Quantum Adaptive Learning Rate

Martin Hoffnagle, Sonia López Alarcón and Cory Merkel
Rochester Institute of Technology, Rochester, NY, USA, 14623

***Abstract*—The training of Neural Networks is a compute intensive task that, in current classical implementations, relies on gradient descent algorithms and a certain learning rate that controls the granularity of the search for a solution. This paper explores a new hybrid quantum-classical approach, which is not only novel for exploding quantum computing to partially solve the problem, but also for being the first approach that adjusts the learning rate with exact information pertaining to the solution of this training problem. The Quantum Adaptive Learning Rate approach is tested in a proof of concept classification problem. Key aspects of the practical implementation of the Harrow, Hasidim and Lloy (HHL) quantum algorithm are discussed.**

1. Introduction

Quantum computing (QC) is an —still— emerging technology that takes advantage of quantum mechanics to offer a computational solution with potential advantages over classical computing. One of the most promising quantum algorithms was proposed by Harrow, Hasidim and Lloy [1], aka *HHL algorithm*, in 2009 to solve systems of linear equations with (potentially) exponential speedup over classical implementations. Needless to say, the scientific and engineering communities were full of hope over this algorithm's potential, including the Machine Learning community.

Today's most advanced Machine Learning algorithms, Neural Networks, are inspired by the brain's complex network of ∼86 billion neurons. Neural networks have made significant achievements (e.g. Krizhevsky et al.'s landmark paper [2] on ImageNet classification with convolutional neural networks (CNN)) in the last decade. However, these improvements come at the cost of the models' size and complexity, with today's state-of-the-art models containing 100s of billions of parameters. Moreover, implementing training and inference of these state-of-the-art models on conventional CPU/GPU hardware incurs large overheads that are incompatible with application domains that have strict size, weight, and power (SWaP) constraints. Quantum computing is currently far from being able to help solve this size problems, but as an emerging technology, creative exploration of its applications can help bridge the gap towards practical usage of this new computational approach. In particular, a single layer NN can be represented as a linear system of equations, and therefore, —one would think— would be a potential candidate for HHL application.

However, when one carefully looks at the HHL algorithm, the true challenges of this approach arise. Beyond the many difficulties of the implementation of HHL on quantum hardware (with noise, error correction, and connectivity issues to name some), the algorithm has strict numerical conditions on the input matrix representing the system of equations, which can critically affect the computational advantage or the precision of the solution. Furthermore, the solution to the system of equations is quantum mechanically built, and encoded in *quantum superposition*. Trying to extract this information into the classical representation of each variable would kill any computational advantage. One of the most insightful papers on the topic by Scott Aaronson [3] reflects on these strict conditions, and their impact on ML.

That being said, if the input matrix can be manipulated and processed, the output of HHL in quantum superposition *does* contain and encode the solution of the system of equations, and other quantum operators can be applied to it to extract some information. Built on these premises, this paper proposes a Quantum Adaptive Learning Rate: a hybrid quantum-classical solution that uses HHL followed by the SWAP test to adjust the classical gradient descent's learning rate based on true knowledge of the distance to the optimal solution of the training problem. This approach is tested on a small real case, the classification of the *Iris test set*.

These main contributions are important for two reasons. *First*, to our knowledge, this is the first proposed approach to adjust the learning rate based not on estimations of how close the exploration of the optimization surface is to an absolute minimum, but based on the certain knowledge of this absolute minimum. The key here is that this absolute solution is represented in a quantum state and cannot be fully extracted. But the quantum SWAP test provides the distance between the current exploration point and the solution, and hence, the gradient descent step can be adjusted accordingly. *Second*, the practical implementation of a real problem on a (simulated) quantum computer provides insight on the limitations of the HHL algorithm, and the required manipulation of the input matrix when it pertains to real world data. This research group was able to successfully find the HHL solution to the training of a single layer NN on the *Iris* dataset. The proposed approach does not intend outperform the classical implementations yet, but to take steps towards a better understanding of quantum algorithms and their applications to real problems.

