Exponential Advantages in Quantum Machine Learning through Feature Mapping

Andrew Nader, Kyle Oppenheimer, and Gary Tom

December 4, 2020

Contents

1 Introduction

Machine learning (ML) has opened the door to many problems we could not previously solve by getting computers to figure out solutions without being explicitly told what to do. Similarly, quantum computers allow us to find solutions to problems that might otherwise be infeasible for classical computers by harnessing the computing power of quantum entanglement and superposition. At the intersection of these fields is quantum machine learning (QML), which makes use of these quantum effects to enhance the field of machine learning and break through classical computational barriers. Many QML algorithms have been proposed that can offer some computational speed-up, however, it is still an open question to determine when we can expect to achieve speed-ups, and in what circumstances exponential speed-ups are possible.

In this report, we take a chronological look at some impactful papers in this area to give a sense of the trajectory of the research and where that might propel us in the future. In Section 2 we describe a quantum recommendation algorithm from 2016 [1], which, along with several other QML algorithms from around the same time, were initially believed to achieve exponential speed-ups. However, these papers were not being particularly rigorous about the input and output to the algorithms and as such it was unclear how exactly to compare them to their classical counterparts. In Section 3, we describe Tang's dequantization results [2], where it was shown that if we make similar assumptions about access to the input in the classical regime, then there are classical algorithms which can match the QML results up to polynomial, not exponential, slowdown. These results brought into question whether the quantum algorithms themselves contributed to the speed-up, or if it was due to assumptions of the quantum data and other small details involved in the end-to-end process.

A natural challenge is then to find quantum machine learning algorithms which can be proven to be exponentially faster than their classical counterparts, irrespective of the dataset to be learned. The most obvious way of doing that would be to use quantum computing to obtain exponential speedups in the training process of an existing classical algorithm (such as inference in probabilistic graphical models), but the research community has not succeeded in doing this yet. However, there have been considerable advancements in using quantum models to try to generate correlations that are hard to compute classically. For example, one way of doing this is to use quantum feature maps that are classically intractable to embed data into a high dimensional Hilbert space where it can be linearly separated by a support vector machine. If these correlations turn out to be useful for some learning problem, then that would provide a considerable advantage over classical ML. In Section 4, we survey a QML algorithm that leverages these quantum feature maps to obtain a rigorous quantum exponential speed-up for a learning problem, given only classical access to the data [3]. We continue to discuss the results of Huang et al. [4], which outlines when a quantum advantage might be possible. Research in this field will likely be focused on finding other results of the same kind, as well as expanding on these results to be of more practical relevance.

2 Quantum Recommendation Algorithm

The algorithm at the center of the hype that took hold of quantum machine learning from 2008 onwards is the HHL algorithm [5], named for its creators, Aram Harrow, Avinatan Hassidim, and Seth Lloyd. The intuition that led to the creation of the algorithm is simple: from a mathematical standpoint, most of machine learning is just a big pile of linear algebraic routines in extremely high

dimensional vector spaces, and quantum mechanics excels at dealing with these high dimensional vector spaces. So it should stand to reason that quantum computing provides substantial speed-ups for linear algebraic subroutines in this setting. The HHL algorithm provides a provably exponential speed-up for matrix inversion, albeit with many caveats which are discussed by Aaronson [6]. The HHL algorithm led to a slew of discoveries in quantum based linear algebraic subroutines, the most important of which is the quantum recommendation algorithm (QRA) [1].

The QRA was extremely important because it solved a practical problem, appeared to provide an exponential speed-up over classical algorithms, and did not share some of the problems that the HHL algorithm had. For example, the HHL matrix inversion procedure for $Ax = b$ does not return the classical state x, but the quantum state $|x\rangle$. In that sense, HHL solves a different problem, giving exponential speed-up for quantum inputs and outputs, whereas the QRA solves the same problem as the classical one.

The classical recommendation system problem is well studied in the literature since it is both theoretically interesting and of practical importance for companies like Netflix and Amazon. The goal is to recommend new products to a user that they are likely to enjoy, given their past likes and dislikes. The main data structure of interest is an $m \times n$ preference matrix P, where m is the number of users, n is the number of products, and the matrix entry P_{ij} represents some kind of utility value that user i has for product j. An example of such a preference matrix for Netflix movies and TV shows is shown in Table 1. Many utility values are missing from the preference matrix and are denoted by "?". This is the case we often encounter in the real world, as we only have the matrix entry P_{ij} if user i has provided a review for product j. If David logs in and is looking for a movie, then it would make sense to recommend Inception, since it appears that Wayne and David have similar tastes and Wayne liked Inception. A simple (in principle, at least) solution then emerges: in order to give recommendations to a user, we would use the available data from "similar" users to reconstruct the utility value for a particular user (i.e, their corresponding row in the preference matrix), and we would then return a high value element from this row.

	Inception	Kim's Convenience	The Avengers	Friends
Wayne	0.8	0.8	0.9	?
Robert	0.95	0.2	0.1	?
Cristiano	0.3	?	?	?
Lionel	?	?	0.9	0.4
David	?	0.9	0.9	?

Table 1: Example of a Netflix preference matrix.

