Improving ML Algorithmic Time Complexity Using Quantum Infrastructure

Aayush Grover

January 4, 2023

Contents

Abstract **2**

1 Introduction to Machine Learning and Quantum Computing **2**
1.1 Moore's Law and State of the Industry 2
1.2 Quantum Computing and Qubits 3
1.3 Quantum Advantage for Algorithms: Why Quantum? 4

2 Algorithmic Time Complexity **4**
2.1 Relevance to Machine Learning 4
2.2 Big O Notation 5

3 Quantum Machine Learning (QML) **6**
3.1 Solving Linear Systems Classically 7
3.2 The HHL Algorithm - A Quantum Alternative 7

4 Quantum Approach to SVMs **8**
4.1 Mathematically Modeling the SVM 8
4.2 The Gap in Classical SVMs 9
4.3 Quantum SVMs (QSVMs) 10

5 Encoding Classical Data for Quantum Machine Learning **10**
5.1 Basis Encoding 11
5.2 Amplitude Encoding 11

6 Conclusion **12**

7 References **13**

Abstract

With the rising popularity of machine learning in the past decade, a stronger urgency has been placed on drastically improving computational technology. Despite recent advancements in this industry, the speed at which our technologies can complete machine learning tasks continues to be its most significant bottleneck. Modern machine learning algorithms are notorious for requiring a substantial amount of computational power. As the demand for computational power increases, so does the demand for new ways to improve the speed of these algorithms. Machine learning researchers have turned to leverage quantum computation to significantly improve their algorithms' time complexities. This counteracts the physical limitations that come with the chips used in our technology today. This paper questions current classical machine learning practices by comparing them to their quantum alternatives and addressing the applications and limitations of this new approach.

1 Introduction to Machine Learning and Quantum Computing

Machine learning has yielded significant advancements in practically every industry, allowing algorithms to utilize large amounts of data to create predictions, automate tasks, and understand complicated trends. As the industry grows, experts discover new ways to use machine learning to execute high-impact tasks; however, this also means that the complexity of these algorithms rises in tandem.

Quantum computing is a technology that fundamentally remodels the way computers approach computation. While it is not inherently faster, it does provide a new approach to computing that significantly improves algorithmic time efficiency for various classical tasks in many circumstances. Naturally, there is a parallel between these two industries with time complexity being one of the biggest points of research for both quantum computing experts and machine learning researchers[1].

1.1 Moore's Law and State of the Industry

In the past, the growth of the AI industry has heavily relied on the growth of the semiconductor industry. For years, there had been enough computational power to support more computationally complex algorithms. Machine learning researchers created computationally intensive algorithms while companies created chips that yielded higher computational power. The growth of this industry has historically been modelled by a law commonly referred to as Moore's Law[5]. Moore's Law is an "empirical relationship", a relationship that relies on observation as opposed to rigorous physics. This law states the following:

> "The number of transistors on a microchip doubles about every two years, though the cost of computers is halved."

This relationship has been accurate since it was deemed law in 1965 by Gordon Moore until recently. Due to the physical limitations on transistor cooling, size, and cost, Moore's Law is thought to no longer apply[6]. The growth of the semiconductor industry will soon no longer be able to sustain the growth in the Machine Learning (ML) industry. Unfortunately, this means that this industry can no longer rely on improvements to these classical systems; however, replacing the classical systems inherent in modern-day ML with new infrastructure rooted in quantum mechanics can eliminate its reliance on Moore's Law.

1.2 Quantum Computing and Qubits

Classical computers run on bits, the most basic unit of information in computing. These bits contain one of two values that represent a logical state[7], which binary representations are often used for.

Quantum computers run on quantum bits (qubits), the most basic unit of *quantum* information, instead. Contrary to classical bits, qubits can contain more than just two values.

"Schrödinger's cat"[9] is a thought experiment that demonstrates this concept. In this thought experiment, a cat is placed in a box with a device that has a 50% chance of killing the cat within the hour. At the end of the hour, the cat is in a superposition: a state of being both dead and alive at the same time.

