Quantum Solutions for Training a Single Layer Binary Neural Network

Sabrina Ly
sjl2178@rit.edu

Quantum Solutions for Training a Single Layer Binary Neural Network

Sabrina Ly

Quantum Solutions for Training a Single Layer Binary Neural Network

Sabrina Ly

May 2022

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Quantum Solutions for Training a Single Layer Binary Neural Network

Sabrina Ly

Committee Approval:

Acknowledgments

Firstly, I would like to thank Dr. Cory Merkel and Dr. Sonia Lopez-Alarcon for their assistance and advisement throughout this process. Their guidance and encouragement in exploring such theoretical topics have made this work possible. I would also like to thank Dr. Nathan Cahill for serving on my thesis comittee. Lastly I would like to thank my friends and family for supporting me through the late nights and pushing me to do my best, especially Andy Meyer who has been my troubleshooting rubber duck through everything.

For my parents, Dareth and Thida Ly, who came from nothing as refugees but still made it possible for me to pursue higher education.

Abstract

Quantum computing is a relatively new field starting in the early 1980s when a physicist named Paul Benioff proposed a quantum mechanical model of the Turing machine introducing quantum computers. Previously, the focus of most quantum computers is in the study of quantum applications instead of broad applications. This is due to the fact that quantum technology is a newer field with many technology constraints, such as limited qubits and noisy environments. However, quantum computers are still capable of using quantum mechanics to solve specific algorithmns with an exponential speed-up in comparison to their classical counterparts. One key algorithm is the the HHL algorithm proposed by Harrow, Hassidim and Lloyd in 2009 [1]. The HHL algorithm outlines a quantum algorithm to solve a linear systems of equations with a best case time complexity of $O(poly(\log N))$ [1], in comparison to the best case time complexity for classical algorithms of $O(N^3)$. The HHL algorithm outlines a use for quantum circuits outside of quantum applications. One such application is in machine learning, as many networks use linear regression in their training algorithm. Currently it is not feasible to solve for weight vector of floating point precision on a quantum computer, but if the weight vector is constrained to binary values 0 or 1 then the problem becomes small enough to implement even on current noisy quantum computers. This work outlines two different circuit designs to solve for 2×2 and 4×4 systems of equations, so long as the matrices follow the eigenvalue constraint of having eigenvalues be powers of 2. In addition, the problem of reading data from the quantum state to classical data is addressed through the use of a swap test between the solution state $|x\rangle$ and an test state $|test\rangle$. By using a swap test vector of all 1s it is possible to find how many ones lay in the solution vector. Once the number of ones is known, the number of possible solution states is reduced. While it is not possible to beat classical algorithms with the noise on current quantum circuits, this work shows it is possible to implement quantum algorithms for non-quantum applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the advent of machine intelligence and deep learning—in addition to the growth in computational capabilities and availability of large amounts of labeled data—the ability of machines in areas such as classification and prediction have grown exponentially. These different deep learning algorithms are computationally intensive due to the fact that they require multiple numerical calculations and several training iterations to arrive at a solution. In, particular, common training algorithms, such as backpropagation, require a signficant number of computation resources. Backpropagation is a method of training a neural network by fine-tuning its weights based on the error rate calculated during the previous training iteration. It stands for backward propagation of errors and calculates the gradient of a loss function with respect to all the weights in the network. The backpropagation algorithm works by computing the gradient of the loss function for a single weight using the chain rule and will compute the gradients one layer at a time.

This operation, while effective, is costly in terms of compute power and memory. Each weight need to be updated based on a calculated value that propogated from the loss. In addition, both the weights of the network and the intermediate values need to be stored in memory, which can be costly depending on the size of the network.

Due to this compute and memory intensive approach, training neural networks can be quite time consuming as calculations must be run on large chunks of data. Thus, a method to decrease training time is needed. With a lowered training time, utilizing and implementing more complex neural networks becomes more feasible.

One potential method to reduce the overall training time of the network is to use a specialized accelerator to compute the updated weights for each training batch. Currently GPUs (graphics processing unit) and TPUs (tensor processing unit) exist in order to speed up processing time. GPUs decrease training time by performing parallel calculations while TPUs address the memory-access issue in order to decrease training time. However, both processing units perform the same intense mathematical calculations as they are still classical approaches; thus it is worth exploring other methods of training computation. Quantum computing is an emerging technology that utilizes quantum properties of entanglement and superposition to implement quantum algorithms and offer a new method of approach different to classical computers. Due to the fact that these computers use quantum mechanics, they operate differently than classical computers, such as being able to change multiple states encoded in a qubit simultaneously [2]. By showing that a quantum computer can be used in place of a classical computer for training a network, the potential exists for quantum computers to be applied to machine intelligence applications.

1.2 Objectives

The goal of this thesis is to establish a proof-of-concept for a potential method of network training using a hybrid training procedure. The hybrid network will use a quantum circuit to compute training parameters while a classical approach is used for supervised training. As the most computationally intense portion of a neural network is the training portion, by using a quantum computer to speed-up training calculations there is a possibility to reduce the overall computational time for a network. In addition, classical computing still possesses many advantages over quantum computing. Thus, by using classical computers for data pre-processing and overhead computations one could work-around the limitations of current quantum computers. Currently, it is not possible for quantum computers to out-perform classical computers outside of the realm of quantum simulation. However, by establishing a proof-of-concept, it will show the potential for quantum computers to be applied in future network training. Towards this goal the following contributions are outlined:

- Implement quantum circuit designs capable of solving specific cases of 2×2 and 4×4 linear systems of equations.

- Establish accuracy of circuit design by comparing the solution state of the circuit vs. expected solution state.

- Outline a method of using a swap test, in addition to the quantum circuit for solving a linear systems of equation, to reduce the search space for weights in a binary quantized neural network.

- Calculate the error between expected number of 1s in a binary network weight vector and actual number of 1s

- Outline future directions and further implementations

Chapter 2

Background

2.1 Quantized Neural Networks Background

2.1.1 Linear Regression

Linear regression is a method of modeling the scalar relationship between variables. The relationship model uses linear predictor functions or line of best fit functions in order to estimate model parameters based on the given data. The linear regression model can be represented by a general single-equation model. [3]

$$Y = a + \sum_{i=1}^{k} m_i X_i$$

Where Y is the predicted value and X is parameter representing the relationship between the independent and dependent variables. a is a constant value and m are the known data points. The summation adds up the multiplication of a parameter and its corresponding data value. In the case of just one model parameter the resulting model will be a straight line of best-fit.

$$Y = mx + b$$

In certain cases, calculating the best-fit line for all the data in the dataset is not the ideal. Several reasons include datasets being too large, requiring larger amount of memory to store data and a longer processing time, or a need to make the model more robust to performance with unknown data. In the case of model robustness, a model may perform well with the given training but may be overfit to that data and does not do well with unknown data prediction. One way to help mitigate overfitting to the data is to use a cost function. Cost functions are used in order to calculate the effectiveness of the prediction model when comparing the calculated values versus the actual values in a test set. A widely-used cost function is least squares regression, which adds up the the square of the difference between the actual dependent variable and calculated expected value. The smaller the value of the calculated cost function, the higher the accuracy of the model with the test data.

2.1.2 Quantized Neural Networks

A strong argument for neural networks is the fact that neural networks using real weights are universal approzimators [4]. However, the success of any algorithmn is its real world applications. In real world applications, theoretical results cannot be directly compared as real numbers are not as readily available in the real world. For example, in analog implementation, while they can implement real numbers, their precision is limited by things such as noise or power dissipation issues [4]. In addition, the precision of a network is proportional to its cost as a larger VLSI area will need to be dedicated to storing values. Other technological issues with increased precision include fabrication limitations, dynamic range problems, defect issues, etc [4]. Thus, due to limitations present in high precision neural networks such as computation speed and power usage, an interest was developed in neural networks that utilized limited precision weights. In the specific case of VLSI implementation, using a limited range of integer values for weights can translate into reduced storage requirements and integer computation can be implemented more efficiently than floating point computation.

When training networks using a low-precision representation of the weights, a quantized function is used to take a real value number and transform it into a quantized version [5]. Depending on the training routine, quantized optimization can occur after a training step or during training. The main approaches to training are currently deterministic versus stochastic. In deterministic rounding, a deterministic quantization function will round a floating point value to the closest quantized value as:

$$Q_s(w) = sign(w) \cdot \Delta \cdot \left[\frac{|w|}{\Delta} + \frac{1}{2} \right] [5]$$

In the equation above, w is a real-valued number, $Q()$ is a quantization function, and Δ denotes the quantization step or resolution. One thing to note is that this won't be the case for binary weights as all weights will be constrained to one of two values. In the case of stochastic rounding the quantization function can be defined as:

$$Q_s(w) = \Delta \cdot \begin{cases} [\frac{w}{\Delta}] + 1 & \text{for } p \leq \frac{w}{\Delta} - [\frac{w}{\Delta}], \\ [\frac{w}{\Delta}] & \text{otherwise,} \end{cases}$$

$p \in [0,1]$ is produced by a uniform random number generator [5]. The operator is non-deterministc and will round up with probability $w/\Delta - [w/\Delta]$.

However, training quantized models can be difficult. In the case when learning rates are small, stochastic gradient methods will only make a small update to weight parameters. Binarization of weights after each training method will round off these updates and cause training to stagnate [5]. In other cases, rounding procedures can lead to poor results as information is lost with lower precision. Thus the tradeoff between training difficulty and resource conservation should be considered during quantized network application.

2.1.3 Quantized Binary Neural Networks

Of particular interest are quantized binary neural networks, which constrain their weights to one bit precision. These weights can either have a value of [-1,1] or [0,1]. By constraining the precision of these weights, less dedicated hardware and overall power use is needed due to the fact that less multiply and accumulates will be necessary during the calculation of updated weights. Generally, multiply and accumulate components (or MACs) take up the most space in neural network hardware because many of them are required to calculate the update weights with floating point precision. In the case of extreme limited precision or quantized binary neural networks, multiply and accumulate components are no longer necessary during the calculation of updated weights as the constrained weight values of [-1, 1] will result in subtraction or addition operations instead of multiply operations.

In the case of binary neural networks, the deterministic and stochastic binarization methods are slightly different. The deterministic binarization function is

$$x^b = sign(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ f - 1 & \text{otherwise,} \end{cases}$$

where x^b is the binarized weight and x is the real-valued variable. In the case of stochastic binarization the quantization function is

$$x^b = sign(x - z) = \begin{cases} +1 & \text{with probability } p = \sigma(x) \\ -1 & \text{with probability } 1 - p, \end{cases}$$

where $z \sim U[-1, 1]$, a uniform random variable and σ is the sigmoid function [6].

In addition to multiply and accumulate components no longer being necessary in quantized binary neural networks, they are also interesting in the case of quantum computing. Quantum computers are capable of performing certain algorithms exponentially faster than classical computers; however, due to technological limitations larger quantum circuits are quite noisy and may not compute accurate results. Using binary weights means the quantum circuit can remain relatively small, taking advantage of the computational speedup while keeping noise levels in the circuit to a minimum. In addition, with classical binary quantized training methods there is no guarantee that the method has found the local minimum or even global minimum of the function. Yet, in the case of quantum computing, the solution will arrive at the global minimum of the function.

2.2 Quantum Background

2.2.1 Quantum Computers

In the early 1980s physicist Paul Benioff proposed a quantum mechanical model of the Turing machine, opening up the field of quantum computers. However it wasn't until 1994 when Peter Shor introduced a quantum algorithm for integer factoring that held the potential for decryption of secure information that quantum computers became of serious interest.

The major setback with current quantum computers is that quantum systems cannot be observed without producing disturbance in the system. So, in order to store and process information in the system, the particular system must be kept isolated from its surroundings [7]. In addition many of the proposed algorithms, such as Shor's algorithm, require that qubits (quantum bits) be as reliable as classical bits [8]. It is possible to protect quantum systems by using quantum error correction (QEC) protocols, but the overhead required in terms of the number of qubits is not feasible with current quantum technology [8]. In response to this, quantum computers called NISQs or Noisy Intermediate-scale Quantum have become popular. Intermediate-scale refers to quantum computers with qubits ranging from fifty to a few-hundred. The significance of the lower bound of 50 qubits is that it is beyond what can be simulated by the most powerful current digital supercomputers using brute force [7]. These NISQ computers acknowledge that current devices are ”noisy” and attempt to implement methods to mitigate the effects of noise while also extracting maximum computational power [8].