2. Background and related work

Hybrid quantum-classical solutions in the ML field are not new. Quantum Machine Learning is a broad field in the intersection of quantum technology and machine learning. It expands from quantum-inspired classical algorithms [4] or ML enhanced quantum technology and tomography [5] to Quantum Neural Networks implementations to classify classical or quantum data [6].

Quantum Neural Networks is an extremely active field of research. In QNN, a hybrid iterative process provides a solution to the training of Neural Networks. This approach starts with the selection of a quantum ansatz and a set of quantum parameters. The ansatz is evaluated quantum-mechanically, while parameters are adjusted in the classical side to tune the quantum ansatz to the training solution. This process is repeated iteratively until the classical side decides the parameters are close enough to the optimal training solution. There are several challenges in this approach, such as the hardware limitation of current quantum computers, the initial selection of the ansatz function and its parameters, the topology of the optimization problem presenting barren platoes and narrow gorges, and the back-and-forth communication between the classical and quantum sides of the implementation. In addition, the specific applications of QNN are still unknown, and they have not yet demonstrated quantum advantage, that is the ability to outperform classical implementations.

The work presented in this paper leverages the HHL algorithm [1]. A plethora of problems can be reduced to solving a linear system of equations, and as such, this quantum acceleration was a source of hope for many fields. Scott Aaronson in his “Quantum Machine Learning: Read the fine print” [3] discusses in detail all the limitations of the HHL algorithm, from precision of the solution to the condition number and sparsity of the input matrix. The encoding of the input data is also a source of computational complexity in the absence of a quantum memory. Nevertheless, based on the HHL solution to a linear regression problem, Zhao et al. proposed a solution to Bayesian deep learning using a quantum computer [7]. This example was a proof of concept on a small, preset system of equations. López Alarcón et al. proposed the use of the SWAP test to calculate the distance between a known test point and the actual solution to the training problem [8]. This work was limited to binary neural networks, and explored how knowledge of a distance to the optimal solution could be used to reduce the search time of a gradient descent approach. The quantum piece of the algorithm was discussed but not implemented in any practical capacity.

One naive way of finding the weights of an artificial neural network (ANN) would be to solve a massive system of equations. In the case of a single layer ANN, this would be a linear system of equations. The classical solution of a linear system of equations has an approximate complexity of $O(N^3)$. With the current size of training data, features and weights, this approach is impracticable, and that is the reason why the training of ANNs is solved as an optimization problem in search of the minimum loss function.

While single-layer neural networks can be used on linear classification and regression problems, they can also solve large classes of non-linear problems by first applying transformations to the input space. For example, so-called extreme learning machines [9] employ a random, untrained hidden layer of neurons to transform the input space into a high-dimensional random feature space. Then, linear combinations of the random features are used to classify the original inputs. Reservoir computing [10] leverages a similar concept, using random dynamical systems to cast time series data into a large feature space which can be learned through simple linear regression.

2.1. HHL algorithm

The HHL algorithm is 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 $A\vec{x} = \vec{b}$. In quantum notation, this linear system of equations is expressed as

$$A\,|x\rangle = |b\rangle \tag{1}$$

where A is a Hermitian operator — a workaround exists when A is not Hermitian, see below— and $\vec{b}$ has to be encoded in a quantum state $|b\rangle$ and, hence, it has to be normalized. The solution to this linear system of equations is, therefore, expressed as

$$|x\rangle = A^{-1}\,|b\rangle \tag{2}$$

The problem boils down to finding A^{-1}, like in any other algorithm to solve linear systems of equations. Given a matrix A of size $N \times N$, classical algorithms would have a computational complexity $O(N^3)$. If the operator were expressed as a diagonal matrix, the calculation of A^{-1} is almost immediate, creating a diagonal matrix in which the eigenvalues λ_i in the diagonal are replaced by their corresponding inverted values, λ_i^{-1}. The computationally intensive portion of this solution in the classical context is precisely to calculate the eigenvalues of A. This, however, can be solved rather easily in a quantum-mechanical manner through the use of the quantum phase estimation in which the eigenvalues are calculated and expressed as a phase.

