A REVIEW OF QUANTUM MACHINE LEARNING

FARIS SBAHI

1. Introduction

Machine learning explores the study and construction of algorithms that are capable of finding patterns in data. Hence, this entails studying the ability of learning models to capture these patterns and generalize, as one would in statistical learning theory, in addition to studying model efficiency, enabled by computational complexity theory. In practice, many of the widely used supervised and unsupervised machine learning models can reasonably be described as linear algebraic data analysis techniques. For example, consider that ordinary least squares regression and ridge regression can be given in terms of a closed-form product of matrices and matrix inverses and that principal component analysis is essentially solving for the largest-eigenvalue eigenvectors of a particular covariance matrix. Furthermore, optimization problems, which are common in machine learning, can often be broken down into a set of basic linear algebraic subproblems.

Since its conception, quantum computation and quantum information has taught us to "think physically about computation" [?]. Well, if quantum mechanics tells us that physical states are mathematically linear algebraic objects (vectors in a Hilbert space), then perhaps this lesson says that the type of computation best suited for quantum physical reality are linear algebraic problems, such as the machine learning ones noted above. This somewhat naive intuition turns out to have value, as we will show in this review.

Hence, this leads us to the idea of quantum machine learning which uses quantum algorithms as part of a larger implementation to outperform the corresponding classical learning algorithms.

Nevertheless, past linear algebraic analysis techniques, there exist other classes of machine learning algorithms such as deep learning built on artificial neural networks and reinforcement learning which models an environment as a Markov decision process [?]. Successes have been achieved in terms of (potential) quantum speedups in these arenas [?], but we won't explore these alternative machine learning routes further in our review.

In this review, we will cover the main theoretical results in terms of quantum algorithms relevant to quantum machine learning. Then, we'll describe specific learning problems that have achieved quantum speedups using these results. Finally, we'll discuss important limitations to these results which may pose substantive challenges to the future of quantum learning theory.

Our goal is to provide a birds-eye view of these concepts and refer the reader to appropriate references where they desire greater detail. We assume that the reader is familiar with the fundamentals of quantum information theory e.g. by means of Chapter 2 of [?].

2. Comparing Machine Learning Performance

If we are to claim that some machine learning algorithms perform better on a quantum computer, we first must decide on a notion of "outperforming".

This is currently characterized by the advantage in runtime obtained by a quantum algorithm over the classical methods for the same task. We quantify the runtime with the asymptotic scaling of the number of elementary operations used by the algorithm with respect to the size of the input, as one does in complexity theory.

The definition of an elementary operation is dependent on the choice of measure of complexity. Query complexity measures the number of queries to the information source for the classical or quantum algorithm. Hence, a quantum speedup results if the number of queries needed to solve a problem is lower for the quantum than for the classical algorithm [?].

3. Speedup Techniques

3.1. **Grover's Algorithm and Amplitude Amplification.**

Suppose we wish to search a space of elements of size N. Clearly this problem is $\Omega(N)$, classically. Rather than searching the elements directly, we can focus on "searching" their indices which are labelled $[0, N-1]$ by using an oracle. So, let $x \in [0, N-1]$ and $|q\rangle$ be an ancillary qubit. Define $f(x) = 1$ if the index is the index of the solution and $f(x) = 0$ otherwise. An oracle is a unitary operation, O, which acts on the computation basis as

$$|x\rangle |q\rangle \rightarrow^{O} |x\rangle |q \oplus f(x)\rangle$$

Hence, if we let $|q\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle$, then we can rewrite this transformation as

$$|x\rangle |-\rangle \rightarrow^{O} (-1)^{f(x)} |x\rangle |-\rangle$$

Grover [?] came up with a quantum algorithm that finds a solution with high probability using $O(\sqrt{N})$ oracle queries (which is known to be optimal on a quantum computer [?]).

For $N = 2^n$, the grover iteration can be given by the linear transformation $G = (2\left|\psi\right\rangle\left\langle\psi\right| - I)O$ where $\left|\psi\right\rangle$ is the equally weighted superposition of states.

So, assuming that the number of solutions $M = 1$, Grover's algorithm essentially prepares N qubits in state $\left|\psi\right\rangle\left|-\right\rangle$ and then applies G for $\frac{\pi}{4}\sqrt{N}$ iterations.

