5-2023

Quantum Acceleration of Linear Regression for Artificial Neural Networks

Martin A. Hoffnagle
mah5414@rit.edu

Quantum Acceleration of Linear Regression for Artificial Neural Networks

Martin A. Hoffnagle

Quantum Acceleration of Linear Regression for Artificial Neural Networks

Martin A. Hoffnagle

May 2023

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 Acceleration of Linear Regression for Artificial Neural Networks

Martin A. Hoffnagle

Committee Approval:

Sonia López Alarcón *Advisor* Date
Department of Computer Engineering

Cory Merkel Date
Department of Computer Engineering

Nathan Cahill Date
School of Mathematical Sciences

Acknowledgments

Thank you to Dr. Sonia López Alarcón, Dr. Cory Merkel, and Dr. Nathan Cahill for serving on my research committee and supporting my research. I would also like to thank Alejandro Pozas-Kerstjens for his help during my research.

Thank you to Research Computing at the Rochester Institute of Technology for providing computational resources that have contributed to the research results reported in this work [1].

Thank you to my friends and family that continue to support and motivate me through my life.

Abstract

Through proofs and small scale implementations, quantum computing has shown potential to provide significant speedups in certain applications such as searches and matrix calculations. Recent library developments have introduced the concept of hybrid quantum-classical compute models where quantum processor units could be used as additional hardware accelerators by classical computers. While these developments have opened the prospect of applying quantum computing to machine learning tasks, there are still many limitations of near and midterm quantum computing. If implemented carefully, the advantages of quantum algorithms could be used to accelerate current machine learning models. In this work, a hybrid quantum-classical model is designed to solve a gradient descent problem. The quantum HHL algorithm is used to solve a system of linear equations. The quantum swap test circuit is then used to extract the Euclidean distance between a test point and the quantum solution. The Euclidean distance is then passed to a classical gradient descent algorithm to reduce the number of iterations required by the gradient descent algorithm to converge on a solution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Advancements in machine learning and the more powerful subsection of deep learning models has allowed for more in depth analysis of data than ever before. Deep learning models allow for computers to classify and detect features in data sets that humans cannot. Currently, specialized devices such as GPUs (graphics processing units) and TPUs (tensor processing units) are used to accelerate machine learning models by computing matrix functions using highly parallel processing techniques. However, while computer hardware continues to advance, deep learning models have become extremely computationally intensive, often taking days to train advanced models.

The relatively new field of quantum computing has shown potential to accelerate searching problems, integer factorization, and quantum simulation. This is possible by applying the quantum properties of superposition and entanglement which allow qubits to affect other qubits. This allows for algorithms not possible using classical computers. One such algorithm, the HHL algorithm, developed by Harrow, Hassidim, and Lloyd aims to solve a system of linear equations [5]. Systems of linear equations are common in machine learning algorithms, especially those that include backpropagation.

Currently there are still many limitations on the amount of noise, accuracy, and

computational size of quantum circuits. However, by composing a hybrid system of both classical computers and quantum computers, quantum algorithms could be used as an accelerator for machine learning models while classical computers could ease the limitations of quantum computers.

1.2 Objective

The use of quantum mechanics allows quantum computers to apply unique algorithms not possible on classical computers. Great interest in quantum computing began when Peter Shor proved that the use of quantum mechanics could be used to factor integers in 1994. Since 1994, quantum factoring and search algorithms have been mathematically proven to have better time complexities than their classical counterparts [6]. However, designing quantum algorithms with better mathematical time complexity is not enough to declare quantum supremacy as quantum hardware must be able to implement the algorithms.

Quantum computers are not able to take full advantage of these quantum algorithms as creating and maintaining a quantum environment is extremely difficult. IBM's state-of-the-art quantum computer has 433 qubits and has further limitations on how many gates can be applied before noise renders the qubits unusable [7]. Limitations of hardware size and high error rates have created the term Noisy Intermediate-scale Quantum (NISQ) computers. Currently there is a great interest in utilizing NISQ computers to their greatest extent. One common idea is to utilize quantum computers as an additional computational unit or accelerator in a larger system, similar to how graphics cards have been incorporated in classical computing in addition to CPUs. Proposed hybrid quantum-classical systems would run computations with classical computers and delegate computations that have an advantageous quantum algorithm to a quantum processor.

By researching quantum-classical hybrid systems, advances could be made to both

deep learning and quantum computing. Quantum computers are not currently capable of computing an entire neural network, and may never be optimized for such general tasks. Instead, this quantum computing research will focus on accelerating a classical neural network by gaining insight to the solution using a quantum processor. Hybrid training of a neural network aims to use quantum algorithms to aid in optimization of the neural network's weights.

Solving for a neural network's ideal weights includes solving a system of linear equations. This can be done using a quantum circuit, but the full solution cannot be extracted. Instead, other information can be extracted from the solution embedded in superposition. This extracted information can aid the linear regression performed in the classical portion of training a neural network. To further the goal of hybrid quantum-classical machine learning models, this research aims to achieve the following. First, the quantum HHL circuit will be implemented, and the capabilities of the HHL algorithm will be tested. Next, a quantum circuit to extract information from the HHL solution state will be designed utilizing a SWAP test circuit. The extracted information will then be used to aid a classical linear regression method.

Chapter 2

Background

2.1 Single Layer Neural Networks

Machine learning aims to build models that can predict an outcome based on a new input. These predictions are made based on samples or training data with known outputs. In essence, machine learning models aim to find a relationship between inputs and outputs that can be used to predict the result of a new input. The process of finding a relationship between the training data and its outputs to predict continuous outcomes is called regression.

Linear regression is a form of regression where the relationship between the inputs and outputs is linear. In the case of a single layer neural network, the training of parameters can be modeled as a linear regression problem.

2.1.1 Gradient Descent

Gradient descent is a popular optimization algorithm used in machine learning. In machine learning, gradient descent is used to train neural networks. It uses an iterative approach to solve for ideal parameters that minimize error. In general, gradient descent utilizes a cost or loss function to determine the error between an estimation for the solution and the true solution. The model for making estimations is then updated using a learning rate that determines how much to update the model [8].

Stochastic gradient descent is a modification of regular gradient descent. Unlike gradient descent, which updates it's parameters by calculating an error based on the entire data set, stochastic gradient descent updates the parameters after calculating an error of a single randomly selected data point or random subset of data points.

One important hyperparameter, a variable to modify the response of a machine learning algorithm, is the learning rate. The learning rate in gradient descent determines how much the estimation is updated in each iteration. This affects how fast the descent converges but a learning rate that is too large can miss the solution. Adaptive learning rates aim to achieve the fastest learning by changing the learning rate as the gradient descent iterates [8].

2.2 Quantum Computing

A quantum computer is a quantum system that applies a controlled quantum mechanical evolution from an initial quantum state to a final quantum state. This controlled evolution is used to solve a computational problem.

The quantum property of superposition is core to quantum computing. Superposition means that unlike a classical computer where a bit is either 0 or 1, a quantum bit or qubit can have many states. The state of a qubit is represented using the Bloch sphere as seen in Figure 2.1.

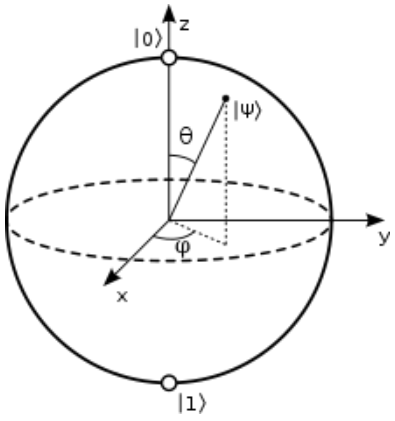


Figure 2.1: Bloch Sphere

The Bloch sphere is used to represent the spin of a qubit. The north and south poles of the Bloch sphere represent $|0\rangle$ and $|1\rangle$ which are similar to the classical bit representations of 0 and 1. However, the other states on the Bloch sphere represent a qubit in superposition: a combination of the states $|0\rangle$ and $|1\rangle$ where each can have a complex number as its coefficient [3].

Each axis of the Bloch sphere represents a different basis. The two orthogonal z-axis states, $|0\rangle$ and $|1\rangle$, are referred to as the computational basis. By default qubits are assumed to be initialized to $|0\rangle$ and measured in the computational basis. Therefore, the mathematical representations of algorithms are usually written using

the computational basis. The x-axis is the Hadamard basis:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.1}$$

The y-axis is referred to as $|R\rangle$ and $|L\rangle$ for Right and Left (other times referred to as $|i\rangle$ and $|-i\rangle$) where

$$|R\rangle = |i\rangle = \frac{|0\rangle + i|1\rangle}{\sqrt{2}} \qquad |L\rangle = |-i\rangle = \frac{|0\rangle - i|1\rangle}{\sqrt{2}} \tag{2.2}$$

Qubits can also be represented as column vectors which is useful to interact with matrix operators [3]. The vector notation of basis states is presented in Table 2.1.

$\|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$\|+\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\|-\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
$\|R\rangle = \|i\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ i \end{bmatrix}$	$\|L\rangle = \|-i\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -i \end{bmatrix}$

Table 2.1: Column vector representation of basis states

Another quantum property key to the power of quantum computing is quantum entanglement. Quantum entanglement means qubits can interact with one another and, when entangled, acting on one qubit can affect the other qubit.

