle) e^{155^\circ i}\end{gathered} \tag{3.15}$$

$$\begin{gathered}|110\rangle \rightarrow .1166\, e^{-149.05^\circ i}|000\rangle + .6732\, e^{-26.33^\circ i}|001\rangle + .07\, e^{-5.34^\circ i}|010\rangle \\ +.0785 e^{146.02^\circ i}|011\rangle + .0761\, e^{-28.07^\circ i}|100\rangle + .0782\, e^{149.31^\circ i}|101\rangle + .7125\, e^{-28.28^\circ i}|110\rangle + .0508\, e^{-72.4314^\circ i}|111\rangle \\ \cong (.6732|001\rangle + .7125|110\rangle) e^{-26^\circ i}\end{gathered} \tag{3.16}$$

To summarize the results of this section, firstly, we divided the eight basis vector into four sets, each of which comprised a pair of vectors. Pair one, two, three and four comprised of the first and eighth, second and seventh, third and sixth, and fourth and fifth, vectors, respectively. On training, we found that pairs one and three were easier to train and the RMS error decreased to 2.19% for pair one (we stopped training at 1000 iterations for pair three, noticing that the RMS kept decreasing). Both pairs two and four were difficult to train, and the RMS error would not go beyond 17%. We also found that pairs one and three, and pairs two and four were matched, i.e., if we found the parameters for pair one to implement a partial e-H gate, we did not have to re-train for pair three for a partial e-H gate. We simply had to negate three parameters, ε_a, ε_c and ξ_{ac}. Next, we used the trained parameters for pair one as a starting point to find parameters for a partial e-H gate for pair four. The RMS error which previously saturated at 17% now descended to 2.19%. Once we had the parameters for a partial e-H gate for pair four, by simply negating the same three parameters, we could implement the e-H gate for pair two. The obvious advantage of this new approach is it is less time consuming, and also allowed us to train for vector pairs for which training had saturated. We will hereafter call this approach the

"phase-flip approach" since we are changing the phases on three parameters by 180 degrees (sign change). In all the cases mentioned above, the training set was as described in Section 3.1, i.e., the e-H gate implementation was based on odd and even number of zeros, where all states with odd number of zeros are assigned to $|0\rangle_L$. In the following, we apply our phase-flip approach towards implementing an e-H gate based on majority number of zeros, i.e., all states with majority of zeros are assigned to $|0\rangle_L$. We trained the 3 qubit system for 5000 epochs and greater with a total time (t_f) of 1ns and a learning rate of 0.25 for all training sets. The RMS error was found to reduce down to 0.82% as shown in Figure 3.8. Table 3.12 shows the training parameters for the second and the seventh vector. Once again, by only flipping phases on the same three parameters, ε_a, ε_c and ξ_{a-c}, we can get parameters for implementing an e-H gate for the fourth pair. The RMS error was 1.14%.

TABLE 3.12

TRAINING PARAMETERS FOR SECOND AND SEVENTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
Δ_a = .2658	ε_a = –.2803	$\xi_{a\text{-}b}$= –1.4353
Δ_b = –.5546	ε_b = –.1665	$\xi_{b\text{-}c}$= 1.3973
Δ_c = .5083	ε_c = –.4620	$\xi_{a\text{-}c}$= –1.4550

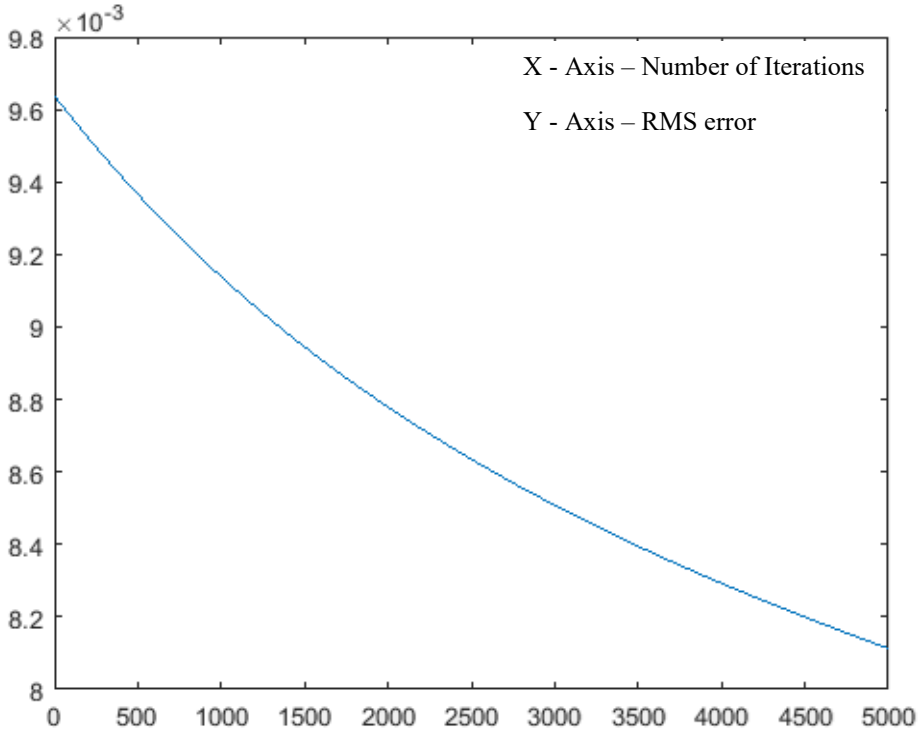


Figure 3.8. RMS error for the majority based output desire for the second and the seventh vector.

Table 3.12 shows the training parameters for the second and the seventh vector. Once again, by only flipping phases on the same three parameters, ε_a, ε_c and ξ_{a-c}, we can get parameters for implementing an e-H gate for the fourth pair. The RMS error was 1.14%.