However, if the number of solutions $M \geq 1$ is unknown, then a variant, known as amplitude amplification [**?**], can be used to find a solution in the solution subspace with high probability using $O(\sqrt{N/M})$ queries.

3.2. **Solving Systems of Linear Equations.** Solving linear systems of equations is a ubiquitous problem in machine learning. As we will discuss, many learning problems, such as least-squares regression and least-squares SVMs, require the inversion of a matrix. Hence, we will describe two common quantum algorithms which lead up to the recent HHL algorithm, named after the algorithm's authors, Harrow, Hassidim, and LLoyd.

3.2.1. *Quantum Fourier Transform.* In the following, we take $N = 2^n$, where n is some integer. Hence, the quantum Fourier transform on the orthonormal computational basis $\{\left|0\right\rangle, \cdots, \left|N-1\right\rangle\}$ for an n qubit quantum computer is defined to be a linear operator with the following action on the basis states,

$$\left|j\right\rangle \to \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} \left|k\right\rangle$$

The gate complexity on a quantum computer is $O(n^2)$ as opposed to $O(n2^n)$, classically, using the technique in [**?**]. Hence, at first glance, the potential quantum speedup via this subroutine is phenomenal for many machine learning and data analysis tasks. For example, speech recognition preprocessing begins with a Fourier transform of the digitized sound.

However, there is no way of determining the Fourier transformed amplitudes of the original state. Hence, the applications are more subtle than one may have expected.

3.2.2. *Phase Estimation.* Suppose a unitary operator U has an eigenvector $\left|u\right\rangle$ with eigenvalue $e^{2\pi i\varphi}$, where the value of φ is unknown.

The phase estimation algorithm uses two registers: one of t qubits in the equal superposition state and another which stores $\left|u\right\rangle$. Then, after a series of controlled-U^{2^j} operations on the jth qubit and an inverse Fourier transform, the first register holds the state $\left|\varphi_1 \cdots \varphi_t\right\rangle$, which is an approximation of φ, whose accuracy is dependent on the size of t.

Hence, the complexity of this algorithm is essentially that of the inverse Fourier transform, $O(t^2)$. This assumes that each controlled-U^{2^j} operation is given by an oracle, which may not hold in practice. Furthermore, we also assume that we can prepare $|u\rangle$ efficiently, which also may not hold in practice. Hence, we often require workarounds to these problems in our applications of Phase Estimation.

3.2.3. *HHL Algorithm.* One such application of Phase Estimation is with respect to solving linear systems of equations. This is the so-called HHL algorithm [**?**]. Here, we will cover the essential details of the algorithm.

The general problem statement of a linear system is if we are given matrix A and unit vector $\vec{b}$, then find $\vec{x}$ satisfying,

$$A\vec{x} = \vec{b}$$

However, assume that instead of solving for x itself, we instead solve for an expectation value $x^T M x$ for some linear operator M. The original description of the algorithm provides runtime bound of $\tilde{O}(\log(N)\kappa^2 s^2/\epsilon)$, where s is sparsity measured by the maximum number of non-zero elements in a row and column of A, κ is the condition number, and ϵ is the approximation precision. Hence, we can only achieve speedup if the linear system is sparse and has a low condition number κ. Ambainis [**?**] and Childs [**?**] improved this depency on κ and s to linear and ϵ to poly-logarithmic.

This compares well considering that the best classical algorithm has a runtime of $O(N^{2.373})$ [**?**]. However, due to the large amount of pre-processing required, the algorithm is not used in practice. Standard methods, for example, based on QR-factorization take $O(N^3)$ steps [**?**].

So, assume that A in our linear system is an $N \times N$ Hermitian matrix. Notice that this is an "unrestrictive" constraint on A because we can always take non-Hermitian matrix A' and linear system $A'\vec{x} = \vec{b}$ and instead solve $\begin{bmatrix} 0 & A' \\ A'^{\dagger} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$. Hence, we we will assume that A is Hermitian from here on.

Recall that because A is hermitian, then we can perform quantum phase estimation using e^{-iAt} as the unitary transformation. This can be done efficiently if A is sparse.

So, we first prepare $|b\rangle = \sum_i b_i |i\rangle$ (the representation of $\vec{b}$). We assume that this can be done efficiently or that $|b\rangle$ is supplied as an input.