2.2.1 Quantum Gates and Circuits

Quantum gates act on qubits and can apply the principles of entanglement and superposition. They are a building block for quantum circuits, like classical logic gates are for classical circuits. Just as classical logic gates can be combined to create more complex circuits, quantum gates can be combined to create more complex quantum circuits. Quantum gates are represented by a matrix that describes its operation. These matrices algebraically describe the effect of the quantum gate on the quantum

state of the qubits it operates on. The Hadamard gate is an extremely important single qubit gate that puts a qubit in uniform superposition. The Hadamard gate is represented by the matrix:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.3}$$

The effect of quantum gates can be manually calculated using linear algebra. For example, the mathematical equation of applying a Hadamard gate to a qubit and multiple representations of the result are described as:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.4}$$

Pauli gates are common single qubit gates that act on a single qubit's phase. There are three Pauli gates that change a qubit's phase by rotating around the Bloch sphere in either the x, y or z axis.

Quantum gates are reversible, unlike classical logic gates, which are irreversible. This means that, unlike classical logic gates, a quantum gate can be "undone" by applying the inverse of the gate. This is because quantum gates can be represented by a matrix that is unitary. A unitary matrix can be defined as one where $UU^\dagger = U^\dagger U = I$ [3]. Reversible gates are powerful in quantum computing because they allow operations and algorithms to be reversed.

While qubits cannot be copied, due to the no-cloning theorem, qubits can be moved. A swap gate will swap the position of two qubits in a circuit. More complex circuits can be created using multi-qubit gates. Controlled gates apply a gate on a qubit with a second qubit controlling the operation.

A quantum circuits connect a series of qubits, quantum gates, and measurements to perform a larger task. An example of a quantum circuit with single qubit gates, two qubit gates, and measurement can be seen in Figure 2.2. A group of qubits, often

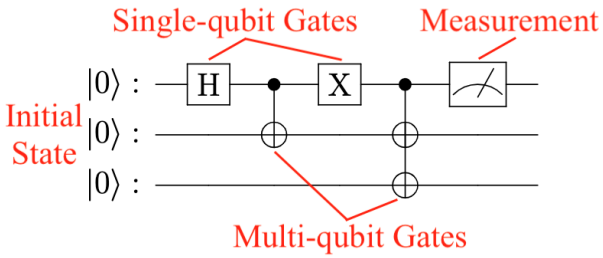


Figure 2.2: Quantum Circuit Example

being used to represent a single data vector, can be grouped together as a quantum register.

Because quantum systems are closed systems, once a qubit is measured, its state collapses. Measurement is a destructive task and thus usually occurs at the end of a circuit to measure a final result. A qubit in superposition can collapse to multiple values under measurement. Therefore, quantum circuits are measured many times to create a probability distribution representing the solution. The number of times a circuit is run and measured to create the probability distribution is referred to as the number of shots. The solution of a quantum circuit generally contains either one or multiple solutions. When a single or few solutions are represented, the largest state in the measurement probability distribution represents the solution. For example, in Grover's algorithm for searching a list, the element being searched for will have the largest amplitude in the resulting probability distribution. Alternatively, algorithms with many solutions can encode a solution in every state of the probability distribution. The HHL algorithm encodes a solution vector where each amplitude in the resulting probability distribution represents an element of the solution vector.

2.2.2 Encoding Data

An important aspect of quantum computing and quantum algorithms is the ability to encode data into qubits. Quantum encoding is the process of embedding classical information into quantum properties and is vital to applying quantum algorithms.

There are several ways to encode information into qubits. A simple quantum encoding method is basis encoding. Basis encoding performs a one-to-one translation of a binary string to a computational basis state [9]. An n-bit string of binary values will be encoded into an n-qubit state:

$$|x\rangle = |i_x\rangle \tag{2.5}$$

where $|i_x\rangle$ is a computational basis state. For example, a binary string $x = 1101$ will be encoded as $|1101\rangle$.

Amplitude encoding is a commonly used method often used to encode vectors or algorithm solutions. In amplitude encoding each value of an N length vector x is encoded into the amplitude of a qubit:

$$|x\rangle = \sum_{i}^{N} x_i|i\rangle \tag{2.6}$$

where $|i\rangle$ is the i-th computational basis state. Amplitude encoding is more powerful than basis encoding because an N length vector can be encoded into n qubits where $n = \log_2(N)$ [10]. Additionally, the encoded values are not restricted to binary or even integer values. However, because the amplitudes of a quantum state must be equal to one, the encoded vector x must be normalized so that $|x|^2 = 1$.

Another method of quantum encoding is phase encoding. A value x is encoded by rotating a qubit's phase such that:

$$|x\rangle = \bigotimes_{i=1}^{n} cos(x_i)|0\rangle + sin(x_i)|1\rangle \tag{2.7}$$

Similar to basis encoding, phase encoding maps each value to one qubit however phase encoding is not limited to binary values. Similar to amplitude encoding, the values must be normalized in [0, 2π] because the encoding is applied to a qubit's quantum

properties. However, preparing a qubit using phase encoding is very efficient as single qubit rotations are easy to implement in quantum circuits [10].

2.2.3 Quantum Fourier Transform

The quantum Fourier transform (QFT) is a quantum algorithm that aims to compute the classical discrete Fourier transform algorithm but on qubits. It is of significance because it is a building block for many other algorithms including quantum phase estimation. The classical discrete Fourier transform uses the formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{i2\pi jk}{N}} \tag{2.8}$$

to map a vector, x, of N complex numbers to another vector, y, of N complex numbers. Similarly, the quantum Fourier transform maps $|x\rangle$ to another quantum state such that:

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{i2\pi jk}{N}} |k\rangle \tag{2.9}$$

The QFT can also be described using a product representation [11] as:

$$\begin{aligned} QFT_N|x\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\right) \otimes \dots \\ \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right) \end{aligned} \tag{2.10}$$

Similar to how the classical Fourier transform is a conversion from time domain to frequency domain, the quantum Fourier transform can be thought of as a conversion from the computational basis to a Fourier basis. This is similar to how the Hadamard gate converts from the computation basis to the Hadamard basis.

A n-qubit implementation of the QFT is shown in Figure 2.3.

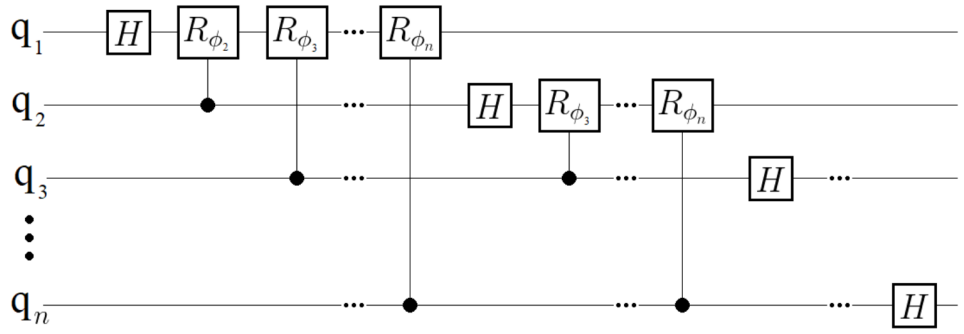


Figure 2.3: n-qubit QFT Circuit [2]

The QFT circuit begins by applying a Hadamard gate on the first qubit:

$$H|x_1x_2...x_n\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + exp\left(\frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3...x_n\rangle \tag{2.11}$$

Next, a two qubit controlled rotation gate $CROT_k$ is applied to the first qubit with the second qubit being the control. $CROT_k$ can be represented as $CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}$ where $UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}$ [11]. After applying $CROT_k$, the circuit state is:

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle \tag{2.12}$$

A further n-1 $CROT_k$ gates are applied to the first qubit controlled by a consecutive qubit. The resulting state can be represented as:

$$\begin{gathered}\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle \\ \text{where } x = 2^{n-1}x_1 + 2^{n-2}x_2 + \ldots + 2^1x_{n-1} + 2^0x_n\end{gathered} \tag{2.13}$$

This application of the Hadamard gate followed by $CROT_k$ gates controlled by the qubits below is repeated for the remaining qubits. The final circuit state is the same as seen in Equation 2.10.

As mentioned in Section 2.2.1, quantum circuits can be reversed. This means to convert from the Fourier basis to the computational basis one can simply apply a

reversed version of the quantum Fourier transform circuit. This is referred to as the inverse quantum Fourier transform.

2.2.4 Quantum Phase Estimation

One key quantum algorithm is the quantum phase estimation algorithm (QPE). Quantum phase estimation aims to estimate the phase or eigenvalue of a eigenvector. More specifically, QPE aims to estimate θ in:

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle \tag{2.14}$$

where U is a unitary operator, $|\psi\rangle$ is an eigenvector and $e^{2\pi i\theta}$ is its eigenvalue. QPE is an important quantum algorithm as it is commonly used in other algorithms such as Shor's algorithm and HHL as covered in Section 2.3.

The circuit implementing the quantum phase estimation algorithm is shown in Figure 2.4.