TABLE 3.13

TRAINING PARAMETERS FOR FOURTH AND FIFTH VECTORS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .2658$	$\varepsilon_a = .2803$	$\xi_{a\text{-}b} = -1.4353$
$\Delta_b = -.5546$	$\varepsilon_b = -.1665$	$\xi_{b\text{-}c} = 1.3973$
$\Delta_c = .5083$	$\varepsilon_c = .4620$	$\xi_{a\text{-}c} = 1.4550$

Equations (3.17) and (3.18) show the output of simulation results for the second and the seventh vector.

$$|001> \rightarrow .028\, e^{36.97°\, i}|000> + .7070\, e^{-75.66°\, i}|001> + .0417\, e^{98.13°\, i}|010> + .0530 e^{-56.54°i}|011> + .0328\, e^{-142.20°i}|100> + .0144\, e^{171.95°i}|101> + .6960\, e^{-} \quad (3.17)$$

$$^{77.55°i}|110\rangle + .0957\ e^{-102.21°i}|111\rangle$$
$$\cong (.7022|000\rangle + .693|111\rangle)e^{-75.66°\ i}$$

$$|110\rangle \rightarrow .0476\ e^{-40.8°\ i}|000\rangle + .6960\ e^{-77.55°\ i}|001\rangle + .0565\ e^{\ 145.30°\ i}|010\rangle$$
$$+.04\ e^{-31.66°i}|011\rangle + .0574\ e^{27.77°i}|100\rangle + .0061\ e^{\ -154.49°i}|101\rangle + .7068\ e$$
$$^{101.21°i}|110\rangle + .0751\ e^{-32.68°i}|111\rangle$$
$$\cong (.6960|000\rangle + .7068|111\rangle)e^{-77.55°\ i} \qquad (3.18)$$

We next extended our scheme to two groups of four vectors each. Equations (3.19) and (3.20) show the input and output pairs. The e-H gate being trained is one for which an odd number of zeros are assigned to $|0\rangle_L$ (Section 3.2). We first trained for the first group of 4 vectors (equation (3.17)). Figure 3.9 shows the least RMS error for a four vector desired output matrix, which was 9.24%. Table 3.14 shows the parameters. Again, by using the phase-flip approach, we could obtain parameters for the group of vectors given by equation (3.19). The same RMS error of 9.24% was obtained.

$$\text{Input} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Desired Output} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \qquad (3.19)$$

$$\text{Input} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{Desired Output} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad (3.20)$$

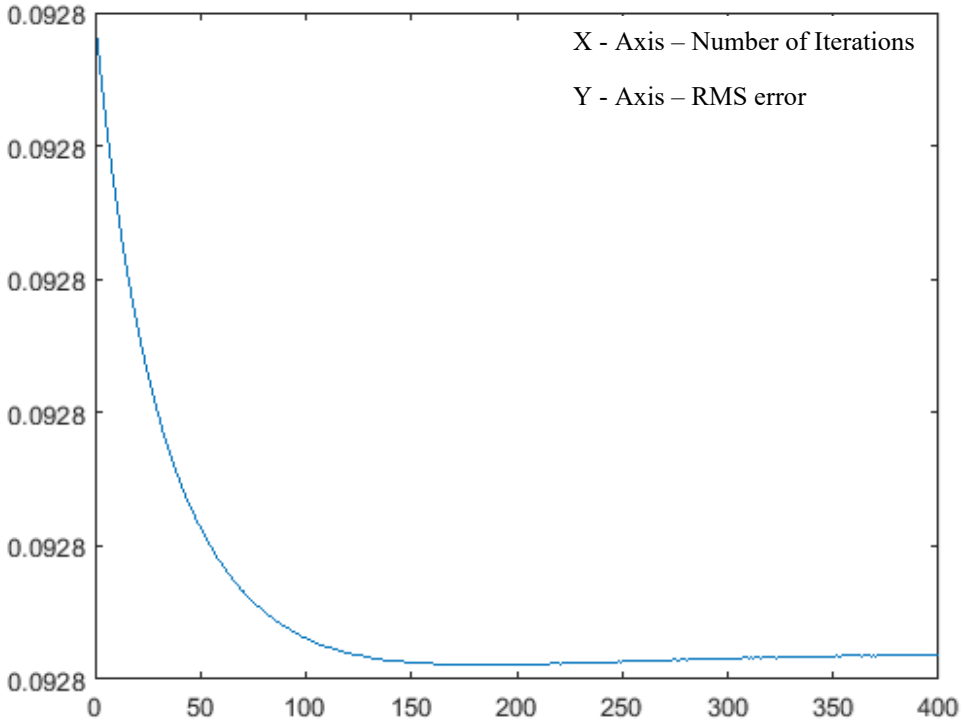


Figure 3.9. RMS error for four-vector training set based on odd and even desired output qubit system.

TABLE 3.14

TRAINING PARAMETERS FOR FIRST FOUR VECTOR PAIRS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3685$	$\varepsilon_a = -.5163$	$\xi_{a\text{-}b} = .5744$
$\Delta_b = .1517$	$\varepsilon_b = .5698$	$\xi_{b\text{-}c} = .2563$
$\Delta_c = .2407$	$\varepsilon_c = .0164$	$\xi_{a\text{-}c} = .0306$

TABLE 3.15

TRAINING PARAMETERS FOR SECOND FOUR VECTOR PAIRS (GHZ)

Tunneling	**Bias**	**Coupling**
$\Delta_a = .3685$	$\varepsilon_a = .5163$	$\xi_{a\text{-}b} = .5744$
$\Delta_b = .1517$	$\varepsilon_b = .5698$	$\xi_{b\text{-}c} = .2563$
$\Delta_c = .2407$	$\varepsilon_c = -.0164$	$\xi_{a\text{-}c} = -.0306$

3.5 Analysis of Vector Outputs with Respect to Each Pair

There is an interesting fact about the outcomes of the odd and even desired outputs which is worth mentioning here. As explained earlier, we successfully trained a three-qubit system pairwise for the first and the eighth vectors with training parameters shown in Table 3.5, and we obtained an RMS error of 2.19%. Initially, we were not able to reduce the RMS error for the second (second and seventh vectors) and fourth pairs (fourth and fifth vectors) below 17%. So we had to change the initial training parameters for these vectors in order to get an optimum RMS error. We used the training parameters used in Table 3.6 as initial values, and obtained successful results, wherein the RMS value decreased to 3% as shown in Figure 3.10.

If we compare Figure 3.10 with Figure 3.4, it is clear that training the three-qubit system with the set of parameters shown in Table 3.5 exhausts the system, and saturates the RMS error at 17%. On the other hand, training the system with the set of parameters shown in Table 3.6 gave a very low RMS error. After we tracked down the training results for the first and the eighth vectors, we concluded that any training parameter set with RMS error below 17% can be used for the second pair and the fourth pair as well, as Figure 3.11 describes.

Here, although we could find the critical point for training sets (star point shown in Figure 3.11), by evaluating training parameter values and phases, we observed that there is too much difference in values of different desired output vectors. Table 3.16 provides a comparison.

