Quantum Error Correction using Quantum Convolutional Neural Network

by

Niloy Deb Roy Mishu
17301081

Fatema Islam Meem
21141074

A. E. M Ridwan
17301208

Mohammad Mushfiqur Rahman
17301097

Mekhala Mariam Mary
17101368

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
BRAC University
June 2021

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Niloy Deb Roy Mishu

Niloy Deb Roy Mishu
17301081

Fatema Islam

Fatema Islam Meem
21141074

A.E.M Ridwan

A. E. M Ridwan
17301208

Mohammad Mushfiqur Rahman

Mohammad Mushfiqur Rahman
17301097

Mariam

Mekhala Mariam Mary
17101368

Approval

The thesis/project titled " Quantum Error Correction using Quantum Convolutional Neural Network " submitted by

1. Niloy Deb Roy Mishu (17301081)
2. Fatema Islam Meem (21141074)
3. A. E. M Ridwan (17301208)
4. Mohammad Mushfiqur Rahman (17301097)
5. Mekhala Mariam Mary (17101368)

Of Summer, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on June 2, 2021.

Examining Committee:

Supervisor:
(Member)

Shahnewaz Ahmed

Shahnewaz Ahmed
Lecturer
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

[illegible]

Sowmitra Das
Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

[illegible]

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Quantum computing with its powerful attributes - entanglement and superposition, is revolutionizing modern computation. Moreover, quantum computation can be applied to a wide range of real-world applications, including cybersecurity and cryptography, computational chemistry, artificial intelligence, prime factorization, and so on. However, the biggest impediment of quantum computation is quantum error. Often the decoherence caused by the environment or any other malfunction creates quantum errors which can arbitrarily change the state of a quantum system and destroying its information content - bit flip & phase flip error, unwanted measurement error, etc. A specific Quantum Error Correction (QEC) code can rectify some particular errors. But, it is usually not optimized for any random error. As of today, 'Deep Learning' techniques in analyzing data have been very promising which is influencing researchers to apply these methods to 'Quantum Computation' problems. We have implemented a Quantum Convolutional Neural Network (QCNN) using Parameterized Quantum Circuit (PQC) on IBM's open source Quantum Computing framework QISKIT. The generic structure of the QCNN consists of variational forms of the encoder and decoder of the error correction code, which is optimized during training. In this way, it constructs a quantum error correction method for a certain error model. We were able to retrieve quantum states with as much as 90% fidelity rate from the experiments. As a result, our model achieves a high fidelity while using relatively few parameters, which can be generalized for any error model.

Keywords: Quantum Computing, Quantum Machine Learning, Qiskit, Deep Neural Network, CNN, QCNN.

Acknowledgement

We are highly indebted to our advisors Mr. Sowmitra Das and Mr. Shahnewaz Ahmed of the Department of Computer Science and Engineering, BRAC University. They were immensely supportive whenever we sought help. Their knowledge, guidance and dedication made this research possible. We would also like to show our gratitude to Mr. Shadman Shahriar and Mr. Wasif Ahmed of the Department of Computer Science and Engineering, BRAC University. They instructed and motivated us with the fundamental knowledge of Quantum Computing. Their guidance was extremely vital for our research.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
$COBYLA$	Constrained Optimization By Linear Approximation optimizer
MLP	Multi-layered Perceptron Network
PQC	Parameterized Quantum Circuit
$QCNN$	Quantum Convolutional Neural Network
QEC	Quantum Error Correction
QML	Quantum Machine Learning
$ReLU$	Rectified Linear Unit
$SPSA$	Simultaneous Perturbation Stochastic Approximation
VQE	Variational Quantum Eigensolver

Chapter 1

Introduction

Quantum computing is the field of study based on principles of quantum theory which focuses on computer-based technology. Quantum theory is a field of physics that interprets the phenomena at the molecular and atomic levels of energy and matter. Quantum computing uses quantum bits to perform computational tasks with has higher efficiency than any classical computation. It enables us the door to have incredible computational capability with obtaining higher performance in any task that needs computation. Over the last few years, researchers have been tremendously using their efforts to build an outstanding quantum computer. Already Quantum Supremacy is achieved by Google (Arute et al., 2019) [1] and many other companies like IBM and Amazon are building full-fledged quantum computers. Moreover, researchers have been already using these quantum computers to use their massive processing power in different fields such as Artificial Intelligence, Machine Learning, Financial Modeling, Cryptography, Medicine development, etc.

1.1 Problem Statement

Quantum computers are have a lot of potential but they still has to wait a long time to gain success as many issues are restricting the path. Normally, classical computers use binary bits to process where quantum computers use 'Qubits' as an example IBM's Paris has 53 qubits and IBM is trying to increase the number. However, the problem is while working with qubits, the quantum system becomes highly fragile and unstable due to noises, faults of quantum decoherence and fails apart before any completion of any program. These effects mostly occur by temperature fluctuations, electromagnetic waves, and other interactions with the outside environment. Because of these noises, the quantum computer fails to protect quantum information as the noises tear apart the nature of the quantum properties. To preserve any quantum information from errors due to decoherence and noises, we need fault-tolerant quantum computation. We need Quantum error correction (QEC) to achieve correct information. Hence, the error correction schemes in a modest execution time are essential for our calculation. Therefore, we hope to achieve a higher accurate error correction with our proposed implementation with QCNN which can be a breakthrough in quantum computing.

1.2 Research Objectives

Quantum computers have the ability to perform more complex calculations at much better performance, including solving problems that are simply beyond the ability of traditional computers. Despite the extraordinary, exceeding computation ability of quantum computing, we have discussed the main drawback of a quantum computer is quantum error. This is because circuits in quantum computers are based on qubits which are extremely fragile and prone to errors due to the interactions with the external environment like decoherence and other quantum noise. Here, decoherence refers to the loss of information from the quantum computer that limits our potential for accurate quantum computation. Therefore, the accuracy of quantum computation depends on how we handle the errors that occur from decoherence and other quantum noise (Gottesman & Daniel, 2009) [2]. As it is seen that quantum shows enormous promise, the quantum industry is in the way to storm the world with a revolution. Nonetheless, this giant leap of reaching the milestone is being held back with a significant challenge: quantum error that we have previously discussed. Hence, this intriguing challenge inspired us to approach this problem which is quantum error correction. To achieve that, we can use QCNN. Normally, every system is loosely coupled with the energetic state of its surroundings and when viewed in isolation, the system's dynamics are non-unitary. Thus the dynamics of the system alone are irreversible. As with any coupling, entanglements are generated between the system and the environment. These have the effect of sharing quantum information with or transferring it to the surroundings. Simply put, to accurately correct the error we need to preserve the coherence by mitigating the decoherence effects (Schindler et al, 2011) [3]. Here, we use Convolutional Neural Networks (CNN) on a quantum computer. CNN has been used widely for data accuracy and the reason why CNN is hugely popular because of its architecture and its flexibility on image data (Saggio et al., 2019) [4]. The Quantum Convolutional Neural Network (QCNN) is motivated by the classical CNN which uses the variational parameters for input sizes only of N Qubits. This grants the implementation of realistic, near-term quantum devices efficiently. The architecture of QCNN combines the multiscale entanglement and helps to achieve successful quantum error correction (Cong et al., 2018) [5]. This development provides a promising way of robust quantum computing.

Chapter 2

Quantum Computing

2.1 Quantum Computing

Quantum mechanics is a mathematical explanation of the construction of quantum theories. In the 1970s, the idea of combining quantum physics and information theory arose. The theory gained traction in 1982, when physicist Richard Feynman argued that traditional computing couldn't handle computations describing quantum processes. This section provides a basic theory of quantum computers by introducing two major principles of quantum mechanics: superposition and entanglement which are crucial for their operation (Noson & Mirco, 2009) [6].

2.2 Qubits and Quantum Gates

Although quantum and conventional computers both complete computational problems, the mechanisms for storing and processing data differ. Classical computation deals with bits. A bit represent a unit of information in a two-dimensional classical system. To express a bit, we represent 0 or, better, the state $|0\rangle$ – as a 2-by-1 matrix with a 1 in the 0's row and a 0 in the 1's row. Similarly , 1 can be illustrated by 2-by-1 matrix with a 0 in the 0's row and a 1 in the 1's row.

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

Quantum computer deals with **qubits**. A quantum bit or a qubit represents an unit of information in two-dimensional quantum system. A qubit as a 2-by-1 matrix with complex numbers:

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

The key distinction between classical bits and qubits is their state. A qubit can stay in a state other than $|0\rangle$ or $|1\rangle$. A qubit can possibly be formed in a linear combinations of states. This property is called **superpositions**:

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle : \text{such that } \|c_0\|^2 + \|c_1\|^2 = 1$$

Entanglement is generally that state which cannot be described or expressed as these states. Vidal, G. er al. (2003) states that entanglement is one of the most fascinating features of quantum theory [7]. Entanglement is vital for quantum information science. Also, it provides a critical role in the study of quantum phase transitions (Saggio et al., 2019) [4]. If the system is in the state,

$$\frac{|11\rangle + |00\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}|11\rangle + \frac{1}{\sqrt{2}}|00\rangle$$

then it indicates that the two qubits are entangled.
Quantum gates are simply operators that act on qubits and manipulates them. The operators that do not distort the normalization of a state are known as Unitary Operators which can be represented by unitary matrices. Some basic gates with their matrix representation as well as the circuit symbols are shown below :

Pauli gates are special as these gates are both unitary and hermitian. These Pauli gates (X,Y,Z) can be applied to any single qubit.

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

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad -\boxed{Y}-$$

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

The pauli-X gate acts as NOT gate. Applying it to a qubit that is in $|0\rangle$ state, will result the qubit to switch in $|1\rangle$ and vice versa. This can be said that the X gate causes the qubit to rotate by π radian along the x-axis. Similarly, the Y and Z gates force the qubit to rotate by π radian respectively along the y-axis and z-axis.

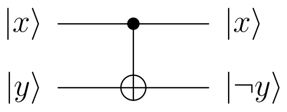
Hadamard Gate is one of the fundamental gates and the most frequently used gate. The matrix representation of this gate is -

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

The beauty of this gate is that we can transform any single qubit from its basis state to superposition states and vice versa by applying hadamard gate. It maps the basis state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Like Classic logic gates, quantum computing also has universal gates.

Controlled-NOT gate is an example of universal gate, can be reversed by itself.

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



Here, $|x\rangle$ is control bit. If $|x\rangle = |0\rangle$, then the bottom output of $|y\rangle$ will be the same as the input. If $|x\rangle = |1\rangle$, then the bottom output will be the opposite.

Toffoli gate, also known as controlled-controlled-not gate is 3 qubit gate. The Matrix representasion of Toffoli gate -

$$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix} \tag{2.1}$$

and the gate representation is given below :

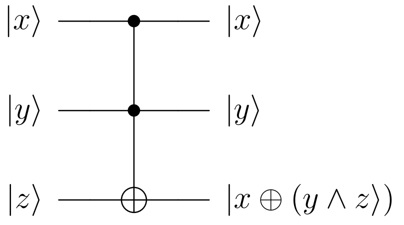


Figure 2.1: Toffoli gate.

This is a controlled-NOT gate with two controlling bits, comparable to the controlled-NOT gate. Only when both of the top two bit are in state $|1\rangle$. We can write this operation as taking state $|x, y, z\rangle$ to $|x, y, z \oplus (x \wedge y)\rangle$. Toffoli gate is interesting, as it is universal. The AND gate is obtained by setting the bottom $|z\rangle$ input to $|0\rangle$. The bottom output will then be $|x \wedge y\rangle$. The NOT gate is obtained by setting the top two inputs to $|1\rangle$.The bottom output will be $|(1 \wedge 1) \oplus z\rangle = |1 \oplus z\rangle = |\neg z\rangle$.