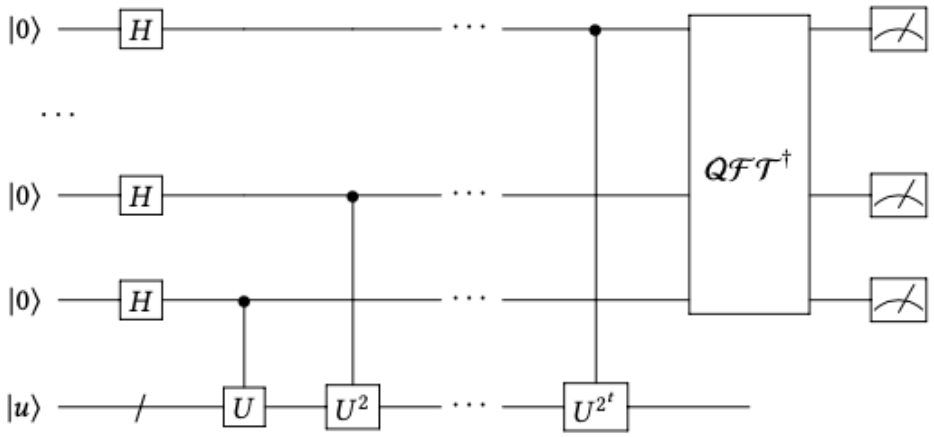


Figure 2.4: Generic Quantum Phase Estimation Circuit [3]

First the eigenvector $|\psi\rangle$ is encoded into a quantum register. A second quantum register, the counting register, of n-qubits is setup to store θ initializing the circuit to

the state:

$$|0\rangle^{\otimes n}|\psi\rangle \tag{2.15}$$

Next, Hadamard gates are applied to the counting register:

$$\frac{1}{2^{\frac{n}{2}}}(|0\rangle + |1\rangle)^{\otimes n}|\psi\rangle \tag{2.16}$$

The unitary from Equation 2.14 must then be applied as a controlled unitary gate, CU. This will apply U to the target qubit if the control bit is $|1\rangle$. This can be done by applying CU^{2^j} where $U^{2^j}|\psi\rangle = e^{2\pi i 2^j \theta}|\psi\rangle$ and $0 \leq j \leq n-1$. The controlled unitary gates are applied n-times to the eigenvector where each qubit in the counting register is used as a control bit [11]. After applying the controlled unitary gates, the circuit is in the state:

$$\begin{aligned} &\frac{1}{2^{\frac{n}{2}}}\left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^{n-1}}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^1}|1\rangle\right) \otimes \left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^0}|1\rangle\right) \otimes |\psi\rangle \\ &= \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{\mathbf{2\pi i}\theta k}|k\rangle \otimes |\psi\rangle \end{aligned} \tag{2.17}$$

Now an inverse quantum Fourier transform can be applied to the counting register. This is done because the circuit state above is in the Fourier basis. Applying the inverse quantum Fourier transform results in:

$$\frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{k=0}^{2^n-1} e^{-\frac{2\pi i k}{2^n}(x - 2^n\theta)}|x\rangle \otimes |\psi\rangle \tag{2.18}$$

The inverse quantum Fourier transform converts from the Fourier basis into the computational basis and leaves the counting register with a n-bit estimation of $|\theta\rangle$. This register can then be further used or measured.

2.3 HHL Algorithm

One of the most important algorithms for quantum machine learning is HHL, named after the authors of the algorithm Harrow, Hassidim, and Lloyd. The aim of HHL is to solve a system of linear equations, a common calculation across machine learning. HHL presents a quantum algorithm that can solve a system of linear equations with a runtime of $O(log(N)s^2\kappa^2/\epsilon)$ [5] when the input matrix A is s-sparse and Hermitian. Classically, a system of linear equations can be directly solved using Gaussian elimination with a time complexity of $O(N^3)$, a much worse time complexity than the quantum algorithm. Classical iterative methods aim to solve systems of linear equations much quicker by converging an initial condition. The conjugate gradient method, with the same restrictions as the quantum algorithm, a s-sparse and Hermitian matrix, can be solved with a time complexity of $O(Ns\kappa log(1/\epsilon))$. The extreme reduction in complexity over direct methods, roughly a factor of N^3 versus $log(N)$, and still large reduction over iterative methods, roughly N versus $log(N)$, has drawn great interest to the HHL algorithm and quantum computing.

The steps of HHL can be broken down into four sections: matrix setup and vector encoding, quantum phase estimation, eigenvalue inversion, and inverse quantum phase estimation. The general circuit design for HHL with these steps is presented in Figure 2.5.

2.3.1 Setup and Encoding

Specifically, the HHL algorithm attempts to solve for $|x\rangle$ given an input matrix A and input vector $|b\rangle$ where $A|x\rangle = |b\rangle$. Because A is an Hermitian matrix, A can be represented by its eigenvalues and eigenvectors as $A = \sum_{j=0}^{N-1} \lambda_j |u_j\rangle\langle u_j|$ where λ_j are the eigenvalues of A and $|u_j\rangle$ are the eigenvectors of A [11]. The Hermitian properties of A also mean A has an orthogonal basis of eigenvectors. This allows $|b\rangle$ to

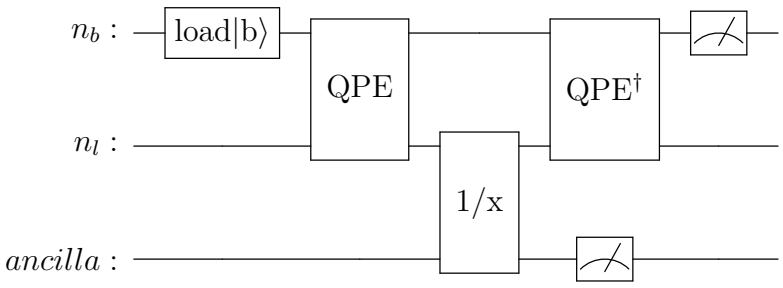


Figure 2.5: General HHL Circuit Design in Stages

be rewritten in the eigenbasis of A as $|b\rangle = \sum_{j=0}^{N-1} b_j|u_j\rangle$. Since A can be represented by its eigenvalues and eigenvectors, then $A^{-1} = \sum_{j=0}^{N-1} \lambda_j^{-1}|u_j\rangle\langle u_j|$. Therefore, the solution to the system of linear equations can also be written as:

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=0}^{N-1} \lambda_i^{-1} b_j |u_j\rangle \tag{2.19}$$

The first step of HHL is to ensure the input matrix A is a Hermitian matrix. If A is not Hermitian, the problem can be easily redefined using a Hermitian matrix where

$$\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \begin{bmatrix} 0 \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{2.20}$$

However, this will double the size of the matrix and thus increase the amount qubits needed.

Next, the input vector $|b\rangle_{n_b}$ must be encoded into a register of qubits. The subscript, n_b, indicates which register the input is in. This step is usually performed using amplitude encoding.

2.3.2 Quantum Phase Estimation

Next, quantum phase estimation is applied to $|b\rangle_{n_b}$. Recalling from Section 2.2.4, the quantum phase estimation algorithm estimates θ in $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$. If QPE were treated as a function, the inputs would be a unitary gate U and the state $|0\rangle|\psi\rangle$, an empty register and a register consisting of an eigenvector. The returned state would be $|\tilde{\theta}\rangle|\psi\rangle$ where $\tilde{\theta}$ is the estimated value of θ.

For HHL, the unitary when applying QPE is defined as:

$$U = e^{iAt} = \sum_{j=0}^{N-1} e^{i\lambda_j t}|u_j\rangle\langle u_j| \tag{2.21}$$

Therefore, the result of QPE with the eigenvector $|u_j\rangle$ and its eigenvalue $e^{i\lambda_j t}$ is $|\tilde{\lambda}_j\rangle|u_j\rangle$ where $|\tilde{\lambda}_j\rangle$ is the estimated value of $\frac{\lambda_j t}{2\pi}$. QPE can then be applied to $|b\rangle$ which in the eigenbasis of A with the added $|0\rangle_{n_l}$ register is $|b\rangle = \sum_{j=0}^{N-1} b_j|0\rangle_{n_l}|u_j\rangle_{n_b}$. The result of which is:

$$\sum_{j=0}^{N-1} b_j|\tilde{\lambda}_j\rangle_{n_l}|u_j\rangle_{n_b} \tag{2.22}$$

which now contains the eigenvalues in register n_l and the eigenvectors in register n_b [3]. The eigenvalues are encoded in register n_l using basis encoding as explained in Section 2.2.2.

2.3.3 Eigenvalue Inversion

The next step of HHL is to apply a rotation often called the eigenvalue inversion step. This step adds the inverted eigenvalues to an ancilla qubit. To do this an auxiliary qubit is added and is rotated conditioned on A's eigenvalues resulting in:

$$\sum_{j=0}^{N-1} b_j|\tilde{\lambda}_j\rangle_{n_l}|u_j\rangle_{n_b}\left(\sqrt{1-\frac{C^2}{\tilde{\lambda}_j^2}}|0\rangle + \frac{C}{\tilde{\lambda}_j}|1\rangle\right) \tag{2.23}$$

where C is a constant such that $|C| \leq \lambda_{min}$ [12]. The ancilla qubit is now measured. While the constant C must be less than the smallest eigenvalue, it should also be as large as possible. The larger the constant C is, the larger the probability the ancilla qubit will be measured as $|1\rangle$ which indicates the solution was calculated successfully [13].

2.3.4 Inverse Quantum Phase Estimation

Next, the inverse quantum phase estimation is applied to undo the eigenvalue computation. This removes the eigenvalue computed earlier leaving register n_l empty:

$$\sum_{j=0}^{N-1} b_j |0\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) \tag{2.24}$$

If the ancilla qubit measured in the previous step is $|1\rangle$ [5], then the solution remains as:

$$\sqrt{\frac{1}{\sum_{j=0}^{N-1} C^2 |b_j|^2 / |\lambda_j|^2}} \sum_{j=0}^{N-1} C \lambda_j^{-1} b_j |0\rangle_{n_l} |u_j\rangle_{n_b} \tag{2.25}$$

which matches the solution in Equation 2.19 but with a normalization factor. Additionally, the norm of the solution $|x\rangle$ is the probability of measuring $|1\rangle$ in the previous step:

$$P(|1\rangle) = ||x||^2 \tag{2.26}$$

2.3.5 Limitations

While the HHL algorithm has potential to show a quantum speedup, HHL has several limitations preventing its real world use. The first challenge of implementing HHL is that quantum computers are still very limited in their size and accuracy. Generic HHL implementations require too many qubits and quantum gates to run on current quantum computers. Reduced circuit versions of HHL have been implemented [14] but have significant disadvantages. The reduced circuits are custom made for a specific

size of input matrix and have limitations on the values in the input matrix. Another inherent limitation of HHL is the output state. The result of the HHL algorithm is a quantum state with the result encoded as amplitudes. Measuring each value would require N measurements, thus breaking the advantage of HHL. This means to effectively use HHL, the result cannot be used directly but rather as a means to measure a specific observable value or trait [15]. The result can also be used as verification step using the swap test to compare to a known value.