The biggest difference between classical computers and most commonly used quantum computers is the usage of qubits instead of classical bits. The smallest unit of data used by classical machines are binary digits also known as bits. A bit is either 0 or 1 but never both at the same time. In quantum computing, quantum bits are the smallest unit of data used and are commonly represented as Bloch spheres [2]. Their behavior differs from classical bits since they use the fundamental concept of superposition. A qubit can be at the state of 1 and 0 at the same time until it's wavefunction collapses to a final state that will decide whether it is 0 or 1. The probability of the qubit being either a 0 or a 1 depends on the state that it is in [2]. Oftentimes the way to represent a qubit is a type of notation called *bra-ket* which represents orthogonal vectors making up the amplitudes of the qubit [2]. This allows for a more accurate representation of the complex states of a qubit as the amplitudes of a qubit correspond to the likelihood of the final state of the qubit once measured.

Qubits are also very fragile as they are susceptible to environmental surroundings. Depending on the technology, they are susceptible to temperature, magnetic field, sound, light, radiation and frequencies. Some of the current technologies used to represent qubits include superconducting qubits, quantum dots, trapped ion, photonice qubits, and more less popular ones. For the purpose of the proposed work, the quantum technology used is superconducting qubits.

The umbrella of quantum computers covers a range of quantum technologies with the particular technology of interest being quantum logic gates.

2.2.2 Quantum Logic Gates

Quantum logic gates function similarly to classical logic gates except for the fact that they use the quantum mechanics of superposition and entanglement. Qubits, the quantum equivalent of bits, pass through different gates and undergo different operations based on which gate they pass through. These quantum gates manipulate the probabilities of the state of the qubit by using linear operators to change their superposition. In order to measure qubits that have passed though these gates, a classical register is tied to each qubit to be measured and multiple measurements will display its state probability.

Quantum logic gates can act on one or multiple qubits, with the basic quantum gates acting on only singular qubits. Basic logic gates manipulate a qubit's three bases: X, Y, and Z-bases. One such gate is the X-gate or not-gate. The X-gate will switch the amplitudes between the states $|0\rangle$ and $|1\rangle$. The X-gate is represented by the Pauli-X matrix $\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$. To see the effect a gate has mathematically, the qubit's statevector gets multiplied by the gate.

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

Other basic gates include the Y and Z gates which perform rotations by π around the y and z-axis of the qubit Bloch sphere respectively. The states $|0\rangle$ and $|1\rangle$ are eigenstates of the Z gate and makes up what is known as the *computational basis*. Another basis is the basis formed by the eigenstates of the X-basis. These two states are the vectors $|+\rangle$ and $|-\rangle$. These two states are represented by the following vectors.

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix}$$

The next gate is the Hadamard or H-gate. The significance of the H-gate is that it can rotate the qubit away from the poles of the Bloch sphere and create a superposition of $|0\rangle$ and $|1\rangle$. The H gate is represented by the following matrix.

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

Applying the H gate to the computational basis states will perform the following transformation:

$$H|0\rangle = |+\rangle$$

$$H|1\rangle = |-\rangle$$

Other quantum logic gates includes ones that act upon multiple qubits. The controlled X gate or CNOT gate is a quantum logic gate that acts upon two qubits: the target qubit and the control qubit. The state of the target qubit will only be flipped if the state of the control qubit is $|1\rangle$. If the control qubit is in the state $0\rangle$ then the value of the target qubit remains unchanged. More advanced control gates can act on multiple qubits and only change a qubit's state if the other qubits are in a certain state or they can be used to swap the states of two qubits. Quantum circuits can all be broken down into these basic single qubit logic gates and control gates.

2.2.3 Qiskit

Qiskit is an open-source SDK for interacting with quantum systems and simulators [2]. It allows you to work with a quantum computer at the circuit level. The appeal of Qiskit is the ease of integration as the framework utilizes python and allows interaction with IBM's quantum simulators and hardware architectures, such as their quantum computer using superconducting qubits. The several simulators within Qiskit allow for modeling of the simple circuit and offer information on the predicted states of the qubits in the circuit throughout simulation without having to measure the final system state. In addition, it offers several options for noise characterization and circuit optimization when working with quantum hardware to reduce circuit noise.

2.2.4 Quantum Fourier Transform

Quantum Fourier Transform or QFT is the quantum implementation of the discrete Fourier transform over the amplitudes of a wavefunction [2]. Several notable algorithms that use QFT are quantum phase estimation and Shor's factoring algorithm.

The discrete Fourier transform acts on a vector ($\vec{x}$) and maps it to a vector ($\vec{y}$). This process is done with the formula (where $\omega_N^{jk} = e^{2\pi i \frac{jk}{N}}$):

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk} \tag{2.1}$$

The quantum Fourier transform instead acts on the quantum state $|X\rangle = \sum_{j=0}^{N-1} x_j \omega_N^{jk}$ and maps it to the state $|Y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$. This process is also done adhering to the equation shown in 2.1. An important thing to note is that only the amplitudes of the state are being affected by the formula. This process can also be expressed by the unitary matrix:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle\langle j|$$

The QFT essentially transforms between the computational basis or Z basis and the Fourier basis. A single-qubit QFT can be accomplised using the hadamard (H gate) as it transforms from the Z-basis states $|0\rangle$ and $|1\rangle$ to the X-basis states $|+\rangle$ and $|-\rangle$. Similarly, all multi-qubit states in the computational basis will have a corresponding state in the Fourier basis with quantum Fourier transform and inverse quantum Fourier transform being the function that transforms qubits between these two state. States in the Fourier basis are often denoted by a $\sim$ to differentiate them.

In the computational basis state the numbers are stored in binary, similar to classical, and are encoded using the state $|0\rangle$ and $|1\rangle$. If the qubit is considered to be a bloch sphere than the the up position would correspond to the $|0\rangle$ state while the down position would correspond to the $|1\rangle$ state. However when considering the X-basis states, instead of the up/down Z-axis it is related to the X-axis and manipulating a qubit's X-state will entail rotating it on the X-axis in the block sphere. Another way to think about it would be rotating it around the Z-axis, which is the up/down axis.

Storing numbers in the Fourier basis would require dictating the angle in which it is rotated around the sphere. To encode the state $|\tilde{5}\rangle$ on four qubits, the least significant qubit is rotated by $\frac{5}{2^n} = \frac{5}{16}$ turns or $\frac{5}{16} \times 2\pi$ radians. The next qubit (going in order from least significant to most significant) will turn double the previous amount or $\frac{10}{16} \times 2\pi$ radians. The angle of rotation will double for the next qubit until all qubits have been rotated properly.

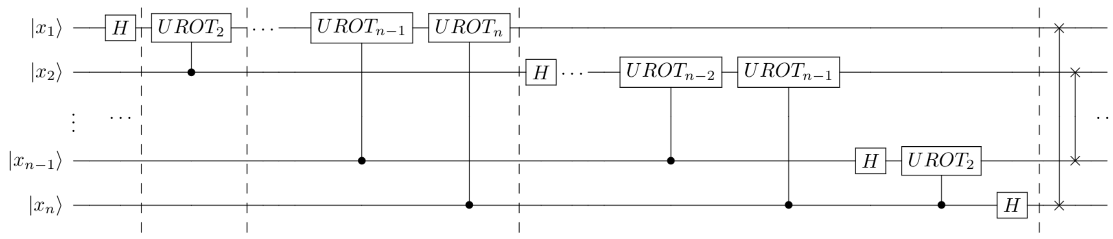

Figure 2.1: Generic Quantum Fourier Transform as outlined in [2]. The qubit x_1 is the least significant qubit while x_n is the most significant qubit.

The generic version of the QFT circuit primarily makes use of two gates: the Hadamard and the two-qubit controlled rotation or $CROT_k$ gate. The first stage of circuit involves placing a Hadamrd gate on the first qubit transforming the state to

$$H_1|x_1x_2\dots x_n\rangle = \frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2}x_1)|1\rangle] \otimes |x_2x_3\dots x_n\rangle$$

The application of the unitary rotation gate on qubit one and controlled by the second qubit will move the state of the circuit to

$$\frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1)|1\rangle] \otimes |x_2x_3\dots x_n\rangle$$

Stage three in the circuit is the application of the last $UROT_n$ gate on qubit one and controlled by qubit n, moving the state to:

$$\frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^n}x)|1\rangle] \otimes |x_2x_3\dots x_n\rangle$$

After the previously mentioned gate sequence is repeated for all qubits the final state of the circuit is

$$\frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^n}x)|1\rangle] \otimes \frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^{n-1}}x)|1\rangle] \otimes \cdots \otimes \frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^2}x)|1\rangle] \otimes \frac{1}{\sqrt{2}}[|0\rangle + \exp(\frac{2\pi i}{2^1}x)|1\rangle]$$

However, the order of the qubits is reversed in the output state so swap gates are neccessary to obtain the correct qubit order. In addition, another thing to note is that as the QFT circuit becomes large, many gates will only perform slight rotations upon the circuit so it may be prudent to ignore certain angle changes below a certain threshold in order to decrease the circuit noise while still obtaining acceptable results.

2.2.5 Quantum Phase Estimation

Quantum Phase Estimation (QPE) is a fundamental subroutine in quantum algorithms. Quantum phase estimation when given a unitary operator U, will estimate θ in $U|\psi\rangle = e^{2\pi i\theta}$ [2]. $|\psi\rangle$ is an eigenvector and $e^{2\pi i\theta}$ is the eigenvalue of the vector. In addition, since U is unitary, all of it's eigenvalues will have the norm of 1 [2].

The QPE algorithm utilizes phase kickback in order to write the phase of U, which is in the Fourier basis, to qubits in a counting register. Phase kickback or kickback is the process in which the eigenvalue being added by a gate for a qubit is "kicked back" into a different qubit [2]. This is done by using a controlled operation. An example of this would be the circuit displayed in Fig. 2.2.

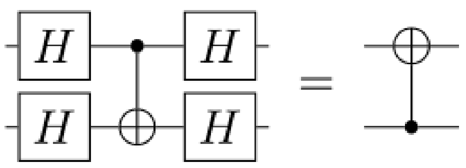

Figure 2.2: Phase Kickback Example

By using the Hadamard gate, the state of the qubit can be transformed from $|0\rangle$ to $|+\rangle$ or $|1\rangle$ to $-\rangle$ and wrapping a CNOT gate in H gates results in the previous circuit.

Once the phase of U is written in the Fourier basis, the inverse Quantum Fourier Transform moves the qubits from the Fourier basis to the computational basis. The computational basis state is necessary as it can be measured.

The general quantum circuit for phase estimation is comprised of t qubits for counting and qubits to hold the state $|\psi\rangle$.

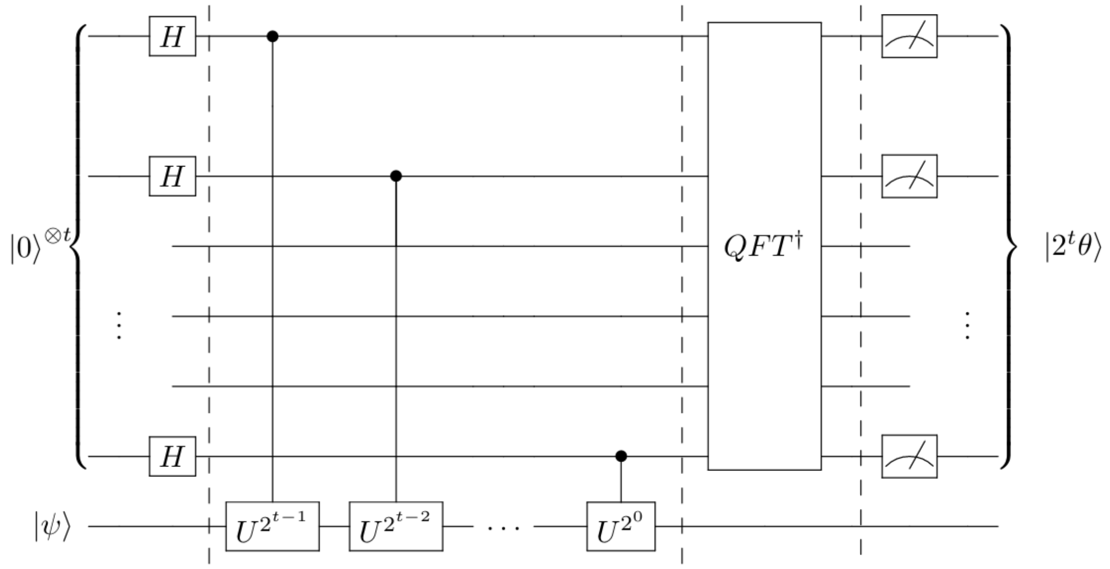

Figure 2.3: Generic Quantum Phase Estimation consisting of three main stages: Walsh-Hadamard transform, Controlled Unitary application, and inverse Quantum Fourier Transform

The first stage of the circuit is the circuit setup while the second stage includes the Walsh-Hadamard transform which essentially places a Hadamard gate on the every qubit in the counting register resulting in:

$$|\psi_1\rangle = \frac{1}{2^{\frac{n}{2}}}(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle \tag{2.2}$$

The second stage includes controlled unitary operators to apply the U operator with eigenvector $|\psi\rangle$ on the target register when the control bit is in the state $|1\rangle$.

$$U^{2^j}|\psi\rangle = U^{2^{j-2}}U|\psi\rangle = U^{2^{j-1}}e^{2\pi i\theta} = ... = e^{2\pi i 2^j\theta}|\psi\rangle \tag{2.3}$$

After the n controlled operations are applied, and where k denotes the integer representation of n-bit binary numbers, the state of the system is now $\frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1}|k\rangle \otimes |\psi\rangle$. The third stage of the circuit uses inverse quantum fourier transform to then recover the state from the fourier basis. Once this is completed the final state of the circuit before measurement is:

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{k=0}^{2^n-1} e^{-\frac{2\pi ik}{2^n}(x-2^n\theta)}|x\rangle \otimes |\psi\rangle \tag{2.4}$$

The final stage of the circuit is the measurement stage. The previous equation will peak near $x = 2^n\theta$ and for the cases in which $2^n\theta$ is an integer, measuring in the computational basis gives the phase of the control register with the highest probability: $\psi_4\rangle = |2^n\theta\rangle \otimes |\psi\rangle$ [2].