If the matrix A is not Hermitian, the matrix and input vector are modified such that:

$$\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \begin{bmatrix} 0 \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{3}$$

which ensures the new matrix is Hermitian. In this case, the size of the matrix is doubled.

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

$$A^T A|x\rangle = A^T|b\rangle \tag{4}$$

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, which has a negative impact on the size of the quantum circuit, as is shown in Section 4.

The circuit implementation can be broken down into six different circuit stages, as shown in Figure 1: loading the $|b\rangle$ input, Quantum Phase Estimation, Eigenvalue Inversion, inverse Quantum Phase Estimation, ancilla qubit measurement, and application of a function $F(x)$ followed by 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.

The HHL algorithm can potentially estimate the function of the solution vector in running time complexity $O(\log(N)s^2\kappa^2/\epsilon)$, given that the matrix A is s-sparse and well-conditioned, where κ denotes the condition number of the

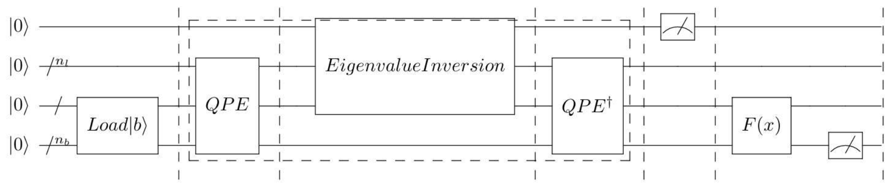


Figure 1: Generic HHL Algorithm with 5 stages: state $|b\rangle$ preparation, Quantum Phase Estimation, Eigenvalue Inversion, inverse QPE, measurement of the flag qubit, and application of some function $F(x)$. Note that only in the case of the measurement in the first register producing the outcome 1 the algorithm succeeds [1].

system, and ϵ the accuracy of the approximation [1]. It is important to take into account that the algorithm assumes the case being considered is one in which the solution $|x\rangle$ does not need to be fully known, but rather an approximation of the expectation value of some operator associated with the solution vector. $|x\rangle$ is in a superposition state of all the independent values of the vector. It is an amplitude encoded solution: the amplitude of each basis state contains each of the weights, solution to the optimization problem. Trying to read each of them would require an amount of time such that it would ruin any quantum advantage.

2.2. Distance to the solution through the SWAP test

With that perspective, the work by Lopez Alarcon et al. [8] proposed to combine HHL with the SWAP test to extract information of the distance between the $|x\rangle$ vector and some other known $|test_x\rangle$ state.

The circuit for a two state comparison is shown in Figure 2. Given a “reference” state, the SWAP test can provide the distance from the solution to the this state, and hence, reduce the classical search of the solution to the hypersphere of distance d from the reference state. The *SWAP test* routine is a quantum algorithm that expresses the scalar product of two input states. Two input states $|A\rangle$ and $|B\rangle$ are compared using controlled-SWAP gates and a control bit.

Previous research [8], describes a framework for how the Euclidean distance can be used to reduce the search space of a classical regression algorithm, improving a neural network's 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. Although this was a promising route, this approach has some downsides: First, the solutions and test cases need to be normalized, and therefore, the distance is known between the normalized vectors, not the true scale ones. Second, exploring the optimization landscape with a new constraint does not necessarily simplify the search process; it may instead produce a new surface which, despite having reduced dimensionality, is not well suited for gradient descent. Last, the approach was tested on binary neural networks, which simplified the problem but also seriously limited the practical test cases to implement quantum mechanically.

3. Quantum Adaptive Learning Rate, QALR

This paper proposes an improvement to the approach described in Section 2.2, in which 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 network's 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 an 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 one. The quantum adaptive learning rate improves the convergence speed of the gradient descent algorithm, when compared to constant learning rates.