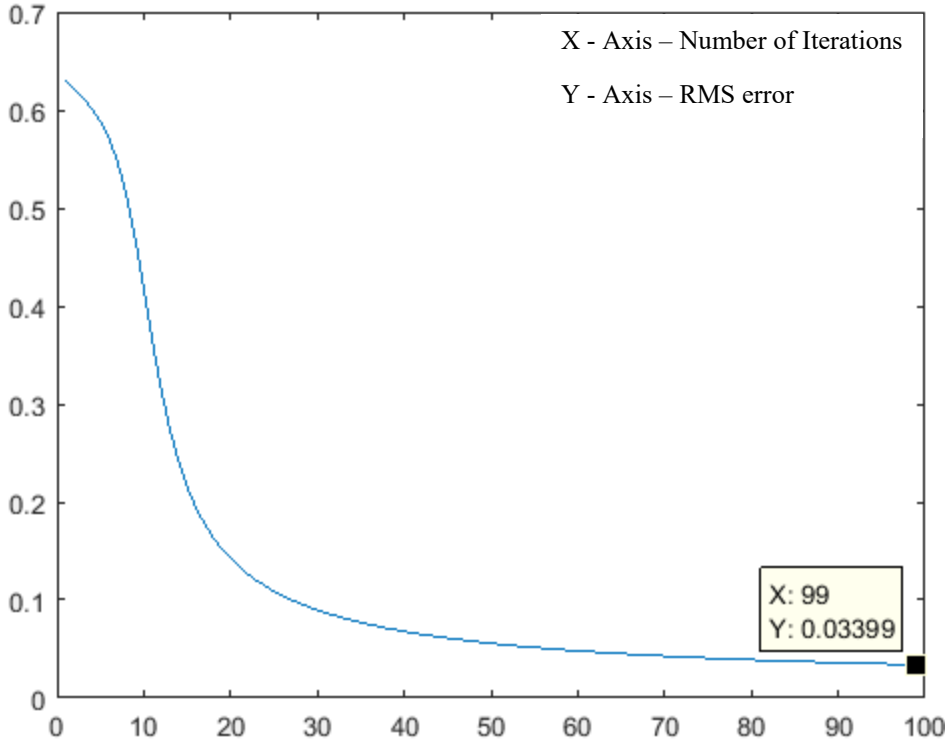


Figure 3.10. Training trend for the second and seventh vector desired output training set.

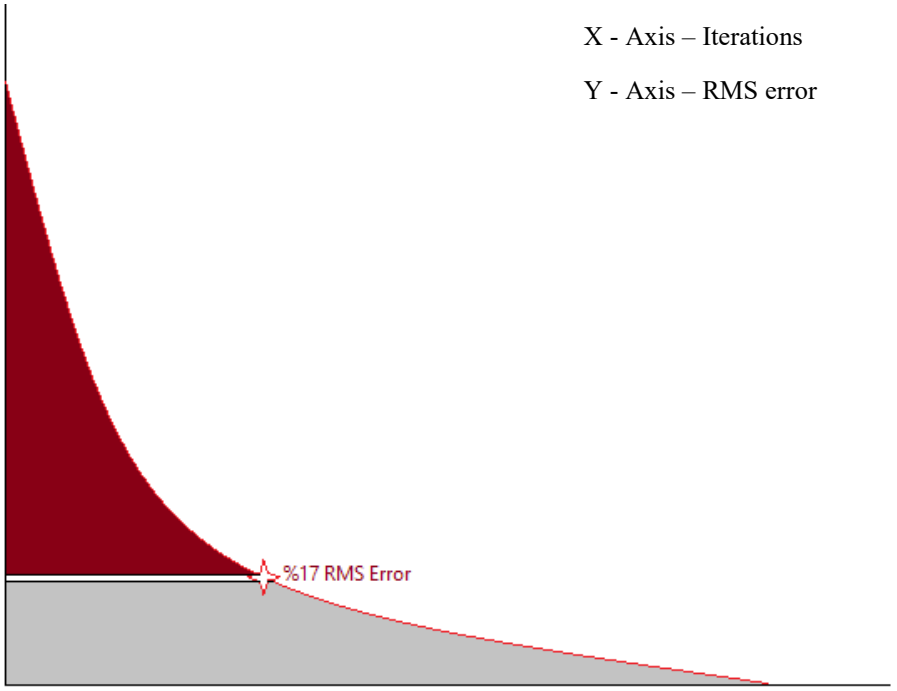


Figure 3.11. All training parameters that reach in the RMS error at the grey area (17% or less) can get used for training all other vectors.

TABLE 3.16

A GENERAL GLANCE OF TRAINING VALUES FOR ODD AND EVEN BASED DESIRED MATRIX (GHZ)

First Pair	$\Delta_a = .3030$, $\varepsilon_a = -.2047$, $\xi_{a-b} = .6069$ $\Delta_b = .3340$, $\varepsilon_b = .4076$, $\xi_{b-c} = .6069$ $\Delta_c = .3030$, $\varepsilon_c = -.2047$, $\xi_{a-c} = -.5133$	$\Delta_a = .2977$, $\varepsilon_a = -.2041$, $\xi_{a-b} = .5249$ $\Delta_b = .3017$, $\varepsilon_b = .2082$, $\xi_{b-c} = .6120$ $\Delta_c = .3372$, $\varepsilon_c = -.4049$, $\xi_{a-c} = -.6067$	**Fourth Pair**
Third Pair	$\Delta_a = .3030$, $\varepsilon_a = .2047$, $\xi_{a-b} = .6069$ $\Delta_b = .3340$, $\varepsilon_b = .4076$, $\xi_{b-c} = .6069$ $\Delta_c = .3030$, $\varepsilon_c = .2047$, $\xi_{a-c} = .5133$	$\Delta_a = .2977$, $\varepsilon_a = .2041$, $\xi_{a-b} = .5249$ $\Delta_b = .3017$, $\varepsilon_b = .2082$, $\xi_{b-c} = .6120$ $\Delta_c = .3372$, $\varepsilon_c = .4049$, $\xi_{a-c} = .6067$	**Second Pair**

3.6 Conclusion

A new scheme in quantum error correction to implement a partial e-H gate by using dynamic learning algorithm has been introduced. This was accomplished by training the quantum system using a back propagation technique, to find the system parameters to implement gate operations directly. We showed how the training algorithm can be used as a tool for finding the parameters to implement a partial e-H gate. Even though we were not able to find an e-H gate for a three-qubit system, we proved that it is possible to train for a partial e-H gate, where vectors were divided into groups of two and four vectors. Pairing the vectors in groups of twos further revealed a strange behavior in the quantum system. We found that two pairs of vectors could be matched, and by training for an e-H gate for one pair, we could find parameters for its matched pair, by simply negating three parameters. We also found that the trained parameters of an e-H gate for one pair could be used as initial parameters to train for another pair, for which the RMS error would not go below a fixed value (17%). This behavior of the quantum system needs to be explored as it can reveal interesting properties of the parameter space, which might be scalable to larger quantum systems.