2.2.6 HHL Algorithm

In 2009, Harrow et. al published a paper describing a quantum algorithm for solving linear systems of equations [1]. The problem can be defined as, given a matrix A and a vector $\vec{b}$, find a vector $\vec{x}$ such that $A\vec{x} = \vec{b}$. For the purposes of the algorithm proposed, the case being considered is one in which the solution $\vec{x}$ does not need to be known but rather an approximation of the expectation value of some operator associated with the solution vector [1].

The HHL algorithm estimates the function of the solution vector in running time complexity $O(\log(N)s^2k^2/\epsilon$ given that the matrix A is s-sparse and well-conditioned [1]. In contrast, a classical computer will solve a s-sparse system of size N in $O(Nsk\log(1/\epsilon))$. In both time complexity equations, k denotes the condition number of the system and ϵ the accuracy of the approximation [2].

The HHL quantum algorithm first encodes the problem into a quantum state by mapping the N entries of $\vec{b}$ onto the $\log_2 N$ qubits. By rescaling the linear system, it is assumed that $\vec{b}$ and $\vec{x}$ are normalized. They can then be mapped to the quantum states $|b\rangle$ and $|x\rangle$. The vector is mapped such that the i^{th} component of $\vec{b}$ corresponds to the amplitude of the i^{th} basis state of the quantum state $|b\rangle$. Thus, the problem is rescaled to $A|x\rangle = |b\rangle$. Due to the fact that matrix A is Hermitian, it has the following spectral decomposition.

$$A = \sum_{j=0}^{N-1} \lambda_i |u_j\rangle\langle u_j|, \ \lambda_j \in \mathbb{R}$$

$|u_j\rangle$ is the j^{th} eigenvector of matrix A and has the eigenvalue λ_j thus:

$$A^{-1} = \sum_{j=0}^{N-1} \lambda_i^{-1} |u_j\rangle\langle u_j|$$

The right hand side of the system can also be re-written in the eigenbasis of A so that quantum state $|b\rangle$ is equal to the following.

$$|b\rangle = \sum_{j=0}^{N-1} b_j |u_j\rangle, \ b_j \in \mathbb{C}$$

After complete execution of the HHL algorithmn, the goal is to have the state of the readout register equal to the following equation.

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1} \lambda_i^{-1} b_j |u_j\rangle$$

The HHl algorithm can be broken down into six different circuit stages: loading the $|b\rangle$ input, Quantum Phase Estimation, Eigenvalue Inversion, inverse Quantum Phase Estimation, ancilla qubit measurement, and $F(x)$ application and measurement. $F(x)$ can be any linear equation in which some quantum mechanical operator is applied in order to obtain an estimate of the expectation value.

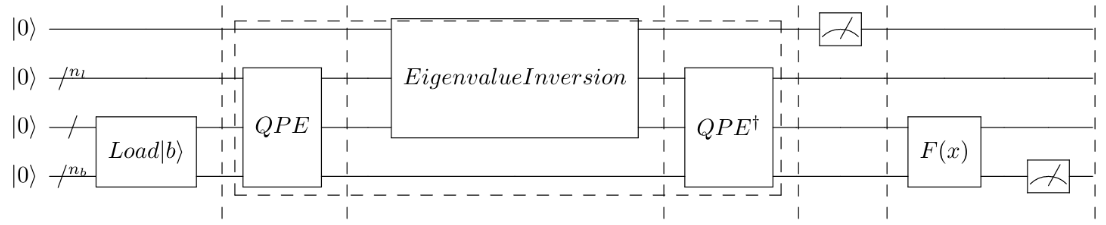

Figure 2.4: Generic HHL Algorithm with 5 states: state $|b\rangle$ preparation, Quantum Phase Estimation as detailed in 2.2.5, Eigenvalue Inversion, inverse QPE, measurement of the flag qubit, and application of some function $F(x)$.

The first stage loads the binary representation of b into the input qubit register. This register will contain the vector solution once the algorithm is finished and is denoted by n_b [2]. Once the data b is encoded into n_b the current state of that register is now $|b\rangle_{n_b}$. Next Quantum Phase Estimation is applied using the following Unitary operator U. The qubit register n_l is the register used to hold the binary representation of the eigenvalues of matrix A. The Quantum Phase Estimation process is better detailed in 2.2.5.

$$U = e^{iAt} := \sum_{j=0}^{N-1} e^{i\lambda_j t} |u_j\rangle\langle u_j|$$

By applying QPE, $|b\rangle$ is decomposed in the eigenbasis of A and the quantum state of the register is now equal to the following equation [1][2].

$$\sum_{j=0}^{N-1} b_j |\lambda_j\rangle_{n_l} |u_j\rangle_{n_b}$$

$|\lambda_j\rangle_{n_l}$ is the n_l-bit binary representation of $\lambda_j\rangle$. The eigenvalue inversion step requires a conditioned Y-basis rotation on $|\lambda\rangle_j$. The Y rotation is a controlled rotation acting upon a ancilla/auxiliary qubit. The ancilla qubit is used to determine whether the eigenvalue inversion was successful as the inversion step uses a unitary and not a linear operation [1]. After this step the state of the system is now equal to the following:

$$\sum_{j=0}^{N-1} b_j |\lambda_j\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}|0\rangle + \frac{C}{\lambda_j}|1\rangle} \right)$$

Here C is a normalization constant that is less than the magnitude of the smallest matrix eigenvalue $\lambda_{\min}$. The next stage in the equation is inverse quantum phase estimation. The application of this stage moves the system stage from the quantum Fourier basis back to the computational basis [2]. If the inverse quantum phase estimation was executed without error than the system state is now equal to:

$$\sum_{j=0}^{N-1} b_j |0\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}|0\rangle + \frac{C}{\lambda_j}|1\rangle} \right)$$

After inverse quantum phase estimation, the ancilla qubit is measured. If the qubit is measured to be $|1\rangle$ then the state of the system after measurement is

$$\left(\sqrt{\frac{1}{\sum_{j=0}^{N-1} |b_j|^2/|\lambda_j|^2}} \right) \sum_{j=0}^{N-1} \frac{b_j}{\lambda_j} |0\rangle_{n_l} |u_j\rangle_{n_b}$$

This state corresponds to the solution vector of the system after accounting for the normalization constant factor introduced earlier.

2.2.7 Swap Test

In certain instances it may be of use to compare the overlap between quantum states. The *SWAP-test* routine is a quantum algorithm that expresses the scalar product of two input states [9]. Two input states $|\phi_1\rangle$ and $|\phi_2\rangle$ are compared using SWAP gates and a control bit. The circuit for a two state comparison is shown in 2.5.

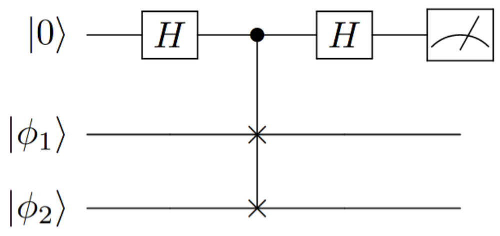

Figure 2.5: 2-state SWAP Test

The ancillary or control qubit is independent of the dimension of the state register and the application of the first Hadamard gate moves the qubit from $|0\rangle$ to the superposition state $|+\rangle$[10]. The CSWAP gates will then exchange the pair of input states $|\phi_1\rangle$ and $|\phi_2\rangle$ if the control qubit is in state $|1\rangle$. The second Hadamard gate will transform the input pair into a superposition of symmetric and anti-symmetric state coupled with the state of the ancillary qubit [10]. These states are represented as follows:

$$\frac{1}{2}|0\rangle(|\phi_1\rangle|\phi_2\rangle + |\phi_2\rangle|\phi_1\rangle) + \frac{1}{2}|1\rangle(|\phi_1\rangle|\phi_2\rangle - |\phi_2\rangle|\phi_1\rangle)$$

The probability of measuring the the ancillary qubit as $|0\rangle$ relates to the inner product of the two states: $|\langle\phi_1|\phi_2\rangle|^2$ [10].

$$p_0 = \frac{1 + |\langle\phi_1|\phi_2\rangle|^2}{2}$$

Thus the two-state swap test essentially performs the following transformation:

$$|0\rangle|\phi_1\rangle|\phi_2\rangle \rightarrow \sqrt{p_0}|0y\rangle + \sqrt{1-p_0}|1\acute{y}\rangle$$

In the above transformation y and $\acute{y}$ are garbage states. The probabilty $P(|0\rangle) = 0.5$ means that the states are orthogonal, while the probability $P(|0\rangle) = 1$ means the two states are identical [9]. In addition, the SWAP test can be used to calculate the Euclidean distance between vectors [9].

2.3 Related Works

In 2019, Google published a paper establishing quantum supremacy [11]. The challenge was to prove that quantum computers could perform certain computational tasks exponentially faster than the same task executed on a classical computer.

A quantum processor with 53 programmable superconducting qubits was constructed that corresponded to a computational state-space dimension of 2^{53}. The Sycamore processor was then tasked with sampling the output of a pseudo-random quantum circuit and then comparing those results to state-of-the-art classical computers [11]. When sampling the quantum circuit's output, a set of bitstrings is produced. However, due to quantum mechanics, the probability distribution of the output bitstrings resembles a speckled intensity pattern produced by light interference in laser scatter, so some bitstrings are more likely to occur in the set than others [11]. As such, using even state-of-the-art classical computers, computing this probablility distribution becomes more difficult as the width and depth of the testing circuit grows. In order to verify the quantum processor functions as expected, cross-entropy benchmarking is used. Cross-entropy benchmarking compares how often each bitsring gets observed experimentally with its corresponding ideal probability that was computed through simulation on a classical computer.

For the largest circuit size implemented, 53 qubits and 20 cycles, obtaining a million samples on the quantum processor took 200 seconds. In the case of the classical circuit, the simulated sampling would take 10,000 years on a million cores and verifying the fidelity of the circuit would take millions of years. This shows that currently even the most powerful classical computers are unable to simulate quantum effects. While this experiment is not very practical in terms of industry implementation and is focused on simulating quantum mechanics, it does establish what has been termed *quantum supremacy.*

While the work in [11] provides a baseline for quantum potential, it does not possess much practical application beyond the world of physics. In, 2009 a paper published by Harrow, Hassidim, and Lloyd detailed a quantum algorithmn for solving a linear systems of equations [1]. Their work showed that under certain cases quantum computers could solve for an approximation of $\vec{x}$ in $poly(\log N)$ time, compared to classical algorithmns which could estimate the same solution in $O(N poly \log(N))$ time [1]. Note: These cases are for when $\vec{x}$ itself is not needed, but for when the approximation of the expectation value of an operator associated with $\vec{x}$ is needed, i.e $\vec{x}^{\dagger} M \vec{x}$. The HHL algorithm outlined an approach for solving a linear systems of equations but does not implement it.

The work introduced in [12] expanded on the HHL algorithm and proposed a circuit design for solving a linear systems of equations. The proposed circuit design was then implemented for a specific 2×2 matrix. Due to quantum computer constraints, the main focus of the experimentation was outlining the effects of changing the accuracy of the controlled rotation angle and not finding $\vec{x}$ [12]. Cao et. al did not provide a method for checking the solution state $|x\rangle$.