2.4 SWAP Test

The SWAP gate is a two qubit gate that swaps the states of two qubits. The controlled swap (CSWAP) gate, or Fredkin gate, adds a control qubit to enable the swap gate. If the control qubit is in the $|0\rangle$ state, then the two target qubits are not affected. However, if the control qubit is in the $|1\rangle$ state, then the swap gate is enabled and the two target qubits swap states. The SWAP test circuit, Figure 2.6, utilizes a SWAP gate and Hadamard gates to determine the distance between two quantum states.

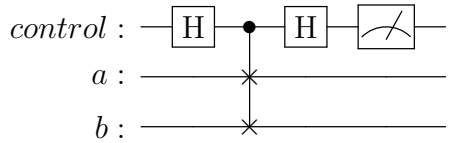


Figure 2.6: SWAP Test Circuit

In the SWAP test, the control qubit is put in superposition using a Hadamard gate before applying the controlled swap gate. This puts the control qubit in a uniform superposition and the overall circuit in a state:

$$|+ab\rangle = \frac{|0ab\rangle + |1ab\rangle}{\sqrt{2}} \tag{2.27}$$

Next, the CSWAP gate is applied. Because the control qubit is in a uniform superposition, the SWAP half occurs swapping $|a\rangle$ and $|b\rangle$ in only the $|1\rangle$ half of the superposition:

$$\frac{|0ab\rangle + |1ba\rangle}{\sqrt{2}} \tag{2.28}$$

The second Hadamard gate is then applied to the control qubit leaving the circuit state as:

$$\frac{1}{\sqrt{2}}\left(\frac{|0ab\rangle + |1ba\rangle}{\sqrt{2}} + \frac{|0ab\rangle - |1ba\rangle}{\sqrt{2}}\right) = \frac{1}{2}\left(|0\rangle(|ab\rangle + |ba\rangle) + |1\rangle(|ab\rangle - |ba\rangle)\right) \tag{2.29}$$

Finally, the control qubit can be measured. The resulting probability of measuring $|0\rangle$ is:

$$P(0) = \frac{1 + |\langle a|b\rangle|^2}{2} \tag{2.30}$$

This means if $|a\rangle = |b\rangle$ then the control qubit will have a 100% probability of measuring $|0\rangle$ [6]. The inner product of $|a\rangle$ and $|b\rangle$ can then be derived as:

$$|\langle a|b\rangle| = \sqrt{2P(0) - 1} \tag{2.31}$$

2.4.1 SWAP Test Application

Many quantum algorithms encode their result in a quantum state, such as in the HHL algorithm where the solution vector is encoded in a qubit's amplitudes. This can make extracting an exact answer difficult. This creates a need for quantum verification to ensure quantum algorithms are functioning properly. One method to verify a result encoded in a quantum state is to use the swap test. Here the swap test is used to determine if two qubit states are equivalent [16]. By applying the swap test, it can be determined if two qubits contain the same quantum state. To verify a quantum algorithm, a known solution is encoded into a qubit. The algorithm can then be performed, and a swap test can be applied to compare the result to the known solution. The swap test will result in a 100% probability of measuring $|0\rangle$ if the algorithm's result matches the known solution. This is a valuable technique as it can verify that quantum algorithms are correct when using regular measurement is not possible such as the HHL algorithm [14].

2.5 Related Works

2.5.1 HHL Read the Fine Print

Aaronson [15] identifies the limitations of the HHL algorithm. The paper explains that without quantum memory, utilizing HHL is not practical as loading in the values would be extremely slow. Even with quantum memory, the values would have to remain fairly uniform. However, the paper does acknowledge that the values could be relatively quickly prepared if they can be described by a formula. The author then points out that unitary transformations must be applied which grows the complexity linearly with the sparsity of the input matrix. This is mentioned to remind that applying the unitary transforms must not take exponential time or the speedup of HHL disappears. The next potential flaw of HHL explained is that the input matrix must be robustly invertible. Similar to the sparsity of the matrix, the ratio in magnitude of the input matrices largest and smallest eigenvalues must not grow exponentially for a speedup to remain.

Additionally, the paper describes HHL not as an algorithm to solve for a system of linear equations but rather an algorithm to gain insight into a system of equations. This description is given due to the limitation that reading out every value of the solution vector would require an additional n steps, breaking any advantage of HHL. However, Aaronson does not aim to dismiss the HHL algorithm. Instead, HHL algorithm is described as "a template for other quantum algorithms" [15] which could use HHL as a means to speedup an application as long as the limitations of HHL are addressed.

2.5.2 Bayesian Neural Networks

Zhao et al. [14] introduce a quantum-classical hybrid model that takes advantage of the HHL algorithm while attempting to avoiding some of its drawbacks. The paper

aims to improve upon Bayesian deep learning, a classically computationally difficult approach to deep learning, through the use of quantum matrix inversion. The hybrid model developed is designed to learn a Gaussian process while avoiding nonexistent quantum technologies. The model avoids the need for quantum memory in a multi-layer case during training by having multiple copies of the encoded covariance matrix.

Additionally, the feasibility of quantum matrix inversion is explored. A circuit is designed to implement the HHL algorithm to solve a 2 by 2 matrix. The circuit is then optimized to solve a specific 2 by 2 matrix. While this does add restrictions to the input matrix, the optimized circuit can be run on current quantum hardware while non-optimized circuits can only be simulated. The optimized circuit uses a swap test to verify the circuit returns the correct solution.

2.5.3 An Introduction to Quantum Machine Learning

Schuld et. al. [6] discuss several potential approaches to applying quantum computing to machine learning problems. One of the possible avenues discussed is quantum versions of the k-nearest neighbor algorithm. The k-nearest neighbor algorithm is a early classification method that classifies an object based on the most frequent class occupied by the k objects closest to the new object. This approach assumes that objects with features close to each other will be similar classes. The first step in a k-nearest neighbors algorithm is to determine the distances to the new object. This is often done using the Euclidean distance.

The authors propose that a quantum processor could be used to calculate the distances between objects. The swap test circuit is proposed as an efficient way of determining the distance between two objects. The use of quantum computing for k-nearest neighbors could prove to be advantageous because the swap test procedure is more efficient than the classical procedure that has a polynomial runtime.

2.5.4 Accelerating the training of single-layer binary neural networks

In a previous work [17], the idea of using the SWAP test to extract information from the HHL algorithm was explored. Since a system of linear equations can be used to represent single layer neural network, the HHL algorithm could be used to find the optimal weights. With the ideal weights in a quantum state, it is suggested that the SWAP test be performed against several fixed reference states. The information obtained can then be used to reduce the computational complexity of the classical search for the optimal set of weights.

To demonstrate how the test states could be used, experiments were performed on the MNIST data set. A single layer binary neural network was used to classify the digits in the MNIST data set into two classes. The binary neural network's weights were rounded using the reduced search space created from select test points. The results indicate that using a reduced search space during training can decrease the number of iterations required for convergence.

Chapter 3

Experiments and Design

3.1 Design

As explained in Section 2.3, HHL is a powerful and important quantum algorithm, however, it also has significant limitations. One limitation is that the algorithm leaves the solution in a quantum state such that the full solution cannot be extracted without ruining the potential speedup advantages. However, while this limits the algorithm's use cases, the resulting quantum state can still be operated on to make interpretations based on the result. Section 2.4 explained how a SWAP test circuit can be used to determine the inner product between two quantum states. The SWAP test can be applied to the HHL circuit to determine the inner product between the quantum solution state of a system of linear equations and a pre-selected test point. Because the quantum states are normalized when encoded, the Euclidean distance between the two state can be calculated from the inner product.

The previous work [17], gives a framework for how the Euclidean distance can be used to reduce the search space of a classical regression algorithm to improve a neural networks ability to find ideal weights. The work utilizes the Euclidean distance between the solution and well selected test points to restrict the search space to a smaller radius hyper-sphere of solutions. In this work, the Euclidean distance is implemented in a classical regression algorithm. A full circuit implementation is

designed and simulated to combine the HHL algorithm and SWAP test to determine the Euclidean distance between the solution to a system of linear equations and a selected test point. The Euclidean distance is then used to speedup a classical regression algorithm to improve a neural networks ability to find ideal weights. This is done by incorporating the quantum circuit between iterations of a classical gradient descent algorithm. As the classical gradient descent algorithm iterates towards a estimate of the true solution, the quantum circuit measures the Euclidean distance between the true solution and the gradient descent's estimated solution. The Euclidean distance between the true solution and current estimated solution is then used to adjust the gradient descent's learning rate. This quantum adaptive learning rate allows for a larger initial learning rate that is reduced as the estimated solution approaches the true solution. The quantum adaptive learning rate improves the convergence speed of the gradient descent algorithm.

3.1.1 Qiskit

Qiskit was used to develop and simulate quantum circuits. Qiskit is an open-source development kit that enables designing quantum circuits using Python [18]. Founded by IBM, Qiskit allows users to run their quantum circuits on both quantum simulators and state-of-the-art IBM quantum computers. The Python environment Qiskit executes in is extremely powerful because it inherently enables hybrid quantum-classical computing. Quantum circuits in Qiskit can be simulated or passed to a quantum processor which will return measurement results. The measurement results can then be analyzed or further processed classically in Python. This process can be repeated enabling algorithms that pass computations between classical and quantum processors multiple times.