CHAPTER 4

CNOT GATE OPERATION BETWEEN UNCOUPLED QUBITS

Most current technologies for quantum computing are based on nearest neighbor arrangements of qubits, wherein any interaction between qubits can only be between nearest neighbors (see Figure 4.1). As such, when implementing a two-qubit gate operation like the CNOT in an N-qubit linear array between two qubits that are not adjacent to each other, a re-ordering of qubits is required to bring the two qubits adjacent to each other. This re-ordering is accomplished using a sequence of Swap operations, each of which comprises of three CNOT gates (see Chapter 1). Depending upon the location of the two qubits in the linear array, this can greatly increase the gate count. The aim of this part of the research was to find system parameters to implement CNOT gate operations between qubits that are not adjacent to each other. Two methods were investigated. The first method was to use the dynamic learning algorithm discussed in Chapter 2 to train the network to find system parameters. The second method, which we call a "brute force method" is a new method we used to scan the parameter space to find the optimal set of parameters. Here, no training is used. We divide the entire parameter space into a fine grid, and evaluate the error at each grid point. Each grid point corresponds to some set of parameter values. The grid points that give the minimum error, are then used as initial parameters, which are then fed to the neural network for training.

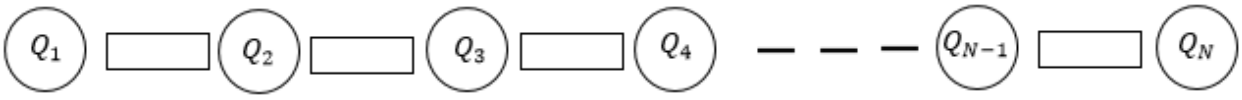


Figure 4.1. Linear nearest neighbor array comprising of N-qubits. Circles represent qubits and rectangles represent couplings between adjacent qubits. Each qubits can only only interact with its nearest neighbors.

4.1 CNOT Gate Operation using The Dynamic Learning Algorithm: 2-Qubit CNOT Gate Operation

Here, we implemented a CNOT gate operation in a two-qubit system by using the dynamic learning algorithm as a tool. To demonstrate this, a two-qubit system is defined by equation (3.5) with N=2. The training set compromised of four input-output pairs. The inputs were the four computational basis states, |00> through |11>. The outputs were the expected state vectors after applying the transformation given by equation (1.10) to the corresponding inputs. For instance, the output states for the input states |01> and |11> were |01> and |11>, respectively. For training, we followed the procedure described in Chapter 2. The total time, t_f=1ns, which is the time step for the gate operation. Table 4.1 shows the parameters of the trained network.

TABLE 4.1

PARAMETERS OF A CNOT GATE OPERATION BETWEEN QUBITS Q_1 AND Q_2 (GHZ)

Δ_a	.0021
Δ_b	.2384
ε_a	.1053
ε_b	.1966
$\xi_{a\text{-}b}$	.1973
RMS error	.0061

4.2 3-Qubit CNOT Gate Operation between Uncoupled Qubits

Suppose we want to implement a CNOT gate between two next-to-nearest neighbor qubits Q_1 and Q_3 (Figure 4.2), without requiring the qubits to be adjacent to each other. Here, Q_3 is the target qubit. The CNOT gate operation between qubits Q_1 and Q_3 can be described as:

$$|q_1 q_2 q_3 >= \begin{cases} |q_1 q_2 q_3 > , & \text{if } |\text{q} \geq |0 > \\ |q_1 q_2 q_3{}' > , & \text{if } |\text{q} \geq |1 > \end{cases} \tag{4.1}$$

where $|q_i>$ corresponds to state of qubit Q_i, i = 1 to 3, and $|q_3'>$ is the complement of state $|q_3>$ (for instance, if $|q_3>$ =|1>, $|q_3'>=|0>$). To implement the gate operation as in Figure 4.2, traditionally, two additional Swap gates are required to bring qubits Q_1 and Q_3 adjacent to each other. If each Swap gate is decomposed into 3 CNOT gates, the gate count can increase to 7. In addition to increasing the computational overhead, such multi-gate decompositions can also increase the probability of propagating errors through the circuit.

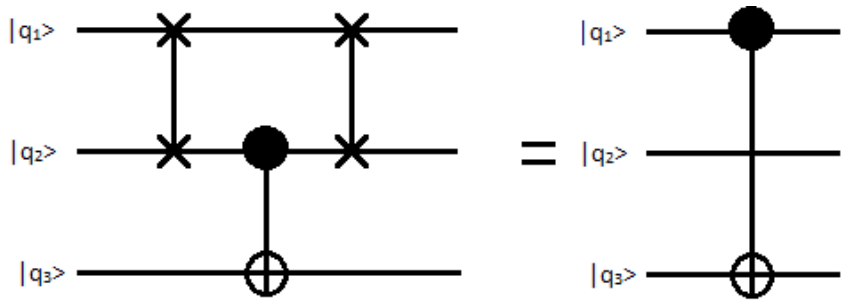


Figure 4.2. Implementation of CNOT gate operation between qubits Q_1 and Q_3 using conventional methods where states of qubits Q_1 and Q_2 are swapped before and after CNOT gate.

The neural network was trained so that a CNOT gate can be implemented between qubits 1 and 3 directly. The same process as training two-qubit CNOT gate operation was followed. The total time, t_f = 5.3ns, which is the time step for the gate operation. Training was stopped when the RMS error was 23.35%. Table 4.2 shows the parameters of the trained network.

TABLE 4.2

PARAMETERS OF A CNOT GATE OPERATION BETWEEN QUBITS Q_1 AND Q_3 (GHZ)

Δ_a	0
Δ_b	.2731
Δ_c	.1148
ε_a	.1053
ε_b	.1601

TABLE 4.2 (continued)

ε_c	.0807
$\xi_{a\text{-}b}$	.1601
$\xi_{b\text{-}c}$	.0715
RMS error	.2335

4.3 4-Qubit CNOT Gate Operation between Uncoupled Qubits

Next, we used the dynamic learning algorithm to implement a CNOT gate directly in a four-qubit system between two remote, uncoupled qubits Q_1 and Q_4 (Figure 4.2), without requiring the qubits to be adjacent to each other. Here, Q_4 is the target qubit. The CNOT gate operation between qubits Q_1 and Q_4 can be described as:

$$|q_1 q_2 q_3 q_4> = \begin{cases} |q_1 q_2 q_3 q_4>, & \text{if } |q_1 \geq |0> \\ |q_1 q_2 q_3 q_4'>, & \text{if } |q_1 \geq |1> \end{cases} \tag{4.2}$$

Once again, to implement the gate operation as in Figure 4.2, four additional Swap gates are required to bring qubits Q_1 and Q_4 adjacent to each other. The dynamic learning algorithm is used to overcome these Swap gates. The total time, t_f = 5.3ns, which is the time step for the gate operation. Training was stopped when the RMS error was 31.36%. Table 4.3 shows the parameters of the trained network.