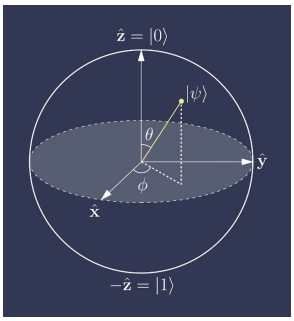

Figure 1: Bloch Sphere

Similarly, a qubit contains a superposition of both the states "0" and "1" rather than just containing one or the other. A superposition can be represented as a vector on the Bloch sphere (see Figure 1). These qubits will always collapse to either 0 or 1 when observed or measured. In Schrödinger's cat, the state of the aforementioned cat be represented as a probability until the moment in which the box is opened. This probability can be represented mathematically as the following superposition:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Here, the vector $|0\rangle$ is a basis vector that denotes the cat being alive, and the vector $|1\rangle$ is a basis vector that denotes the cat being dead in Dirac notation.

1.3 Quantum Advantage for Algorithms: Why Quantum?

Outside of machine learning, quantum computing has shown how replacing classical infrastructure with its quantum counterpart can significantly improve classical algorithms. An example of this is Grover's Algorithm[2]. This algorithm references a popular classical algorithm that searches an unstructured array of elements for an element with a certain value and returns the index of the element.

With classical computing, this algorithm takes $O(n)$ time, meaning the time taken to run the algorithm increases linearly with the number of inputs. However, quantum computing allows this same algorithm to be executed in $O(\sqrt{n})$ time which makes it significantly faster. It is crucial to note that the quantum counterpart for this algorithm constructs a dramatic speedup. Algorithms with this characteristic are known to have the "quantum advantage".

2 Algorithmic Time Complexity

Algorithmic Time Complexity[10] is ultimately the crux of this paper. The role that quantum computing plays in machine learning can control the practical impacts that ML algorithms have. To make these impacts tangible, it is important to understand what computational complexity is.

The time complexity of an algorithm is known to be the *'worst case scenario'* time taken for an algorithm to run for a given size of inputs denoted with N by convention.

> "Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input."[11]

Various modifications of the same algorithm can yield different time complexities based on structural differences and the number of operations needed to run the algorithm, an important concept in computer science in the context of improving algorithms. As stated earlier, this becomes especially relevant when there is not enough computational power to support algorithms that are not optimized to their highest possible efficiency.

2.1 Relevance to Machine Learning

While the concept of time complexity is often in the scope of most algorithm-related discussions in the field of computer science, it holds a special place in the world of machine learning. Time complexity refers to the relationship between the number of items within an algorithm and the amount of time it takes to run the algorithm. For example, it is evident that sorting numbers in an array with 100 elements would take less time than sorting numbers in an array with 1000 elements.

Many algorithms do not require significantly large input sizes while others might. That being said, even when programmers attempt to accomplish tasks such as sorting through 5000 elements in an array, these tasks are not very computationally-intensive due to the impressive power of modern computers. A lot of the time, realistic applications of computer science do not require algorithms with unreasonably large input sizes, removing the emphasis on time complexity; however, this would no longer be the case if a computer had to go through 1 000 000 elements instead, or maybe even 1 000 000 000.

Suppose there are two algorithms, both of which sort 1 000 000 numbers from highest to lowest. One of them requires n computations while the other requires n^2 computations. The first algorithm would only require 1 000 000 computations, while the second algorithm would require 1 000 000 000 000. Due to the quadratic nature of the second algorithm, a very large dataset yields an unreasonable amount of computations. This increases the relevance of time complexity.

Machine learning requires large datasets to identify patterns and 'learn' from the data. This means that it requires the largest input sizes in comparison to almost every other discipline of computer science. While other types of algorithms can be satisfied with poor time complexities and still remain practical, machine learning cannot. Poor time complexities can render machine learning algorithms entirely impractical, forcing programmers to choose between including more data for a better-informed algorithm or fewer data to allow for the algorithm to run in a reasonable amount of time. Note that running machine learning algorithms is not just a matter of seconds or even minutes. A lot of the time, training ML algorithms takes *hours* or even *days*.

2.2 Big O Notation

In computer science, the Big O notation refers to the growth rate of a certain algorithm as a function of the number of inputs n.