Denote by $|\psi_j\rangle$ the eigenvectors of A with associated eigenvalues λ_j. Hence, we can express $|b\rangle$ as $|b\rangle = \sum_j \beta_j |\psi_j\rangle$. So, we initialize a first register to state $\sum_j \beta_j |\psi_j\rangle$ and second register to state $|0\rangle$. After applying phase estimation, we then have the joint state $\sum_j \beta_j |\psi_j\rangle \left|\widetilde{\lambda}_j\right\rangle$, where $\widetilde{\lambda}_j$ is an approximation of λ_j. We'll assume that this approximation is perfect from here on, for the sake of demonstration.

Next we add an ancilla qubit and perform a rotation conditional on the first register which now holds $|\lambda_j\rangle$. The rotation transforms the system to

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

for some small constant $C \in \mathbb{R}$ that is $O(1/\kappa)$.

Hence, we can undo phase estimation to restore the second register to $|0\rangle$.

One sees that if we measure the ancillary qubit in the computational basis, we'll evidently collapse the state to $|1\rangle$ with probability $\Omega(1/\kappa^2)$. However, using amplitude amplification this can be upper-bounded to $O(1/\kappa)$.

Finally, we can make a measurement M whose expectation value $\langle x| M |x\rangle$ corresponds to the feature of x we wish to evaluate.

The concessions we noted along the way clearly limit the algorithm's applicability to practical problems. We have three essential caveats to achieving exponential speedup: (1) A must be sparse and have a condition number that scales at most sublinearly with N, (2) $|b\rangle$ must be loaded in quantum superposition in $\log(N)$ time, and (3) $|x\rangle$ isn't actually be read out, but instead an expectation is computed.

Limitation (1) has been partially resolved by the work of [**?**] to achieve a quadratic speedup for dense matrices.

Limitation (2) may be solved by quantum RAM, which then has its own limitations discussed in the next section.

Limitation (3) is a general issue with regards to reading out classical information from a quantum state at the conclusion of a quantum algorithm because we would need at least N measurements to retrieve this classical data. Hence, this would eliminate the potential for exponential speed-up.

Hence, after observing these limitations we may wonder if there can exist a classical algorithm with the same caveats that can achieve the same runtime. Importantly, in the original paper the authors showed that HHL is universal for quantum computation. Hence, we can encode any quantum algorithm e.g. Shor's algorithm into a system of roughly 2^n linear equations in 2^n unknowns, and then use HHL to solve the system (i.e., simulate the algorithm) in polynomial time. Thus, provided we believe any quantum algorithm achieves an exponential speedup over the best possible classical algorithm, HHL can theoretically achieve such a speedup as well [**?**].

To conclude, HHL is a logarithmic time quantum algorithm for matrix inversion, a task arising in many learning problems. However, a number of caveats that include the requirement of a logarithmic access time to the memory and the impossibility of retrieving the solution vector with one measurement lead to the question of whether classical or parallel algorithms that make use of the same assumptions obtain similar, or better, runtimes in practice. Hence, we will need empirical data to further address this question.

3.3. **Quantum Random Access Memory.** The essence of machine learning is analyzing a vast amount of data. Hence, we must address the question of how classical data is encoded in quantum superposition, a concern brought up in our previous discussion of the matrix inversion algorithm.

So, assume that we have an classical vectors $\{v_1, \cdots, v_n : v_1 \in \mathbb{R}^m\}$ that need to be encoded for use as part of a quantum algorithm. Quantum random access memory (qRAM) can encode these classical vectors in superposition into $\log(nm)$ qubits in $\log(nm)$ time using its "bucket-brigade architecture"[**?**]. The basic idea is to use a three-level tree structure where the nm qubit "leaves" contain the entries of the n vectors in $\mathbb{R}^m$.

One of the central limitations of qRAM is that the number of resources scales as $O(nm)$ i.e. exponentially in the binary representation of n, m. There have been open questions of the viability of this model of memory, in practice. In particular, if each of the qubits must be error-corrected, then it seems entirely impractical for general use. Some of this concern has been answered by proponents who have showed that, given a certain error model, algorithms that require to query the memory a polynomial number of times (e.g. the HLL algorithm above) might not require fault-tolerant components. Still, the amplitude amplification algorithm above does require this error correction.