TABLE 4.3

PARAMETERS OF TRAINED NETWORK TO REALIZE A CNOT GATE OPERATION BETWEEN QUBITS Q_1 AND Q_4 (GHZ)

Δ_a	0
Δ_b	.3891
Δ_c	1.9511
Δ_d	.1245
ε_a	1.5

TABLE 4.3 (continued)

ε_b	.9551
ε_c	.9264
ε_d	.0006
$\xi_{a\text{-}b}$	.9548
$\xi_{b\text{-}c}$	.7038
$\xi_{c\text{-}d}$	.1866
RMS error	.3136

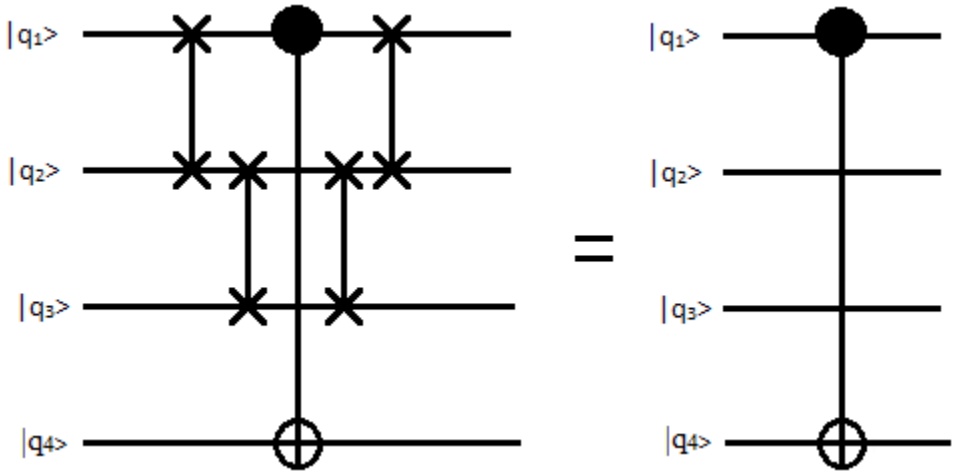


Figure 4.3. Implementation of CNOT gate operation between qubits Q_1 and Q_4 using conventional methods where the states of qubits Q_1, Q_2 and Q_3 are swapped before and after the CNOT gate.

4.2 Brute Force Method

In this section, we add a new scheme to the dynamic learning algorithm for implementing the CNOT gate operation between remote qubits. So far in this chapter, we showed how to implement a CNOT gate operation in a two, three and four-qubit system using the dynamic learning algorithm. As the number of qubits increase, the number of system parameters also increase, which makes the training process complex and hard to converge. Equation 4.3 shows

the number of parameters in each system with respect to the number of qubits in the quantum system.

$$P = 3N - 1 \tag{4.3}$$

where N is the number of qubits and P is the number of parameters need to be trained. For example, to train a four-qubit quantum system, we need to adjust eleven different parameters such as tunnelings, biases and couplings, which makes the training process very complicated. To avoid this problem, we added the brute force method as a first step of the dynamic learning algorithm to ease the training process. The brute force method helps the dynamic learning algorithm to find an optimum initial set of parameters for training. The goal behind this technique is oftentimes during training, it was found that the RMS results showed great variation depending on what parameters were initially used. Therefore, we decided to find a method of finding and assigning the most optimal set of initial parameters for training. By adding this method to our algorithm, first, we increase the chance of achieving an RMS error in a shorter training period, and second, we can investigate the behavior of a quantum gate within a fixed period of frequency range for all parameters.

Figure 4.4 shows how a "trainable" quantum system uses the dynamic learning algorithm to gradually progress towards the global minimum of the system. Here, each contour corresponds to an error value (without using brute force). All points along a contour constitute a set of parameter values that give that value of RMS error. If the network is not trainable, the system just gets stuck on one of the error contours and never reaches the global minimum. In Figure 4.5, we illustrate what the brute force method does to the parameter space. In this method, the entire parameter space is divided into a grid, and the error is evaluated at each grid point or node. Each node of the grid represents an initial set of parameters. The brute force method scans the entire

parameter space, and evaluates the RMS error at each node for one iteration. Of all the error values calculated at each node, it finds the node at which the minimum RMS error occurs. This node then constitutes the optimal set of parameters, which is then fed to the dynamic learning algorithm as initial parameters.

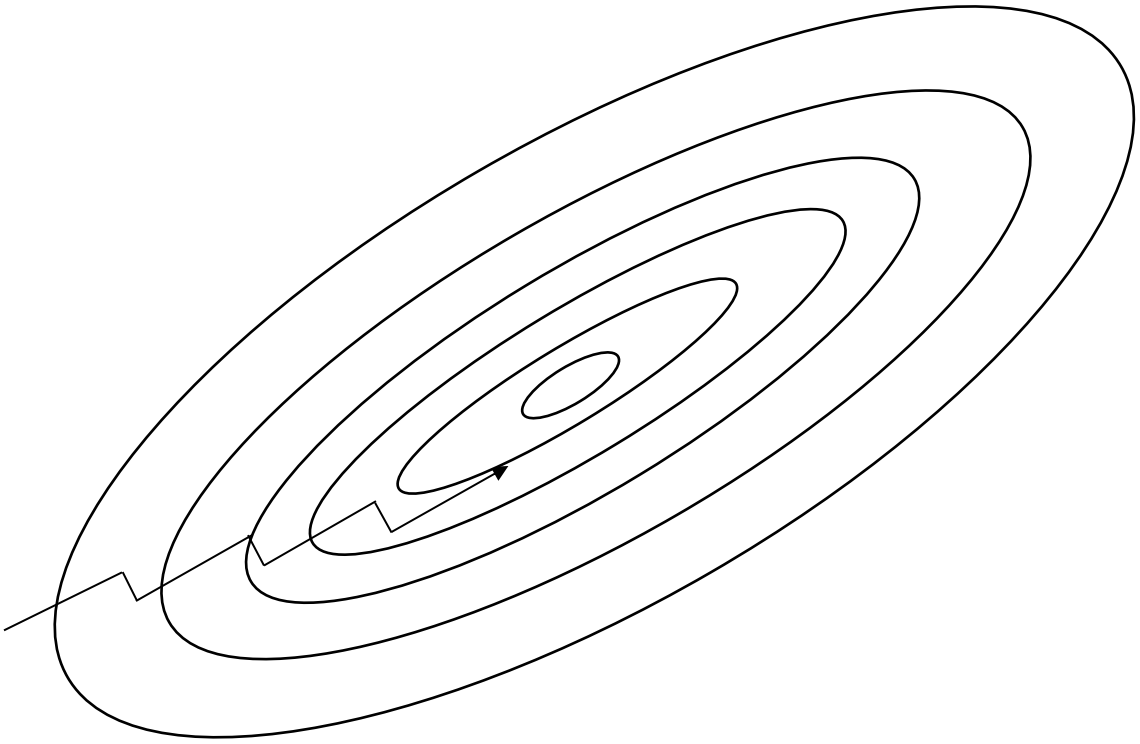


Figure 4.4. Learning contour with training trend progresses toward minimum local error.