Rotational Gates are the operators that are used to rotate any single qubit along the x,y and z-axis from its origin point in the surface of the Bloch sphere. With the help of R_X, R_Y and R_Z, a qubit can be rotated by θ with respect to x-axis, y-axis and z-axis. Their matrix representation is shown below :

$$R_X(\theta) = \begin{bmatrix}\cos\theta & -i\sin\theta\\ -i\sin\theta & \cos\theta\end{bmatrix}$$

$$R_Y(\theta) = \begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}$$

$$R_Z(\theta) = \begin{bmatrix}e^{-i\frac{\theta}{2}} & 0\\ 0 & e^{i\frac{\theta}{2}}\end{bmatrix}$$

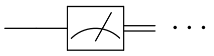
There are also two-qubit rotational gates that are used to control both qubits with the same parameter - θ. R_{YY}, R_{XX} and R_{ZZ} are examples of such operators. For an instance, if R_{XX} is applied to a two-qubit, it will rotate both qubits by θ with respect to x-axis. Similarly, R_{YY} and R_{ZZ} rotate both qubit with respect to y-axis and z-axis. The matrix representation of R_{YY}, R_{XX}, $R_{ZZ}$is given below :

$$R_{XX}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & 0 & -i\sin(\theta) \\ 0 & \cos(\theta) & -i\sin(\theta) & 0 \\ 0 & -i\sin(\theta) & \cos(\theta) & 0 \\ -i\sin(\theta) & 0 & 0 & \cos(\theta) \end{bmatrix} \tag{2.2}$$

$$R_{YY}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & 0 & i\sin(\theta) \\ 0 & \cos(\theta) & -i\sin(\theta) & 0 \\ 0 & -i\sin(\theta) & \cos(\theta) & 0 \\ i\sin(\theta) & 0 & 0 & \cos(\theta) \end{bmatrix} \tag{2.3}$$

$$R_{ZZ}(\theta) = \begin{bmatrix} e^{-i\theta} & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & e^{-i\theta} \end{bmatrix} \tag{2.4}$$

Measurement is being performed to calculate out any qubit state to any basis state $|0\rangle$ or $|1\rangle$. This operation is done by also gates that are not only non-unitary but also irreversible. These gates are denoted by-



Generally, states are measured and collapse into $|0\rangle$ or $|1\rangle$ state. As an example,

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle \tag{2.5}$$

Here $||c_0||^2 + ||c_1||^2 = 1$; where $||c_0||^2$ indicates the probability to find $|\psi\rangle$ in state $|0\rangle$. Similarly, $||c_1||^2$ implies the probability of finding the $|\psi\rangle$ in state $|1\rangle$. The value of $||c_0||^2$ and $||c_1||^2$ can be anything between 0 and 1 before measurement whereas the value of $||c_0||^2$ will be either 0 or 1, same as for $||c_1||^2$, after measurement. This indicates if our states collapse into either $|0\rangle$ or $|1\rangle$.

2.3 Quantum Error Correction

Now that we have seen about quantum states, we need to understand how quantum error correction works. Quantum Error Correction (QEC) is based on the concept of the classical theory of error correcting codes. In classical information theory, error correction is a vital idea which is fundamentally analogous to quantum error correction. Thus, the merger of quantum physics and the classical theory of error correcting codes gives rise to quantum error correction.

Generally, classical error correcting codes use an operation to diagnose which error corrupts an encoded state, so is quantum error correction (QEC) code. If a qubit has been corrupted it can be identified by the operation. The procedure is known as the syndrome measurement. It can analysis and inform us to which qubit was disrupted and also how it was affected. The effects can either be a bit flip or a phase flip or both (corresponding to the Pauli matrices X, Z, and Y). Let us introduce the errors that can be identified by the QEC.
Bit flip : A bit flip error indicates an error where the qubits computational state flips its bits. It can flip from 0 to 1 or 1 to 0. This error is equivalent to X-gate.

$$X|0\rangle = |1\rangle$$
$$X|1\rangle = |0\rangle$$

This can be corrected with the help of a bit-flip code . Here a 3 qubit circuit which corrects 1 qubit with help of 2 ancillary qubits. At first the computational state of the main qubit is transferred to other ancillary qubits using CNOT. If any error occurs the first qubits will be flipped. Similarly CNOT gates will be applied to the other ancillary qubits again. After that to correct the state of the first qubit a Toffoli gate has been applied.

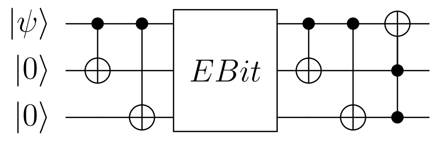


Figure 2.2: Bit Flip Error Correction Code

Phase flip : A phase flip error changes the phase of an qubit and a Z-gate is identical to this error.

$$Z|0\rangle = |0\rangle$$
$$Z|1\rangle = -|1\rangle$$

Similarly, this can also be corrected with phase flip code.

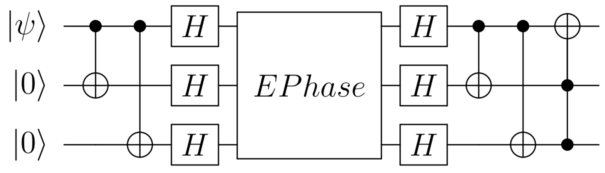


Figure 2.3: Phase Flip Error Correction Code

Like the bit flip code, the phase flip code first transfers the state of the main qubit to the ancillary qubits using CNOT gates. However, in this code qubits are put into superposition using a Hadamard gate to be affected by the phase error. Then again another hamard gate is applied to every qubit to take them out of the superposition.

As the phase of the main qubit has been shifted, it can be corrected using CNOT gates and Toffoli gates respectively. Here the main qubit is the target and ancillary qubits are the control.

Sometimes, the Bit-phase flip can be seen together at the same time which is equivalent to the Y-gate.

As it is seen that a QEC needs to be constructed in such a way that it can identify and correct a qubit that is affected by a bit flip or phase flip or both. From this problem, A QEC was first developed by Peter Shor in 1995 which is a nine qubit error correcting code [8].

The Shor code can fix error for a bit-flip, a phase-flip or both error for a single qubit.

The Shor code works with a 9 qubit circuit where 8 ancillary qubits needed to correct 1 qubit. To put in a nutshell, we will acknowledge the 1st qubit as a main qubit that needs to be corrected and 1 to 8 bit are the ancillary qubits. Shor code will act very similar to bit flip and phase flip as we use and rearrange the same gates like we use in there.

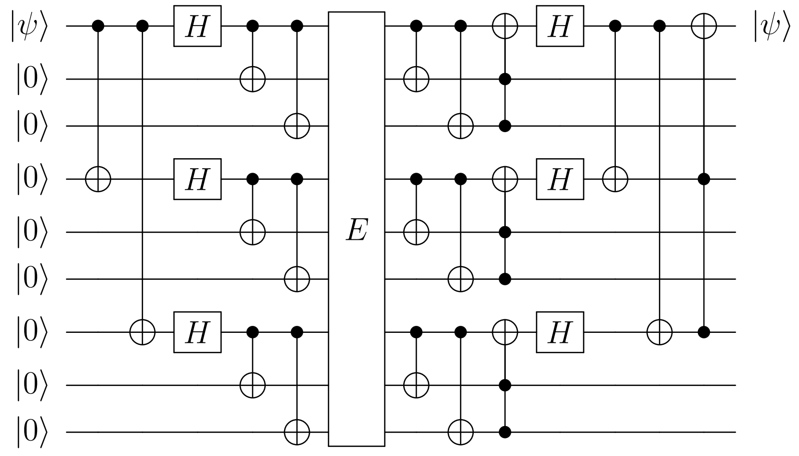


Figure 2.4: The Shor Code

Firstly, the Shor code takes the computational state of the main qubit and transfers it to the 3rd and 6th qubit that will be used to phase errors. With the help of a Hadamard gate, these qubits are set into a superposition state. The main qubit along with 3rd and 6th qubits transfer their states to ancillary qubits with the help of CNOT gates. This is done to correct the bit flips. After that the main qubits' state are being transferred to the 1st and 2nd ancillary qubit. Similarly, the 3rd qubit changes its state to the 4th and 5th and 6th changes its state to the 7th and 8th.

Afterwards, a bit flip or phase flip may occur on the input qubit. In the diagram above (Fig. 2.4), this segment is signified as E. Subsequently, the previous step is repeated. Thereafter, Toffoli gates are applied on the input qubit as well as the 3rd and 6th qubits. Here the control qubits are serving as phase correction ancillary qubits.

Later, Hadamard gates are applied to the input qubit as well as the 3rd and 6th qubit to map them out of superposition. Afterwards, the 3rd and 6th qubit go through CNOT gates where the control qubit is the input qubit. Finally a toffoli gate is implemented to the input qubit which is controlled by the 3rd and 6th qubit.

The Pauli group is a group of Pauli matrices. Some Pauli operators of Pauli group does not change a quantum state. Those operators belong to **Stabilizer** set. Zygelman, B. (2018) [9] suggests that we can find out those operators that does not cause any error in a quantum state, that causes fixable error and that causes unfixable error using stabilizer analysis.

Chapter 3

Machine Learning Basics

Mitchell (1997) [10] suggests that in machine learning, a computer program experiences some specific class of tasks, and its performance is evaluated. The computer program is said to be learning if the result of iterative performance evaluation improves through experience. There are several machine learning tasks like - Classification, Regression, Anomaly detection, etc. The deep learning model varies from task to task. For example, Natural Language Processing can be done using **Deep Feedforward Network** and **Convolutional Neural Network (CNN)** is generally used in image classification. Deep learning model and some properties of machine learning are discussed below.

3.1 Deep Feedforward Network

Deep feedforward networks, also called **feedforward neural networks**, or **Multilayer Perceptrons (MLPs)**, are the widely used deep learning model. According to Rosenblatt (1958) [11], the multi-layered perceptron is the simplest kind of neural network to classify linearly separable patterns. For any classification task, a feedforward neural network maps, $y = f(x;\theta)$ (Goodfellow et al. 2016) [12]. It learns the value of θ in an iterative process which will result in the best function approximation. Here, θ consists of **weights** and **biases**. The computational unit of deep feedforward neural network is ***neuron*** or ***perceptron***. Multiple neurons form in a layer. Moreover, These layers of neurons are associated with weights and biases. These layers of functions are composed together into a network or chain where the output of a layer is passed as input to the next layer. In a feedforward neural network (Fig: 3.1), the first layer is called **input layer** where inputs are given to the network. The final layer of the network is **output layer** which measures the performance of the model. There can be multiple **hidden layers** in the middle. For example, the function in a layer is denoted as,

$$h = g(W^T x + b) \tag{3.1}$$

where, h is the output of the layer, g is the activation function to achieve non-linearity, W is the weight matrix, x is the input and b refers to the bias. Output h is forwarded as a input to the next layer and so on, until *output layer* is reached. Finally, *output layer* will produce the final output and error of the model will be evaluated through **cost function**. According to Rumelhart et al. [13], **back-propagation** or **backprop** algorithm optimizes the error through gradient computation using chain rule of calculus. After computing the gradient, weights of the layer's are updated in a backward manner (Haykin S. 2009) [14]. This forward and backward propagation continues until the model completes learning and the error is minimized.

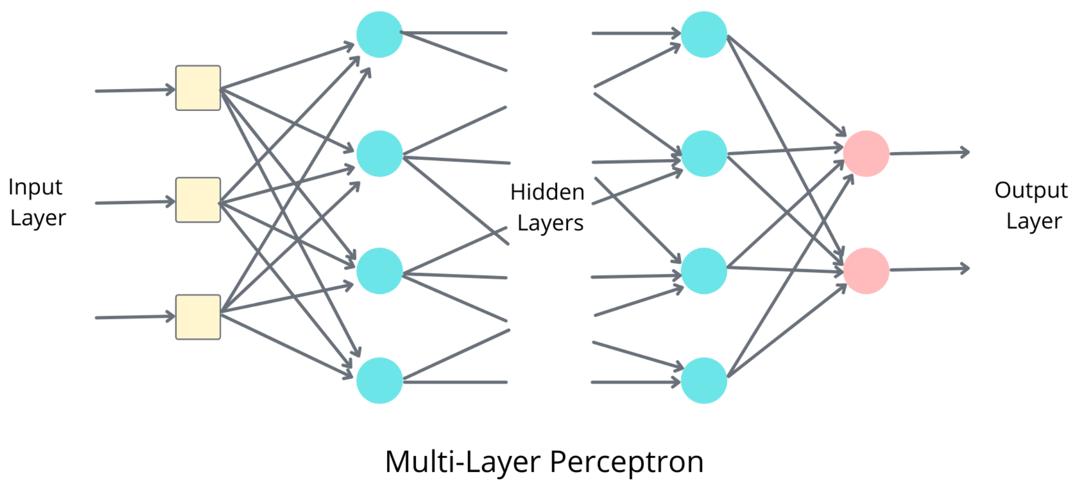


Figure 3.1: Deep Feedforward Network

3.2 Cost function:

Cost function is a result of the difference between the value predicted by the model and the target. For example:

$$Cost function(J) = \sum_{1}^{m} \frac{1}{2}(y - y')^2$$

Here, y = true value; y' = predicted value; m = number of examples; i = index of sample.

3.2.1 Types of cost function:

There are different types of cost function. Choosing the correct cost function depending on the model is another challenge of machine learning (Mulla, 2021) [15].

1. **L1 cost function:** The total sum of the absolute differences between the estimated values and the corresponding target values is the L1-norm cost function. It is more robust than the L2-norm.

$$J(\theta) = \sum_{i=1}^{m} |y_i - y_i'|$$

2. **L2 cost function:** The sum of the squares differences between the estimated values and the respective target values is the L2-norm cost function. It is more stable than the L1 norm when a single input data is changed slightly.

$$J(\theta) = \sum_{i=1}^{m} (y_i - y_i')^2$$

3. **Mean Squared Error (MSE):** One of the most widely used and firstly explained cost functions. Each partial error in MSE is equal to the square area formed by the mathematical distance between the measured spots.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (y_i - y_i')^2$$

and many more.