> "Different functions with the same growth rates may be represented using the same O notation"[12]

This notation helps us differentiate between algorithmic complexity by outlining whether algorithms are linearly, exponentially, logarithmically, or polynomially proportional to the number of inputs n.

The relationships between input size and time can be seen in Figure 2.

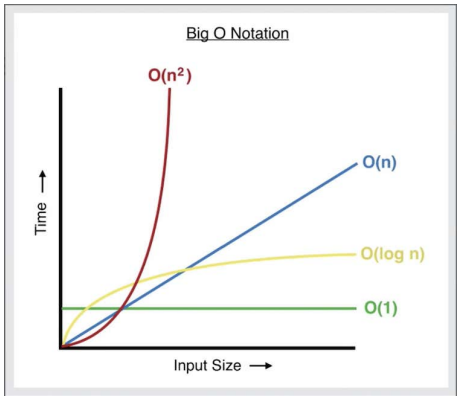

Figure 2: Big O Notation Graphs

A common analogy used to describe this is using different methods to transfer data. Suppose a set of data needs to be sent from one house to another. There are two options: transfer the data through the internet or walk over with a USB stick that has the data on it. Here, these options are analogous to different algorithms that complete the same task. With the first example, the amount of time it would take to transfer the data is dependent on the number of files there are. Transferring more data through the internet requires more time. The relationship here is linear as the amount of time it takes to send over the data has a linear relationship with the number of files being sent over. This would be represented as $O(n)$. With the second option, the amount of time it takes to travel over to the other house is the same regardless of the amount of data on the USB stick. Here, the time taken is constant and not proportional to the input size. This can be represented as $O(1)$. The notation does not necessarily outline the specific function that models the relationship between input size and time. Instead, it provides a general understanding of the nature of the relationship.

Note that there are other factors that can affect time complexity. For example, giving a USB stick to somebody a few blocks away versus sending it across the entire country are two very different things that take very different amounts of time. That being said, the *relationship* between input size and time remains the same: constant. Big O notation is not designed to outline the exact time complexity, but rather the growth rate between time complexities. This becomes especially useful to compare algorithms with incredibly large data sets. There may be moments where algorithms that run in $O(n^2)$ time may take less time than algorithms that run in $O(n)$ time. However, when the input sizes become as large as they tend to be in ML, it can objectively be said that certain algorithmic complexity growth rates are superior to others, creating a clear and tangible comparison between algorithms. This will contextualize the rest of this paper and create clear benchmarks of comparison in the discussion of machine learning algorithms and their quantum counterparts.

3 Quantum Machine Learning (QML)

Quantum Machine Learning (QML) utilizes quantum computing to decrease the time complexity for various machine learning problems. A major component of

this quantum speedup is its ability to significantly improve computation for linear algebra operations known as Basic Linear Algebra Subroutines (BLAS)[13]. These include solving linear systems, matrix multiplication, linear combinations, Fourier transforms, etc. It is important to note that machine learning is almost entirely based on linear algebra and calculus. Many of the operations required for training machine learning models include these subroutines. Quantum computing can solve linear algebra problems faster than any classical system. Acquiring exponential speedups to these linear algebraic operations creates large implications for the future use of the machine learning algorithms rooted in these subroutines.

One of the most relied-upon linear algebra operations in the field of machine learning is the ability to solve linear systems through matrix operations. Solving linear systems by representing them as matrices is a concept utilized in various machine learning algorithms such as linear regression and support vector machines. Classically, one of the largest limitations to the applications that these algorithms can have is the time complexity for solving linear systems; however, an algorithm developed by Aram Harrow, Avinatan Hassidim, and Seth Lloyd creates an exponential speedup for solving linear systems compared to the fastest classical counterpart. This algorithm is known as the HHL Algorithm[14].

3.1 Solving Linear Systems Classically

Solving linear equations can be modeled classically with the equation $A\vec{x} = \vec{b}$ where A is a matrix, $\vec{x}$ is a vector of n variables, and $\vec{b}$ is a vector of solutions. The question aims to solve for $\vec{x} = A^{-1}\vec{b}$, commonly done classically with algorithms such as *"Gaussian Elimination"* which has a time complexity of $O(n^3)$. The best algorithms take approximately $O(n)$ time, much better than exponential time but much slower than potential quantum alternatives.