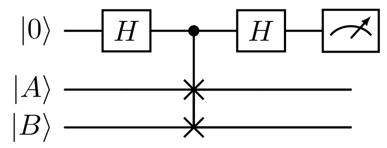


Figure 2: The standard SWAP test. The resulting state of the circuit is (without normalizations) $|0\rangle(|AB\rangle + |BA\rangle) + |1\rangle(|AB\rangle - |BA\rangle)$, and thus the probability of obtaining outcome $|1\rangle$ on the measurement is $\frac{1}{4}(\langle AB| - \langle BA|)(|AB\rangle - |BA\rangle) = \frac{1}{2} - \frac{1}{2}|\langle A|B\rangle|^2$, where if $|A\rangle = \sum_i a_i |i\rangle$ and analogously for $|B\rangle$, then $\langle A|B\rangle = \sum_i a_i^* b_i$. This allows computing the overlap of $|A\rangle$ and $|B\rangle$ with additive error ϵ.

3.1. Gradient descent

Gradient descent can be used to find the least squares solution to linear regression problems of the form $A\vec{w} = \vec{b} + \vec{v}$, where A is an $n \times m$ matrix representing a dataset of n m-dimensional examples, $\vec{b}$ is the expected outputs associated with each example, $\vec{w}$ is the vector of true parameters, and v is a Gaussian noise variable. We define a mean squared error loss function as $\mathcal{L} = \frac{1}{2n}\sum_{p=1}^{n}(\hat{b}^{(p)} - b^{(p)})^2$, where $\vec{\hat{b}}$ is the output of our linear model based on an estimate of the parameters: $\vec{\hat{b}} = A\vec{w^*}$. Applying gradient descent to the loss function, we get the parameter update for each iteration as

$$\vec{w^*} := \vec{w^*} - \eta \times \frac{1}{n}\sum_{p=1}^{n}(\vec{a}^{(p)} \cdot \vec{w^*} - b^{(p)})\vec{a}^{(p)} \quad (5)$$

where $\cdot$ is the vector dot product. Over many iterations, $\vec{w^*}$ is guaranteed to converge to the global minimum of the mean squared error loss function provided that the learning rate is less than $2/L$, where L is the Lipschitz constant of the loss gradient. More specifically, the learning rate hyperparameter η plays a key role in controlling the convergence rate. This learning rate is fundamental to the efficiency of the gradient descent algorithm. A large learning rate may result in random exploration that misses the absolute minimum, while small learning rates will not only make the search slow, but can also get stuck in local minima for non-convex loss functions. For that reason, many gradient-based optimization algorithms employ learning rates that adapt based on the local topography of the loss landscape and/or the trajectory of the solution estimate. For example, methods such as momentum, adagrad, RMSprop, and second order techniques like Newton's method are able to speed up convergence of gradient-based optimization [11]. In this work, we show that convergence of gradient descent can be further enhanced by leveraging knowledge of the Euclidean distance from a particular test point to the solution.