3.1.2 Quantum Circuit Design

Before creating the HHL circuit, the input matrix A is verified to be Hermitian. If the matrix is not Hermitian, the matrix and input vector are modified such that:

$$\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \begin{bmatrix} 0 \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{3.1}$$

which ensures A is Hermitian.

Another approach to setting up the input matrix is to transform the system of linear equations to

$$A^T A|x\rangle = A^T|b\rangle \tag{3.2}$$

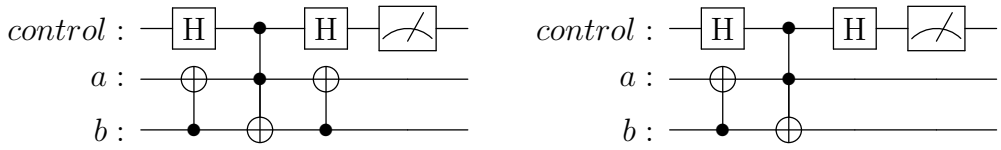
Modifying the input matrix to be $A^T A$ means the input matrix will always be a square matrix, which is a requirement for the HHL algorithm. This is useful for machine learning datasets which usually have many more rows, or data entries, than columns, or features. However, this method will increase the range of the matrix entries and increase the complexity of the eigenvalues. This will require the quantum phase estimation step of the HHL algorithm to use more qubits to represent the matrix's eigenvalues.

The Qiskit library includes an implementation of the HHL algorithm. The included function creates a quantum circuit that implements the HHL algorithm for given a matrix and a vector. This function was used because of its powerful implementation of the HHL algorithm. This version of the HHL algorithm includes eigenvalue scaling. A scaling factor is applied as a time evolution to the input matrix to make the eigenvalues simpler to encode in the quantum phase estimation stage. This is used to reduce the number of qubits used and reduce the error in the quantum phase estimation stage. For example, if both eigenvalues are a factor of $\frac{1}{3}$, the eigenvalues can be scaled to integers. This reduces error and qubits because integers can be easily

encoded using basis encoding while a fraction, especially those with many decimal places, cannot be encoded without error.

The SWAP test circuit was then designed to compare two quantum states with a generic number of qubits: the test point state and the solution state. The circuit begins by normalizing and encoding a given test point. The swap test circuit is then implemented as outlined in Section 2.4. Each qubit in the encoded test point is swapped with a qubit in the solution state. This implements a single or multi-qubit SWAP test that can determine the inner product for states that encode across multiple qubits.

Additionally, the SWAP test circuit can be modified to use a controlled-not and Toffoli gate instead of a traditional swap gate as seen in Figure 3.1b.



(a) Deconstructed SWAP Test Circuit **(b)** Reduced Gate SWAP Test Circuit

Figure 3.1: SWAP Test Circuit Variations

This can be done because the decomposition of the controlled-SWAP gate normally used in the SWAP test is a controlled-not gate followed by a Toffoli gate and another controlled-not gate [16]. However, the final controlled-not gate does not have an impact on the control qubit used in the SWAP test circuit. This means only the first controlled-not gate and Toffoli gate are required to determine the inner product of the two test states. This will reduce the number of gates by one controlled-not gate but will not retain the test and solution states. The missing final controlled-not gate means the test and solution states are not returned to their original state. This reduces the number of gates in the overall circuit if the test and solution states are no longer needed after the SWAP test. Finally, a measurement gate is added to the

control qubit in the SWAP test. The result from measurement represents the inner product of the two states.

Because the Qiskit implementation of HHL simply returns the quantum circuit to perform the algorithm, further additions can be made to the circuit. First, a measurement gate must be added to the ancilla qubit. Then, the SWAP test circuit can be combined with the HHL circuit. The complete circuit for a four-by-four matrix is shown in Figure 3.2. The qubits of SWAP test circuit implementing the control

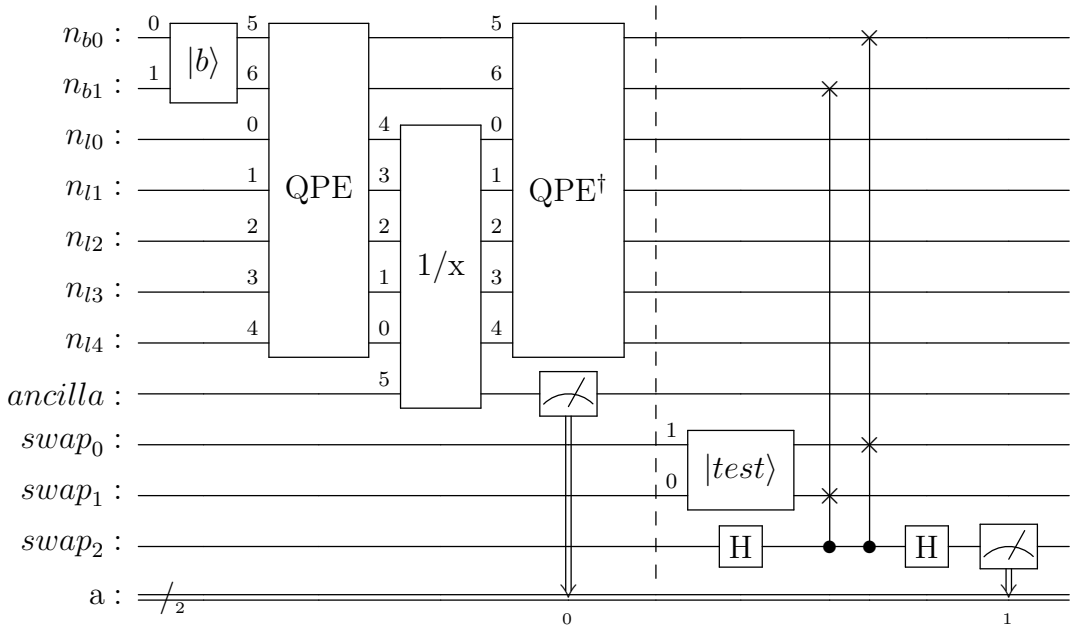


Figure 3.2: Complete HHL and SWAP Circuit Implementation

and test state are appended to qubits not used in the HHL circuit. The qubits of the SWAP test circuit that are being swapped with the solution state are mapped to the top qubits of the HHL circuit. This will swap the prepared test state with the solution state computed by the HHL circuit.

3.1.3 Measurement

After simulating the circuit, the results will return a distribution of four possibilities. These results are called counts and the total number of counts is equal to the number

of shots simulated. During a single shot, each measurement gate in a quantum circuit will return one bit indicating if the measurement gate measured a $|0\rangle$ or $|1\rangle$. Since there are two measurements being made, each shot will return two bits. One bit represents the measurement result of the ancilla qubit, and the other bit indicates the measurement result of the SWAP test.

If enough shots are used, the normalization factor applied during the eigenvalue rotation stage of the HHL algorithm can be determined by measuting the ancilla qubit. The normalization factor will be equal to:

$$normalization factor = \sqrt{\frac{P_a(|1\rangle)}{\text{total shots}}} \tag{3.3}$$

where $P_a(|1\rangle)$ is the number of times the ancilla qubit was measured as $|1\rangle$.

After performing the HHL portion of the circuit, the ancilla qubit is measured. The HHL result is only valid when the ancilla qubit is measured as $|1\rangle$. Therefore, when the inner product between the test state and solution state is calculated, the SWAP test measurement results are only used if the ancilla qubit is measured as $|1\rangle$. Using the equation derived in Section 2.4, the inner product of the solution, $|s\rangle$, and test vector, $|t\rangle$, resulting from the swap test is:

$$\langle s|t\rangle = \sqrt{2 \cdot \frac{P_s(|0\rangle)}{P_s(|0\rangle) + P_s(|1\rangle)} - 1} \tag{3.4}$$

where P_s is the number of times the SWAP test measurement gate is measured as $|0\rangle$ or $|1\rangle$.

Next, the Euclidean distance between the solution and test state is determined. The Euclidean distance of two vectors can be represented as:

$$Ed^2 = |a|^2 + |b|^2 - 2\langle a, b\rangle \tag{3.5}$$

If the vectors a and b are normalized to 1, then the Euclidean distance can be reduced to:

$$Ed^2 = 2 - 2\langle a, b\rangle \tag{3.6}$$

Since the vectors used in the swap test are normalized, the Euclidean distance between the solution and test point can be defined as:

$$|Ed| = \sqrt{2 - 2\langle s|t\rangle} \tag{3.7}$$

3.1.4 Accelerating Neural Networks: Quantum Adaptive Learning Rate

For a system of linear equations, linear regression converges on a solution for the vector $\vec{x}$ where $A\vec{x} = \vec{b}$. A simple gradient descent algorithm was implemented to perform the linear regression on a linear system of equations. This iterative process updates the solution by calculating the error between $\vec{b}$ and $A\vec{x}^*$, where $\vec{x}^*$ is the current estimation of the true solution $\vec{x}$. The estimated solution is updated using the equation below in each iteration:

$$\vec{x}^* = \vec{x}^* - lr * A^T(A\vec{x}^* - \vec{b}) \tag{3.8}$$

where lr is the learning rate, a hyperparameter in gradient descent. Over many iterations, $\vec{x}^*$ converges to a value close to the true solution $\vec{x}$. The linear regression function finishes when the mean squared error between $\vec{b}$ and $A\vec{x}^*$ is below a threshold, ε.

To aid the gradient descent, the SWAP test distance between the estimation $\vec{x}^*$ and the HHL solution is performed. The result from the swap test is the Euclidean distance between the normalized vector of the current iteration's estimation and the normalized true solution to the system of linear equations. The Euclidean distance across iterations is then used as a metric to adjust the gradient descent's learning rate.