Furthermore, it may be only fair to compare qRAM to a parallel classical architecture, given that we are allowing resources to scale exponentially.

Considerable, too, is the fact that data must be distribute fairly uniformly over the quantum register or else qRAM is no longer efficient [**?**].

In conclusion, qRAM comes with a significant number of considerations in itself and hence should be subjected to empirical investigation to determine its genuine quantum speed-up in practice.

4. Applications

4.1. **Principal Component Analysis.** First, we consider the ubiquitous principal component analysis (PCA) algorithm. PCA reduces the dimensionality of data by transforming the features to uncorrelated weightings of the original features ("principal components"). This requires simply finding the eigenvalues and eigenvectors of the data covariance matrix.

Hence, let $v_j \in V$ where V is a d-dimensional vector space such that $d = 2^n = N$ and assume that $|v_j| = 1$ for simplicity. Hence, the covariance matrix of the data is given by $C = \sum_j v_j v_j^T$ whose eigenvalues and eigenvectors we seek to discover. So, suppose we can prepare quantum state $v_j \rightarrow |v_j\rangle$, choosing classical data vector v_j uniformly at random. For example, we can use qRAM, discussed above. Then, because of our random selection, the resulting quantum state has density matrix $\rho = \frac{1}{N}\sum_j |v_j\rangle \langle v_j|$ which we observe to be the covariance matrix, up to an overall factor.

Then, as Lloyd, Mohseni and Rebentrost describe [**?**], one way to perform quantum PCA use that given n copies of ρ we have the ability to apply the unitary operator $e^{-i\rho t}$ to any state σ with accuracy $O(t^2/n)$. This can be done by using repeated infinitesmal applications of the SWAP operator on $\rho \otimes \sigma$.

Hence, using $e^{-i\rho t}$, we can perform phase estimation on *rho*. So, let $|\psi_i\rangle$ be the eigenvectors of ρ with associated eigenvalues λ_i and so $\rho = \sum_j \lambda_j |\psi_j\rangle \langle\psi_j|$. Therefore, phase estimation gives u

$$\sum_j \lambda_j |\psi_j\rangle \langle\psi_j| \otimes \left|\tilde{\lambda_j}\right\rangle \left\langle\tilde{\lambda_j}\right|$$

where $\tilde{\lambda_j}$ is an approximation of λ_j. Hence, if we measure the second register and repeat this procedure several times, the most frequent outcome is the first principal component $|\psi_{max}\rangle$, and so on.

The algorithm works best when a small number of principal components have significantly greater eigenvalues than the rest. Hence, ρ is well represented by the subspace spanned by the principal components. In particular, let $m << d$ be the dimension of the subspace of principal components which represent ρ within some bound. Then, the algorithm has runtime $O(m \log n)$.

Hence, this gives a limitation: if the eigenvalues of the covariance are roughly equal and of size $O(1/d)$, then the algorithm reduces to scaling in time $O(d)$ (as does the classical algorithm). Furthermore, we evidently inherit the limitations of qRAM, if we use it for the state preparation above.

4.2. **Support Vector Machines.** Support vector machines seek to find an optimal (maximum margin) separating hyperplane between two classes of data in a dataset. Hence, if the training data is linearly separable, each class can be found on only one side of the hyperplane. In particular, let $\{x_1, \cdots, x_n\}$ be a set of observations where x_i is a d-dimensional vector with associated labels $y_i \in \{0, 1\}$. Hence, we seek a linear function $f(x) = \vec{w} \cdot \vec{x} - \vec{b}$ where $\vec{w}$ is a weight vector and $\vec{b}$ is a bias vector s.t. $y_i f(x_i) > 0, \forall i$. Furthermore, we seek to maximize the minimum $y_i f(x_i)$, across all i.