In [13] the authours outlined a method of implenting Bayesian deep learning on a quantum computer. The Bayesian approach to deep learning typically includes learning a direct mapping to probabilistic outputs. Bayes' Theorem is used to to determine these conditional probabilities. One advantage that Bayesian methods have over other machine learning methods is that they often provide estimates of the uncertainty associated with the prediction. Recently, deep feedforward neural networks have been used in tandem with Gaussian processes, allowing for training without backpropogation [13]. Zhao et. al [13] leveraged a quantum algorithm designed to be used for Gaussian processes in order to develop a new algorithm for Bayesian deep learning on quantum computers. The kernel matrix properties in the Gaussian processes allowed for efficient execution of the quantum matrix inversion component of the algorithm. It was shown that this method provided an a polynomial speedup when compared to classical algorithms.

The matrix inversion operation performed was performed on quantum computer using Qiskit. This circuit was an adapted version of the quantum circuit proposed by [12] and solved a nontrivial linear system of equations. The Bayesian training implemented in Zhao et. al [13] calculated the mean and variance of the predictive distribution of the given Gaussian process model by inverting the covariance matrix, which was done using the previously mentioned quantum circuit.

Recent developments in connection between Bayesian models, Guassian processes, and deep feedforward neural networks have led to interest in implementing quantum algorithms for calculations due to its probabilistic nature and $O\log(N)$ calculation time.

While the main focus of the work was to analyze gate and measurement noise and its effect on the circuit noise, of particular interest is the swap test used to check the accuracy of the result calculated in the problem-specific quantum circuit for a 2×2 system.

In, [14] a proposed circuit design using a 7-qubit circuit was implemented using IBM's quantum experience Qiskit. The circuit solves a four variable regression problem utilizing elementary quantum gates. In addition, they used a group leader optimisation algorithm in order to create a low-cost circuit approximations for the Hamiltonian simulation. The optimization algorithm was used to decompose a more complex $\mathbf{U}$ gate into elementary gates and then a scipy optimization algorithm was used to approximate the R_x and R_{zz} gates rotation values. This circuit was then tested and compared against a predicted solution with a 0.1666 error (the L2-norm between calculated and predicted values). The main focus of [14] was an implementation of quantum circuit that could solve a specific systems of linear equations case for the purpose of linear regression. The introduced quantum circuit could solve a

4×4 system. Once the solution was encoded into the quantum state the information is then processed into classical data and then the simulated experiment solution was compared against the predicted solution. While this work proposed a quantum circuit design for 4×4 matrices, it does not provide a subroutine for $|b\rangle$ state preparation or reading out the solution vector.

Other related works include a paper published by Maria Schuld et.al. Maria Schuld et. al [15] explores the strategy in which data can be encoded into a model which will influence the expressive power of quantum circuits when used as function approximators. Quantum computers can be used for supervised learning when treating quantum circuts as models that map data inputs to predictions [15]. Specifically in this case, a partial Fourier series is represented by a quantum model in which the accessible frequencies are determined by the data encoded into the gates in the circuit. In order to access more complex frequency spectra data, encoding gates are repeated multiple times in the model. The models considered in this paper consist of a *data encoding (circuit) block* and a *trainable (circuit) block*. It is assumed that all input features are encoded by gate using Hamiltonian transforms.

It was then concluded that for both integer or non-integer frequencies, how expressive a quantum model is is determined by the frequency spectrum of the quantum model and expressivity of the coefficients controlled by the model. In addition, many quantum machine learning algorithms which use other data encoding strategies already perform an implicit pre-processing of the data and then use the time-evolution encoding studied in this paper, meaning the results of this paper are also applicable with those algorithms. One thing to note is that it is suggested that in comparisons between classical and quantum models the same pre-processed features should be passed to both models.

This paper explores the use of models that use a quantum circuit to map data inputs to predictions [15]. It details data encoding on a quantum circuit and discusses practical implications for quantum machine learning. While the focus was on data encoding and prediction for a partial Fourier series, the data encoding method could be expanded for solving linear systems of equations.

On the side of classical neural networks. A concern for many Deep Neural Networks (DNN) is that while they have achieved state-of-the-art results in a wide range of tasks, this was done with large training sets and models. GPUs are required with these large training sets and complex models due to their computations speed. However in most cases, these GPUs are power intensive and not intended to be used for low-power devices. This gap in application has led to interest and development in dedicated hardware that could implement these deep learning neural networks without the large models that require GPUs. Courbariaux et. al demonstrated a method of training deep neural networks with binary weights during propagation [16]. These binary weights were constrained to either -1 or 1. By implementing this method many multiply and accumulate operations during training were replaced with simple accumulate operations, decreasing the power and overall size requirements of the device. This is due to the fact that multipliers are often the most space and power-hungry components within hardware. The method proposed during this paper is called BinaryConnect and implements a method of training a DNN with binary weights during forward and back propagation. The precision of the stored weights in which by the gradients are accumulated remains unchanged. This BinaryConnect method of training deep neural networks offers a space in which quantum computing algorithms can be applied. An issue with current quantum computing is that precision can often be lost in computation due to its probabilistic nature, so using a quantum algorithm on a multi-precision problem does not seem like a viable option currently. Thus, a problem domain in which precision is restrained while speed is desired is an ideal application space for quantum technology.

Draghici [4] presents results which describes the relationship between the difficulty of the problem set and the weight range necessary required to ensure a solution. The difficulty of the problem of a set is described by the minimum distance between the patterns of the different classes and the weight range specifies the precision range required, meaning that the the network has the capability to solve the given problem set using weights within the specified range precision. The networks tested within this study used integer weights. One thing to note is that how the network is trained is not addressed, but rather, how well the network performed with the constrained integer weight values. The purpose of the results obtained in [4] was to be used in conjunction with a chosen specified training method in order to obtain the best trade-off between network performance and efficiency.

In [5] three different methods of training quantized neural networks are explored, Deterministic Rounding, Stochastic Rounding, and Binnary Connect. Binary Connect is a method described in another related work and the algorithm uses gradient updates with full-precision but binarizes the weights before gradient computation [16]. Deterministic Rounding will push a floating point value to the nearest specified quantized value while Stochastic Rounding is non-deterministic and will round its arguments with probability [5]. The Binary Connect and Stochastic Rounding methods were analyzed analytically under both cases of convergence and non-covergence.

In the case of convergence results both Binary Connect and Stochastic Rounding had a prediction accuracy that was bound by the coarseness of the discretization. In the case of non-convergent problem sets, the Stochastic Rounding method was unable to exploit a greedy local search unlike in the case of conventional stochastic methods. Both of the previously mentioned training methods were tested with VGG-like and Residual networks with binarized weights on the problem of image classification [5]. Adam was used as the baseline as it gave better overall results than SGD. The results of the experiment show that Binary Connect with Adam showed comparable performance to the full-precision model. Stochastic Rounding with Adam outperformed

Deterministic Rounding with ADAM. There was also a performance gap between the Binary Connect and Stochastic Rounding methods across all the used models and datasets, showing that keeping track of the real-valued weights (with Binary Connect) would produce better results.

Li et. al [5] details the different methods in training a quantized neural networks and the limitations in using such training methods. The analysis of different methods and the advantages to using a quantized neural networks are also covered in this work. Yet, the analysis presented in this paper focuses on purely classical methods and does not include any hybrid training solutions for a quantized neural network.

Chapter 3

Experimentation and Architecture

3.1 Architecture

3.1.1 Circuit Architecture

The HHL algorithmn previously described in 2.2.6 specifies a generic formula for implementing a linear systems of equations solver using a quantum computer. Given a Hermitian matrix A, the algorithm transforms the matrix into a unitary operator e^{iAt} [1]. The transformation is possible if the the matrix A is s-sparse, meaning it has s nonzero entries per row. In the event that A is not a Hermitian matrix, define a variable such that

$$C = \begin{pmatrix} 0 & A \\ A^{\dagger} & 0 \end{pmatrix}$$

Since C is Hermitian, the original problem can be written as $C\vec{y} = \left(\begin{smallmatrix} \vec{b} \\ 0 \end{smallmatrix}\right)$ and now $y = \left(\begin{smallmatrix} 0 \\ \vec{x} \end{smallmatrix}\right)$. This transformation is necessary as rest of the HHL algorithm functions under the assumption the matrix A is necessary [1].

The original algorithm proposed in [1] does not require knowledge of the eigenvalues or eigenvectors of A prior to execution [2] as the matrix A is applied to the circuit through the use of the controlled unitary $U = e^{iAt}$ on the quantum state $|b\rangle$ (which holds the vector $\vec{b}$) for a superposition of different t values. [12]. After this applied unitary, as mentioned previously in 2.2.6, the state of the system becomes

$\sum_j \beta_j |\lambda_j\rangle|u_j\rangle$. Here $|\lambda_j\rangle$ is state that holds an encoded approximation to the eigenvalue λ_j of the matrix A. The unitary e^{iAt} shares the same eigenvectors with the matrix A while its eigenvalues are $e^{i\lambda_j t}$ [14].

However, while previous knowledge of the eigenvalues of the matrix is not necessary for the HHL algorithm, having a matrix with eigenvalues that are powers of 2 allows for the phase estimation subroutine to generate states that exactly encode the eigenvalues [12]. In addition, knowing the matrix has pre-specified eigenvalues simplifies the subroutine to find their reciprocals [12]. Thus, only matrices with specific eigenvalues are explored in this work.

From the generic HHL design shown in Figure 2.4 two specific cases were implemented in this work in order to solve specific cases of 2×2 and 4×4 matrices. The architecture for the 2×2 circuit was taken from [12] and is shown in Figure 3.1.

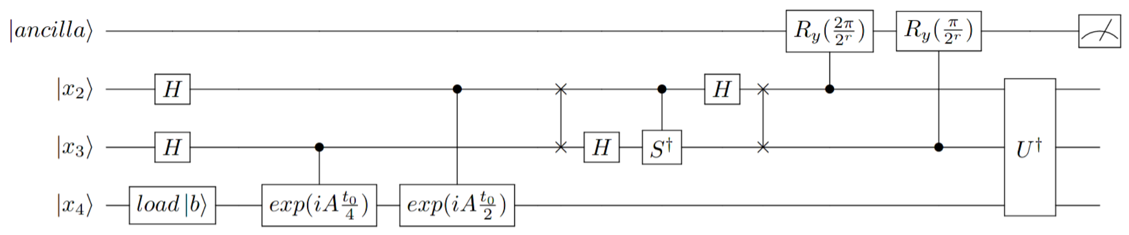

Figure 3.1: Circuit for 2×2 Solver: the circuit is comprise of one aniclla qubit, two clock register qubits x_2 and x_3, and one register b qubit. The controlled unitary $\exp(iA\frac{t_0}{2})$ is applied during the phase estimation and has values $t_0 = 2\pi$. The variable r is detailed in 3.1.1.4

The 2×2 circuit uses one ancilla qubit, two clock register qubits, and one qubit to hold $|\vec{x}\rangle$. The $U^\dagger$ gate shown in the circuit, represents applying all the gates previously applied prior to the controlled rotation on the ancilla qubit in reverse order. The second circuit use one ancilla qubit, four clock register qubits, and two qubits to hold $|\vec{x}\rangle$. The first part of the circuit is shown in Figure 3.2 and the second portion after the controlled rotation gates is shown in Figure 3.3. The circuit design for the 4×4 matrix solver originated from [14],[12].