There is a simple method to visualize how the brute force method evaluates parameter values at each node. The method can be thought of as creating a truth table for all initial parameters. For example, in very simple case assume the frequency range for each parameter in the initial set is 0-250 MHZ, and grid size (resolution) is 250 MHZ. Here, in order to avoid having a big truth table we introduced only 3 variables named B, C and D in Table 4.4.

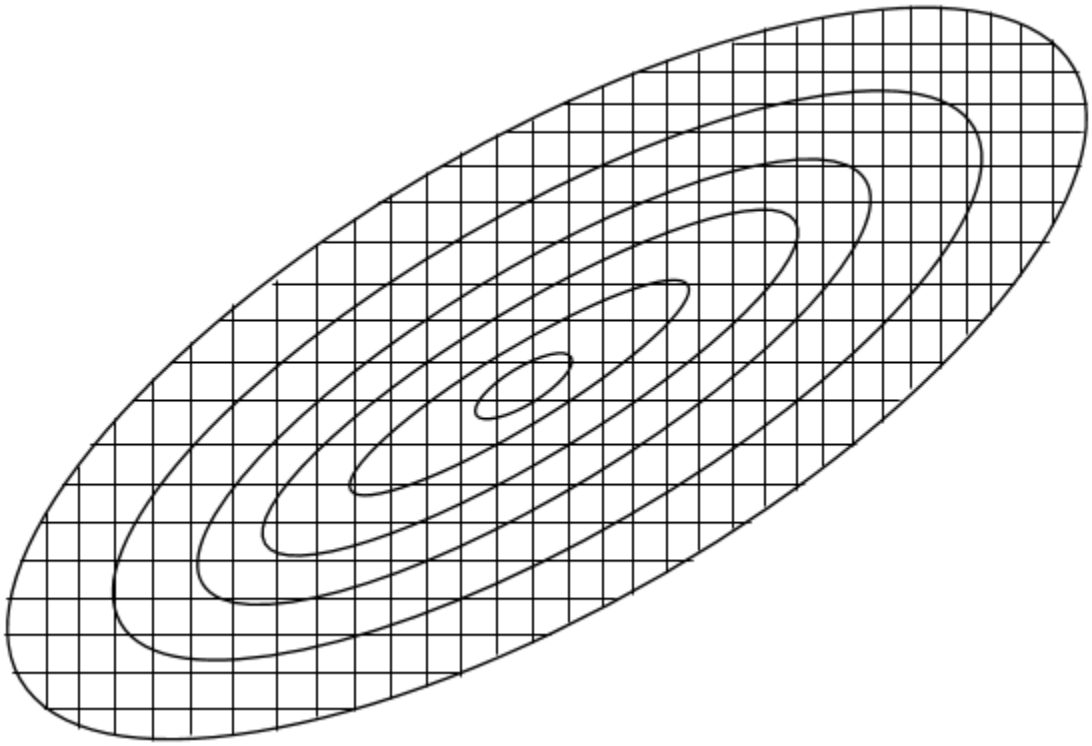


Figure 4.5. Plotting initial training spots in contour using brute method.

Each variable is a parameter of the system Hamiltonian. Initially, all parameter values are zero (first row of truth table). Next, parameter D is “switched on”. Since the step size value is 250 MHz (or 0.25 GHz), it is assigned a value of 0.25 (GHz) in Table 4.4. Next, parameter D is “switched off”, but parameter C is “switched on”, which explains the values in the third row. This process is carried out until we cover the entire truth table, which is equivalent to covering the entire grid in parameter space. Note that here we chose a large resolution in order to demonstrate how the brute force technique employs a truth table approach for covering the entire grid. Each row of the truth table corresponds to one node, and hence, to one iteration for calculating the RMS error at that node. As the number of parameters increase, the number of iterations increase.

TABLE 4.4

TRUTH TABLE SAMPLE FOR BRUTE FORCE FEATURE (GHZ)

B	C	D
0	0	0
0	0	.25
0	.25	0
0	.25	.25
.25	0	0
.25	0	.25
.25	.25	0
.25	.25	.25

Equation (4.4) describes the number of iterations (initial parameters) with respect to the number of qubits:

$$A = S^P \tag{4.4}$$

where A is the number of iterations for each training set, S is the number of steps in a fixed frequency range of parameters, and P is the number of parameters for the quantum system that is calculated using equation (4.3).

2-Qubit CNOT Gate Operation using Brute Force Method

Figure 4.6 shows the training results using the brute force method for finding the optimum initial training set of parameters for a two-qubit quantum system. Here, the resolution for the grid was 25MHz, and the range for each parameter was from 0 to 2GHZ. That is, each parameter was incremented in steps of "25MHz" starting from 0 all through 2 GHz. The manner in which each parameter was incremented followed a truth table similar to the one described in

Table 4.4, only much larger. Note that the finer the grid (higher the resolution), the higher the chance of finding the optimal node, the parameters for which were used as initial parameters for training. However, it was found that the optimal nodes also happened to be points at which the RMS error was almost zero without training. As such, we did not need to train the network any further, and the brute force method itself was sufficient to find the trained system parameters. However, this was not the case for the three- and four-qubit system, as is expected. In systems with larger number of qubits, training the initial set of parameters is needed because adding more qubits to the system brings more complexity to the system.

Here we trained two-qubit quantum system where (according to equation (4.4) P=5, with S=9 and A=59049. Figure 4.6 shows the RMS error calculated at each node using the brute force method. The best RMS error occurred in the 51040^{th} iteration, and Table 4.2 lists the parameter values. Once again, simulations were run to verify these results (Hamiltonian given by equation (3.5) for N = 2). Simulations confirmed the CNOT gate operation for the parameter values listed in Table 4.2.