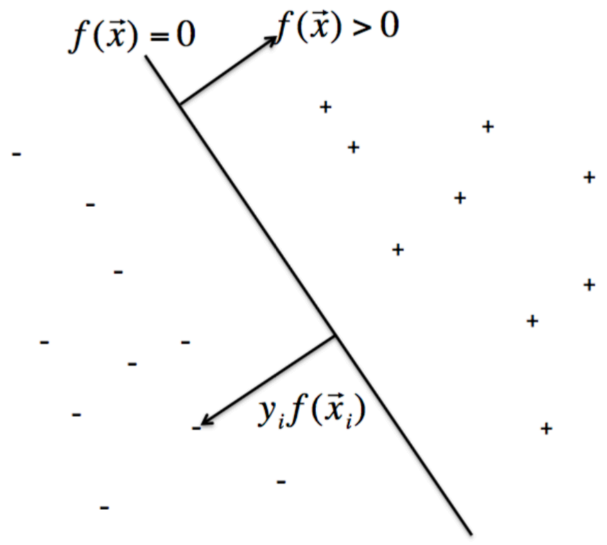

FIGURE 1. Visualization of a separating hyperplane and its associated margin taken from [?].

Much of the appeal of SVM is in its generalization to nonlinear hypersurfaces by replacing the constituent inner products with a kernel that implicitly maps the observations to another high dimensional space [?].

Solving SVM generally is performed by optimization of the dual problem which is a quadratic optimization problem. Hence, in [?], the authors show that quantum evaluation of inner products leads to exponential speed-ups in terms of d.

Full exponential improvements, however, can be achieved (with respect to d and n) in the case of least-squares SVMs [?]. In this case, finding a solution requires optimizing a least squares minimization. Hence, this type of minimization reduces to solving a set of linear equations. So, after data has been inputted efficiently (perhaps using qRAM), a modified version of the matrix inversion algorithm above can be used, incorporating methods from the PCA algorithm above to prepare the linear system.

The key point is, all the operations required to construct the maximum-margin separating hyperplane and to test whether a vector's classification can be performed in time that is polynomial in $\log n$.

4.3. **Gaussian Processes.** In [?], the authors demonstrate an application of HHL algorithm with respect to Gaussian process regression (GPR), another supervised learning method. In effect, GPR is a stochastic generalization of ordinary regression. In particular, let $\{x_1, \cdots, x_n\}$ be a set of observations where x_i is a d-dimensional vector with associated scalar targets y_i. In GPR, we consider the distribution over latent functions $f(x)$ which can return the correct labelling, y, given an inputted data-point. However, we further assume that this labelling is subject to Gaussian noise i.e. $f(x) = y + \epsilon$, where ϵ is the Gaussian noise. Hence, GPR uses Bayesian inference to return this distribution of latent functions such that they are consistent with the observed data.

Now, given this distribution and some test input x', we can predict its output by computing (1) a linear predictor known as the predictive mean and (2) the variance of the predictor with respect to x'. These values then give the distribution the y' that is consistent with training data. Hence, the quantum speedup comes from the fact that both of these values can be computed using a modification of the HHL algorithm, as described in [**?**]. Furthermore, the size of the output is independent from the size of the dataset which the authors use to show that, combined with the efficiency of HHL, can given exponential speedups in terms of d. Again, we require that the data can be loaded in efficiently initially e.g. using qRAM.

4.4. **Optimization.** Unsurprisingly, optimization methods are fundamental to many machine learning algorithms. Again, we can apply our quantum set of tools to several important optimization problems, including semi-definite programs (with potential super-polynomial speedups) and constraint satisfiability programs. Even easier to see, we can directly use our demonstrated results to solve quadratic programming problems which reduce to a linear system. If $N \times N$ matrices that define the linear system are sparse and low rank, we can use HHL and yield a solution in $\log N$ time–an exponential speedup.

Furthermore, iterative optimization, such as by means of gradient descent can be implemented by modifying PCA. In this case, several copies of a quantum state representing the solution are used to iteratively improve the solution. We may expect improvements in this type of optimization to lead to improvement in training neural networks on a quantum computer.

Interestingly, the quantum optimization algorithm finds approximate solutions for combinatorial optimization by "alternating qubit rotations with the application of the problem's penalty function" [**?**].

4.5. **Topological Data Analysis.** By now, we've seen that the presented algorithms suffer from a common limitation: the classical data must be loaded into a quantum superposition efficiently given that the speedups come from performing the computational steps after data is loaded in. However, this issue can be avoided in cases where the data points can be computed efficiently individually, as opposed to loading all of a large dataset in. In terms of machine learning, this can happen when the computation component of the algorithm requires exploring a comparatively small subset of the original dataset. In other words, if we can explore a subset of the input data to determine the distribution and other descriptive features of the overall data, we can then have the quantum algorithm generate the combinatorially larger space in quantum parallel, thereby computing the quantum dataset efficiently. This idea was used in the context of topological data analysis in [**?**].