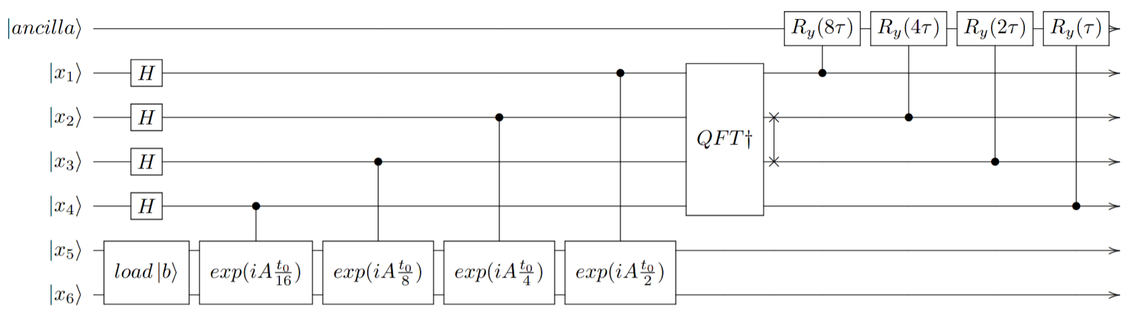

Figure 3.2: Circuit for 4×4 Solver

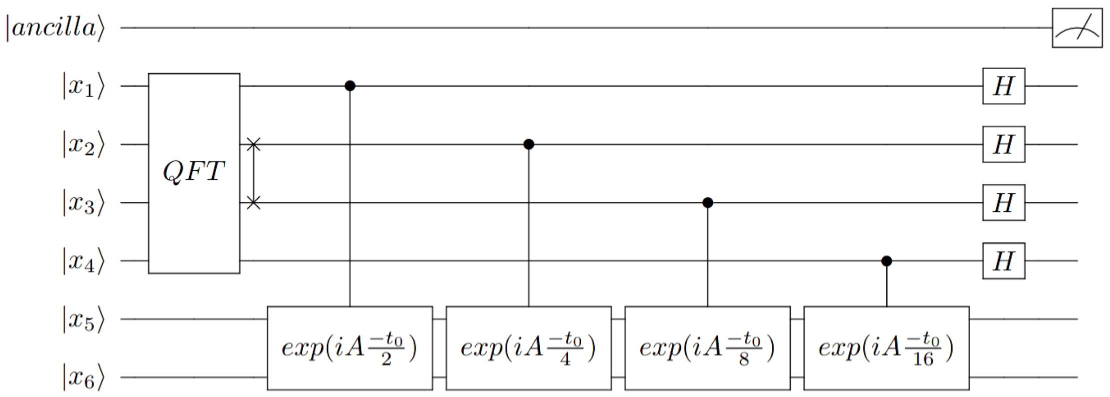

Figure 3.3: Circuit for 4×4 Solver pt.2

The two previous circuit designs solve the linear systems of equations but do not extract the solution vector from the state of the system. By adding a swap test to the circuit, the state of the qubit holding $|\vec{x}\rangle$ can be tested by measuring it's fidelity across a known state vector. This design is described in more detail in 2.2.7.

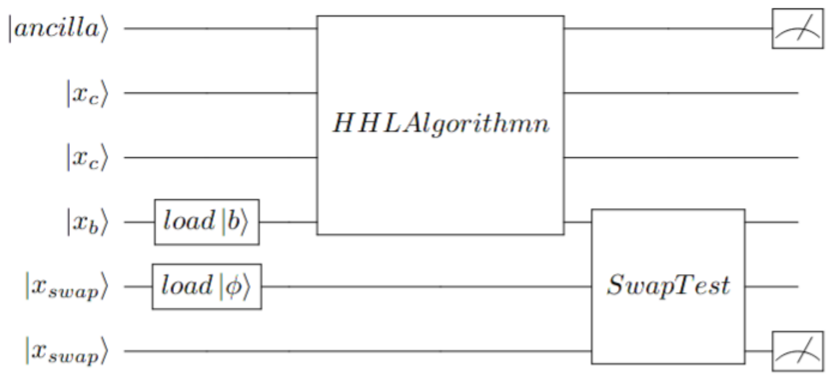

Figure 3.4: Full Circuit Design with Swap Test: the first steps load the state $|b\rangle$ and swap test state into the circuit, the second step applies the modified HHL algorithm, and the last step applies the swap test for measurement

As shown in Figure 3.4, the input vector $\vec{b}$ is encoded into the qubit and after the circuit execution the same qubit now holds the solution state vector. The state $|\phi\rangle$ is simultaneously loaded into the swap qubit so that its fidelity can be compared against the computed solution state. The following subsections detail each subroutine present in the circuit.

3.1.1.1 Preparation of $|b\rangle$

An important subroutine for the quantum circuit includes encoding the classical information into a quantum state. Several methods exist for encoding data into a qubit including basis encoding, amplitude encoding, and angle encoding. However, the process of loading data onto a quantum device is not a trivial task as unlike in classical computers information cannot be loaded several times without erasing it [17]. Due to the no-cloning theorem [18], noisy quantum operations [7], and decoherence-bounds [19] state preparation in a quantum device becomes difficult. The no-cloning theorem states that it is not possible to perform a copy of an arbitrary quantum state as when a quantum operation is applied its input is transformed or collapsed [18]. So, even if the data is represented using basis state encoding it will be difficult to copy the data without corrupting it [17]. Thus for this circuit design, the classical information will be encoded using amplitude encoding. Amplitude encoding stores the classical data in the amplitudes of the states of the qubits. A toy example would be encoding the values $[1, 0]$ into a single qubit. After encoding the classical date the state of the qubit will be $|0\rangle$ as the expanded notation is $|0\rangle = 1|0\rangle + 0|1\rangle$ [2]. The first value in the classical vector gets stored in the amplitude of the $|0\rangle$ state while the second value gets stored in $|1\rangle$. Thus, loading an input vector $\vec{x} = (x_0, \ldots, x_{N-1})$ into the amplitudes of a quantum system requires $\log_2(N)$ qubits [17]. The state of the system after encoding can be represented as:

$$x_0|0\rangle + \cdots + x_{N-1}|N-1\rangle$$

In order to encode classical data into a quantum state vector, the classical information must be normalized. This is due to the fact that the squares of the amplitudes of a qubit must equal one when added together [2]. The squared value of an amplitude of a qubit's state is equal to the probablity of the qubit collapsing to that state when measured [2].

Due to the issue of decoherence [19], it is important to implement optimal quantum circuit decomposition. In addition, the more gates present in a circuit introduces more gate noise affecting the results of the circuit [7]. Bergholm et. al introduced a method of local state preparation using uniformly controlled one-qubit gates and detailed implementing general n-qubit gates using elementary gates [20]. Then in [21], the authors describe a unitary transformation which maps any given state of a n-qubit quantum register into another one. By employing uniformly controlled rotations described in [20], Moettoenen et. al presents a quantum circuit of $2^{n+2} - 4n - 4$ $CNOT$ gates and $2^{n+2} - 5$ one-qubit elementary rotations to effect the previously described state transformation [20].

The state preparation of $|b\rangle$ subroutine used in this circuit design has been modified from an algorithm presented in [17]. Given an N-dimensional vector x, where $n = \log_2(N)$ the vector is loaded into a quantum register using the following process.

Algorithm 1 gen_angle(x)

input: A vector x with dimension $N = 2^n$

output: Angles to generate the amplitude encoding circuit

```
1:  Check if length of x is power of 2, if it is not break
2:  if $size(x) > 1$ then
3:      Create an auxiliary vector $new\_x$ with dimension $N/2$
4:      for $k \leftarrow 0$ to $length(new\_x)$ do
5:          new_x[k] = $\sqrt{|x[2k]|^2 + |x[2k+1]|^2}$
6:      inner_angles = $gen\_angles(new\_x)$
7:      Create a vector $angles$ with dimension $N/2$
8:      for $k \leftarrow 0$ to $length(new\_x)$ do
9:          if $new\_x[k] \neq 0$ then
10:             if $x[2k] > 0$ then
11:                 angles[k] = 2asin($\frac{x[2k+1]}{new\_x[k]}$)
12:             else
13:                 angles[k] = $2\pi$ - 2asin($\frac{x[2k+1]}{new\_x[k]}$)
14:         else
15:             angles[k] = 0
16:     if inner_angles is not None then
17:         angles = inner_angles + angles
18:     else
19:         angles = angles
20:     return angles
```

Algorithm 2 gen_circuit($angles$)

```
input: $N - 1$ dimensional vector angles = $gen\_angles(x)$
output: Quantum circuit to lad $x$ in the amplitudes of a quantum system
1: circuit = quantum circuit with $n = \log_2(N)$ qubits $q[0], \ldots, q[n-1]$
2: if $length(angles) == 1$ then
3:     $Ry(angle[0], 0)$
4: else
5:     for $k \leftarrow$ to $N - 2$ do
6:         j = level(k)
7:         index(k, j, q)
8:         $CR_y(angle[k], [q[0], \cdot, q[j-1], q[j]])$
9:         index(k, j, q)
10: return circuit
```

As stated previously the amplitude encoding subroutine shown in algorithms 1 and 2 were taken from [17] but modified to include the case in which only one qubit is required to encode the classical data. The amplitude encoding algorithm includes two parts: angle generation and circuit generation. The angle generation function takes in $\vec{x}$ and finds the angles to perform rotations that lead $|0\rangle_n \equiv |0\rangle^{\otimes n}$ to the encoded state. The second function will take in these angles and generate a quantum circuit of Y-Rotation and controlled Y-rotation gates from the calculated angle rotations. One thing to note is that the input vector into the function to generate angles functions must be normalized in order to encode the data into a quantum state vector.

The generate angles function divides the 2^n input vector into 2^{n-1} 2-dimensional subvectors and creates a new 2^{n-1} dimensional vector new_x with the norms of the generated subvectors. While the size of the generated sub vector is greater than 1, the generate angles function gets called recursively with the new sub vector until no more sub vectors are created. After the last recursive call, angle θ gets appended to the angle array such that $\sin(\theta/2) = \frac{x[2k+1]}{new\_x[k]}$ and $\cos(\theta/2) = \frac{x[2k]}{new\_x[k]}$. The premise of the angles algorithm is that the angle vector is a complete binary tree [17] with the overall cost of the generate angles function being $O(N)$ as each *gen_angles* call will perform $\log_2(N)$ recursive calls.

The circuit generation function receives the $N-1$ array of vector angles generated by the previously described function and outputs a circuit with the gates to load the classical data into the qubits. The function will first check if only one qubit is needed, if so, it will only add one Y-rotation gate with the computed angle. In the case where multiple qubits are needed the root of the binary angle tree generated by the *gen_angles* function defines the first rotation and then a top-down approach is used where the rotation of an angle is defined by the angles of the next level down in the tree. As defined in [17] the cost to compute the necessary angles and generate the circuit is $O(N)$. $O(N)$ multi-controlled gates are applied sequentially at a circuit depth of $O(N)$.

3.1.1.2 Quantum Phase Estimation Design

The Quantum Phase Estimation subroutine consists of three main parts: Walsh-Hadamrd Transform, Hamiltonian simulation, and inverse Quantum Fourier Transform. The Walsh-Hadamard Transform is accomplished by applying a Hadamard gate onto every qubit in the counting register in order to move away from the poles and create a superposition of $|0\rangle$ and $|1\rangle$ [2].

The inverse Quantum Fourier transform subroutine is well-known and described in detail in 2.2.4. Although one thing to note is the most significant bit and least significant bit for the generic circuit in 2.2.4 and the circuit design presented in this thesis are different. This is due to the fact that qiskit and many physicists will list q_0 as their most significant qubit if their circuit is represented as $|q_0 q_1, \ldots, q_n\rangle$ [2]. However, for most computer notation q_0 is the least significant bit and the circuit was designed with that notation in mind.

The most difficult portion of the quantum phase estimation subroutine is the Hamiltonian simulation. For general non-sparse $N \times N$ Hamiltonian H, it is not possible to simulate e^{-iHt} in $poly(||Ht||, \log N)$ [12]. In addition, since the initial HHL algorithm [1], there have been improvements by using a linar combination of unitary operators to yield a cost scaling that is $poly(logN, 1/\epsilon)$ and nearly linear in $||Ht||$ [22]. For the purposes of this circuit design, the specific gate decomposition was found using a heuristic method introduced in [23], [24]. A group leaders optimization algorithmn was used to find the Hamiltonian situation for both the 2×2 and 4×4 matrices cases. For the 2×2 matrix the gate decomposition was detailed in [12] while the 4×4 matrix Hamiltonian gate decomposition was taken from [14]. In addition, since the specific case considered in this circuit design only considers matrices with eigenvalues that are powers of 2, they can be implemented exactly [12]. After implementation the state of the system will be

$$\sum_{j=1}^{N} \beta_j |u_j\rangle |(\tilde{\lambda_j} t_0 / 2\pi)\rangle.$$

However, if we chose our evolution time as $t_0 = 2\pi$. Then the state of the system will be

$$\sum_{j=1}^{N} \beta_j |u_j\rangle |\tilde{\lambda_j}\rangle.$$

In addition, since there are a sufficient number of qubits to encode the eigenvalues with phase estimation, $|\tilde{\lambda_j}\rangle = |\lambda_j\rangle$ [14]. The state of the system can be rewritten as

$$\sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle.$$

Figure 3.5 shows the decomposition of the hamiltonian simulation for eigenvalues of one and two into elementary gates.