3.2.2 Activation function:

An activation function is a vital part of an artificial neural network; it determines whether or not such a neuron should be activated. The activation function of a node in an artificial neural network determines the output of the whole node given input or group of inputs. The activation function adds non-linearity to the model as well.

Types of activation function:

1. **Binary Step Activation Function:** The binary step function is an activation function that is based on a threshold. It activates a neuron after a particular threshold is reached and vice-versa.

2. **Sigmoid Activation Function:** The logistic function is the sigmoid activation function. It is a non-linear activation function. The function takes any real value as input. Later, the function squash the input and returns a value between 0 and 1. However, this function causes vanishing gradient problem for very high or very small input.

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

3. **Hyperbolic Tangent Activation Function:** The hyperbolic tangent function could be utilized as an activation function in neural networks as an alternative to the sigmoid function. Modeling inputs with highly negative, neutral, and highly positive values is easier with zero-centered inputs.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

4. **Softmax Activation Function:** Softmax is a function that converts integers and logits into probabilities. This activation function is used only for the output layer, for neural networks that must categorize inputs into numerous groups.

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{m} e_j^x}$$

3.2.3 Capacity, Overfitting and Underfitting

According to Goodfellow et al. (2016) [12] the pathway to algorithm optimization has two main focuses. The first goal is to create an algorithm with the least amount of **training** error possible. The second purpose is to reduce the size of the undesired gap between **training** and **test error** as much as possible. **Underfitting** is when an algorithm fails to adequately minimize training error. In addition, **overfitting** is when a method by its nature increases the error between training and test data. The **capacity** of an algorithm determines whether it will **overfit** or **underfit**. The hypothesis space of the learning algorithm's function is called capacity. For instance, a polynomial of degree 2: $\hat{y} = w_1x + w_2x^2 + b$ has more capacity than a polynomial of degree 1: $\hat{y} = wx + b$.

3.2.4 Hyperparameters

Hyperparameters are settings that can be used to regulate the behavior of an algorithm. The learning algorithm does not change the values of hyperparameters, and these hyperparameters are optimizable. Learning rate, batch size, number of epochs, number of iteration, number of hidden layers, momentum, etc are some training or learning algorithm-related hyperparameters. It is crucial to choose the right types of hyperparameters for the model.

- **Learning rate:** The learning rate of a neural network model determines how quickly or slowly it learns a problem. In gradient descent, if the learning rate is very high then the output will oscillate rapidly and if it is very low then convergence time will increase.

- **Batch size:** The batch size measures the accuracy of the error gradient estimation while training neural networks. Moreover, it determines how many items from the data are included in the training model. However, additional memory space is needed for the higher the batch size.

- **The number of epochs:** The number of epochs is the number of occasions the algorithm examines the whole data set. When the algorithm has viewed all of the samples in the dataset, it has completed one epoch. Each time it transmitted a batch of data by the neural network, it executed one loop.

- **The number of iteration:** The number of times the algorithm sends a batch of data is referred to as an iteration. In neural networks, this refers to the forward and backward passes. Every time a batch of data is transmitted through the neural network, one iteration is completed.

- **The number of hidden layers:** The algorithm's input and output layers are separated by hidden layers. One hidden layer in a computing system is highly powerful. The complexity of the code and the time grows as more hidden layers are added.

3.3 Gradient-Based Optimization

Gradient-Based Optimization is a key application of machine learning to minimize error. The **local maximum** indicates a point where the value of a function,

$f(x)$ is higher than all neighboring points. Similarly, in a **local minimum** point, value of a function, $f(x)$ is lower than all neighboring points. In a **global minimum** point, $f(x)$ has the absolute lowest value. The **saddle point** is neither maxima nor minima. Visual representation is in Figure: 3.2. The goal of machine learning is to converge error in a point closer to the global minimum [12].

Using first derivative test, we can reduce $f(x)$, hence the error. To do that x needs to be shifted in small steps according to the learning rate with the opposite sign of the derivative ($\frac{dy}{dx}$ or $f'(x)$). This technique is known as **gradient descent** (Cauchy, 1847) [16].

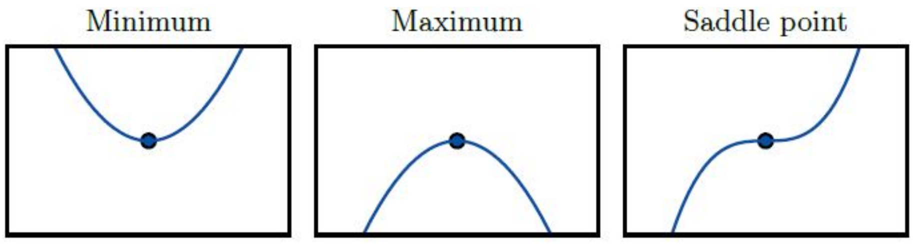


Figure 3.2: Three types of critical points where slope is zero. (Goodfellow et al. 2016)

The **second derivative test** denoted as $\frac{d^2y}{dx^2}$ or $f''(x)$ determines the curvature of a function. Moreover, this can also be used to determine whether a **critical point** where $f'(x) = 0$, is a local maxima, a local minima or a saddle point. In a critical point, when $f''(x) > 0$, it means $f'(x)$ will increase with steps moving rightwards and decrease as steps moving leftwards. Therefore, it is determined as a local minima. Similarly, where $f''(x) < 0$, x is local maxima and where $f''(x) = 0$, x is saddle point or a part of flat region.

3.4 Convolutional Neural Network

According to LeCun, Y. (1989) [17], Convolutional Neural Networks or CNN, are a type of neural network that is used to analyze input using a defined grid-like architecture. Unlike the feedforward network, CNN not necessarily needed to be a fully connected network meaning not every neuron is connected with the other neurons of the next layer. The architecture of a CNN depends on the specific type of data (O'Shea, K. & Nash, R., 2015) [18]. CNN has three stages: convolution stage, detector stage and pooling stage (Fig: 3.3). Several layers of these stages are applied throughout the network. Let's review the stages:

3.4.1 Convolution Stage

To perform convolution, at first inputs are given to the architecture. Unlike, multi-layer perceptron (MLP), CNN does not use matrix multiplication of weights or parameters with each input. CNN introduces a set of parameters known as **kernel**. This set of parameters are several orders of magnitude smaller than the input size. Moreover, the kernel has a specific dimension that is applied to the same dimension

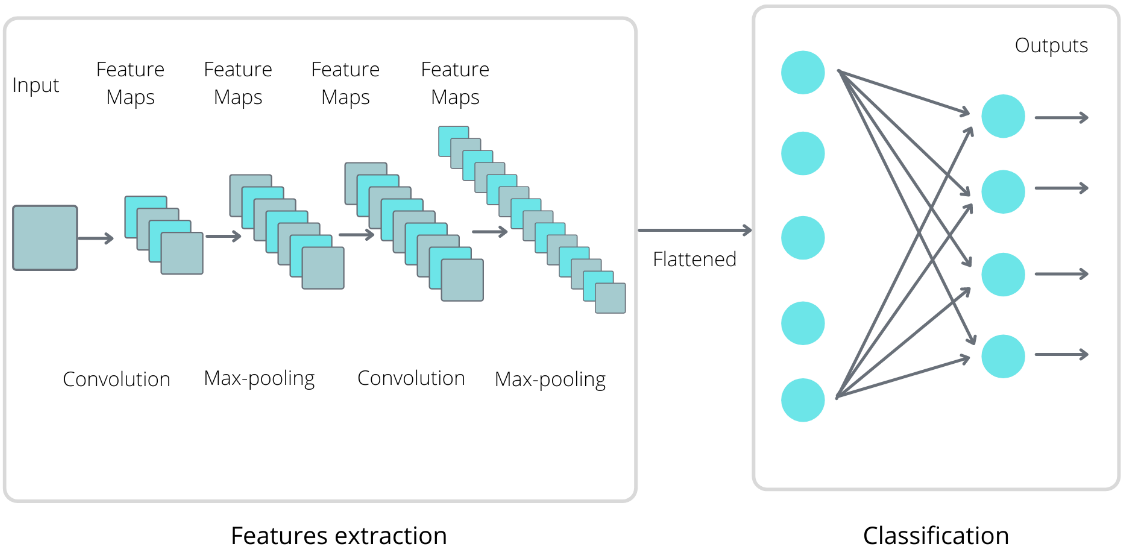


Figure 3.3: Convolutional Neural Network Architecture

of input in one stride. As a result, the network has a sparse connection of parameters. Besides, The same kernel is applied to the whole input space and filter the inputs. Several convolutions are performed by the kernel in parallel in the layer using stride and zero paddings. [12]. Finally, the output **feature map** has a reduced dimension than the original input.

3.4.2 Detector Stage

At the end of the convolution stage, the output has a set of linear activation. A non-linear activation function like Rectified Linear Unit (ReLU) is used in each linear activation. This is done to achieve non-linearity in the network.

3.4.3 Pooling Stage

The output of previous stages then goes through a **pooling function**. The pooling function pulls the concise attributes of the nearby outputs at a given range. This further reduces size and complexity of the network. However, due to sparse connectivity important features remain intact although size is decreased. There are different types of pooling. The most used one is **max pooling**. It pulls the maximum value from a specific rectangular neighborhood (Zhou and Chellappa, 1988) [19]. Other pooling operations like weighted average as well as $L2$-Norm of a rectangular neighborhood are also popular.

3.4.4 Fully-Connected Layer

Previous stages of CNN are known as **feature extraction**. For classification, a fully connected layer needs to be introduced. This has the similar architecture of a normal feedforward neural network. The output data are flattened and passed as a fully connected network. As usual, neurons of two adjacent layers are connected here.

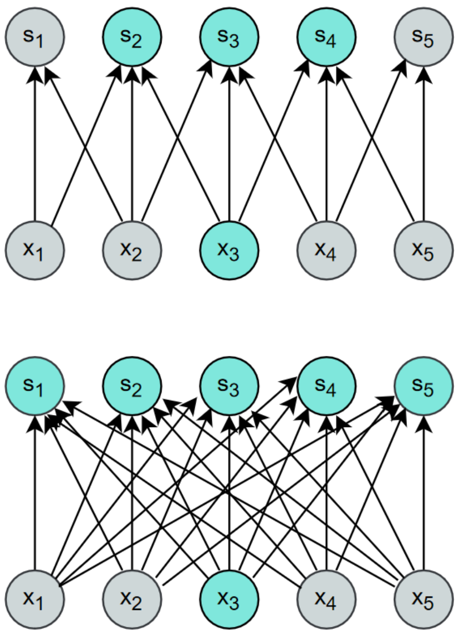


Figure 3.4: Input units of **x**, known as **receptive field** creates output unit S_3. In top, only 3 input units forms s_3 by convolution with kernel of width 3. This is sparse connectivity. In bottom, matrix multiplication of all input units forms s, which is not sparse. [12]

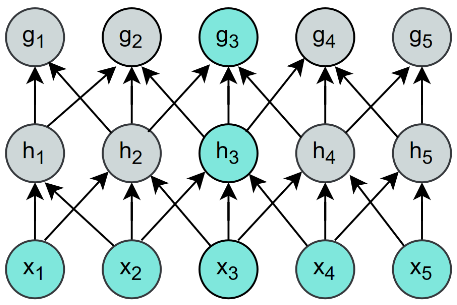


Figure 3.5: Deeper layer receptive field units has more connectivity than the shallower layer. [12]

3.4.5 Advantages

The major advantage of CNN over MLP is **parameter sharing**. The parameter sharing implies that instead of learning a different set of parameters for each location, it just learns one set. Unlike MLP, where every output unit is connected with every input unit, CNN has **sparse connectivity** (Fig. 3.4). As the kernel is significantly smaller than the input, there are very few parameters to store. Therefore, in terms of memory requirements and statistical efficiency, convolution is thus vastly superior to dense matrix multiplication in MLP. In addition, CNN maintains **equivariant representations** property. It implies that a little translation in the input does not make much difference to the value of most pooled outputs. Although, the direct connection between two adjacent layer units is very sparse, however, the deeper layer units are more connected to the inputs than the shallower layers (Fig. 3.5). This happens because of the strided convolution and pooling operation.

Convolutional Neural Network is very handy for the machine learning tasks where parameters are shared across the model and sparse connectivity is absolutely necessary. In many cases, CNN provides more computational efficiency than any fully connected neural network. Although, CNN mainly works with images for segmentation and many more. However, in our paper, we will use the basic architecture of CNN and combine it with quantum mechanics to implement a **Quantum Convolutional Neural Network (QCNN)**.

Chapter 4

Parameterized Quantum Circuit & Quantum Convolutional Neural Network

4.1 Qiskit

IBM Quantum is the industry who took the initiative to construct universal quantum computers for trading first, engineering and science. This attempt embraces promoting the entire quantum computing technology stack. Moreover, this makes an impact on exploration the applications to increase the accessibility of quantum computing. IBM Quantum has build various of its quantum systems within easy reach to the general. To create this scope, IBM introduces open systems such as Qiskit and IBM Quantum Experience to access multiple quantum computing systems. We can access the IBM Quantum computer with just a Qiskit IBMQ account. It has offered learning opportunities for students and researchers. (IBMQ, 2021) [20]

The IBM System design is divided into three parts- End user, Classical Components and Quantum Devices.