3.2 The HHL Algorithm - A Quantum Alternative

This problem can be modelled through quantum mechanics with the Harrow-Hassidim-Lloyd Algorithm. This algorithm has a time complexity of $O(\log \mathrm{n})$, an exponential speedup to the best classical algorithms for this problem. This problem can be modeled quantumly with the equation $A|x\rangle = |b\rangle$ where the question aims to solve for $|x\rangle = A^{-1}|b\rangle$. However, the exponential speedup only occurs under the conditions that A is a Hermitian matrix, a matrix equal to its own conjugate transpose, and the user isn't interested in the actual values of $|x\rangle$ but rather the result of applying an operator onto it. Reading out the values of the solution vector would change the algorithmic time complexity to $O(N)$, matching the best classical algorithm for this problem since the result of the HHL algorithm is a quantumly encoded representation of the solution vector $|x\rangle$. Despite these restrictions, this algorithm changes the way linear systems can be solved within

machine learning algorithms, ultimately impacting the holistic time complexity of the algorithms.

4 Quantum Approach to SVMs

Support Vector Machines (SVMs)[15] use a unique technique of data representation in order to solve classification machine learning problems. Data is mapped to an N-dimensional feature space where its manipulated so that a hyperplane, a subspace with $N-1$ dimensions, can separate the data into different classes. Oftentimes, this is done using the "kernel trick" where data that is not linear is projected into a higher dimensional space to allow it to be divided linearly by a hyperplane. As a result, new data that is mapped onto the feature space can be classified based on the physical positioning of this hyperplane. The model uses backpropagation techniques to define an accurate hyperplane in order to solve classification problems.

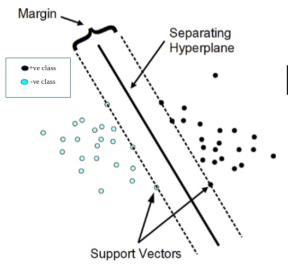

Figure 4: SVM Model

4.1 Mathematically Modeling the SVM

In classical SVMs, a hyperplane is created with the equation $\vec{w} \cdot \vec{x} - b = 0$. Here, $\vec{w}$ is a vector of weights and b is the bias for an n-dimensional hyperplane. Our training data is defined by the following vectors

$$\begin{bmatrix} \vec{x_1} \\ \vdots \\ \vec{x_n} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

where each $\vec{x_i}$ represents the i^{th} training example and each y_i is either 1 or -1, defining which class $\vec{x_i}$ is a part of. The following equation is the expanded equation of a hyperplane:

$$w_1 x_1 + \cdots + w_n x_n - b = 0$$

An SVM model aims to optimize our weights and biases to separate the data within our feature space. It does so by maximizing the margins between the support vectors and the hyperplane. The equations for these support vectors are:

$$\vec{w} \cdot \vec{x} - b = 1 \quad \text{and} \quad \vec{w} \cdot \vec{x} - b = -1$$

The margins between the hyperplane and the support vectors are characterized by the magnitude of the vector orthogonal to our hyperplane between the hyperplane

and the support vectors, This vector is also known as the normal vector. Since the weights vector is the coefficient vector for our hyperplane, the normal vector is defined by the vector $\vec{w}$.

Utilizing this information, the margin can be mathematically derived. We can take any $\vec{x}$ on our hyperplane and calculate the distance required to reach the support vector travelling along it. Knowing that $\vec{w} \cdot \vec{x} - b = 0$, we can add a coefficient k to a unit vector in the same direction as our normal vector $\vec{w}$ to reach $\vec{w} \cdot \vec{x} - b = 1$ and isolate for the margin k.