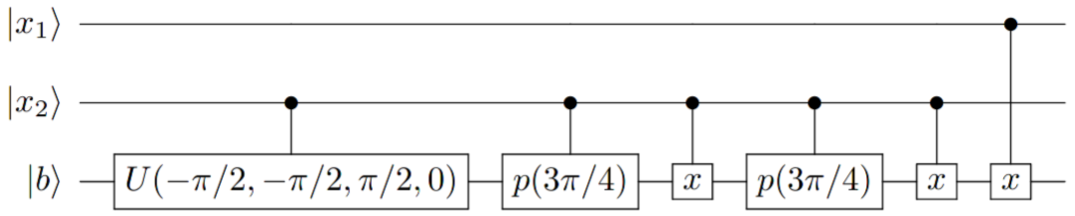

Figure 3.5: Hamiltonian Simulation for 2×2 Circuit: U is the controlled u gate and p is the controlled phase gate

For the Hamiltonian Simulation for the 4×4 matrix circuit, the simulation is applied on two qubits as two qubits are used to hold the output vector. Every gate applied in the gate decomposition is controlled on the corresponding counting qubit in the clock register. The hamiltonian operator $e^{iA\frac{2\pi}{16}}$ gate decomposition was found using the previously mentioned group leaders optimization algorithm heuristic and then the operators for $e^{iA\frac{2\pi}{8}}$, $e^{iA\frac{2\pi}{4}}$, and $e^{iA\frac{2\pi}{2}}$ were found by multiplying the angle shifts in the rotation gates by the necessary factor [12]. The specific gate decomposition for the hamiltonian $e^{iA\frac{2\pi}{16}}$ is shown in Figure 3.6.

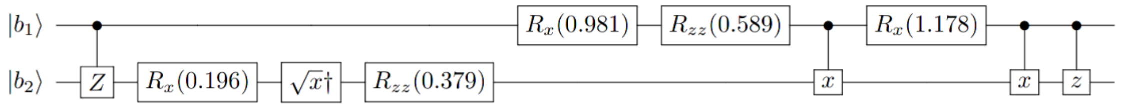

Figure 3.6: Hamiltonian Simulation for 4×4 Circuit

One thing to note is Figure 3.6 shows the gate decomposition of only the hamiltonian portion. Every gate in the circuit decomposition is also controlled by the clock register qubit the hamiltonian is being applied on. The other three hamiltonian gates are identical to the one shown in Figure 3.6 but the rotation values differ. The rotation values for all the Hamiltonian were taken from [14] and were found using the *scipy.optimize.minimize* function. The parameters for the rotation gates in the hamiltonian are shown in Table 3.1.

4×4 Hamiltonian Parameters	
Hamiltonian Simulation	Parameters
$e^{iA\frac{2\pi}{16}}$	[0.19634953, 0.37900987, 0.9817477, 1.87900984, 0.58904862]
$e^{iA\frac{2\pi}{8}}$	[1.9634954, 1.11532058, 1.9634954, 2.61532069, 1.17809726]
$e^{iA\frac{2\pi}{4}}$	[-0.78539816, 1.01714584, 3.92699082, 2.51714589, 2.35619449]
$e^{iA\frac{2\pi}{2}}$	[-9.01416169e-09, -0.750000046, 1.57079632, 0.750000039, -1.57079633]

Table 3.1: Table of Hamiltonian Parameters

The matrices applied by the elementary gates are shown below in order to clarify the operations being applied.

The X, Y, Z, and H gates represent the standard Pauli operators ϕ_x, ϕ_y, ϕ_z and Hadamard gates. The S and T (also known as $\frac{8}{\pi}$) gates are represented by

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} T = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\frac{\pi}{4})w \end{pmatrix}$$

The gate $\sqrt{x}^{\dagger}$ gate is known as the square root of X conjugate gate or $V\dagger$ gate. The matrix representation for this gate and its non-conjugate form are:

$$V^{\dagger} = \frac{1}{2}\begin{pmatrix} 1-i & 1+i \\ 1+i & 1-i \end{pmatrix} V = \frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$$

The rotation matrices are defined as follows:

$$R_x(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & i\sin(\frac{\theta}{2}) \\ i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix} R_y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & \sin(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\theta) \end{pmatrix} R_{zz}(\theta) = \begin{pmatrix} \exp(i\theta) & 0 \\ 0 & \exp(i\theta) \end{pmatrix}$$

The gate matrices were taken from [12].

3.1.1.3 Eigenvalue Inversion

The critical portion of a quantum circuit for solving linear systems of equations includes the subroutine for eigenvalue inversion. The work shown in [12] outlines a generic circuit for solving a linear systems of equations that includes a subroutine for eigenvalue inversion. This subroutine includes use of two additional registers and a controlled global phase shift R_{zz} in order to transform the state of the two registers to $\sum_p \sum_s \exp(i\frac{p}{2^m}t_0)|s\rangle|p\rangle$. In this state $|p\rangle$ and $|s\rangle$ represent the basis states of the two additional registers (M and L respectfully) used in the eigenvalue subroutine registers. The $|\lambda_j\rangle$ states stored in the clock register after the inverse Fourier transform is then used as a control for the Hamiltonian simulation applied on register M[12]. The applied Hamiltonian simulation is a diagonal matrix with diagonal elements $(1, 2, \ldots, 2^{m-1})$. The k_l-th qubit of register L is then used as the control qubit for the $\exp[-ip(\frac{\lambda_j}{2^m}\frac{1}{2^{l-k_l}}t_0)]$ simulation applied on register M. The binary values stored in register L then determine the time parameter t in the overall Hamiltonian $\exp(-iH_0t_0)$. The state of the system after the Hamiltonian situation is now

$$|0\rangle \otimes \sum_{j=1}^{n} \sum_{p=0}^{2^m-1} \sum_{s=0}^{2^l-1} \beta_j \exp[i\frac{p}{2^{m+l}}t_0(2^l - \lambda_j s)]|s\rangle|p\rangle|\lambda_j\rangle|u_j\rangle.$$

In the case of $t_0 = 2\pi$, the state $|s\rangle$ will be concentrated on the state $|\frac{2^l}{\lambda_j}\rangle$ [12]. This state can then be used to rotate the ancillab bit with the controlled angle shift.

However, for the purposes of this thesis, only a set of constrained eigenvalues are used in order to simplify the circuit design. By limiting the the eigenvalues to specific powers of 2, it is possible to accomplish eigenvalue inversion through a serious of swap gates. In the case of a 2×2 matrix the eigenvalues of the circuit should be 1 and 2. While for 4×4 matrices the eigenvalues should be 1, 2, 4, and 8. In these cases the eigenvalues are of the form $\lambda_i = 2^{i+1}$ [14]. For the 2×2 matrix case with eigenvalues of 1 and 2, a swap gate performed on the 2 register qubits will peform the neccessary eigenvalue inversion. By performing a swap gate between qubits $|x_2\rangle$ and $|x_3\rangle$ the system is transformed to the state $\beta_1|10\rangle|u_i\rangle + \beta_2|01\rangle|u_2\rangle$. The state encoding $|x_2x_3\rangle = |10\rangle$ can now be interpreted as the inverted eigenvalue $2\lambda_1^{-1} = 2$ and $|01\rangle$ as the value $2\lambda_2^{-1} = 1$ [12]. In the case of the 4×4 circuit, a swap gate is used to on the first and third qubit of the counting register [12]. The eigenvalue $\lambda_4 = 8$ is encoded as the state $|1000\rangle$. After the swap gate is applied the eigenvalue becomes $|0010\rangle$ which is equal to $\frac{1}{8} \times 2^4 = 2$ [12]. Thus for both circuit cases one swap gate is used to invert the eigenvalues of the system.

3.1.1.4 Controlled Y-Rotation

The next subroutine applied to the circuit applies controlled Y-rotation gates to the ancilla qubit conditioned on the clock register qubits. By applying rotation angles such that $\theta = \tilde{\theta_j} = 2^{1-r}\pi/\lambda_j$, the approximate angle $\theta_j = 2\arcsin(C/\lambda_j)$ is applied. C is the normalization constant discussed in 2.2.6 and is assumed to be $C \leq \min_j |\lambda_j|$. The reason C is needed is that it is a scaling factor to prevent the controlled rotation from being unphysical [25]. C can also be defined as $C = O(1/\kappa)$ [14].

The variable r is related to the rotation value of the circuit. As explored in [12],[13] r cannot be too small or else the small angle approximation $\tilde{\theta_j}$ of θ_j becames invalid.

However, an r too large reduces the probability of obtaining the solution and measuring the ancilla qubit as 1. Also, a larger value of r means implementing finer angles in the rotation gates which is also difficult with current quantum computer [7]. According to [12], the minimum angle resolution realizable can be defined as ω. Thus, r can range between $\log_2(2\pi)$ and $\log_2(\pi/\omega)$ [12]. For the purposes of our circuit design the r variable is chosen to be an 4, 5, or 6 depending on the circuit. These rotation angles are applied to the ancilla qubit and controlled on the state of the clock register.

3.1.1.5 Undo Subroutine

The last portion of the circuit for solving a linear systems of equations undoes the phase estimation subroutine. By undoing the quantum phase estimation the state of the system returns from $|\lambda_j\rangle$ back to $|0\rangle$ [12]. This will now bring the state of the system to

$$\sum_j (\sqrt{1 - \frac{C^2}{\lambda_j^2}}|0\rangle + (\frac{C}{\lambda_j})|1\rangle)\beta_j|0\rangle|u_j\rangle.$$

When measurement of the ancilla qubit is measured to be $|1\rangle$ then the state of register B is now

$$\sum_j C\frac{\beta_j}{\lambda_j}|u_j\rangle \propto |x\rangle.$$

The subroutine for undoing quantum phase estimation is quite simple. All previous gates prior to the controlled-Y rotation gates need to applied in reverse order. For example, if a swap gate was the gate applied prior to the controlled rotation subroutine then it is the first gate applied after the rotation, making sure the gates are applied to the same qubits. One thing to note is that when undoing phase rotations the inverse of θ is applied. So if a rotation of $\theta = \pi$ is applied than the a rotation of $\theta = -\pi$ is applied to reverse it.

3.1.1.6 Swap Test Subroutine

The last portion of the circuit includes the swap test subroutine. The swap test subroutine checks the fidelity between the predicted weight values and the weight values calculated by the circuit using swap gates and a control register. As discussed in Section 2.2.7, the probabability p_0 of measuring state $|0\rangle$ relates to the inner product between the two compared states.

By using a swap test, one is able to get an estimation of the weight values encoded in the solution state of the circuit without having to convert the solution state from quantum data to classical data. In [1] the original Harrow, Hassidim, and Lloyd quantum algorithm for solving a linear systems of equations, under ideal situations the circuit could be executed in $poly(logN)$ time. However, the HHL algorithm performs under the assumption that one is not interested in the value $\vec{x}$ itself but in an expectation value $\vec{x}^T M\vec{x}$ [1]. In this case M is a linear operator. In order to read out all the components of the quantum state $\vec{x}$ would require performing the measurement procedure at least N times, reducing the exponential speedup introduced. Schuld et. al proposed a method of writing the desired result state into selected entries of an ancilla's density matrix [26] and in [27] Wang introduces a three-stage algorithm where the second stage determines the parameter sign and the last stage determines whether the calculated vector $B \in \mathbb{R}^d$ is closer to $\hat{B}$ or $-\hat{B}$. In [26] the swap test is not used because their output domain exteneded into negative number. In contrast, a swap test is viable for the circuit design in this thesis as the interest is only in binary weights in the domain of [0,1].

By constraining the weights to a binary value, the expected solution is discrete and the implementation of the swap test subroutine becomes less complex. One difficulty with the SWAP test is accurately preparing the states to be compared. Yet, by using binary weights the expected value state becomes simple to implement. An expected weight value of [0,1] would require putting the state of the comparison qubit into $|1\rangle$, which is the state representation of $0|0\rangle + 1|1\rangle$. Figure 3.7 shows how the swap test is placed at the end of the circuit to test the computed $|\vec{x}\rangle$ state.

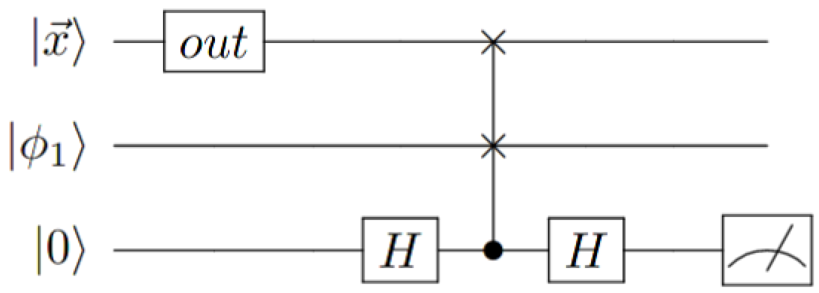

Figure 3.7: Swap Test Subroutine where $|x\rangle$ holds the computed solution state and $|\phi_1\rangle$ holds the test state

In the case of multiple qubits being used to hold the encoding of the solution vector, the same number of qubits needs to be used for each state comparison. Each qubit then gets compared to its corresponding qubit in the comparison state. For example, if qubits $|x_1 x_2\rangle$ hold the calculated state $|\vec{x}\rangle$ and qubits $|x_3 x_4\rangle$ hold the comparison state ϕ then qubit x_1 and x_3 will be swapped and x_2 and x_4 get swapped. The modified swap test circuit accounting for multiple qubits being needed to encode a state vector is shown below in Figure 3.8.