The distance indicates how near the linear regression is to the correct solution. Even though this distance is normalized and not the true distance between the solution and test point, the distance can indicate if large steps can be used or if finer iterations are required to obtain a correct estimate. The learning rate, lr, is updated to:

$$lr = \frac{lr}{1 + (1 - \frac{Ed}{\sqrt{2}}) * \beta} \tag{3.9}$$

where Ed is the Euclidean distance determined by the SWAP test and β is a user modifiable hyperparameter. The hyperparameter β can be increased so the Euclidean distance has a larger impact on the learning rate or reduced to make the impact on the learning rate more gradual. This formula works similar to adaptive learning rates using time based decay. The Euclidean distance is divided by the square root of two to set the range between zero and one. This value is then subtracted to one so that the denominator increases as the Euclidean distance decreases. The result is an adaptive learning rate that decreases as the Euclidean distance decreases. To reduce the number of SWAP tests used, the learning rate can be updated periodically instead of every iteration. Figure 3.3 illustrates the hybrid model and which operations are performed on a classical processor or on a quantum processor.

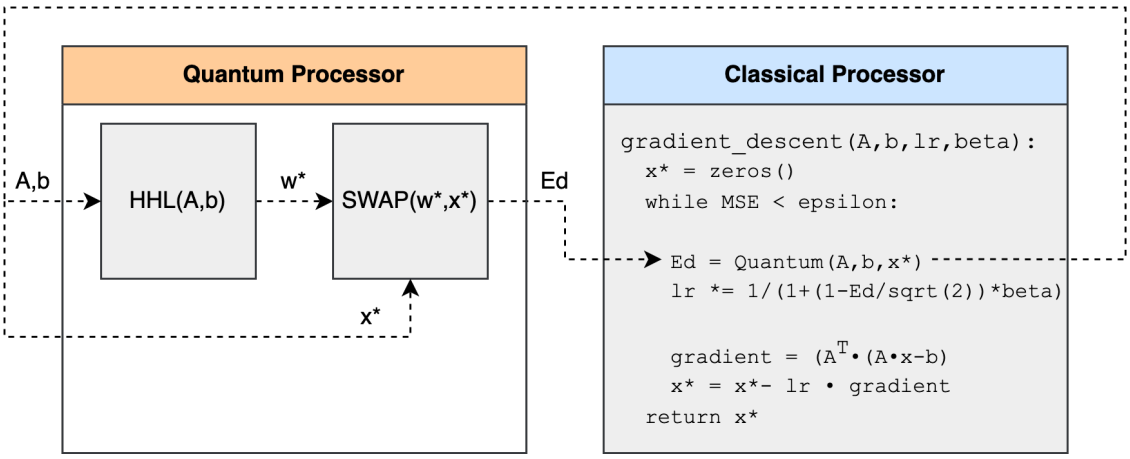


Figure 3.3: Hybrid Quantum-Classical Model for Gradient Descent

While other methods of variable learning rates exist such as learning rate scheduling and adaptive learning rates, these methods use estimates and pre-defined schedules to adjust the learning rate. This quantum method adjusts the learning rate using a true metric of how close the estimate is to the true solution. This means the learning rate can be heavily modified without overshooting the solution.

3.2 Experiments

All experiments were simulated using Qiskit's Aer simulator without noise. While this is not a realistic representation of current quantum hardware, it is necessary as current quantum hardware cannot run the full, generic implementation of the HHL algorithm. Others [19, 14] have implemented optimized versions of the HHL algorithm that can be performed on current quantum hardware, however, these reduced circuits add additional restrictions to the input matrices. This work aims to explore the attributes of the HHL algorithm and potential applications in hope that future quantum hardware will be capable of computing larger quantum circuits. Since quantum circuit results are represented as a probability distribution, a quantum circuit must be run many times to create an accurate distribution of the results. Each run is called a shot, and all simulations in this work used 20,000 shots.

When simulating experiments, the full HHL solution was extracted as well. This would not normally be done as it breaks the computational advantage of the HHL algorithm and can only be done in simulation. However, extracting the full solution allows for better verification of the HHL solution and allows the error between the expected and HHL solution to be calculated. This was done by saving the statevector of the circuit after the HHL algorithm was simulated. The HHL solution can then be extracted from the statevector and normalized.

3.2.1 Model under Ideal Conditions

To demonstrate the combined HHL and SWAP test circuit, the following simple system was tested:

$$\frac{1}{4}\begin{bmatrix} 15 & 9 & 5 & -3 \\ 9 & 15 & 3 & -5 \\ 5 & 3 & 15 & -9 \\ -3 & -5 & -9 & 15 \end{bmatrix} \vec{x} = \frac{1}{2}\begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \end{bmatrix} \tag{3.10}$$

This system of linear equations was selected because it is an ideal system to test the HHL algorithm. The eigenvalues of the matrix are 8, 4, 1, and 2, which are easy to encode in the QPE stage of HHL [19]. This reduces error in the HHL algorithm and will demonstrate the model's capabilities under ideal conditions. Two test vectors were used and the error was recorded in Table 3.1. The statevector error is the L2 error between the normalized true solution and the extracted statevector solution. The SWAP error is the L2 error between the measured Euclidean distance from the SWAP test and the true Euclidean distance between the normalized solution and test point.

Table 3.1: Ideal Matrix Statevector and Swap Test Euclidean Distance Error

Test Vector	Statevector Error	SWAP Error
[0.5,0.5,0.5,0.5]	0.0	0.00565
[0.85896 0.23426 0.23426 0.39043]	0.0	0.0

The results in Table 3.1 show little error. This is because the matrix's eigenvalues can be perfectly represented in the QPE stage of the HHL algorithm. In ideal circumstances the SWAP test can be used to gain an accurate measurement of the Euclidean distance between a test point and the solution of a system of linear equations. While the experiment was an ideal case, the circuit used 16 qubits and had a depth of 888. Since this is far too large a circuit to run on current quantum hardware, all experiments were run in simulation.

3.2.2 Quantum Phase Estimation Counting Qubits

One significant source of error in the HHL algorithm is in the quantum phase estimation stage. In this circuit stage, the eigenvalues are estimated and stored in a quantum register. Since this stage performs an estimation of the eigenvalues and not an exact calculation, there will likely be error. However, the amount of error in this stage can vary depending on the size of the quantum register being used to represent the calculated eigenvalues. Selecting the size of this quantum register can greatly change the amount of error caused by expressing an eigenvalues with a limited number of qubits. The size of the quantum register will have further implications on the number of qubits and quantum gates used in the quantum phase estimation stage. This will increase the depth of the circuit and increase the run time or simulation time of the circuit.

The standard implementation of the HHL algorithm automatically selects the size of the quantum phase estimation stage based on the input matrix's condition number. To manually test the impact of register size, the quantum phase estimation stage of the circuit was modified so the size of the quantum register could be manually selected. Experiments using different sizes of quantum registers were simulated to determine the effect of the register size on the error of the HHL result and the error on the SWAP test distance.

Two simple matrices were tested to see the impact of the register size:

$$A_1 = \begin{bmatrix} 1.25 & 0 \\ 0 & 1 \end{bmatrix} A_2 = \begin{bmatrix} 1.125 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.11}$$

The eigenvalues of A_1 are 1.25 and 1, and the eigenvalues of A_2 are 1.125 and 1. The L2 error between the true Euclidean distance and SWAP test Euclidean distance, and the L2 error between the true solution and statevector solution were recorded in

Table 3.2.

Table 3.2: Error vs Register Size

	Matrix A_1		Matrix A_2	
Register Size	**SWAP Error**	**Statevector Error**	**SWAP Error**	**Statevector Error**
3	0.2936	0.0963	0.1904	0.0161
4	0.1645	0.0236	0.1995	0.0696
5	0.0	0.0	0.2314	0.0854
6	0.0	0.0	0.2147	0.0605
7	not tested	not tested	0.0	0.0
8	not tested	not tested	0.0	0.0

As expected, unless there are enough qubits to perfectly represent the matrix's eigenvalues using basis encoding, there will be error. Once the number of qubits is increased such that the eigenvalues can be exactly ecoded, there is no longer error caused by the HHL circuit. Additionally, if extra, unnecessary qubits are added there is no negative effect on the result.

To test the impact of the register size when the decimals cannot be exactly represented, the following system of equations was used

$$\begin{bmatrix} 2.5 & 3 \\ 1 & 4 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.12}$$

which was modified to be Hermitian. This matrix was selected due to its complex eigenvalues of

$$\lambda = [5.53637, -5.53637, 1.26437, -1.264371]$$

These eigenvalues are difficult to encode because they cannot be exactly encoded with a reasonable amount of qubits. The HHL algorithm used a scaling factor on the

eigenvalues of $s = 1.26436$ which resulted in post scaling eigenvalues equal to

$$\frac{\lambda}{s} = [4.37877, -4.37877, 1, -1]$$

Eigenvalue scaling was enabled to increase the accuracy of experiment as seen in Section 3.2.3. Even post scaling, an extreme number of qubits would be required to perfectly represent the eigenvalues in this experiment. Experiments were run increasing the number of qubits used in the counting register during the quantum phase estimation stage of the HHL algorithm. The true solution was used as the test vector in the SWAP test so the expected Euclidean distance would be zero. The L2 error between the true solution and the statevector solution, and the L2 error in the SWAP test were recorded in Table 3.3.