$$\vec{w} \cdot (\vec{x} + k\frac{\vec{w}}{||\vec{w}||}) - b = 1$$

Here, $\frac{\vec{w}}{||\vec{w}||}$ is a unit vector in the same direction as $\vec{w}$ and k is the magnitude of this unit vector. Knowing that $\vec{w} \cdot \vec{x} - b = 0$, this equation can be further simplified:

$$(\vec{w} \cdot \vec{x} - b) + k(\frac{\vec{w} \cdot \vec{w}}{||\vec{w}||}) = 1$$

$$0 + k\frac{||\vec{w}||^2}{||\vec{w}||} = 1$$

$$k = \frac{1}{||\vec{w}||}$$

This means that our margin k is $\frac{1}{||\vec{w}||}$. The distance between each support vector and the hyperplane needs to be the same in order to ensure proper classification from this model, meaning that our overall margin is $2 \cdot \frac{1}{||\vec{w}||}$, simplified to $\frac{2}{||\vec{w}||}$. The optimization of an SVM model attempts to maximize this equation, synonymous to minimizing $||\vec{w}||$. For hard-margin SVM problems, the following constraints are maintained in order to ensure that the data is classified correctly:

$$\vec{w} \cdot \vec{x} - b \geq 1 \text{ for } y = 1 \quad \text{and} \quad \vec{w} \cdot \vec{x} - b \leq 1 \text{ for } y = -1$$

4.2 The Gap in Classical SVMs

With larger and larger datasets, the limitations of the classical SVM models become evident with training time increasing quicker and quicker. When our data is mapped to higher dimensional feature spaces, the computational power required for minimizing $||w||$ increases alongside it. With more and more training data, the sheer amount of computations is increased as the loss function used for training the weights and biases of the hyperplane is dependent on each individual data point in the training dataset.

4.3 Quantum SVMs (QSVMs)

The bottleneck for models like SVMs is the classical, computationally-intensive means of optimization. In Quantum SVMs (QSVMs), the input data is encoded into qubits and various algorithms such as the HHL algorithm significantly speed up linear algebra computation utilized during optimization. Despite initializing the model classically, mapping the inputs and hyperplane onto qubits allows for quantum optimization techniques to rapidly decrease the time frame that these models train in. These results can then be decoded, resulting in a well-optimized SVM model that functions in the same way as classical SVMs despite the significant improvement in its training time.

Classical Support Vector Machines utilize kernels in order to map data onto a new domain allowing for it to be classified with a hyperplane. Some examples of these kernels include:

$$\text{Polynomial:} \quad k(x_i, x_j) = (x_i \cdot x_j)^d$$

$$\text{Sigmoid:} \quad k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c)$$

$$\text{Radial Basis Function (RBF):} \quad k(x_i, x_j) = \exp(-\gamma |x_i - x_j|^2)$$

Aside from quantum optimization, QSVMs can also implement quantum kernels that allow for the quantum advantage to occur within classical SVM models. Rather than operating on data classically, quantum operations on quantumly-encoded data are done to create mapped states. The dot products of these states are then taken, functioning similarly to classical kernels. The ZZFeatureMap and PauliMatricesMap are popular quantum kernels that use quantum gates to map the data in order to be classified classically[19].

5 Encoding Classical Data for Quantum Machine Learning

In the previous section, it is briefly stated that classical input data is encoded into qubits for Quantum SVMs, with the exception that the input data begins in a quantum form. In many QML algorithms, classical data must be encoded into qubits. That being said, the computational power required for this step holds great relevance in the conversation of algorithmic time complexity, being a large contributing factor as to whether or not a machine learning algorithm truly has a quantum speedup[1]. While many quantum algorithms in their worst-case scenario require exponential time to encode classical data into qubits, many algorithms that promise quantum speedup assume that this data can be quantumly encoded in linear or logarithmic time[21]. This is an aspect of QML that still requires a large

amount of research to truly allow for quantum computing to reach the potential it could have in this industry.

5.1 Basis Encoding

One of the methods in which classical data is encoded for use by quantum systems is basis encoding. This digital encoding method utilizes the computational basis within a quantum system by directly mapping data in the form of binary strings into its corresponding quantum state. For example, $x = 01$ would be represented as $|x\rangle = |01\rangle$. A dataset can be represented as superpositions of these computational basis states

For example, let us take a dataset X consisting of n-dimensional vectors for a total of a vectors.

$$X = x_1, x_2, \ldots, x_a$$

The dataset after basis encoding is represented with the following superposition:

$$|X\rangle = \frac{1}{\sqrt{a}} \sum_{i=1}^{a} |x_i\rangle^{[22]}$$

The following dataset $X = \{010, 101, 011\}$ would be represented quantumly as:

$$|X\rangle = \frac{1}{\sqrt{3}} |010\rangle + \frac{1}{\sqrt{3}} |101\rangle + \frac{1}{\sqrt{3}} |011\rangle$$

This form of encoding is largely preferred when the data has to be arithmetically manipulated within the quantum algorithm.