Users can simulate in their own local environment. They can use instruction on a classical computer, which enters to a quantum computer through the IBM cloud. The instructions travel as microwave pulses. These pluses has frequencies and shapes which can control qubits and change their quantum states. They have to be in negative 459 degrees fahrenheit to travel over cables to reach the qubits. This transformation of the temperature needs about milliseconds. When the pluses interact with the qubits, the results are sent over the cables to the classical computer. In this process, the results are sent back to us (Quantum Computing System, 2021) [21].

Qiskit is a library of python to support quantum computing through the classical computer and IBM cloud. It is supported in Ubuntu, macOS and Windows. Moreover, we can use python virual environments to explore Qiskit. We are using Qiskit, as it is very convenient to use for quantum computing.

4.2 Parameterized Quantum Circuit

Parameterized Quantum Circuit (PQC) is a special type of quantum circuit that produces various results depending on the parameters given. The parameterized circuit optimize the value of the parameters (θ) iteratively to reach the minima. A **variational form** is a fixed form that uses parameterized circuits. Different unitary gates, $U(\theta)$ which is using single or multiple parameters are applied here. This kind of implementation of variational form on a quantum computer requires varying the ansatz systematically. Let's assume a starting state, $|\psi\rangle$. Usually, it is a vacuum state $|0\rangle$. A variation form of $U(\theta)$ is applied to it and it generates, $U(\theta)\,|\psi\rangle \equiv |\psi(\theta)\rangle$, the output state. Thus, an ansatz produces an expectation value $\langle\psi(\theta)|H|\psi(\theta)\rangle \approx Egs \equiv \lambda_{min}$ by iterative optimization over $|\psi(\theta)\rangle$ for any target state λ_{min}.

Any possible state $|\psi\rangle$, where $|\psi\rangle \in C^N$ and $N = 2^n$ can be generated by a variational form of N qubit. Let's assume a case where the number of qubits, $N = 1$. The U3 unitary gate has three parameters: θ, ϕ and λ. And it can be represented in the following form:

$$U3(\theta,\phi,\lambda) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

A variational form associated with a mentioned U3 gate can produce any possible state for the single qubit case.The circuit is given below :

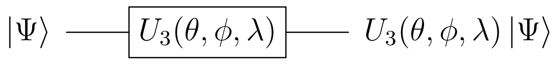

$|\Psi\rangle$
$U_3(\theta,\phi,\lambda)$
$U_3(\theta,\phi,\lambda)\,|\Psi\rangle$

Figure 4.1: Single Qubit Variation Form

Additionally, this $U3$ gate is in universal variational form and has 3 optimizable parameters. It can generate any arbitrary state to ensure that the optimization process runs smoothly. Figure: 4.2 denotes the universal parameterized 2 qubit circuit (Shende et al. 2003) [22].

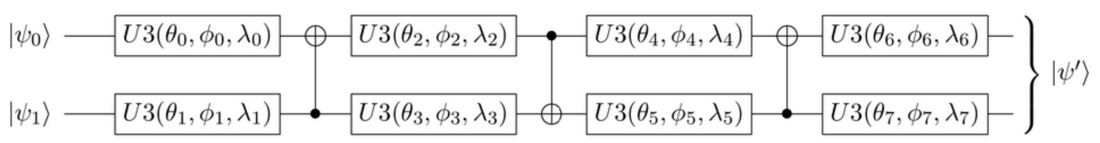

$|\psi_0\rangle$
$|\psi_1\rangle$
$U3(\theta_0,\phi_0,\lambda_0)$
$U3(\theta_1,\phi_1,\lambda_1)$
$U3(\theta_2,\phi_2,\lambda_2)$
$U3(\theta_3,\phi_3,\lambda_3)$
$U3(\theta_4,\phi_4,\lambda_4)$
$U3(\theta_5,\phi_5,\lambda_5)$
$U3(\theta_6,\phi_6,\lambda_6)$
$U3(\theta_7,\phi_7,\lambda_7)$
$|\psi'\rangle$

Figure 4.2: Two Qubit Variation Form

There is a controlled version of U3 gate called Controlled-U3 or CU3 gate. It is applied for generic single qubit rotation. According to the Qikskit-Textbook [23], the matrix representation of CU3:

$$CU_3(\theta,\phi,\lambda)q_0,q_1 = I\otimes|0\rangle\langle 0| + U_3(\theta,\phi,\lambda\otimes|1\rangle\langle 1| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\frac{\theta}{2}\right) & 0 & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ 0 & 0 & 1 & 0 \\ 0 & e^{i\phi}\sin\left(\frac{\theta}{2}\right) & 0 & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

(4.1)

Controlled-U3 Circuit is visualized in figure: 4.3.

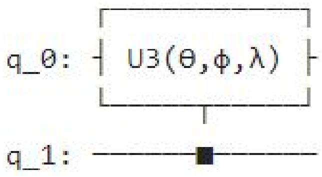


Figure 4.3: Controlled-U3 Circuit

4.3 Optimizers

An **Optimizer** updates the parameters of a variational form iteratively in order to minimize error.
Gradient decent is a popular optimization which updates every parameter in the direction of the largest local change in energy. The number of evaluations performed depends on the number of optimization parameters present. **Qiskit** offers multiple build-in optimizers for parameter optimization. Qiskit Aqua library contains a variety of classical optimizers for use by quantum variational algorithms (IBMQ Optmizers, 2021) [24]. These optimizers can be divided into two categories:

1. Local Optimizers: A local optimizer is the function which finds an optimal value within the neighboring set of a postulant solution.

2. Global Optimizers: A global optimizer is the function which finds an optimal value among all possible solutions.

We are going to use some local optimizers in out model. Some local optimizers:

1. COBYLA: For constrained tasks, COBYLA (Constrained Optimization By Linear Approximation optimizer) is used. It is useful in situations where the objective function's derivative is unknown. Per optimization iteration, one objective function is used. As a result, the number of assessments is unaffected by the parameter set. Furthermore, the objective function is noise-free and minimizes the number of evaluations conducted. Thus, the use of COBYLA is suggested.

2. SPSA: SPSA (Simultaneous Perturbation Stochastic Approximation) is a stochastic approximation methodology for optimizing systems with numerous unknown parameters. It is used for large-scale population models, adaptive modeling, and simulation optimization. The gradient of the goal function is calculated using two measurements. This method perturbs all of the parameters at the same time, in an ad hoc manner. SPSA is a recommended as the classical optimizer while running a variational form circuit in either a noisy simulator or on real hardware.

3. ADAM: Adam and AMSGRAD optimizers (ADAM) is a gradient-based optimization approach that uses adaptive lower-order moment estimates. It is unaffected by gradient rescaling on the diagonal. It also has non-stationary objective functions, as well as noisy and sparse gradients.

4. POWELL: Boundaries and limitations are not considered by the Powell algorithm. Powell is a *conjugate direction approach* where it does one-dimensional reduction on each directional vector in a sequential manner and at each iteration of the primary minimization loop, it is updated. No derivatives are taken because the function being reduced does not have to be differentiable.

4.4 Quantum Convolutional Neural Network

QCNN (Quantum Convolutional Neural Network) proceeds like a Convolutional Neural Network which is narrated in section 3.4. QCNN also needs to perform tasks such as using filters, feature map, pooling, activation function and so on (Cong et al., 2018) [5]. As previously stated in section 4.2, parameterized quantum circuit uses variation form of optimizable gates. It is needless to perform same gate operation among all qubits. Therefore, PQC holds the sparse connectivity property of CNN. Furthermore, the parameters are frequently shared across the circuit, which provides a computational advantages on optimization. For convolution and pooling, PQC employes several unitary or controlled unitary gates. Despite the fact that there are two types of activation functions, the non-linear activation function is used predominantly. Although, quantum operations are linear unitary, to introduce non-linearity, QCNN appertains measurement.

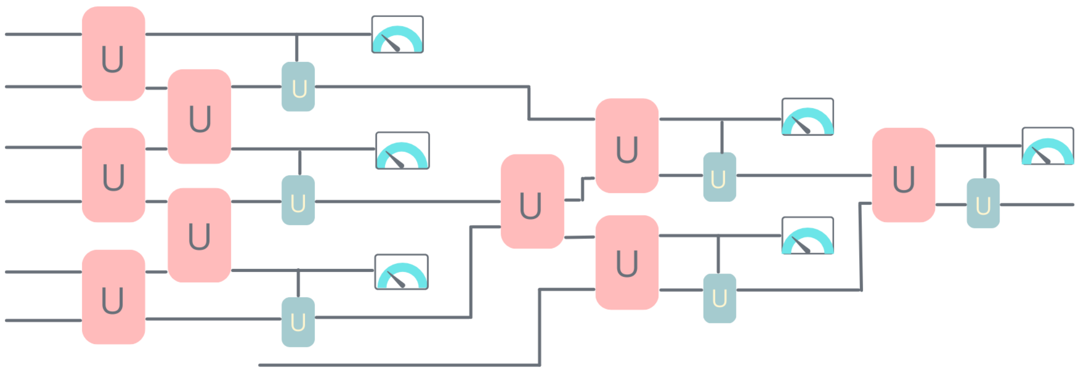
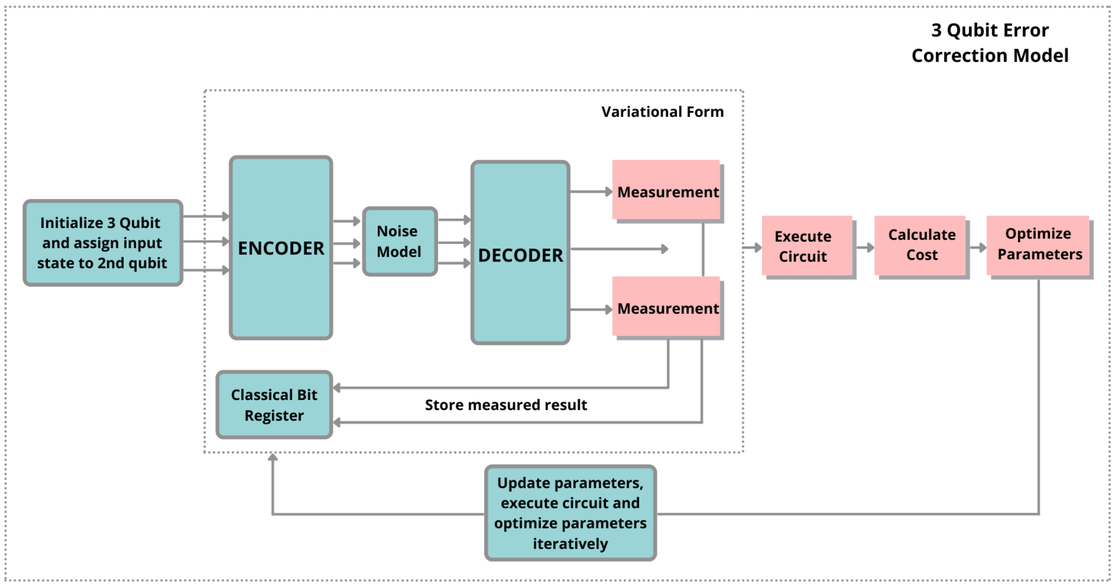


Figure 4.4: Quantum Convolutional Neural Network

With clearer visual demarcation, the QCNN is depicted in the schematic Fig. 4.4. Here, Convolutional operations are performed using generic two-qubit unitary gates. Two-qubit unitary gates are manoeuvred for conserving locality. Because a qubit can interact with another qubit within its neighborhood. Besides, controlled unitary gates signalize pooling in QCNN, since at this stage controlled unitary operation are performed based on some particular qubit. Pooling operation transfers the control qubit's information to the target qubit. This procedure is designated for one layer. For further layers, QCNN uses the same procedure frequently until it obtains the output.

Chapter 5

Architecture



3 Qubit Error Correction Model
Variational Form
Initialize 3 Qubit and assign input state to 2nd qubit
ENCODER
Noise Model
DECODER
Measurement
Measurement
Execute Circuit
Calculate Cost
Optimize Parameters
Classical Bit Register
Store measured result
Update parameters, execute circuit and optimize parameters iteratively

Figure 5.1: Error Correction Model

Fig. 5.1 symbolizes the flow diagram of error correction model.

5.1 Architecture

Using QCNN for 3-qubit error correction

Using QCNN, we are introducing quantum error correction code. For error correction of 3-qubit bit-flip and phase-flip, our model circuit will obtain the **encoder part**, **noise channel** and **decoder circuit**.

In Fig. 5.2, we have shown the general approach of QCNN architecture for quantum error correction [5].

In the case of **encoder circuit**, our input state is q_1 (any random state). q_0 and q_2 are our ancilla bit. Our input state will be copied to q_0 and q_2 by using controlled unitary gates. This procedure is similar to representing the qubits into physical qubits in the Shor code. Later, q_0, q_1 and q_2 will enter our **convolutional layer**.