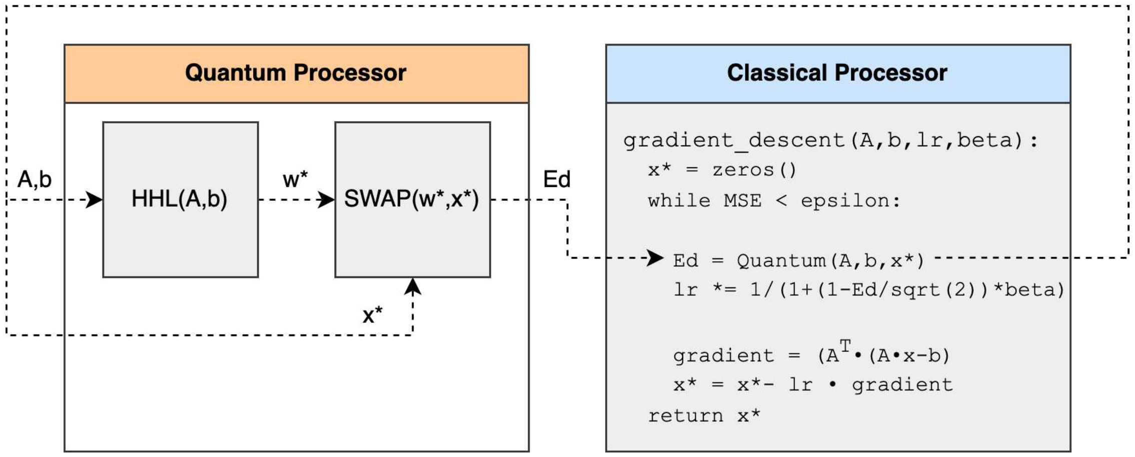
3.2. Quantum Adaptive Learning Rate

Unique to the approach in this paper is that the learning rate is defined based on true knowledge of the problem's solution. To aid the gradient descent, the SWAP test distance between the estimation $\vec{w}^*$ 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 close 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.

In this approach, the learning rate, η, is updated following the expression:

$$\eta_{n+1} = \frac{\eta_n}{1 + (1 - \frac{Ed}{\sqrt{2}}) * \beta} \quad (6)$$

where Ed is the Euclidean distance determined by the SWAP test and β is a user modifiable parameter. This formula works similar to adaptive learning rates using time based decay [11]. The Euclidean distance is divided by the square root



```
gradient_descent(A,b,lr,beta):
  x* = zeros()
  while MSE < epsilon:
    Ed = Quantum(A,b,x*)
    lr *= 1/(1+(1-Ed/sqrt(2))*beta)

    gradient = (A^T•(A•x-b)
    x* = x*- lr • gradient
  return x*
```

Figure 3: Pseudo-code representation of the quantum adaptive learning rate (lr in the code). The hybrid Quantum-Classical model for the gradient descent algorithm is represented, separating the quantum steps from the classical ones.

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 with the Euclidean distance. To reduce the number of SWAP tests used, the learning rate can be updated periodically instead of every iteration. Figure 3 illustrates the hybrid model, separating classical from quantum computations.

4. HHL limitations: encoding the eigenvalues

As explained in Section 2.1, and in previous work [3], HHL is a powerful quantum algorithm, however, it also has significant limitations. Section 2.1 explained how the matrix should be well-conditioned, as reflected by $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$. This is inherent to the matrix. But, further, the eigenvalues have an impact on the overall error depending on the size of the register that encodes them. This section explores the limitations associated with the eigenvalues of the matrix representing the system of linear equations. To this end, implementations were tested under noiseless simulation using IBM's *Qiskit* tools [12]. The focus of these experiments is to explore the inherent difficulties of the algorithm itself, and their impact on accuracy, rather than the challenges of the physical implementation. Since quantum circuit results are presented 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.

4.1. Model under ideal conditions

To demonstrate the combined HHL and SWAP test circuit, the following standard system with well defined eigenvalues (8, 4, 1, and 2) was tested:

$$\frac{1}{4}\begin{bmatrix} 15 & 9 & 5 & -3 \\ 9 & 15 & 3 & -5 \\ 5 & 3 & 15 & -9 \\ -3 & -5 & -9 & 15 \end{bmatrix} \vec{x} = \frac{1}{2}\begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \end{bmatrix} \quad (7)$$