5.2 Amplitude Encoding

Amplitude encoding is an analogue encoding method in which classical data can be quantumly encoded. It often holds a large advantage when optimizing for information density and trying to store a large amount of data in a small number of qubits. This method utilizes the amplitude of a quantum state in order to encode classical data and is largely preferred for machine learning algorithms, utilizing $\log_2(n)$ qubits.

With amplitude encoding, the amplitudes of a normalized vector are utilized as the coefficients for our basis vectors. For example, suppose we have a normalized vector

$$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

Here, $||x|| = 1$. The amplitude encoded vector would look like the following:

$$|x\rangle = x_0 |0\rangle + x_1 |1\rangle$$

Utilizing this same concept, we can encode the following vector:

$$\alpha = (3, 5, 1)$$

The normalized version of the vector α is:

$$\alpha_{\text{norm}} = \frac{1}{\sqrt{35}}(3, 5, 1)$$

These can then be utilized as the coefficients for our superposition, resulting in our final encoded vector:

$$|\alpha\rangle = \frac{3}{\sqrt{35}} |00\rangle + \frac{5}{\sqrt{35}} |01\rangle + \frac{1}{\sqrt{35}} |10\rangle$$

A remarkable improvement is made in the number of qubits required when using amplitude encoding as opposed to basis encoding. That being said, the nature of the encoded superposition makes this method harder to manipulate arithmetically.

6 Conclusion

Despite the indisputable potential for disruption in the machine learning industry with quantum computing, this technology is not where it needs to be to achieve these disruptions. Quantum computers are incredibly susceptible to noise and do not currently have enough qubits to fully improve time complexity at a scale required to have any real relevance; however, it is a field that is growing rapidly, a field that will one day be where it needs to be to have these impacts. With the considerable research being done in the intersection between QC and AI/ML, researchers prepare themselves for the day in which their research transforms from theory to reality. Quantum is highly regarded as a potential solution for solving the computational limitations machine learning algorithms currently face. As one of the largest tools with the ability to disrupt the ML industry, there is no doubt that the future of machine learning is quantum.

7 References

[1] royalsocietypublishing.org/doi/full/10.1098/rspa.2017.0551

[2] en.wikipedia.org/wiki/Grover's_algorithm

[3] en.wikipedia.org/wiki/Quantum_machine_learning

[4] ai.googleblog.com/2021/06/quantum-machine-learning-and-power-of.html

[5] en.wikipedia.org/wiki/Moore's_law

[6] brainspire.com/blog/end-of-moores-law-whats-next-for-the-future-of-computing

[7] en.wikipedia.org/wiki/Bit

[8] devopedia.org/qubit

[9] en.wikipedia.org/wiki/Schrödinger's_cat

[10] en.wikipedia.org/wiki/Time_complexity

[11] mygreatlearning.com/blog/why-is-time-complexity-essential

[12] en.wikipedia.org/wiki/Big_O_notation

[13] en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

[14] en.wikipedia.org/wiki/Quantum_algorithm_for_linear_systems_of_equations

[15] en.wikipedia.org/wiki/Support_vector_machine

[16] aliceliu2004.medium.com/quantum-support-vector-machines-a-new-era-of-ai-1262dd4b2c7e

[17] editor.analyticsvidhya.com/uploads/83153SVM.png

[18] cs.cmu.edu/ tom/10701_sp11/slides/Kernels_SVM2_04_12_2011-ann.pdf

[19] medium.com/mit-6-s089-intro-to-quantum-computing/quantum-support-vector-machine-qsvm-134eff6c9d3b

[20] quantumalgorithms.org/chap-classical-data-quantum-computers.html

[21] quantumzeitgeist.com/quantum-encoding-an-overview/

[22] pennylane.ai/qml/glossary/quantum_embedding.html

[23] catalogimages.wiley.com/images/db/pdf/9780470525890.excerpt.pdf