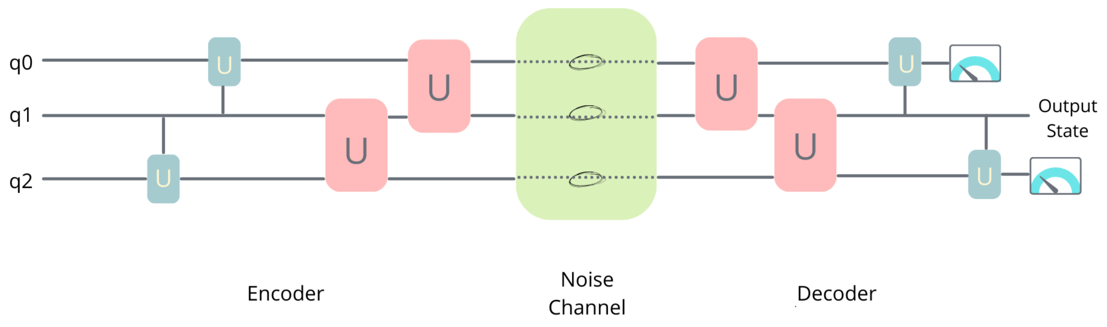


Figure 5.2: QCNN in error correction code

As stated, qubits can interact with another qubit within its locality; our designed model obtains two two-qubit unitary gates. Firstly q_1 and q_2 enter the two-qubit unitary gate and then q_0 and q_1 enter two-qubit unitary gate. This circuit leads us to an encrypted state.

Coming through the encoder circuit, q_0, q_1 and q_2 enter the noise channel. We consider our **noise model** causes error to only one qubit at each iteration.

Building the decoder circuit should have concerned the idea of reversing the quantum circuit. As, all quantum circuits are unitary operations and $UU^\dagger = 1$, $U^\dagger = U^{-1}$. Thus, our decoder circuit will be exactly the mirror reflection of our encoder circuit with the exact parameters in reverse order.

When q_0, q_1 and q_2 reach the **decoder circuit**, they pass through the **convolutional layer** again. Firstly q_0 and q_1 pass through a two-qubit unitary gate. Next, q_1 and q_2 pass through the second two-qubit unitary gate. Later, **pooling** is performed on our state using controlled unitary gates. We only measure our ancilla bits (q_0 and q_2) to operate the activation function as QCNN. Therefore in this manner, we build an error correction circuit for the 3-qubit system using QCNN, and our desired state remains in q_1. Our circuit can be enhanced by appertaining **optimizers**.

We have tried to implement our proposed error correction concept in several approaches using QCNN.

5.2 Proposed architecture version 1.0

Fig. 5.3 symbolize our designed model with 54 parameters. This architecture represents our approach to make the encoder circuit at the very beginning. Afterward, we have constructed our decoder circuit by inverting our encoder circuit.

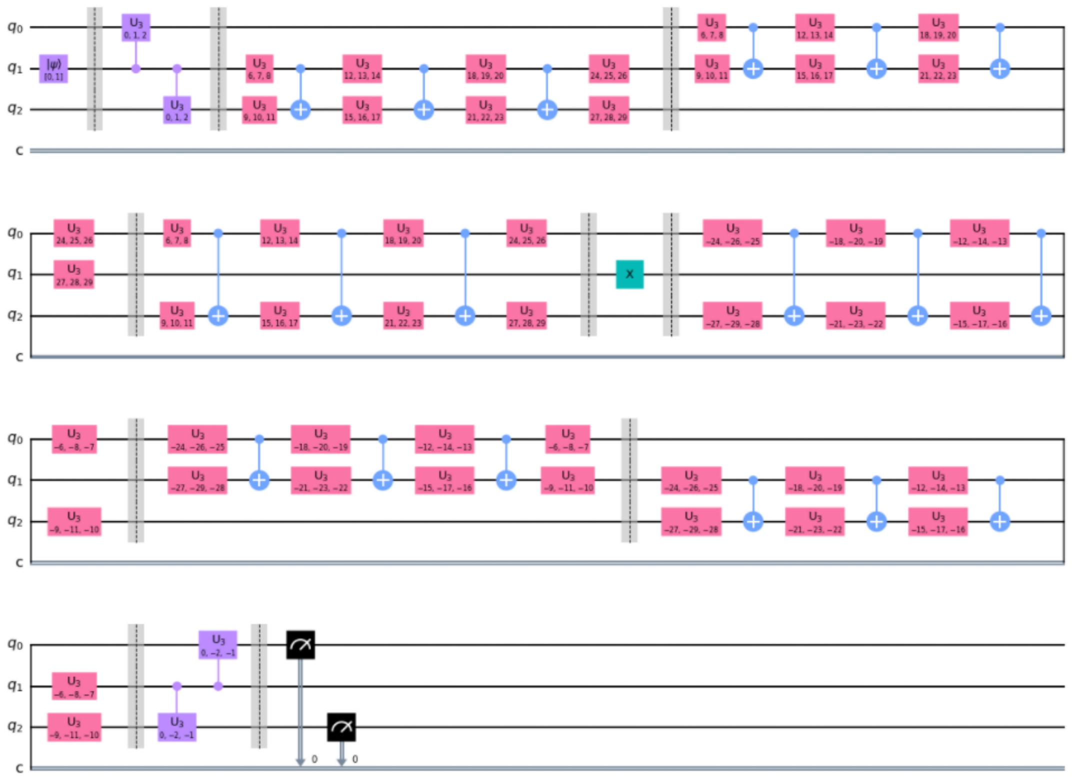


Figure 5.3: Proposed architecture version 1.0

5.2.1 Encoder Circuit

Cong at el. (2018) have introduced the encoder as MERA, which is exactly inverse of decoder [5].

Our encoder circuit contains two controlled unitary gates. Each of the controlled unitary gates has 3 parameters which is stated in PQC section [Fig. 4.3].

Moreover, the encoder circuit contains sixteen 1-qubit unitary gates as we need two 2-qubit unitary gate. Each 2-qubit unitary gate is equivalent to eight 1-qubit gate with 3 parameters which is described in PQC section [Fig. 4.2] (Shende et al. 2003) [22].

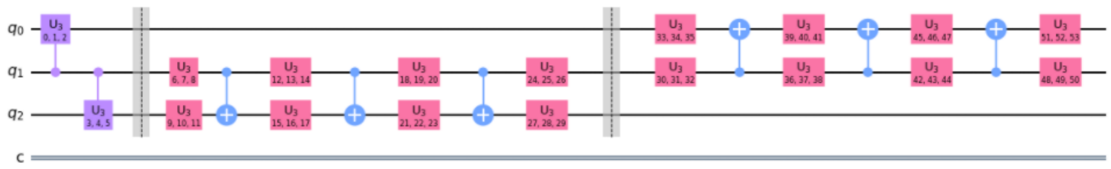


Figure 5.4: Encoder circuit Version 1.0 built in qiskit library

The encoding circuit takes total 54 parameters (48 parameters for sixteen 1-qubit gates and 6 parameters for two controlled unitary gates.) [Fig. 5.4].

5.2.2 Decoder Circuit

The decoder circuit is built based on the encoder circuit. As narrated earlier in section 4.4, our decoder and encoder circuits are $E^{\dagger} = E^{-1} = D$. Thus, if we pass any state in our channel, our output state is approximately the same as the target state with no error.

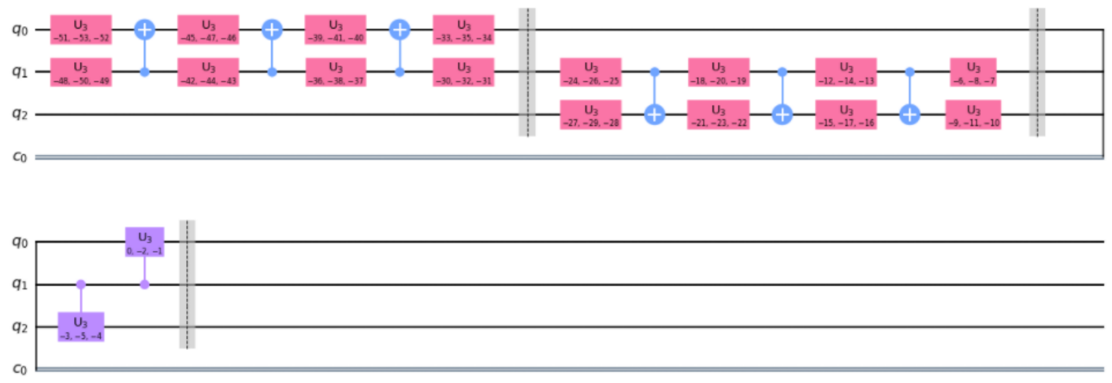


Figure 5.5: Decoder circuit version 1.0 built in qiskit library

We can see in Fig. 5.5, our parameters for decoding circuit is exactly inverse of encoder circuit.

5.3 Proposed architecture version 2.0

Fig. 5.6 represents our proposed architecture version 2.0 where we are using 21 parameters. Here we have built our decoder circuit first and then inverted the decoder to generate the encoder.

5.3.1 Decoder Circuit

Tucci R. (2005) has presented a one-qubit unitary circuit and a two-qubit unitary circuit architecture [25]. We have added those two circuits to our decoder circuit. Our decoder circuit also contains pooling layers as well as convolution layers.

One Qubit Unitary Circuit

We have used R_x, R_y and R_z gate to create one qubit unitary circuit in Fig. 5.7.

Two Qubit Unitary Circuit

We have built two qubit unitary circuit by adding:

1. One qubit unitary gate
2. R_{zz} gate
3. R_{yy} gate

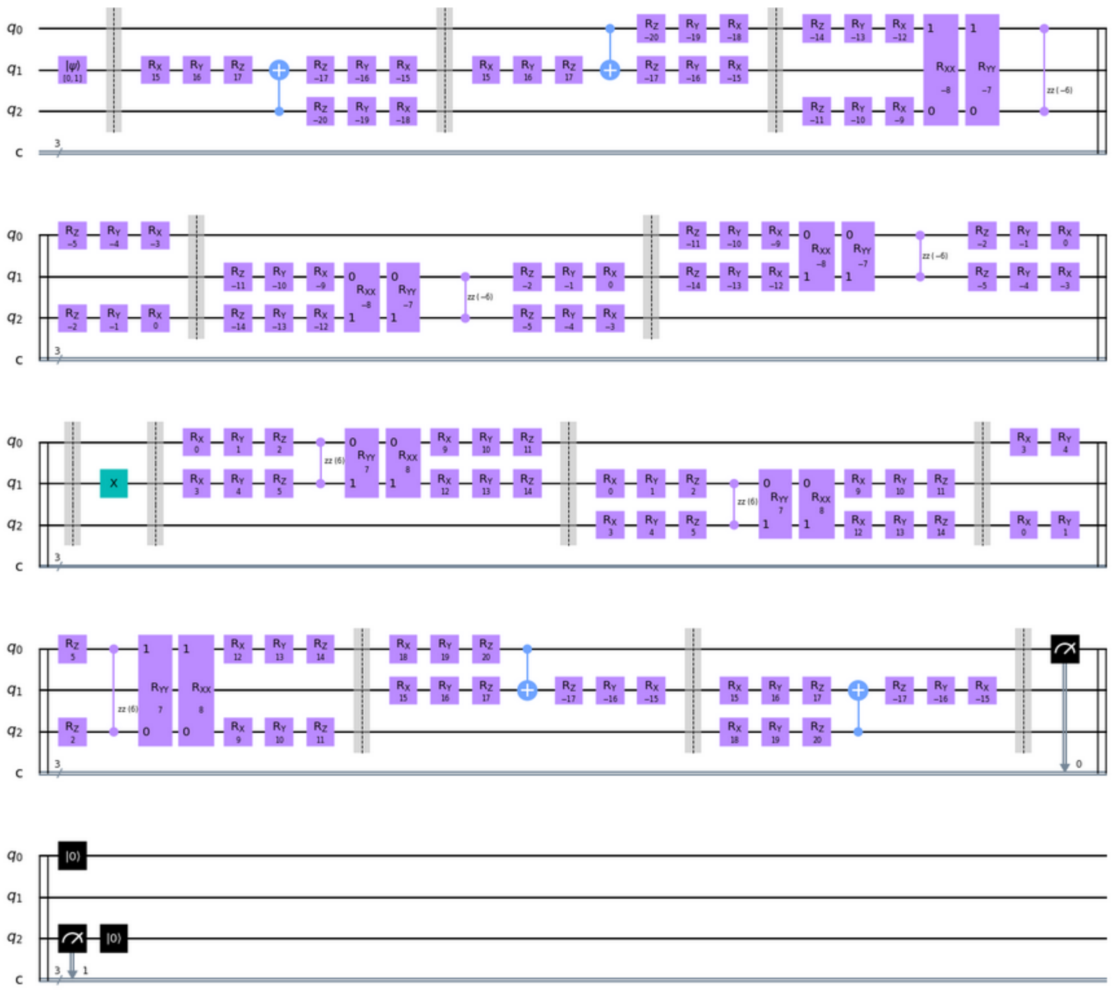


Figure 5.6: Proposed architecture version 2.0

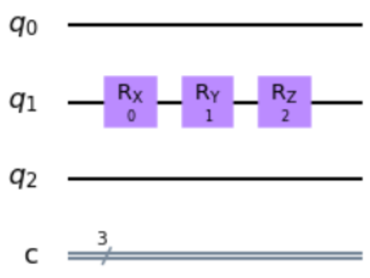


Figure 5.7: One qubit gate built in qiskit library

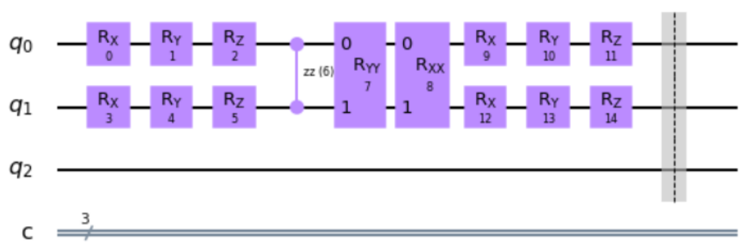


Figure 5.8: Two qubit unitary circuit built in qiskit library

4. R_{xx} gate

In Fig. 5.8, we have shown our 2-qubit unitary circuit. This circuit requires 15 parameters.