There are two remarkable features about the brute force methods, which are significant. One is that, in [36], Kumar et al. proposed an analytical solution for a CNOT gate in a two-qubit system. The solution obtained using the brute force technique was the exact solution as obtained in [36], even though two completely different approaches were used. This is worth investigating because it may mean that analytical solutions perhaps always exist that allow one to find system parameters for a CNOT gate in multi-qubit gate operation. While it is hard to find an analytical solution using the method proposed in [36] for higher than two-qubit systems, the brute force method might offer a way, which needs to be investigated. The second significant feature is that, on observing Figure 4.6, we notice a periodic pattern in the RMS error values. This needs to be

investigated to see if a periodic trend always exists, which might offer a way to evaluate parameter values at low error points.

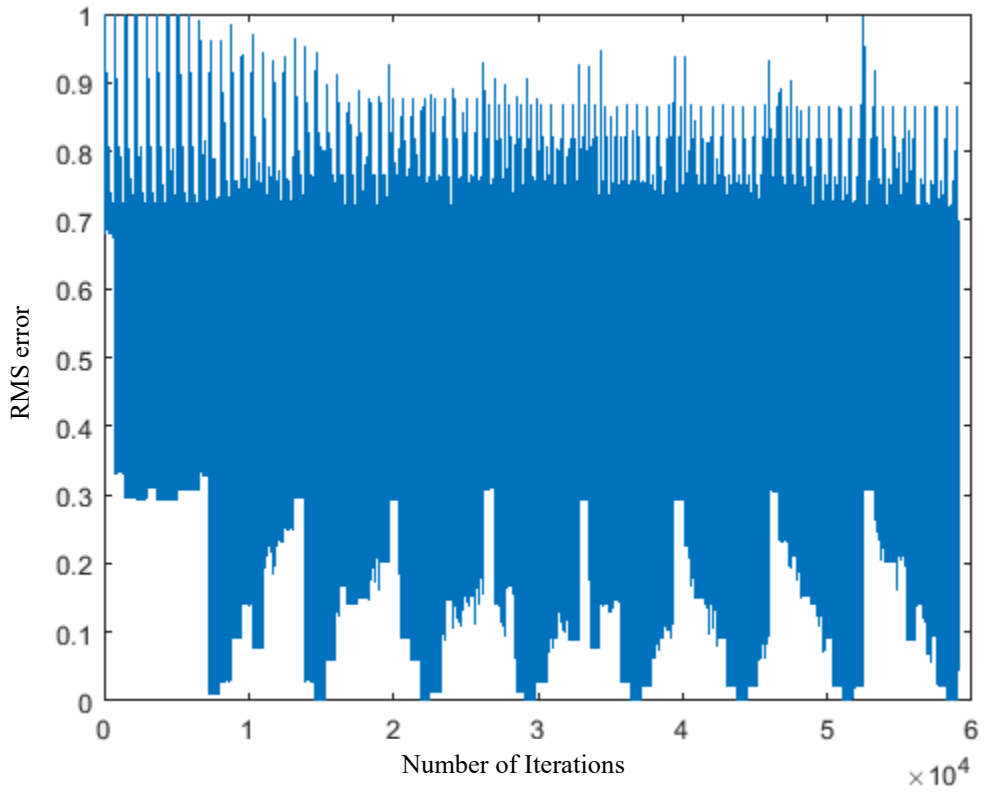


Figure 4.6. Brute force plot in a two-qubit quantum system.

TABLE 4.5

PARAMETERS OF TRAINED TWO QUBIT NETWORK TO REALIZE A CNOT GATE OPERATION USING BRUTE FORCE FEATURE (GHZ)

Δ_a	0
Δ_b	.25
ε_a	0
ε_b	1.75
$\xi_{a\text{-}b}$	1.75
RMS error	8.8406e-6

3-Qubit CNOT Gate Operation using Brute Force Method

Here, we trained the three-qubit quantum system using parameters from the two-qubit quantum system (Table 4.1). In other words, we added 3 new parameters to the existing parameters of the two-qubit system. Each of these parameters corresponded to the third qubit (bias and tunneling for the third qubit, and an additional coupling between qubits 2 and 3). The brute forced method was used to find the best initial parameters for training. However, while using the brute force method, only the three new parameters were varied, while all parameters from the two-qubit system were kept unchanged. Once we found the node at which the RMS error was minimum, the parameters corresponding to this node were used as initial parameters for the neural network. Table 4.5 lists the parameters for the trained network. Here, the total time, t_f = 5.3ns, which is the time step for the gate operation. The RMS error decreased to4.7%. Recall that when we trained the neural network directly (Table 4.2 lists the parameters), the RMS error did not go below 23.35%. Using the brute force method could reduce the RMS error for implementing CNOT gate operation in the three-qubit quantum system from 23.35% to only 4.7%, while both systems trained under identical conditions (learning rates, etc.).

TABLE 4.6

PARAMETERS OF TRAINED NETWORK TO REALIZE A CNOT GATE OPERATION BETWEEN QUBITS Q_1 AND Q_3 (GHZ)

Δ_a	0
Δ_b	.362
Δ_c	1.974
ε_a	1.5
ε_b	.822
ε_c	.5162
$\xi_{a\text{-}b}$	.75

TABLE 4.6 (continued)

$\xi_{b\text{-}c}$	1.5
RMS error	.047

4-Qubit CNOT Gate Operation using Brute Force Method

Here, we implemented a CNOT gate in a four-qubit quantum system using the brute force method for finding the initial parameters. We followed the same process as implementing a CNOT gate operation in a three-qubit system. Table 4.6 lists all the parameters for the trained network.

TABLE 4.7

PARAMETERS OF TRAINED NETWORK TO REALIZE A CNOT GATE OPERATION BETWEEN QUBITS Q_1 AND Q_4 (GHZ)

Δ_a	.0044
Δ_b	.0118
Δ_c	1.8897
Δ_d	.5022
ε_a	1
ε_b	.3942
ε_c	.1234
ε_d	.1138
$\xi_{a\text{-}b}$	.3980
$\xi_{b\text{-}c}$	.0972
$\xi_{c\text{-}d}$	.5004
RMS error	.148

Once again, by adding brute force method to the dynamic learning algorithm, we could reduce the RMS error from 31.36% to 14.8% for implementing the CNOT gate operation in a four-qubit quantum system, while both systems trained under identical conditions.