This is a standard first step in testing HHL, but in this case, the intention is to examine the error in the Euclidean distance measurement to a test vector. It should be noted that to be able to measure this distance accurately in the computational basis, the solution should be in the $\mathbb{R}$ realm. Given that the eigenvalues are well defined — known beforehand, easily invertible powers of two—, the system of equation's solution is exactly known, free of error. The circuit used 16 qubits and had a depth of 888 quantum steps. The results of the Euclidean distance calculation and its error are shown in Table 1, calculating the distance in two instances: the solution vector against itself, and the solution vector against a random test vector (first and second rows respectively). The distance is shown for each one of these. The error is calculated as the L2 error between the measured output of the SWAP test and the true output containing the Euclidean distance between the normalized solution and test point. The distance error shows that the Euclidean distance is exactly correct, (no error) when measured against the solution of the problem. In this case the distance is 0 with error 0. However, the second case with some random test vector, there is an small error of approximately of 0.0056 over the normalized state vector output of the SWAP test. Since the experiments are done through noiseless simulation, this error is only due to the sampling out of the state vector, and a higher number of shots will further reduce this small error.

Table 1: Ideal matrix swap test euclidean distance error to two test vector cases, the normalized exact solution (first row), and a random test vector (second row). The distance error is measured against the normalized ideal SWAP test output.

Test Vector	Euclidean Distance	Distance Error
0.8590, 0.2342, 0.2342,0.3904	0.0	0.0
0.5,0.5,0.5,0.5	0.5312	0.0056

4.2. Number of qubits for Quantum Phase Estimation

One significant source of error in the HHL algorithm is in the quantum phase estimation (QPE) stage. In this circuit's 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. The number of qubits for phase estimation, in turn, has a critical impact on the width and depth the circuit. Either way, not knowing what the eigenvalues are, one can only guess how many qubits are needed for their quantum phase estimation. The objective of this section is to evaluate the impact of this parameter on the general HHL implementation.

To manually test the impact of QPE register size, experiments using different sizes of quantum register were simulated to determine its effect 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:

$$A_1 = \begin{bmatrix} 1.25 & 0 \\ 0 & 1 \end{bmatrix} A_2 = \begin{bmatrix} 1.125 & 0 \\ 0 & 1 \end{bmatrix} \quad (8)$$

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 the HHL statevector solution were recorded in Table 2.

Table 2: Error depending on the quantum phase estimation register size, for matrices A1 and A2 as represented in Equation 8. The distance error is measured against the normalized ideal SWAP test output.

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

As expected, unless there are enough qubits to perfectly represent the matrix's eigenvalues in the phase encoding, there will be error associated to both the distance and the state vector. Once the number of qubits is increased such that the eigenvalues can be exactly encoded, there is no longer error caused by the HHL circuit.

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

$$\begin{bmatrix} 2.5 & 3 \\ 1 & 4 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (9)$$

which was modified to be Hermitian.

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

These eigenvalues cannot be exactly encoded with a reasonable amount of qubits. The HHL algorithm uses a scaling factor on the eigenvalues of $s = 1.26436$ in this case, which resulted in post scaling eigenvalues equal to

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

This approach allows for *some* of the eigenvalues to be accurately represented, thus improving the chances of accurate final results. Even post scaling, a significant number of qubits would be required to perfectly represent the eigenvalues in this case. Experiments were run increasing the number of qubits used in the 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 distance through the SWAP test are recorded in Table 3.

Table 3: Error depending on the quantum phase estimation register size, for non-trivial eigenvalues. The distance error is measured against the normalized ideal SWAP test output.

Register Size	Distance 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 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. since additional qubits 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.

While increasing the register size may improve the results, it will also have a negative impact on the width and depth of the circuit. This means selecting the size of the quantum phase estimation register is not a trivial step in utilizing the HHL algorithm.

4.3. HHL applied to the Iris data set

The Iris data set is a simple data set often used to test classification in machine learning.

Figure 4: Iris data set visualization.

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

$$\begin{bmatrix} 5.2 & 6.6 & 5.8 & 6.3 \\ 2.7 & 2.9 & 2.8 & 2.9 \\ 3.9 & 4.6 & 5.1 & 5.6 \\ 1.4 & 1.3 & 2.4 & 1.8 \end{bmatrix} \vec{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{10}$$

which was altered to be Hermitian as explained in Section 2.1 before performing the HHL algorithm. The resulting matrix used was an eight by eight matrix with the eigenvalues

λ = [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 eight qubits for the QPE counting register to represent the eigenvalues. This resulted in a quantum circuit with a depth of 34,200 steps and 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 4: Error With and without eigenvalue scaling. The distance error is measured against the normalized ideal SWAP test output.