Table 3.3: Error vs QPE Register Size

Register Size	**SWAP Error**	**Statevector Error**
4	0.3931	0.00031
5	0.1817	0.0001
6	0.2944	0.0188
7	0.2991	0.0063
8	0.0	0.0001
9	0.0	0.001

Table 3.3 indicates that increasing the number of qubits in the quantum phase estimation stage can reduce the error in the final HHL solution even if the eigenvalues are not exactly represented. However, the relationship between error and register size is not linear. Adding qubits can aid in reducing error because this allows for the encoded eigenvalues to be better represented in a limited number of qubits. However, the reduction in error will not be linear because an additional qubit will not always better represent a value. For example, the number 0.625 can be perfectly represented with four binary digits: 0.101. However, if fewer binary digits are used, the estimation will remain the same with two or three binary digits: 0.1 and 0.10 both estimate 0.625

to 0.5. With enough qubits the error drops significantly. This indicates that even if the eigenvalues are not exactly represented by the quantum phase estimation, a high enough accuracy can be achieved for an accurate SWAP test result. One must note that while increasing the register size may improve the results, the complexity of the circuit will increase. This means selecting the size of the quantum phase estimation register is not a trivial step in utilizing the HHL algorithm.

3.2.3 Scaling Eigenvalues

The Iris data set was used for this experiment [20]. The Iris data set is a simple data set often used to test classification in machine learning.

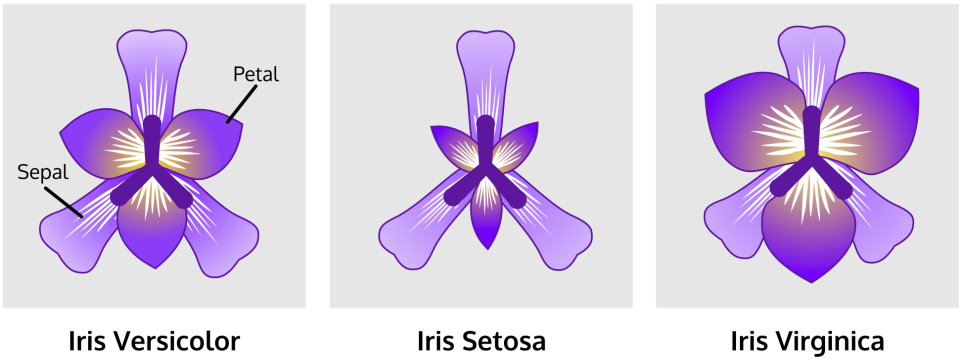


Figure 3.4: Iris Data Set Visualization [4]

The data set consists of three classes and four features. The three classes represent three distinct iris plant species. The four features are the sepal length, sepal width, petal length, and petal width of an iris flower. This data set is strong candidate for testing the HHL algorithm because it is a simple, widely used, and tangible data set. The following system of linear equations was created by randomly selecting four

samples from the Iris data set

$$\begin{bmatrix} 5.2 & 6.6 & 5.8 & 6.3 \\ 2.7 & 2.9 & 2.8 & 2.9 \\ 3.9 & 4.6 & 5.1 & 5.6 \\ 1.4 & 1.3 & 2.4 & 1.8 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{3.13}$$

which was altered to be Hermitian as explained in Section 3.1.2 before performing the HHL algorithm. The resulting matrix used was an eight by eight matrix with the eigenvalues

$$\lambda = [16.75792, -16.75792, 1.07088, -1.07088, 0.48816, 0.23879, -0.23879, -0.48816]$$

Two experiments were run with this system of linear equations, one with eigenvalue scaling enabled and one without scaling. The first experiment with eigenvalue scaling used a scaling factor of $s = 0.23879$ resulting in post scaled eigenvalues of

$$\frac{\lambda}{s} = [70.17819, -70.17819, 4.48461, -4.48461, 2.04429, 1, -1, -2.04429]$$

Both experiments used the default HHL implementation which used eight qubits for the QPE counting register to represent the eigenvalues. This resulted in a quantum circuit with a depth of 34200 and used a total of 16 qubits: three to encode the input vector, eight for the QPE stage, one ancilla qubit, three to encode the test vector, and one control qubit for the SWAP test.

Table 3.4: Error With and Without Eigenvalue Scaling

	SWAP Error	**Statevector Error**
Scaling Enabled	0.3645	0.0122
Scaling Disabled	0.5815	0.5371

The results in Table 3.4 show that scaling the eigenvalues has a positive impact

on the accuracy of the HHL algorithm. Because of the significant reduction in error after scaling the smallest eigenvalues to one, it indicates that the minimum eigenvalue contributes greatly to the system of equations. When using the same number of qubits, scaling is an effective way of reducing error. However, in practice it would not be realistic to scale the eigenvalues of the matrix to exact integers because it would require prior knowledge of the matrix's eigenvalues. One can assume, as seen Section 3.2.2, that if enough qubits were available, scaling would not be necessary as the error could instead be reduced by increasing the number of qubits used. However, due to a lack of available qubits and limitations on simulating quantum circuits, scaling is used in this research achieve more accurate results with fewer qubits.

3.2.4 Aiding Gradient Descent

Both the ideal matrix, Equation 3.10, and Iris matrix, Equation 3.13, were tested in fully simulated experiments executing the full hybrid gradient descent model. Classical gradient descent was performed on the ideal matrix with a learning rate of $lr = 0.01$ and $lr = 0.03$. The hybrid model was performed with an initial learning rate of $lr = 0.03$ and $\beta = 0.3$. The learning rate was updated using the SWAP test every 10 iterations. By only updating the learning rate every 10 iterations, the number of SWAP tests required is reduced. If implemented into a larger regression model, the learning rate could be updated every epoch similar to step decay learning rates. The learning rate and loss were recorded in Figure 3.5.

As expected the learning rate is updated similar to a step decay adaptive learning rate. Across the 100 iterations of gradient descent, 11 SWAP tests were performed. Compared to the true Euclidean distance between the true solution and gradient descent estimation, the 11 SWAP tests had an average L2 error of 0.01211. Compared to the constant learning rate, the quantum adaptive learning rate (QALR) achieves a lower loss and quicker descent. This means with the quantum adaptive learning

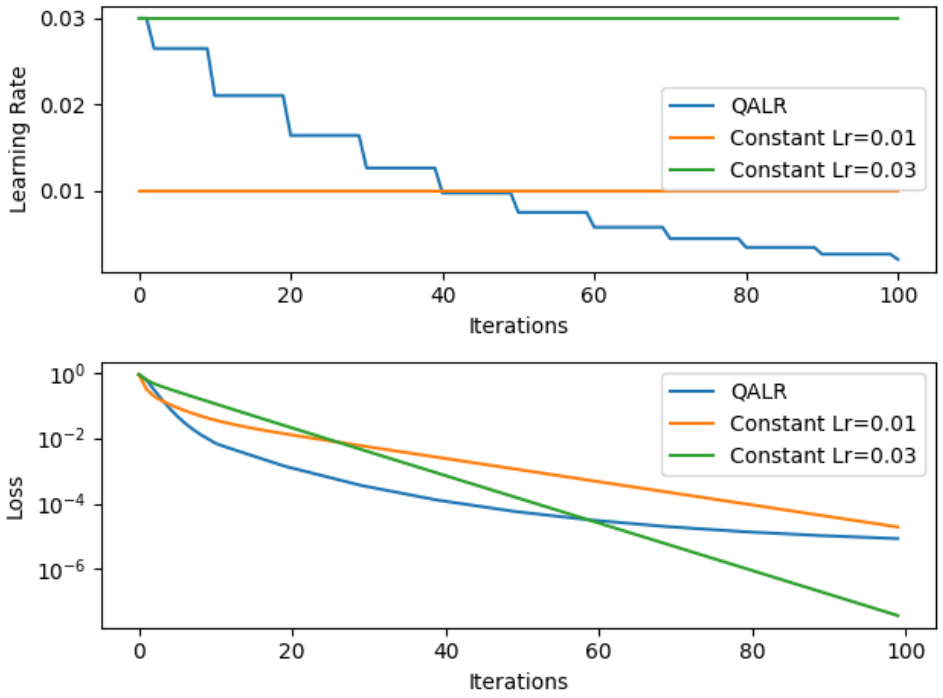


Figure 3.5: Learning Rate and Loss vs Iterations (Ideal Matrix)

rate the same loss is reached in fewer iterations than the constant learning rate. The loss with a constant learning rate of $lr = 0.01$ does near the loss seen in the quantum adaptive learning rate toward the final iterations, and the constant learning rate of $lr = 0.03$ surpasses the adaptive learning rate around 60 iterations. This is because the quantum adaptive learning rate becomes increasingly small causing its loss to slow during later iterations. The constant learning rate of $lr = 0.01$ and quantum adaptive learning rate would intersect to have the same loss around iteration 110.

For the Iris matrix, the classical gradient descent was performed with a constant learning rate of $lr = 0.001$ and $lr = 0.003$. An initial learning rate of $lr = 0.003$ and $\beta = 0.5$ were used for the hybrid gradient descent. Due to the amount of error seen in Table 3.4, even with scaling enabled, the number of qubits used in the QPE stage was increased from 8 to 12.