4.3 Conclusion

In this chapter, we introduced a new method to implement a CNOT gate operation in a two-qubit and multi-qubit quantum system using LNN architecture. First, we trained the quantum systems using a dynamic learning algorithm as described in Chapter 2 for implementing CNOT gate operations between uncoupled qubits. Second, we added the brute force method to the dynamic learning algorithm to achieve lower RMS errors. The brute force method allowed us to scan the parameter space to find the optimal set of initial parameters, which were then fed into the learning algorithm for further training. The helped reduce the RMS error drastically. Finally, to demonstrate the improvement of our learning algorithm using brute force method, we compared all outcomes for two-qubit, three-qubit and four-qubit quantum systems, with and without using the brute force method. There are several significant features of this method like periodicity in the error in parameter space and convergence to actual analytical solutions that need to be explored as future work.

REFERENCES

REFERENCES

[1] A. Khalid, *A Gentle Introduction to Quantum Computing*, School of Science and Engineering Lahore University of Management Sciences, Lahore, Pakistan, 2012.

[2] R. D. Wolf, "Quantum computing lecture notes," Centrum Wiskunde & Informatica, Amesterdam, Netherlands, 2016.

[3] R. C. Garigipati, Doctor of Philosophy Dissertation "Implementation of quantum gate operarions using a dynamic learning algorithm," Wichita State University, Wichita, Kansas 2015.

[4] S. Daraeizadeh, Master of Science Thesis, "Efficient implementation of multi-control Toffoli gates in linear nearst neighbor arrays," Wichita State University, Wichita, Kansas 2014.

[5] X. Ni, Master of Science Thesis "Quantum computation with commuting operations," Technical University of Munich 2012

[6] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N, Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A,* vol. 52, p. 3457, 1995.

[7] G. Song, and A. Klappeneckerm, "Optimal realizations of controlled unitary gates," *Quantum Information and Computation,* vol. 3, p. 139, 2002.

[8] M. A. Nielsen, and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University , 2000.

[9] D. Maslov, G. W. Dueck, D. M. Miller and C. Negrevergne, "Quantum circuit simplification and level compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 3, p. 436, 2008.

[10] S. S. Bullock, and I. L. Markov, "Smaller circuits for arbitrary N-qubit diagonal computations," *Quantum Information & Computation,* vol. 4, no. 1, p. 27, 2004.

[11] G. Song, and A. Klappenecker, "Optimal realizations of controlled unitary gates," *Quantum Information and Computation,* vol. 3, no. 2, p. 139, 2003.

[12] F. Vatan, and C. Williams, "Optimal quantum circuits for general two-qubit gates," *Physical Review A,* vol. 69, no. 3, p. 032315, 2004.

[13] D. Maslov, and M. Saeedi, "Reversible circuit optimization via leaving the boolean domain," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 6, p. 806, 2011.

[14] R. Wille, D. Große, G W. Dueck, and R. Drechsler, "Reversible logic synthesis with output permutation," *22nd International Conference on VLSI Design*, p. 189, 2009.

[15] W. Hung, X. Song, G. Yang, J. Yang and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 9, p. 1652, 2006.

[16] P. Gupta, A. Agrawal and N. Jha, "An algorithm for synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 11, p. 2317, 2006.

[17] S. J. Devitt, W J. Munro and K. Nemoto, "Quantum error correction for beginners," *Reports on progress in physics,* vol. 76, no. 7, p. 076001, 2013.

[18] A. M. Steane, "Efficient fault-tolerant quantum computing," *Nature,* vol. 309, p. 124, 1999.

[19] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing,* Vol. 26, p. 1484, 1997.

[20] J. Zhang, R. Laflamme, and D. Suter, "Experimental implementation of encoded logical qubit operations in a perfect quantum error correcting code," *Physical Review Letter,* vol. 109*, p. 100503,* 2012.

[21] D. Touchette, H. Ali, and M. Hilke, Doctor of Philosophy Dissertation "Five-qubit quantum error correction in a charge qubit quantum computer," Doctor of Philosophy Dissertation, Yale University 2010.

[22] M. Nakahara, and T. Ohmi, *Quantum Computing from Linear Algebra to Physical Realization,* CRC Press 2008.

[23] R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, vol. 21, p. 467, 1982.

[24] M. Schlosshauer, *The quantum-to-classical transition and decoherence,* Springer*,* 2014.

[25] B. E. Kane, "A silicon-based nuclear spin quantum computer," *Nature,* vol. 393, p. 133, 1998.

[26] E. C. Behrman, J. E. Steck, P. Kumar, and K. A. Walsh*,* "Quantum algorithm design using dynamic learning," *Quantum Information and Computation,* vol. 8, p. 12, 2008.

[27] R. Josa, "Fidelity for Mixed Quantum States," *Journal of Modern Optics,* vol. 41, p. 2315, 1994.

[28] M. Friesen, P. Rugheimer, D. E. Savage, M. G. Lagally, D. W. van der Weide, R. Joynt, and M. A. Eriksson, "Practical design and simulation of silicon-based quantum dot qubits," *Physical Review B,* vol. 67, p. 121301, 2003.

[29] T. D. Ladd, J. R. Goldman, F. Yamaguchi, Y. Yamamoto, E. Abe, and K. M. Itoh, "An all silicon quantum computer," *Physical Review Letters,* vol. 89, p. 017901, 2002.

[30] E. Novais, and A. H. C. Neto, "Nuclear spin qubits in a pseudo-spin quantum chain," *Physical Review A,* vol. 69, p. 062312, 2004.

[31] L. Tian, and P. Zoller, "Quantum computing with atomic josephson junction arrays," *Physical Review A,* vol. 68, p. 042321, 2003.

[32] S. H. W. Ploeg, A. Izmalkov, A. M. Brink, U. Huebner, M. Grajcar, E. Il'ichev, H.-G. Meyer, and A. M. Zagoskin, "Controllable coupling of superconducting flux qubits," *Physical Review Letters,* vol. 98, p. 057004, 2007.

[33] J. Lantz, M. Wallquist, V. S. Shumeiko, and G. Wendin "Josephson junction qubit network with current-controlled interaction," *Physical Review B,* vol. 70, p. 140507, 2004.

[34] R. Stock, and D. F. V. James, "Scalable, high-speed measurement-based quantum computer using trapped ions," *Physical Review Letters,* vol. 102, p. 170501, 2009.

[35] R. V. Meter, T. D. Ladd, A. G. Fowler, and Y. Yamamoto, "Distributed quantum computation architecture using semiconductor nanophotonics," *International Journal of Quantum Information,* vol. 8, p. 295, 2010.

[36] P. Kumar, and S. Skinner, "Universal quantum computing in linear nearest neighbor architecture," *Quantum Information and Computation,* vol. 11, p. 0300, 2010.