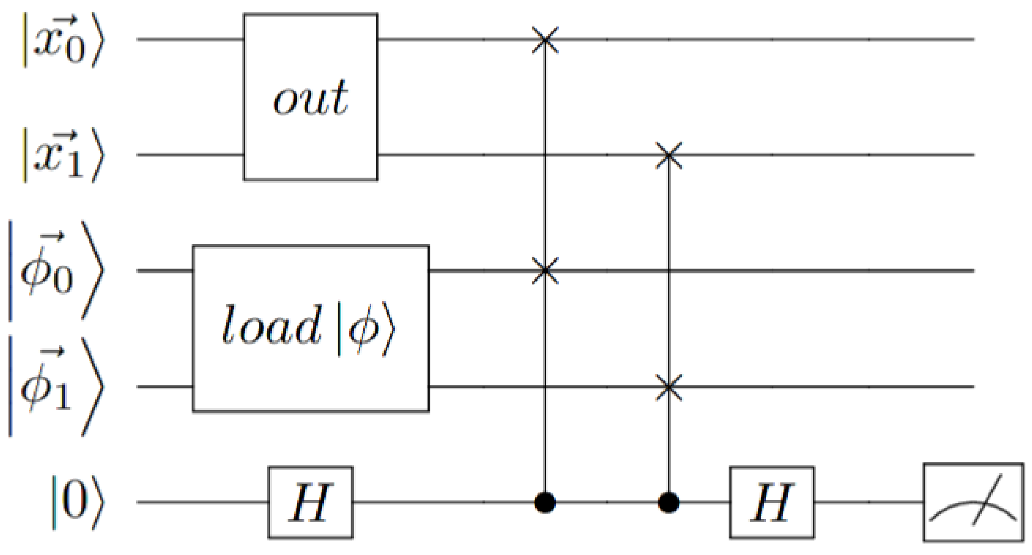

Figure 3.8: Swap Test with State Encoded in Multiple Qubits

[10] introduces a method of comparing multiple states in a multiple swap test. By using multiple control ancilla control bits, different states will be compared depending on the value of the control bit. The work in [10] uses swap gates to re-order the input states so that every possible pair of input states will be brought to the first two registers. Once the states that need to be compared are in the first two registers then a regular two-state swap test is performed. Which two states get swapped depends on the values of the control qubits. By testing mutiple states with one circuit, the search space of the possible weights is reduced as the number of possible weights is finite with quantized neural networks.

3.1.2 Training Algorithmn

While current quantum technology is unable to implement circuits large enough to solve systems of significant size, this work proposes a potential training flowchart shown in Figure 3.9.

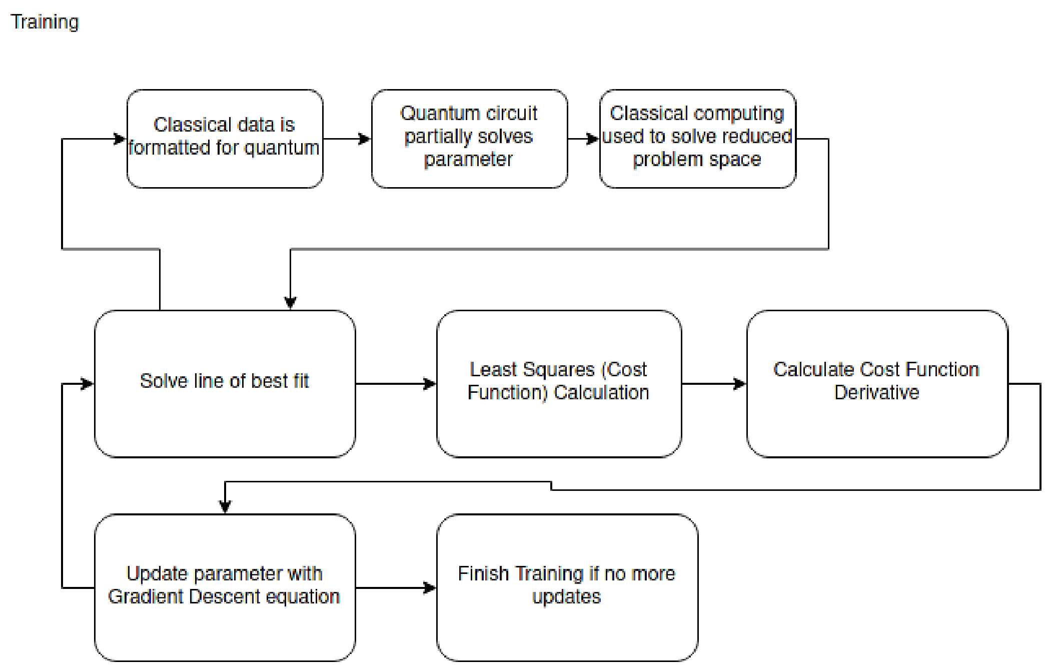
Training
Classical data is formatted for quantum
Quantum circuit partially solves parameter
Classical computing used to solve reduced problem space
Solve line of best fit
Least Squares (Cost Function) Calculation
Calculate Cost Function Derivative
Update parameter with Gradient Descent equation
Finish Training if no more updates

Figure 3.9: Potential Training Flowchart

Figure 3.9 outlines how a quantum circuit could be used to partially solve for the weight vectors in order to reduce the problem space for the classical computer. A classical computer is still needed to control the data flow, update the network, and calculate the cost function. Thus, the quantum circuit will only be used in the linear regression portion to partially solve the linear systems of equations. It is possible to partially solve the weight vector as the neural networks is a quantized network, meaning there is a discrete number of weights.

3.1.3 Estimating Number of Ones in Weight Vector

In the specific case being considered, all weights in the weight vector are constrained to values of either 0 or 1 as it is a binary neural network. As a result the expected output from the quantum circuit will have solution vector where the amplitudes are only zero or one.

As detailed in 2.2.7, the probability of measuring the ancilla qubit of a swap test as 0 is related to the fidelity between the two states being compared. If the two states being compared are $|x\rangle$ and $|test\rangle$, where $|x\rangle$ is the solution state and $|test\rangle$ is the test vector, the the probability of measuring 0 in the ancilla qubit is:

$$P(0) = \frac{1 + 1|\langle x|test\rangle|^2}{2} = \frac{1}{2} + \frac{1}{2}|\langle x|test\rangle|^2$$

Since the weight vector only has the value of 0 or 1, the amplitude of each weight will have a value of $\frac{1}{\sqrt{N}}$ if the weight is 1 or 0 if the weight is 0. Note: N is the number of 1s in the circuit. This is due to the fact that the weight vectors encoded in a qubit are normalized and thus will produce a uniform superposition of weights when only values of 0 or 1 are encoded. If the test state is a fully uniform superposition of ones, meaning it only contains values of normalized ones, then a swap test with the binary solution state will produce the following relationship.

$$N + \Delta = |\langle x|test\rangle|^2 W$$

In the previous equation, Δ represents the difference in the number of ones between the test vector and the solution vector and W represents the total number of weights encoded in the vector. Using this equation it is possible to solve for the number of ones present in the solution vector.

3.2 Experimentation

3.2.1 2×2 Matrix Solver Circuit

The experiment was first conducted with a quantum circuit capable of solving specific cases of a 2×2 linear systems of equations. Given matrices A and b it is possible to solve for x as $Ax = b$. Assuming one is given the following matrices:

$$A = \frac{1}{2}\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}; \vec{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Quick calculations show that $\vec{x} = 2 \times [3/8 \ -1/8]^T$. The 2 is multiplied by the solution vector to get the actual solution vector as the value $\frac{1}{2}$ is a constant that can be ignored in calculating $\vec{x}$. (The solution to the circuit is not necessary to be known beforehand but is useful for testing the accuracy of the circuit design.) As discussed previously in 3.1.1.1, the input vector $\vec{b}$ is normalized so that $|b_1|^2 + |b_2|^2 = 1$. The normalization is necessary to encode the values of $\vec{b}$ into the state $|b\rangle = b_1|0\rangle + b_2|1\rangle$ [12], [28]. In this specific case encoding the value of $\vec{b}$ is trivial as the state $|0\rangle$ already corresponds to the quantum state $1|0\rangle + 0|1\rangle$. However, for other other states that cannot be encoded so simply, the $|b\rangle$ state preparation subroutine detailed in 3.1.1.1 is used.The eigenvalues of A are $\lambda_1 = 1$ and $\lambda_2 = 2$ following our previous set constraints in 3.1.1.3. In addition, as mentioned previously, eigenvalues that are power of 2 will lead the phase estimation subroutine to generate states that exactly encode the eigenvalues [12]. The eigenvectors corresponding to λ_1 and λ_2 are $|u_1\rangle$ and $|u_2\rangle$ respectively. The two clock qubits x_2 and x_3 can exactly encode the eigenvalues as $|x_2x_3\rangle = |01\rangle$ and $|x_2x_3\rangle = |10\rangle$. Thus after execution of the phase estimation subroutine, the state of the system (excluding the ancilla qubit) is $|x_1x_2x_3\rangle = \beta_1|01\rangle|u_i\rangle + \beta_2|10\rangle|u_2\rangle$. The state amplitudes β_1 and β_2 correspond to the expansion of the $|b\rangle$ coefficients in the eigenbasis of matrix A.

The next portion of the circuit includes the eigenvalue subroutine. As outlined in 3.1.1.3, inverting the eigenvalue is a difficult process but because the chosen test circuit has eigenvalues of one and two the subroutine consists of simple swap test. The eigenvalue inversion brings the state of the system to $\sum_{j=1}^{2} \beta_j 2\lambda_j^{-1}|u_j\rangle$. The states encoded in $|x_2x_3\rangle$ then control a Y-rotation on the ancilla qubit. Since the inverted eigenvalues are encoded in $|x_2x_3\rangle$, the rotation is dependent on the state $2|\lambda_j^{-1}\rangle$. The rotation angle θ, is $\tilde{\theta_j} = 2^{1-r}\pi/\lambda_j = 2C/\lambda_j$. This angle will approximate $\theta_j = 2\arcsin(C/\lambda_j)$ [12]. The rotation variable r is discussed in 3.1.1.4. The final portion of the circuit applies the inverse of the the previously applied gates as detailed in 3.1.1.5. After the circuit finishes execution the state $|x\rangle$ will now be encoded in the qubit $|x_4\rangle$.

If the ancilla qubit (also known as the flag qubit) is measured to be one, then the rotations were performed correctly and the correct state is encoded in the fourth qubit. Reading back out the state of the qubit $|x_4\rangle$ from the quantum state to classical data is difficult and requires a run time of at least $O(\log N)$ [1]. However, qiskit allows for the statevector of the circuit to be accessed without *measurement* under ideal simulations. In this instance, *measurement* refers to process of measuring the value of a qubit, collapsing its state to either 0 or 1. The process of collapsing a qubit also collapses its superposition so that its value is now always 0 or always 1. But by using Qiskit's *Statevectorsimulator* one can index into the state of the system and look at its superposition values. In this particular case, the solution vector is encoded into the amplitude states corresponding to $|1000\rangle$ and $|1001\rangle$. Note: x_1 is the least significant bit and x_4 is the most significant bit, adhering to classic computer notation. Thus the statevectors of 1 and 9 must be indexed in order to reconstruct the solution vector. This is due to the fact the circuit correctly computes the state $|x\rangle$ when the flag qubit is measured to be 1 and the two clock register qubits are 0, giving us the two cases of $|1000\rangle$ and $|1001\rangle$ [2]. The normalized solution vector is

$$x = [0.147 \ \ -0.049].$$

As can be seen in above, the state amplitudes are not the expected values of the solution vector. This is due to the fact that the values have to be normalized in order to encode them into the qubit. By dividing by the norm of the vector and multiplying by the euclidean norm in order to undo the normalization applied by the circuit, the expected solution vector can be reconstructed [2].

$$x = [0.3747299 \ \ -0.1240997].$$

As shown previously, multiply the the calculated solution vector by 2 approximately gives the expected vector $\vec{x}$. While checking the solution to the system by accessing certain amplitudes in the state vector is not a practical solution (as it is run in ideal simulations and one can only measure a qubit by collapsing the system) it is useful for ensuring correctness of circuit design.