The results in Figure 3.6 resemble those in the previous experiment. The loss drops lower and faster than the constant learning rate meaning a lower loss can

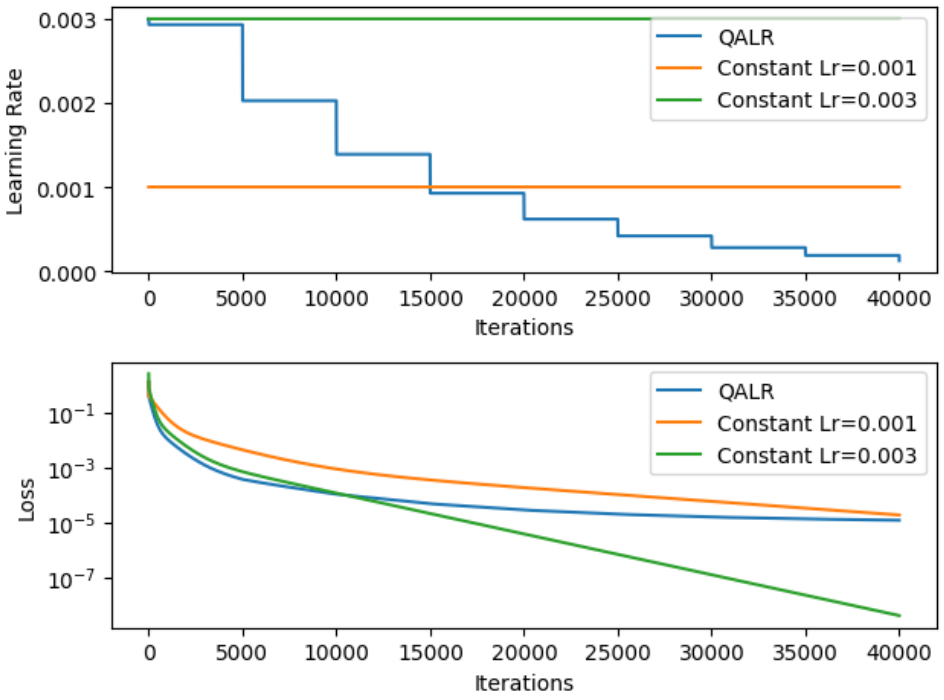


Figure 3.6: Learning Rate and Loss vs Iterations (Iris Matrix)

be achieved in fewer iterations. However, the gap between the quantum adaptive learning rate and constant learning rate of $lr = 0.003$ is not as wide. Again, the loss of the constant learning rate of $lr = 0.001$ begins to approach the quantum adaptive learning rate as the adaptive learning rate nears zero. The constant learning rate and quantum adaptive learning rate would intersect to have the same loss after about 45000 iterations. The learning rate was updated every 5000 iterations and performed the SWAP test nine times. The Euclidean distance determined by the SWAP test had an average L2 error of 0.03233 compared to the true Euclidean distance.

While both experiments had little error, it is worth noting that the SWAP test returned an Euclidean distance of zero for the last few updates. Small distances, approximately less than 0.05 in these examples, returned an Euclidean distance of zero. Increasing the number of shots would improve the precision of the Euclidean distance from the SWAP test. However, this increases the circuit execution time and was deemed not necessary as the extremely small distances towards the end of the

experiment have minimal impact on the learning rate. This indicates that as the distance between the estimated and true solution becomes increasingly small, the quantum approach becomes less impactful. A more efficient algorithm could switch to a classical adaptive learning rate once the Euclidean distance from the SWAP test becomes minimal.

By incorporating the quantum adaptive learning rate into classical gradient descent, the training process of a linear regression problem is improved. The loss is reduced quicker than a constant learning rate indicating that the quantum adaptive learning rate improves the speed at which a gradient descent converges on ideal weights.

While the focus of this work was to implement a proof-of-concept model, some research was made into the runtime and speedup of this model. The HHL circuit is kept from previous works such that the HHL portion of the circuit would scale the same. Since the true solution does not change during gradient descent, the HHL portion of the model should only need to be performed once. A previous work [17] explored the cost of the SWAP test. Experiments demonstrated that with a carefully selected test point, the SWAP test was not computationally expensive. Foulds et. al. [16] indicates that the SWAP test can have better performance than quantum state tomography especially when the two states compared are more entangled. Even though the SWAP test is not computationally expensive, efforts must be made to mitigate the number of SWAP test. This was explored in this work by limiting the SWAP test to select iterations during gradient descent and the possibility of using a reduced SWAP test when the quantum states are no longer needed.

Chapter 4

Conclusion

4.1 Future Work

Based on the experiments performed in this work, additional research could be done to best utilize the Euclidean distance in the quantum adaptive learning rate. While this work did integrate the Euclidean distance into an adaptive learning rate, the implementation was based on classical learning rate decay methods. If the quantum Euclidean distance could be incorporated in a more advanced optimizer, further improvements could be made to learning regression.

Additionally, any future improvements made to the HHL algorithm could improve the near term feasibility of this model. Further research could be made into quantum tomography to improve the SWAP test precision, especially with small decimals. Finally, a robust method of selecting the size of the quantum phase estimation register size would be greatly beneficial to the HHL algorithm. Good register size selection would reduce the error of the HHL algorithm while keeping the circuit size as small as possible which is extremely important for near term and noisy quantum processors.

4.2 Conclusion

Through this work, the abilities of the HHL algorithm and SWAP test were explored. By testing the HHL circuit, the importance of the input matrix's eigenvalues were

determined. The quantum phase estimation stage of HHL has a great impact on the circuit's performance because this stage estimates the eigenvalues of the input matrix. By applying a time evolution to the input matrix and scaling the matrix's smallest eigenvalues, the error occurred during the quantum phase estimation could be greatly reduced. The error occurred in this stage could also be reduced by increasing the register size used to represent the eigenvalues. By increasing the register size, the error occurred could be minimized to acceptable amounts. However, increasing the register size does not have linear relationship to a reduction in error, so selecting the size is not a trivial task.

When the HHL circuit was correctly designed, the SWAP test could accurately determine the inner product between the solution to a system of linear equations determined by the HHL circuit and a prepared test point. Based on the inner product, the Euclidean distance between the solution and test point can be calculated. In cases with ideal eigenvalues and complex eigenvalues, if enough qubits were used for the QPE register, the SWAP test was able to successfully determine the Euclidean distance with minimal error.

The combined HHL and SWAP test circuit was then incorporated into a classical gradient descent algorithm. The combined quantum circuit was able to calculate the Euclidean distance between the normalized solution and the gradient descent algorithm's current estimate for the solution. This normalized Euclidean distance was incorporated into the gradient descent algorithm as a quantum adaptive learning rate. The quantum adaptive learning rate was able to reduce the loss occurred in the gradient descent algorithm better and quicker than a constant learning rate.

While current noise rate in quantum hardware limit the ability to run this complete model on current quantum hardware, this model presents a blueprint for incorporating quantum computing in regression algorithms. The hybrid nature of the model explores the use of future quantum processors as accelerators in classical computing systems.

Bibliography

[1] Rochester Institute of Technology, "Research computing services," 2022. [Online]. Available: https://www.rit.edu/researchcomputing/

[2] D. Koch, L. Wessing, and P. M. Alsing, "Introduction to coding quantum algorithms: A tutorial series using qiskit," 2019.

[3] A. J., A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'malley, D. Oyen, S. Pakin, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. J. Swart, J. G. Wendelberger, B. Yoon, R. Zamora, W. Zhu, S. Eidenbenz, A. Bärtschi, P. J. Coles, M. Vuffray, and A. Y. Lokhov, "Quantum algorithm implementations for beginners," *ACM Transactions on Quantum Computing*, vol. 3, no. 4, pp. 1–92, jul 2022. [Online]. Available: https://doi.org/10.1145%2F3517340

[4] "Iris dataset." [Online]. Available: https://www.codecademy.com/courses/machine-learning/lessons/machine-learning-clustering/exercises/iris-dataset

[5] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, oct 2009. [Online]. Available: https://doi.org/10.1103%2Fphysrevlett.103.150502

[6] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, oct 2014. [Online]. Available: https://doi.org/10.1080%2F00107514.2014.964942

[7] "Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two," https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two.

[8] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[9] M. Schuld, "Supervised quantum machine learning models are kernel methods," 2021.

[10] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Expanding data encoding patterns for quantum algorithms," in *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, 2021, pp. 95–101.

[11] D. Dervovic, M. Herbster, P. Mountney, S. Severini, N. Usher, and L. Wossnig, "Quantum linear systems algorithms: a primer," 2018.

[12] Y. Cao, A. Daskin, S. Frankel, and S. Kais, "Quantum circuit design for solving linear systems of equations," *Molecular Physics*, vol. 110, no. 15-16, pp. 1675–1680, aug 2012. [Online]. Available: https://doi.org/10.1080%2F00268976.2012.668289

[13] H. J. M. J. au2, A. Zaman, and H. Y. Wong, "Step-by-step hhl algorithm walkthrough to enhance the understanding of critical quantum computing concepts," 2023.

[14] Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost, and P. Wittek, "Bayesian deep learning on a quantum computer," *Quantum Machine Intelligence*, vol. 1, no. 1-2, pp. 41–51, may 2019. [Online]. Available: https://doi.org/10.1007%2Fs42484-019-00004-7

[15] S. Aaronson, "Read the Fine Print," *Nature Physics*, vol. 11, pp. 291–293, 04 2015.

[16] S. Foulds, V. Kendon, and T. Spiller, "The controlled SWAP test for determining quantum entanglement," *Quantum Science and Technology*, vol. 6, no. 3, p. 035002, apr 2021. [Online]. Available: https://doi.org/10.1088%2F2058-9565%2Fabe458

[17] S. L. Alarcon, C. Merkel, M. Hoffnagle, S. Ly, and A. Pozas-Kerstjens, "Accelerating the training of single layer binary neural networks using the HHL quantum algorithm," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, oct 2022. [Online]. Available: https://doi.org/10.1109%2Ficcd56317.2022.00070

[18] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[19] Y. Cao, A. Daskin, S. Frankel, and S. Kais, "Quantum circuit design for solving linear systems of equations," *Molecular Physics*, vol. 110, no. 15-16, pp. 1675–1680, aug 2012. [Online]. Available: https://doi.org/10.1080%2F00268976.2012.668289

[20] E. Anderson, "The irises of the gaspe peninsula," *Bulletin American Iris Society*, vol. 39, pp. 2–15, 1935.