Pooling Circuit

Here pooling circuit requires:

1. 2 One-qubit unitary circuits in two separate qubits
2. Controlled-NOT gate
3. Inverse of one-qubit unitary circuit with inverted parameters

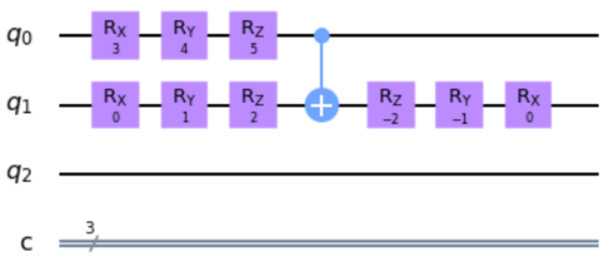


Figure 5.9: Pooling circuit built in qiskit library

In Fig. 5.9, we have used 6 parameters for pooling circuit.

We can see in Fig. 5.10, when q_0, q_1 and q_2 reach the **decoder circuit**, they pass through the **convolutional layer**. Firstly, q_0 and q_1 pass through a 2-qubit unitary circuit. Then q_1 and q_2 pass through the second 2-qubit unitary circuit. Afterwards, q_0 and q_2 pass through the third 2-qubit unitary circuit. These three 2-qubit unitary circuits of convolutional layer needed 15 parameters.
Subsequently, **pooling** is performed on our state by using the pooling circuit. q_0 and q_1 enter our fisrt pooling circuit. Then q_2 and q_1 go through our second pooling circuit. Here we need 6 parameters for our pooling function as QCNN. At the last, We measure our ancilla bits (q_0 and q_2) to operate the activation function as QCNN. Therefore in this manner, we build a decoder circuit with only 21 parameters.

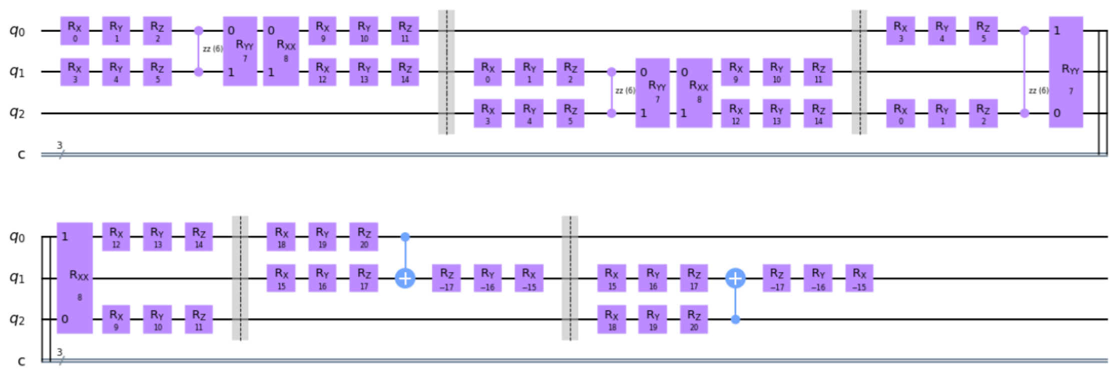


Figure 5.10: Decoder circuit version 2.0 built in qiskit library

5.3.2 Encoder Circuit

Here our encoder circuit is the reverse of the decoder circuit with inverted parameters.

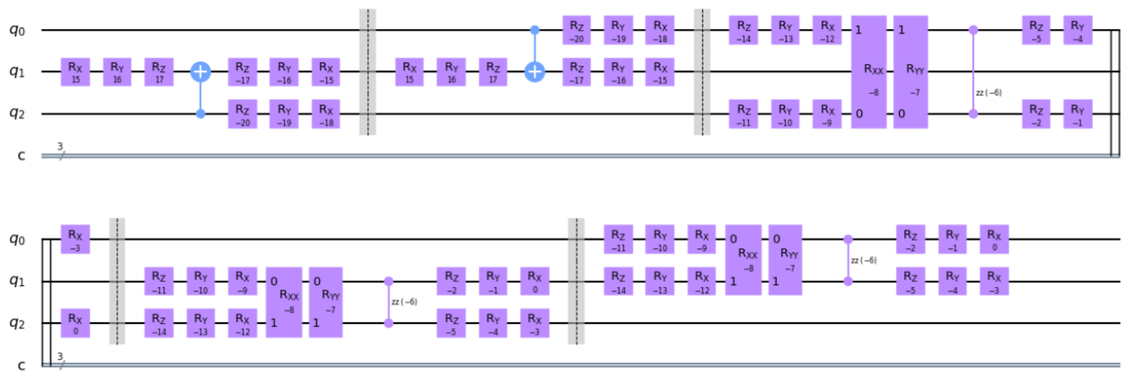


Figure 5.11: Encoder circuit version 2.0 built in qiskit library

In Fig. 5.11 encoder circuit accommodates **inverse of pooling circuit** and **inverse of convolutional layer**.

5.4 Noise Model

The noise model is built based on the concept that our circuit can only correct the 1-bit flip error and only 1 phase flip distinctively. We have constructed dummy noise models using X gate and Z gate. Our noise model implies an error in one qubit out of three at each iteration. Therefore, a maximum of 33% of error is applied to the circuit at each iteration.

5.4.1 Noise for bit flip

Bit flip are caused if one bit changes due to noise. Either $|1\rangle$ becomes $|0\rangle$ or $|0\rangle$ becomes $|1\rangle$, generates bit flip.
We have already discussed it in section 2.3.

$$X\left|0\right\rangle = \left|1\right\rangle$$
$$X\left|1\right\rangle = \left|0\right\rangle$$

Observing this effect of X gate, we have applied X gate as representative noise for bit flipping.

5.4.2 Noise for phase flip

Phase flip is induced whenever an qubit faces change in their relative phase. Like, $\left|1\right\rangle$ becomes $-\left|1\right\rangle$.
We have considered this in section 2.3.

$$Z\left|1\right\rangle = -\left|1\right\rangle$$
$$Z\left|+\right\rangle = Z(\tfrac{1}{2}(\left|0\right\rangle + \left|1\right\rangle)) = \tfrac{1}{2}(\left|0\right\rangle - \left|1\right\rangle) = \left|-\right\rangle$$

Analysing the changes, we have made use of Z gate as foretaste of noise for phase flipping.
Assume a state, $\left|\psi\right\rangle = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$. Now applying our noise models, we can analysis the change.

$$X\left|\psi\right\rangle = X(\alpha\left|0\right\rangle + \beta\left|1\right\rangle) = \alpha\left|1\right\rangle + \beta\left|0\right\rangle \text{ which refers to bit flip.}$$

$Z\left|\psi\right\rangle = Z(\alpha\left|0\right\rangle + \beta\left|1\right\rangle) = \alpha\left|0\right\rangle - \beta\left|1\right\rangle$ which indicate the change in relative phase.

Thus we have used X and Z gates for noise model.

Chapter 6

Experiments & Results

6.1 Experiments

To experiment with our model, we used a batch optimization approach with different batch sizes. First of all, we randomly generated two-dimensional data according to the given batch size. Next, we normalized those two-dimensional data to convert these into quantum states. Hence, these quantum states of a given batch size are our experiment's input batch. Finally, the target states are equivalent to the input states as our goal is error correction.

In addition, the initial parameters of the circuits are chosen randomly. The parameters bounded by the region, $0 \leq \theta \leq 2\pi$.

As discussed in Section 4.3, lots of open-source optimizers are available in Qiskit. We experimented with our model using *COBYLA*, *SPSA*, and *POWELL* optimizers.

Moreover, it is crucial to choose appropriate cost functions during experiments. We used the $L2$-Norm cost function predominantly to perform better concerning other available cost functions. The cost function:

$$J(\theta) = \sqrt{\sum_{i=1}^{m} |y_i - y_i'|^2}$$

Here, y_i' = output state; y_i = target state; m = batch size; i = index of input in the batch.

For the $L2$-Norm cost function, the global phase of both output state and target state needs to be removed. It means,

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

to remove global phase, this vector needs to be transformed into,

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix}$$

We used the following procedure to remove the global phase.

$$\begin{aligned} &\quad \alpha' = |\alpha| \\ if &\quad (\alpha == 0), \\ &\quad \beta' = |\beta| \\ else&, \\ &\quad \beta' = \frac{\beta}{\alpha} \cdot \alpha' \end{aligned}$$

Besides that, Premaratne & Matsuura (2020) [26] suggest a particular cost function that uses inner product property to determine error for variational quantum circuits. As this cost function uses an inner product, therefore no global phase removal is required. The cost function:

$$\epsilon = 1 - |\langle \psi' | \, \psi \rangle|^2 \tag{6.1}$$

Here, ψ' = output state and ψ = target state.

Also, other hyperparameters such as batch size, number of epochs, number of iterations per epoch, and so on are appropriately selected.

In the batch optimization, for every input state in that batch, 4 circuits are executed at each iteration. 4 circuits because of no error and individual error on qubits q_0, q_1 and q_2. After completing one iteration, the cumulative cost of these four circuits for all input states is calculated, and the optimizer does optimization. When the training ends, the training error is calculated by dividing the total cost by the batch size multiplied by 4. If $L2$-Norm cost function is used, then it needs to be further divided by $\sqrt{2}$. Because the range of the difference between two quantum state lies between 0 to $\sqrt{2}$

6.2 Results

The architecture is tested with numerous batch of input states and hyperparameters. The experimental realization is discussed in this section.

To begin, a batch of 50 input states is created. The model has been experimented with 100 iteration and 100 epoch. The chosen optimizer is COBYLA. For phase-flip error correction, the model retrieved the input state with 90% accuracy (Approx.). The result is graphically demonstrated in Fig. 6.1.

Similarly, The bit-flip error correction approach achieves an accuracy of nearly 86.85%. The hyperparameters are 50 Batch Size, 200 Iteration and 50 Epoch. The result is graphically demonstrated in Fig. 6.2.

The SPSA optimizer is tested with the hyperparameters of 50 Batch Size, 100 Iteration, and 100 Epoch to draw a comparison. The bit-flip error correction model introduces 89.29% (Approx.), and the phase-flip error correction model introduces 90.1% (Approx.) fidelity rate. See Figure: 6.3 for graphical representation.