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

The results in Table 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. Even with scaling, and with a significant reduction, the error is still over 0.36 out of a normalized SWAP state vector, possibly too high to derive useful information in a real application. 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 4.2, 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. But this would further increase the size and resources necessary for the implementation. As is shown in Section 5, this is still useful information when used in conjunction with QALR.

5. Quantum Adaptive Learning Rate Results

The ideal matrix, Equation 7, and Iris matrix, Equation 10, were tested in fully simulated experiments executing the hybrid gradient descent model. Values for β and the number of iterations between SWAP tests were assumed to be adequate based on the rage of all other parameters, but more work can be done in the future to tune these to each specific problem.

Classical gradient descent was performed on the *ideal matrix* with a learning rate (lr[1]) 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 are recorded in Figure 5.

[1] lr used from here on for consistency with the algorithm and figures

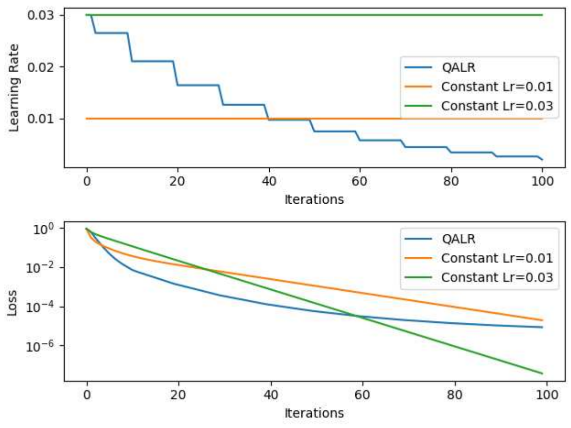


Figure 5: Learning Rate (lr) and Loss vs Iterations (Ideal Matrix)

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 that with the quantum adaptive learning 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 4, even with scaling enabled, the number of qubits used in the QPE stage was increased from 8 to 12.

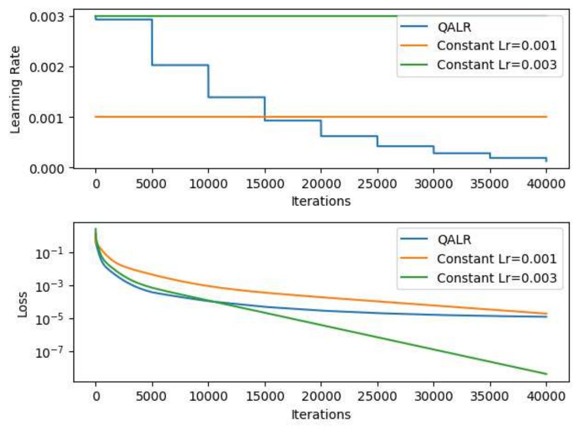


Figure 6: Learning Rate (lr) and Loss vs Iterations (Iris Matrix)

The results in Figure 6 resemble those in the previous experiment. The loss drops lower and faster than the constant learning rate meaning a lower loss can 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.

5.1. Circuit and problem complexity

The HHL circuit is kept from previous works [7], [8] and no notable changes were made to this part of the circuit. 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 [8] explored the cost of the SWAP test. Experiments demonstrated that with a carefully selected test point, the SWAP test was not computationally expensive. Even though the SWAP test is not computationally expensive, efforts must be made to mitigate the number of SWAP tests. This was explored in this work by limiting the SWAP test to select iterations during gradient descent, and the possibility of using a SWAP test with reduced Toffoli gates when the quantum states are no longer needed, since the swapped states are not necessarily preserved in this implementation.

6. Conclusions

HHL is a powerful but limited and highly conditioned quantum algorithm. This paper looks at some of these limitations, and uses HHL as a way of extracting partial information on the solution to the training problems of neural networks. 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.