Topological features, in particular, seem promising for this goal given that they are independent of the metric of choice and hence capture essential features of the data. In a discrete set, topological features are given by features that persist across spatial resolutions. These persistent features, formalized through persistent homology, are less likely to be artifacts of noise or parameterization. For example, the holes, voids, or connected components are examples of such features. The number of these types of features are defined for simplical complex (roughly a closed set of simplexes) as "Betti numbers".

In the paper, the authors show how to generate quantum states to encode the simplexes using logarithmically fewer qubits. Furthermore, they show that the Betti numbers can be efficiently represented using this representation. Important assumptions were made, in particular such that the quantum state encoding the simplexes can be generated efficiently. One satisfying condition is if the number of simplexes in the complex are exponentially large. Hence, in at least some cases, we may extract exponential speed-ups in this type of topological analysis.

5. Challenges

The first obvious challenge to quantum machine learning is that the execution of quantum algorithms requires quantum hardware that does not exist at present.

Aside from this, as we've seen by now, many of the quantum algorithms which are used to potentially give quantum speedups come with several caveats and limitations to their use. As noted in [**?**], implicitly in [**?**], and repeatedly elsewhere, we can condense the general caveats into a few fundamental problems:

(1) The Input State Preparation Problem. The cost of reading in the input can in some cases dominate the cost of quantum algorithms. In many cases, we hoped qRAM would solve this problem. Understanding this factor is an important ongoing challenge.

(2) The Readout problem. Retrieving the solution in terms of classical information after performing quantum algorithms requires learning an exponential number of bits through repeated trials. We've shown that learning an expectation value of a linear operator can be a sound alternative (e.g. in the case of matrix inversion).

(3) The true cost problem. We require empirical study to show the number of gates or elementary operations to carry out a quantum algorithm, in practice. For example, we encountered this issue in the fundamental phase estimation algorithm (i.e. is it fair to assume that a U^{2^j} gate is an elementary operation?). Nevertheless, our query complexity bounds seem to indicate great advantages in specific cases, as we've noted.

Throughout this review, we've considered learning classical data which introduces problems (1) and (2), given that we have to interface between classical and quantum states. Hence, we may consider the case of applying quantum algorithms to quantum data. This is covered in detail in [**?**].

[19] Seth Lloyd. Quantum algorithm for solving linear systems of equations. In *APS March Meeting Abstracts*, 2010.

[20] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data. *Nature communications*, 7:10138, 2016.

[21] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.

[22] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631, 2014.

[23] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *arXiv preprint arXiv:1804.09139*, 2018.

[24] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[25] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[26] Cynthia Rudin. Support vector machines. *Duke Course Notes*.

[27] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2):022342, 2016.

[28] Changpeng Shao. Reconsider hhl algorithm and its related quantum machine learning algorithms. *arXiv preprint arXiv:1803.01486*, 2018.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[30] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505, 2012.

[31] Leonard Wossnig, Zhikuan Zhao, and Anupam Prakash. Quantum linear system algorithm for dense matrices. *Physical review letters*, 120(5):050502, 2018.

[32] VU Thi Xuan. Cr04 report: Solving linear equations on a quantum computer. 2016.

[33] Zhikuan Zhao, Vedran Dunjko, Jack K Fitzsimons, Patrick Rebentrost, and Joseph F Fitzsimons. A note on state preparation for quantum machine learning. *arXiv preprint arXiv:1804.00281*, 2018.

[34] Zhikuan Zhao, Jack K Fitzsimons, Michael A Osborne, Stephen J Roberts, and Joseph F Fitzsimons. Quantum algorithms for training gaussian processes. *arXiv preprint arXiv:1803.10520*, 2018.

[35] Yarui Zheng, Chao Song, Ming-Cheng Chen, Benxiang Xia, Wuxin Liu, Qiujiang Guo, Libo Zhang, Da Xu, Hui Deng, Keqiang Huang, et al. Solving systems of linear equations with a superconducting quantum processor. *Physical review letters*, 118(21):210504, 2017.