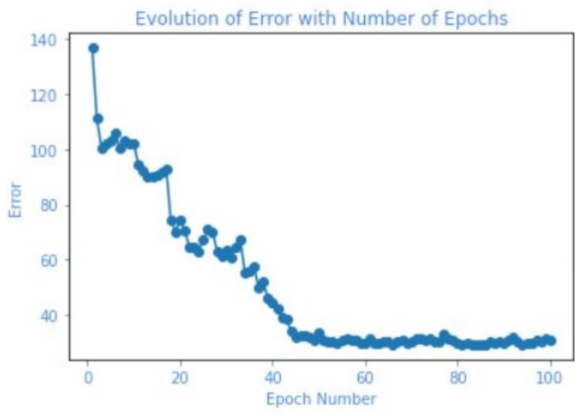


Figure 6.1: Result of phase-flip error correction for 50 Batch Size, 100 Iteration and 100 Epoch. (COBYLA)

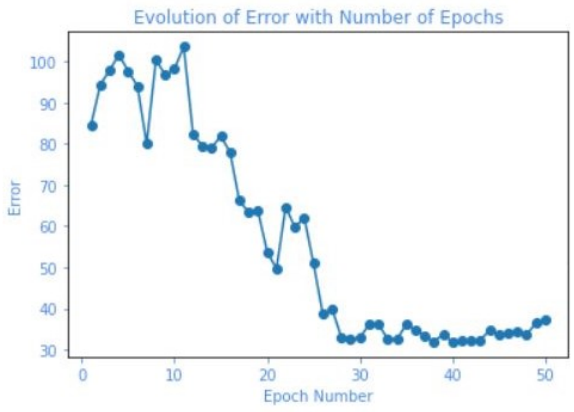


Figure 6.2: Result of bit-flip error correction for 50 Batch Size, 200 Iteration and 50 Epoch. (COBYLA)

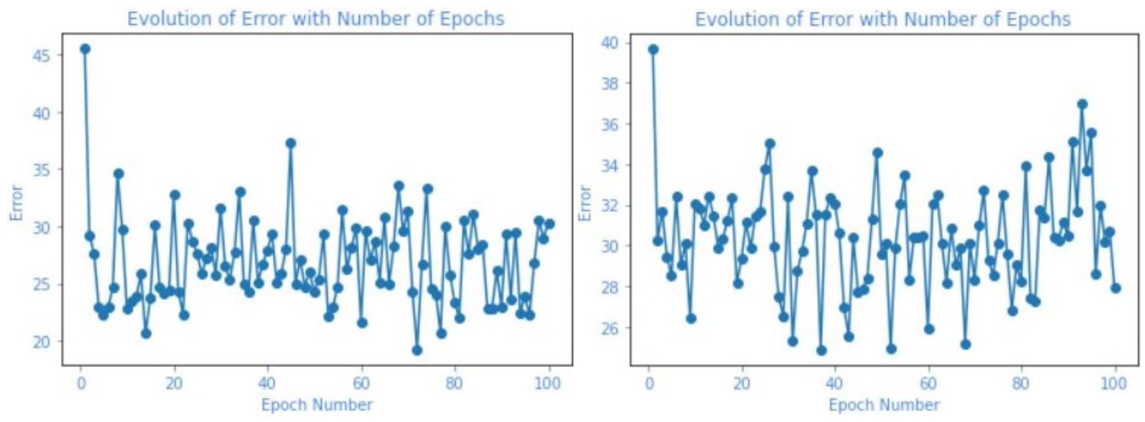


Figure 6.3: Result of bit-flip (left) & phase-flip (right) error correction for 50 Batch Size, 100 Iteration and 100 Epoch. (SPSA)

Optimizer	Noise	Iteration	Number of Epoch	Accuracy (Approx.)
COBYLA	Phase-flip	100	100	90%
SPSA	Phase-flip	100	100	90.10%
COBYLA	Bit-flip	200	50	86.85%
SPSA	Bit-flip	100	100	89.29%

Table 6.1: Comparison of result between COBYLA & SPSA for batch size 50.

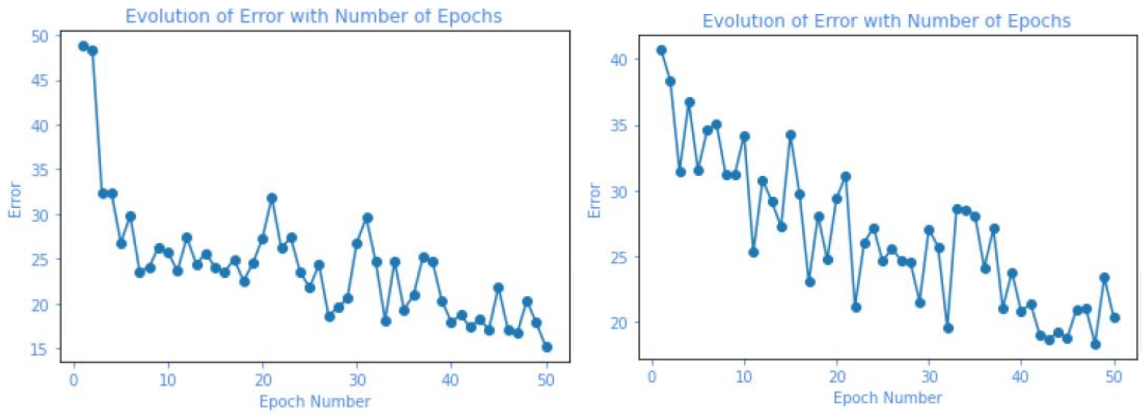


Figure 6.4: Result of bit-flip (left) & phase-flip (right) error correction for 20 Batch Size, 200 Iteration and 50 Epoch. (COBYLA)

Table: 6.1 shows the comparison between COBYLA and SPSA optimizers for batch size 50.

Besides that COBYLA optimizer is experimented with other hyperparameters too. For instance, Fig. 6.4 demonstrates result of bit-flip (left) and phase-flip (right) error correction for 20 Batch Size, 200 iteration and 50 Epoch. The bit-flip QEC code has an 86.53% (Approx.) fidelity rate. And the phase-flip QEC code has an almost 82% fidelity rate.

To make a comparison, the SPSA optimizer has experimented with the hyperparameters of 20 Batch Size, 100 Iteration, and 50 Epoch. The bit-flip QEC code introduces 70% (Approx.), and the phase-flip QEC code introduces 81.5% (Approx.) fidelity rate. See Figure: 6.5 for graphical representation.

Optimizer	Noise	Iteration	Number of Epoch	Accuracy (Approx.)
COBYLA	Phase-flip	200	50	82%
SPSA	Phase-flip	100	50	81.5%
COBYLA	Bit-flip	200	50	86.53%
SPSA	Bit-flip	100	50	70%

Table 6.2: Comparison of result between COBYLA & SPSA for batch size 20.

Table: 6.2 shows the comparison between COBYLA and SPSA optimizers for batch size 20.

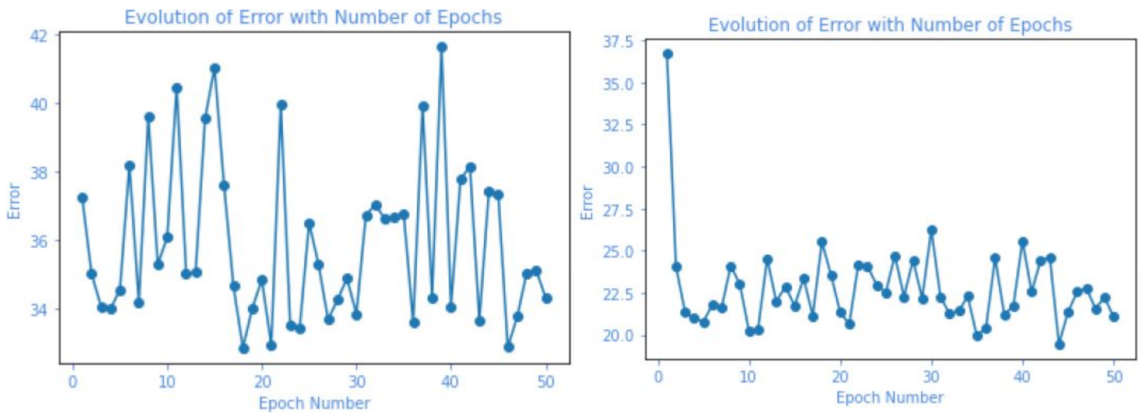


Figure 6.5: Result of bit-flip (left) & phase-flip (right) error correction for 20 Batch Size, 100 Iteration and 50 Epoch. (SPSA)

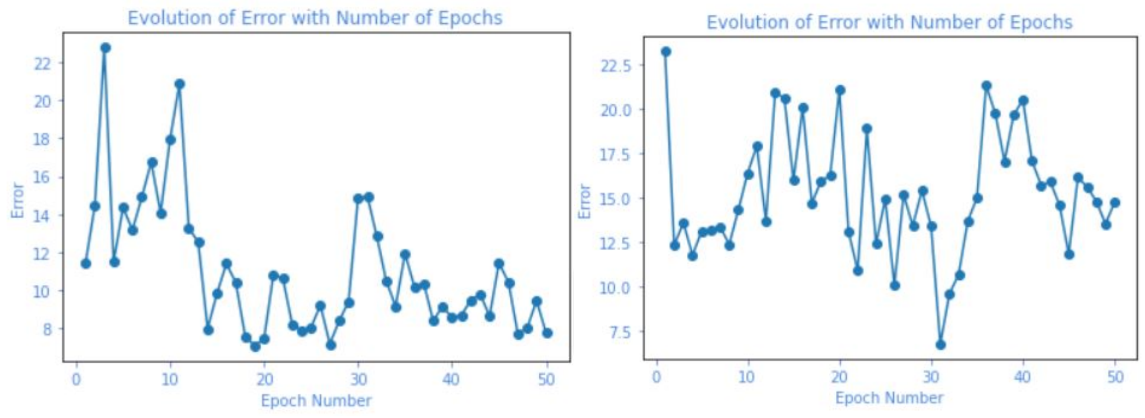


Figure 6.6: Result of bit-flip (left) & phase-flip (right) error correction for 10 Batch Size, 200 Iteration and 50 Epoch. (COBYLA)

To see how the model reacts to smaller batch size, we experimented COBYLA & SPSA with 10 batch size, 200 iteration & 50 epoch. For COBYLA, the bit-flip QEC code introduce 86.25% (Approx.), while the phase-flip QEC code introduces 73.93% (Approx.) fidelity rate. For SPSA, the bit-flip QEC code introduces 86.74% (Approx.), while the phase-flip QEC code introduces 84% (Approx.) fidelity rate. See Fig. 6.6 (COBYLA) & 6.7 (SPSA) for graphical representation. Also, see the Table. 6.3 for comparison between the optimizers.

Optimizer	**Noise**	**Iteration**	**Number of Epoch**	**Accuracy (Approx.)**
COBYLA	Phase-flip	200	50	73.93%
SPSA	Phase-flip	200	50	84%
COBYLA	Bit-flip	200	50	86.25%
SPSA	Bit-flip	200	50	86.74%

Table 6.3: Comparison of result between COBYLA & SPSA for batch size 10.

Table: 6.3 shows the comparison between COBYLA and SPSA optimizers for batch size 10.

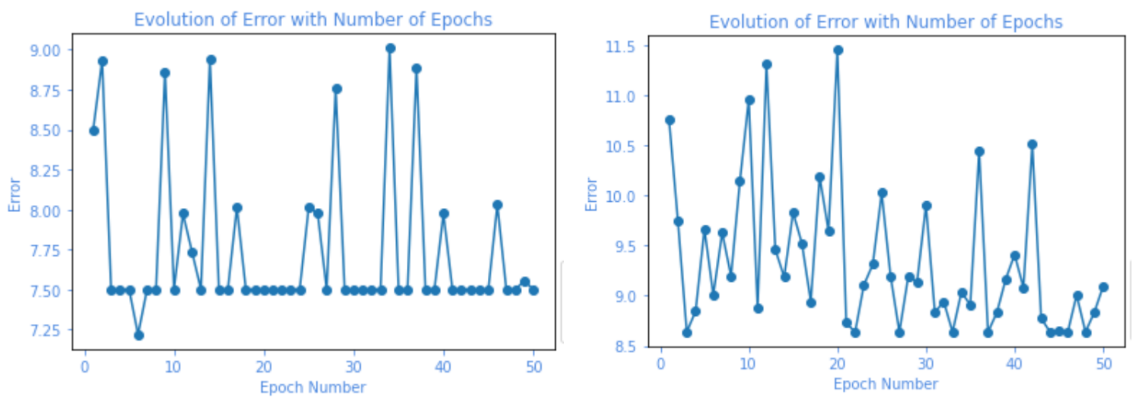


Figure 6.7: Result of bit-flip (left) & phase-flip (right) error correction for 10 Batch Size, 200 Iteration and 50 Epoch. (SPSA)

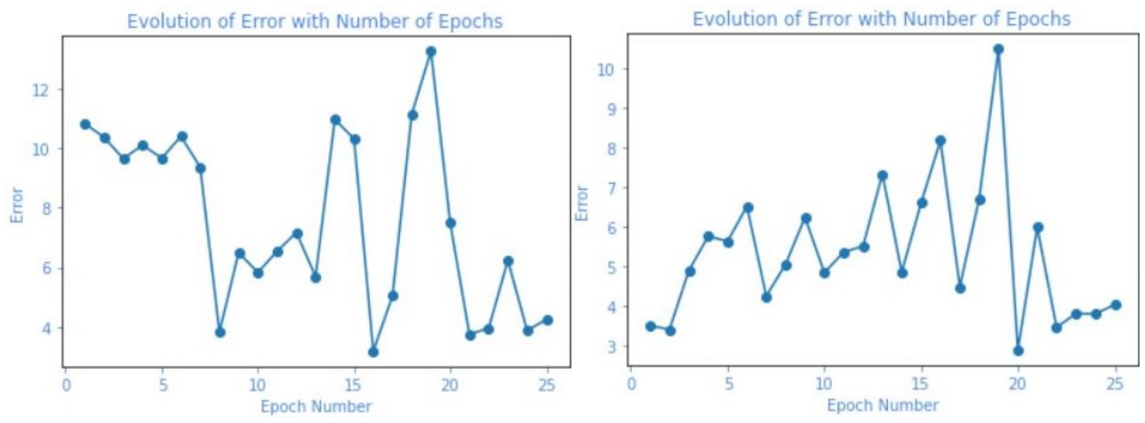


Figure 6.8: Result of bit-flip (left) & phase-flip (right) error correction for 5 Batch Size, 100 Iteration and 25 Epoch. (POWELL)

The POWELL optimizer takes huge time to complete each iteration of a given circuit. Therefore, a tiny batch size of 5 input states has experimented with 100 iteration and 25 epoch. It achieves a fidelity rate of 85% for bit-flip error and 85.75% for phase-flip error (See Fig. 6.8). The same experiment is performed on the COBYLA optimizer, and it generates a fidelity rate of 86.35% for the bit-flip and 88.08% for the phase-flip error (See Fig. 6.9). Lastly, the SPSA optimizer generates a fidelity rate of 71.50% for the bit-flip and 58.88% for the phase-flip error (See Fig. 6.10).