3.2.2 4×4 Matrix Solver Circuit

The previously described experiments were then tested again with a quantum circuit capable of solving 4×4 matrices with the same specific constraints. The circuit consists of 1 ancilla qubit, 4 clock register qubits, and 2 qubits to hold the solution to the system. The entire circuit uses 7 qubits as shown in 3.2. For a four variable systems of equations the following matrices were chosen.

$$A = \frac{1}{4}\begin{pmatrix} 15 & 9 & 5 & -3 \\ 9 & 15 & 3 & -5 \\ 5 & 3 & 15 & -9 \\ -3 & -5 & -9 & 15 \end{pmatrix}; \vec{b} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

This specific case was chosen as it adheres to the constraint that the eigenvalues of the circuit have the form $\lambda_i = 2^{i-1}$. The expected solution vector is $\vec{x} = \frac{1}{32}[-1\ 7\ 11\ 13]^T$. Once again in this particular case the state $|b\rangle$ preparation is trivial as it only requires placing a Hadamard bit on qubits x_5 and x_6 to encode the state into the b register. For non-trivial vectors encoding is done using a series of controlled Y-rotation gates are applied on x_5 and x_6 with angles calculated using the subroutine detailed in 3.1.1.1.

The eigenvalues of matrix A are $\lambda_1 = 1$, $\lambda_2 = 2$, $\lambda_3 = 4$, and $\lambda_4 = 8$. These eigenvalues can be exactly encoded using the four qubits x_1, x_2, x_3, and x_4. The eigenvectors corresponding to the previous eigenvalues can be represented by $|u_1\rangle$, $|u_2\rangle$, $|u_3\rangle$, and $|u_4\rangle$. After the execution of the phase estimation subroutine, the state of the system without the ancilla qubit is $|x_1x_2x_3x_4x_5x_6\rangle = \beta_1|0001\rangle + \beta_2|0010\rangle + \beta_3|0100\rangle + \beta_4|1000\rangle$. Since $|b\rangle = \sum_{j=1}^{4} \beta_j|u_j\rangle$, the state amplitudes are the expansion of the $|b\rangle$ coefficients. After the inversion subroutine, the state of the systems is $\sum_{j=1}^{2} \beta_j 2\lambda_j^{-1}|u_j\rangle$. Next the Y-rotation subroutine is implemented on the ancilla qubit according to the state encoded in $|x_1x_2x_3x_4\rangle$. Similar to the circuit for the 2×2 system, the rotation is dependent on the state $2\lambda_j^{-1}\rangle$ with the rotation angle being $\tilde{\theta_j} = 2C/\lambda_j$ as outlined in 3.1.1.4. Lastly, the process is undone through application of the inverse of all gates applied before the Y-rotation subroutine according to 3.1.1.5.

The process for verifying whether the circuit performed as expected is is similar to the one used for the 2×2 circuit. The state of the system after circuit execution was accessed using Qiskit's *Statevector simulator*, except that this time four states were accessed. The four states corresponding to the conditions that the flag qubit is 1 while the clock register qubits are 0 represent the circuit solution vector. The normalized results are

$$x = [-0.01\ 0.084\ 0.132\ 0.157]$$

After undoing the normalization process by diving by the norm of the vector and multiplying by the euclidean norm of $\sqrt{340}$ the calculated solution vector of the solution is shown in

$$x = [-0.82929169\ 6.96605024\ 10.94665037\ 13.01987961].$$

The calculated solution vector by the constant $\frac{1}{32}$ matches the expected result.

3.2.3 Full Circuit with Swap Test

Both the 2×2 and 4×4 circuit designs were then tested with the swap test detailed in 3.1.3.

The circuits detailed in Figure 3.1 and Figure 3.2 were expanded to include a swap test at the end of the circuit as shown in Figure 3.4. The fidelity of the solution state $|x\rangle$ and the test state $|test\rangle$ was tested through controlled swaps in the swap test subroutine as detailed in 2.2.7.

As mentioned in 3.1.3 it is possible to estimate the number of ones in the solution vector by swapping the solution state with a test state of all ones. Since the probability of measuring the ancilla qubit as zero is

$$P(0) = \frac{1}{2} + \frac{1}{2}|\langle x|test\rangle|^2$$

it is possible to find $|\langle x|test\rangle|^2$ by setting the equation to

$$|\langle x|test\rangle|^2 = 2(P(0) - \frac{1}{2})$$

Then since the number of ones in the solution vector can be found by

$$N + \Delta = |\langle x|test\rangle|^2 W$$

where $N + \Delta$ is the total number of ones in the solution vector, the probability of $P(0)$ can be used to find the number of ones in the solution vector. In the previous equation, N is the number of ones in the test vector and Δ is the difference between the number of ones in the test and the solution vector.

The 2×2 circuit was then tested under three cases where the expected solution vector was [0, 1], [1, 0], and [1, 1] with the swap vector being [1, 1]. The probability $P(0)$ was then constructed by taking the number of times the circuit measured *01* divided by number of times the circuit measured *01* and *11*. *01* is the number of times the circuit measured success as the 0 corresponds to success in the swap test while the 1 measures success in the circuit as indicated by the ancilla qubit. *11* measures the amount of times the ancilla qubit indicated success but the swap test indicated the states are not similar.

Each test cases was run 100 times with the circuit being executed for 1024 shots eact time and the average result across the 100 times was recorded and shown in the table below.

Vector	Value	**Error**
0 1	1.04	0.18
1 0	0.95	0.21
1 1	2.0	0.0

Table 3.2: Calculated number of ones in solution vector for different solution vectors with $Dim = 2$

The error was calculated by mean squared error. The test was then repeated for the 4×4 circuit design. However, since there are more permutations of a 4 dimension binary vector the Table 3.3 averages the calculated number of ones for all test cases with the same number of ones. For example, there are four test case vectors with only one one ([1 0 0 0], [0 1 0 0], [0 0 1 0], [0 0 0 1]). So those four test cases were further averaged by combining the results for those test cases and dividing by four. Those results are displayed below.

Expected Number of Ones	Value	**Error**
1	0.95	0.46
2	2.02	0.616
3	2.97	0.416
4	3.99	9e-4

Table 3.3: Calculated number of ones in solution vector for different solution vectors with $Dim = 4$

From both Tables 3.2 and 3.3 it is shown that it is possible to calculate the number of ones in a binary weight vector using a swap test.

Chapter 4

Conclusion

4.1 Future Work

Further areas of interest for quantum circuits for solving linear systems could include a tolerance correction algorithm. While currently quantum circuit are noisy, it may be of interest to see if one could use a classical algorithm to correct the results of the circuit. In addition, could it be possible to solve for a linear systems of equations that have eigenvalues close to the constrained eigenvalue powers of 2 within an acceptable error level.

In addition, it could be worth exploring the mathematical predictions for finding the state of a solution vector by using a swap test to calculate the distance between the two states and narrowing down the possibilities of states the solution vector could be. Also of particular interest is the overall time complexity analysis. While this work makes some comparisons between classical and quantum implementations, it is not explored in-depth as it is not feasible to beat classical implementations with current quantum circuits.

4.2 Conclusion

While the circuit results were shown with any one specific test case, they can be implemented for any 2×2 or 4×4 matrix that follows the specific eigenvalue constraint.

The only values that get encoded in the circuit design outlined in Figure 3.1 and Figure 3.2 are the eigenvalues and state $|b\rangle$. Thus to solve for any other matrix that follows the eigenvalue constraint, one would just need to change the input $\vec{b}$. One thing to note is that while the circuits are capable of solving for any matrix adherring to the specific constraints, the parameters for the hamiltonian simulation of the 4×4 circuit were fine-tuned for the particular case in [14] and may have larger errors for some cases.

In terms of solving for linear regression, the eigenvalue constraint of powers of 2 is applied to the matrix A to solve the system $Ax = b$. Dutta et. al introduces a data set X that is positive definite and has the unique solution $(X^T X)^{-1} X^T y$ [14]. Thus the matrix A would be equal to $X^T X$ and $y = X^T y$. The 4×4 circuit shown in Figure 3.2, 3.3 is capable of solving the matrix A since it follows the specific eigenvalue constraint. Thus, while current quantum technology constraints do not allow for any Hermitian matrix ciruit solver to be implemented, as detailed in [1], there is potential for using a quantum circuit to solve for linear regresion.

In addition, for the specific case of a binary neural network it is shown that it is possible to estimate the number of predicted ones in a solution vector by comparing the solution state to a test state as shown in 3.2.3. By using a swap test to implement the number of ones, information about the solution vector can be read out without performing quantum tomography. As quantum tomography is a costly procedure that require measurements in all three bases, using a swap test to read information about the solution state reduces the number of measurements needed overall for the circuit. Since a binary neural network has a finite number of possible weights, knowing how many ones are within a solution vector would allow reduce the search space for finding the weight vector. This method would take advantage of the quantum algorithm speed-up without adding on the additional minimum time complexity $O(N)$ required to read out the solution state from a qubit and translate it into classical date.

The circuit proposed in 3.1 is not capable of outperforming classical computers as it has the constraint of specific eigenvalues and preparing state $|b\rangle$ takes $O(N)$. However, once quantum circuits reach the size of 50 qubits they will outperform classical computers, as 50 qubits is beyond what the most powerful current digital supercomputers can simulate using brute force [7]. At 50 qubits the circuit will be able to encode and process 2^{40} or 1.09×10^{12} data points given 9 clock register qubits with eigenvalue precision of 2^9 and 1 ancilla qubit. Thus, it is the hope that as larger quantum circuits become more viable and more efficient state preparation algorithms are discovered it is possible to take full advantage of the $poly(\log N)$ speedup outlined in the HHL algorithm [1].

Bibliography

[1] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), Oct 2009.

[2] The Qiskit Team and The Qiskit Team. Single qubit gates. Data 100 at UC Berkeley, Mar 2021.

[3] Michael A. Poole and Patrick N. O'Farrell. The assumptions of the linear regression model. *Transactions of the Institute of British Geographers*, (52):145–158, 1971.

[4] Sorin Draghici. On the capabilities of neural networks using limited precision weights. *Neural Networks*, 15(3):395–414, 2002.

[5] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding, 2017.

[6] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018.

[7] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[8] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum (nisq) algorithms. 2021.

[9] D. V. Fastovets, Yu. I. Bogdanov, B. I. Bantysh, and V. F. Lukichev. Machine learning methods in quantum computing theory. 2019.

[10] Xavier Gitiaux, Ian Morris, Maria Emelianenko, and Mingzhen Tian. Swap test for an arbitrary number of quantum states, 2021.

[11] Babbush R. Arute F. Arya K. and et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, oct 2019.

[12] Yudong Cao, Anmer Daskin, Steven Frankel, and Sabre Kais. Quantum circuit design for solving linear systems of equations. 110(15-16):1675–1680, aug 2012.

[13] Zhikuan Zhao, Alejandro Pozas-Kerstjens, Patrick Rebentrost, and Peter Wittek. Bayesian deep learning on a quantum computer. *Quantum Machine Intelligence*, 1(1-2):41–51, May 2019.

[14] Sanchayan Dutta, Adrien Suau, Sagnik Dutta, Suvadeep Roy, Bikash K. Behera, and Prasanta K. Panigrahi. Quantum circuit design methodology for multiple linear regression. *IET Quantum Communication*, 1:55–61(6), December 2020.

[15] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. The effect of data encoding on the expressive power of variational quantum machine learning models. 2021.

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. 2016.

[17] Israel F. Araujo, Daniel K. Park, Francesco Petruccione, and Adenilton J. da Silva. A divide-and-conquer algorithm for quantum state preparation. *Scientific Reports*, 11(1), mar 2021.

[18] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. 299:802–803, 1982.

[19] Richard J. Hughes, Daniel F. V. James, Emanuel H. Knill, Raymond Laflamme, and Albert G. Petschek. Decoherence bounds on quantum computation with trapped ions. *Physical Review Letters*, 77(15):3240–3243, oct 1996.

[20] Ville Bergholm, Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. Quantum circuits with uniformly controlled one-qubit gates. 71, 2004.

[21] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. *Quantum Inf. Comput.*, 5:467–473, 2005.

[22] Andrew M. Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information & Computation*, 12:901–924, 2012.

[23] Anmer Daskin and Sabre Kais. Group leaders optimization algorithm. *Molecular Physics*, 109(5):761–772, Mar 2011.

[24] Anmer Daskin and Sabre Kais. Decomposition of unitary matrices for finding quantum circuits: Application to molecular hamiltonians. 134:144112, 2011.

[25] Danial Dervovic, Mark Herbster, Peter Mountney, Simone Severini, Naïri Usher, and Leonard Wossnig. Quantum linear systems algorithms: a primer. 02 2018.

[26] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2), Aug 2016.

[27] Guoming Wang. Quantum algorithm for linear regression. 96, 2017.

[28] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Physical Review A*, 83(3):032302, mar 2011.