The HHL quantum algorithm is highly dependent on several parameters, such as the sparsity and conditioning of the system's matrix to mention two. This paper has looked at the impact the of quantum phase estimation register's size on the accuracy of the results. Improving accuracy would come at the cost of a more complex circuit that requires more resources. But that is if the range of the eigenvalues were known, which will not be, and one can only estimate the best size for this register.

On future work, it is possible to use linear algebra tools to help estimate upper and lower bounds for the eigenvalues, without compromising the computational complexity of overall solution. The β step size and the number of iterations between SWAP tests should also be further evaluated.

There is a number of classical adaptive learning rate methods that could potentially outperform the method proposed here. However, classical adaptive learning rates typically have additional computational overhead, and that requires of the tuning/storage of extra hyperparameters. For example, the *momentum* technique introduces a new momentum parameter that needs to be set carefully to avoid oscillatory or divergent behavior. Other approaches, like *AdaGrad* , require tuning and storage of learning rates for every model parameter, which could be a significant overhead for large model sizes. Finally, *second-order* techniques require the computation of the Hessian matrix which is often not feasible for very large model sizes [13]–[15]. Quantum adaptive learning rate (QALR) has its own difficulties, but makes an adaptive decision based on certain knowledge of the optimal solution. A fair question to ask is whether it is worth enlisting a quantum computer to solve this problem, instead of paying the price of additional classical computational resources and complexity. This work does not evaluate the computational complexity of the problem as a whole (classical+quantum mechanical), which is also left for a theoretical analysis of these algorithms in the future.

Acknowledgments

The work of Sonia López Alarcón is supported by the National Science Foundation under Award No. 2300476.

■ REFERENCES

1. A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum Algorithm for Linear Systems of Equations,” *Physical Review Letters*, vol. 103, no. 15, Oct 2009.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
3. S. Aaronson, “Read the Fine Print,” *Nature Physics*, vol. 11, pp. 291–293, 04 2015.
4. E. Tang, “A Quantum-Inspired Classical Algorithm for Recommendation Systems,” in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 217–228. [Online]. Available: https://doi.org/10.1145/3313276.3316310
5. T. Schmale, M. Reh, and M. Gärttner, “Efficient Quantum State Tomography with Convolutional Neural Networks,” *npj Quantum Information*, vol. 8, no. 1, p. 115, Sep. 2022. [Online]. Available: https://www.nature.com/articles/s41534-022-00621-4
6. M. Cerezo, K. Sharma, A. Arrasmith, and P. J. Coles, “Variational Quantum State Eigensolver,” *npj Quantum Information*, vol. 8, no. 1, pp. 1–11, Sep. 2022. [Online]. Available: https://www.nature.com/articles/s41534-022-00611-6
7. Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost, and P. Wittek, “Bayesian Deep Learning on a Quantum Computer,” *Quantum Machine Intelligence*, vol. 1, pp. 41–51, 2019.
8. S. Lopez 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,” *IEEE International Conference on Computer Design (ICCD’22)*, Oct. 2022, arXiv:2210.12707 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2210.12707
9. G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: theory and applications,” *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.
10. M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.
11. I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
12. Qiskit contributors, “Qiskit: An open-source framework for quantum computing,” 2023.
13. H. Yu and B. M. Wilamowski, *Neural Network Training with Second Order Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 463–476. [Online]. Available: https://doi.org/10.1007/978-3-642-23172-8_30
14. G. S. Na, “Efficient learning rate adaptation based on hierarchical optimization approach,” *Neural Networks*, vol. 150, pp. 326–335, Jun. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608022000478
15. A. Senior, G. Heigold, M. Ranzato, and K. Yang, “An empirical study of learning rates in deep neural networks for speech recognition,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Vancouver, BC, Canada: IEEE, May 2013, pp. 6724–6728. [Online]. Available: http://ieeexplore.ieee.org/document/6638963/