Table: 6.4 shows the comparison between COBYLA, SPSA and POWELL optimizers for batch size 5.

Table: 6.5 shows the comparison of average optimization time between COBYLA, SPSA and POWELL optimizers.

The optimization process requires certain times to complete. Depending on the optimizer, different amounts of time are needed. Moreover, the time it requires to complete the training process is determined by the device on which the model is trained. Based on the above experiments, a comparison between the time needed for

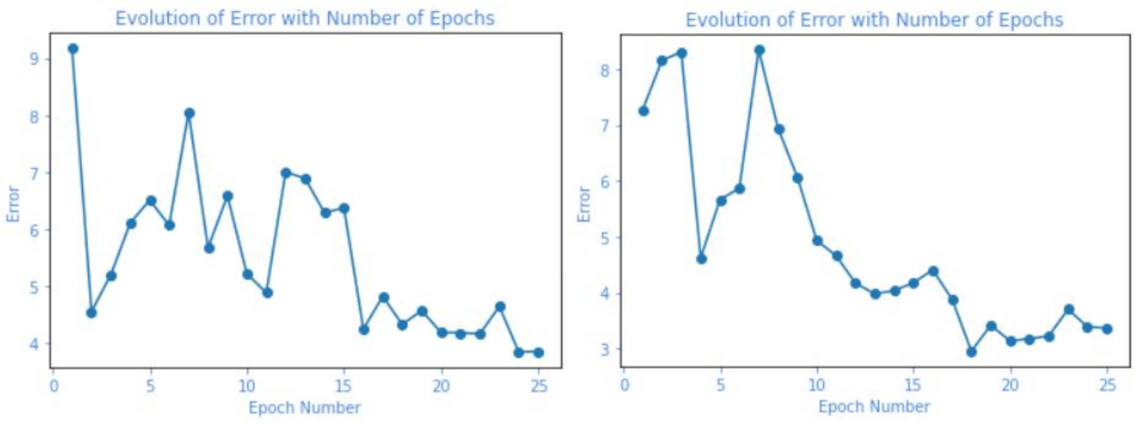


Figure 6.9: Result of bit-flip (left) & phase-flip (right) error correction for 5 Batch Size, 100 Iteration and 25 Epoch. (COBYLA)

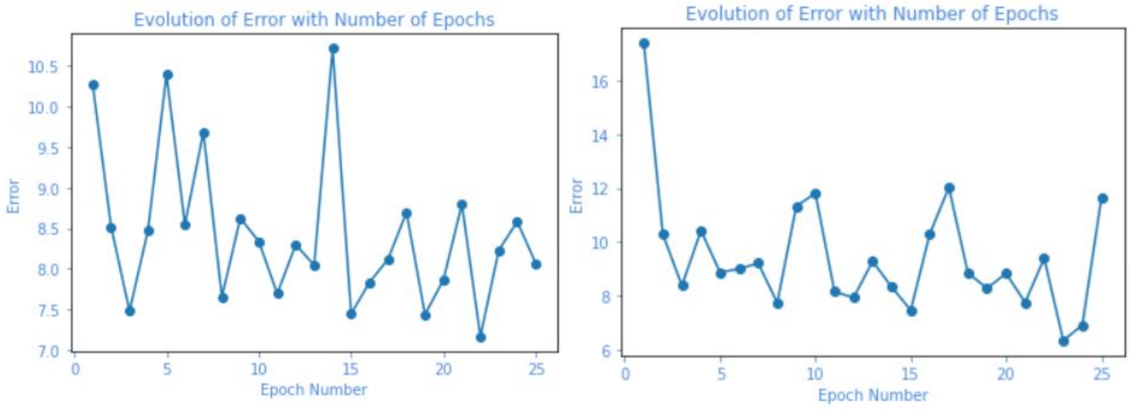


Figure 6.10: Result of bit-flip (left) & phase-flip (right) error correction for 5 Batch Size, 100 Iteration and 25 Epoch. (SPSA)

Optimizer	**Noise**	**Iteration**	**Number of Epoch**	**Accuracy (Approx.)**
COBYLA	Phase-flip	100	25	88.08%
SPSA	Phase-flip	100	25	58.88%
POWELL	Phase-flip	100	25	85.75%
COBYLA	Bit-flip	100	25	86.35%
SPSA	Bit-flip	100	25	71.50%
POWELL	Bit-flip	100	25	85%

Table 6.4: Comparison of result between COBYLA, SPSA & POWELL for batch size 5.

Optimizer	**Batch size**	**Iteration**	**Number of Epoch**	**Average optimization time (in seconds)**
COBYLA	5	100	25	1408
SPSA	5	100	25	3435
POWELL	5	100	25	11224
COBYLA	10	200	50	44579
SPSA	10	200	50	73596
COBYLA	20	200	50	58967
SPSA	20	100	50	75350
COBYLA	50	100	100	50547
SPSA	50	100	100	76083

Table 6.5: Comparison of average optimization time between COBYLA, SPSA & POWELL.

optimization for different optimizers is given in Table: 6.5. Due to CPU utilization, these numbers may differ from device to device. However, it does provide some insight into the efficiency of such optimizers.

The above experimental findings provide a broad overview of the model's capabilities. The maximum fidelity rate is achieved roughly 90% from our experiments. The findings imply a relation between batch size and other hyperparameters. Hence, desynchronization could produce bottleneck issues in the model. Experiments show that a bigger batch size combined with proper iteration and epoch speeds up the fine-tuning of parameters. Further, the imbalance between iteration and epoch may cause underfitting. However, an abundance of iteration and epoch can cause the model to memorize the input states, leading to overfitting. Therefore, cautiously choosing the hyperparameters may avoid generalization error. On the other hand, experiments with various optimizers yield a variety of results. The comparison of optimization time between optimizers from Table: 6.5 indicates that the POWELL optimizer requires extensive time to optimize. This much time is too expensive to use in our model. SPSA demands a shorter amount of time than POWELL. However, the COBYLA optimizer has a faster efficiency than either of these two. Regarding effectiveness, SPSA often pulls off a reasonable accuracy rate. But it has a considerable amount of spikes during the optimization process. Conversely, COBYLA consistently tries to minimize the error while maintaining a high accuracy rate. As a result, COBYLA outperforms the other two optimizers for this model. c

The source code of our model can be found at: Github Link

Chapter 7

Conclusion

In our research, we demonstrated an error correction scheme using Quantum Convolutional Neural Network (QCNN). We presented two distinct architectures of QCNN that use different parameterized quantum circuits. Further, we did experiments using different hyperparameters and optimizers. Finally, we were able to retrieve quantum states at an almost 90% fidelity rate.
In addition, we applied both basic noise types, bit-flip, and phase-flip, individually. The shared parameters across the model are optimized iteratively. With enough iteration, our parameters are fine-tuned by the optimizer, and the error is converged to a minima. It is inferred from the experiment that this model could generate an error correction scheme for any arbitrary error. As a result, our approach will render the necessity of a hand-made error correcting circuit superfluous for any arbitrary error.
Although our Quantum Error Correction (QEC) code can reduce the bit-flip and phase-flip error separately but independently, it is conceivable to create a QEC code that minimizes both of the errors concurrently. Going forward, developing a 9-Qubit error correction model that can maneuver both errors simultaneously can be a future direction. Moreover, the built-in optimizers from the Qiskit library are adopted in our model. Another important work could be to craft a more suitable optimizer for this particular error correction task. Focusing on improving these areas may enhance the performance of our model significantly.

Bibliography

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. Bardin, R. Barends, R. Biswas, S. Boixo, F. Brandao, D. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, and J. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct. 2019. DOI: 10.1038/s41586-019-1666-5.

[2] D. Gottesman, "An introduction to quantum error correction and fault-tolerant quantum computation," May 2009. DOI: 10 . 1090 / psapm / 068 / 2762145.

[3] P. Schindler, J. Barreiro, T. Monz, V. Nebendahl, D. Nigg, M. Chwalla, M. Hennrich, and R. Blatt, "Experimental repetitive quantum error correction," *Science (New York, N.Y.)*, vol. 332, pp. 1059–61, May 2011. DOI: 10.1126/science.1203329.

[4] V. Saggio, A. Dimić, C. Greganti, L. Rozema, P. Walther, and B. Dakić, "Experimental few-copy multipartite entanglement detection," *Nature Physics*, vol. 15, Sep. 2019. DOI: 10.1038/s41567-019-0550-4.

[5] I. Cong, S. Choi, and M. Lukin, *Quantum convolutional neural networks*, Oct. 2018.

[6] S. Coutinho and M. Mannucci, "Quantum computing for computer scientists noson s. yanofsky and mirco a. mannucci," *Sigact News - SIGACT*, vol. 40, pp. 14–17, Jan. 2009. DOI: 10.1145/1711475.1711479.

[7] G. Vidal, J. Latorre, E. Rico Ortega, and A. Kitaev, "Entanglement in quantum critical phenomena," *Physical review letters*, vol. 90, p. 227 902, Jul. 2003. DOI: 10.1103/PhysRevLett.90.227902.

[8] P. Shor, "Shor, p. w. scheme for reducing decoherence in quantum computer memory. phys. rev. a 52, r2493-r2496," *Physical review. A*, vol. 52, R2493–R2496, Nov. 1995. DOI: 10.1103/PhysRevA.52.R2493.

[9] B. Zygelman, *A First Introduction to Quantum Computing and Information.* Jan. 2018, ISBN: 978-3-319-91628-6. DOI: 10.1007/978-3-319-91629-3.

[10] T. M. Mitchell, *Machine Learning.* New York: McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 0033-295X. DOI: 10.1037/h0042519. [Online]. Available: http://dx.doi.org/10.1037/h0042519.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016, ISBN: 9780262035613. [Online]. Available: https://books.google.co.in/books?id=Np9SDQAAQBAJ.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0.

[14] S. Haykin, *Neural Networks and Learning Machines*, ser. Neural networks and learning machines v. 10. Prentice Hall, 2009, ISBN: 9780131471399. [Online]. Available: https://books.google.com.bd/books?id=K7P36lKzI%5C_QC.

[15] M. Z. MULLA, *Cost, activation, loss function - neural network - deep learning. what are these?* https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de, Accessed: 2021-05-29, 2020.

[16] A. CAUCHY, "Methode generale pour la resolution des systemes d'equations simultanees," *C.R. Acad. Sci. Paris*, vol. 25, pp. 536–538, 1847. [Online]. Available: https://ci.nii.ac.jp/naid/10026863174/en/.

[17] Y. Lecun, "Generalization and network design strategies," English (US), in *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Elsevier, 1989.

[18] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *arXiv e-prints*, arXiv:1511.08458, arXiv:1511.08458, Nov. 2015. arXiv: 1511.08458 [cs.NE].

[19] Y. Zhou and R. Chellappa, "Computation of optical flow using a neural network," *IEEE 1988 International Conference on Neural Networks*, 71–78 vol.2, 1988.

[20] IBM, *Access systems with your account*, https://quantum-computing.ibm.com/docs/manage/account/ibmq, Accessed: 2021-05-30.

[21] *Quantum computing systems*, https://www.ibm.com/quantum-computing/systems/, Accessed: 2021-05-30.

[22] V. V. Shende, I. L. Markov, and S. S. Bullock, "Minimal universal two-qubit controlled-not-based circuits," *Phys. Rev. A*, vol. 69, p. 062 321, 6 Jun. 2004. DOI: 10.1103/PhysRevA.69.062321. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.69.062321.

[23] *Qiskit.circuit.library.cu3gate — qiskit 0.26.2 documentation, 2021*, https://qiskit.org/documentation/stubs/qiskit.circuit.library.CU3Gate.html, Accessed: 2021-05-29.

[24] IBMQ, *Aqua optimizers*, https://qiskit.org/documentation/apidoc/qiskit.aqua.components.optimizers.html, Accessed: 2021-05-30.

[25] R. Tucci, "An introduction to cartan's kak decomposition for qc programmers," Aug. 2005.

[26] S. Premaratne and A. Matsuura, *Engineering the cost function of a variational quantum algorithm for implementation on near-term devices*, Jun. 2020.