In real world applications and for companies like Amazon and Netflix, the preference matrix is huge, with millions of users and entries, so the recommendation problem becomes challenging. The matrix reconstruction cannot be done in general, but it can be done if we make the assumption that our preference matrix has a low rank approximation. There is empirical evidence that this assumption holds true, and in general consumers fall into a small number of categories based on preferences. Armed with this low rank assumption, classical algorithms can solve the recommendation problem in time polynomial in the matrix dimensions. However, to make a recommendation, we do not need to reconstruct the entire row for a user; we only need to be able to sample a high value element from that row. This is exactly what the QRA does in time poly-logarithmic in the matrix dimensions!

Let us make this more precise. First, a new matrix T is defined by replacing the unknown values

in P by 0. Then a uniform sub-sample $\hat{T}$ of T is defined, where $\hat{T}_{ij} = \frac{T_{ij}}{p}$ with probability p and 0 otherwise. An analysis shows that being able to sample from $\hat{T}_k$ for small k is enough, where $\hat{T}_k$ is a rank k approximation to $\hat{T}$. Note that the row $(\hat{T}_k)_i$ for user i from which we need to sample is closely related to the singular value decomposition, since $(\hat{T}_k)_i$ is the projection of $\hat{T}_i$ onto the top-k singular row singular vectors of $\hat{T}$. This implies a possible way to get the sample we need: compute the projection in time poly-logarithmic in the matrix dimensions, and then measure it in the computational basis. The QRA does this,[1] but on a matrix that is "close" to $\hat{T}_k$. Still, this is sufficient for generating good recommendations.

There is a final detail, however; quantum algorithms operate on quantum data, not classical data, but our preference matrix is assumed to be classical. So there has to be some way of encoding the classical data into a quantum state without sacrificing the exponential speed-ups. Kerenidis and Prakash resolved this by defining a binary tree data structure for $m \times n$ matrix A that is able to encode the classical rows A_i into quantum states $|A_i\rangle$ in time poly$(\log mn)$. This is called the *state preparation procedure*. It seems now that we have a QRA that offers an exponential speed-up over known classical algorithms, but the two necessary assumptions for the QRA, the low rank approximation assumption, and the assumption of an efficient state preparation procedure, will prove to be detrimental to the quantum advantage provided by the algorithm.

3 Dequantization Result

At the time of publication, the Kerenidis and Prakash quantum recommendation algorithm was one of the more promising candidates of QML algorithms that promised exponential speed-up over classical algorithms. Unlike other QML algorithms, such as principal component analysis (PCA) [7], and supervised clustering [8], QRA was able to achieve exponential speed-up without necessitating sparse or well-conditioned data, making it more applicable and comparable to the classical problem.

However, in 2019, Tang published a classical algorithm for recommendation systems that performed only polynomially slower than QRA, demonstrating that Kerenidis and Prakash's algorithm does not give an exponential speed-up over classical recommendation algorithms [2]. More importantly, Tang's work provides a framework for *dequantizing* QML algorithms. Since then, a number of quantum-inspired classical algorithms have been developed for various machine learning tasks [9, 10, 11]. Here we will briefly summarize the dequantization result by Tang.

3.1 Quantum-inspired Classical Recommendation Algorithm

Like many QML algorithms, QRA relies on the quantum state preparation assumptions: when given some input vector v, one can prepare a corresponding quantum state $|v\rangle$. Tang's key insight is that the quantum state preparation assumption requires a data structure that permits l^2-norm sampling of the data. With l^2-norm sampling, a classical algorithm can efficiently pinpoint portions of input vectors and matrices with the most weight, something that is implicitly encoded in the prepared quantum states for QML.

We define a vector $x \in \mathbb{C}^n$ to have *query and sampling access* if we can query x_i for any given $i \in [n]$; and we can independently sample $i \in [n]$ from a probability distribution defined as $D_x(i) = x_i^2/\|x\|^2$. Here $\|x\|$ is a normalization factor, and is assumed to be known for both classical

[1] Well, actually, it computes the projection of the vector onto the space spanned by the row singular vectors with singular value greater than some threshold, but that's good enough.

and quantum algorithms. This sampling technique is the classical analogue of the quantum state prepared for QRA. The distribution $D_x(i)$ is the same as making an i^{th} measurement on the quantum state $|x\rangle$. Similarly, we can define a matrix to have query and sampling access if the row vectors have query and sampling access.

From here we can produce three dequantized results which will contribute to the final classical algorithm. Firstly, we can efficiently estimate the inner product between two vectors $x, y \in \mathbb{C}^n$, where we have sampling and query access to x, and just query access to y. We can estimate $\langle x, y\rangle$ up to an error of $\|x\|\|y\|\varepsilon$ with at least $1-\delta$ probability, with a runtime of $O(\text{poly}(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}))$. This is an application of a median-of-means estimate. Computationally, the estimate would be the median of $6\log\frac{1}{\delta}$ copies of the mean of $\frac{9}{2\varepsilon^2}$ copies of the random variable y_i/x_i, where x_i has $i \sim D_x(i)$.

Secondly, consider a matrix $V \in \mathbb{C}^{n\times k}$ where we have sampling and query access, and the spectral norms $\|V\|$. For vector $w \in \mathbb{C}^k$ as input, we can output a sample of Vw (ie. the rows) with $O(\text{poly}(k))$ queries. This result relies on rejection sampling to generate the target D_{Vw} samples from another distribution $D_{V^{(j)}}$, for which we have sampling access.

Finally, with query and sampling access, we can efficiently generate a low-rank approximation using a classical algorithm, which is modified from an algorithm by Frieze, Kannan, and Vempala (FKV) [12]. For a matrix $P \in \mathbb{C}^{m\times n}$ with query and sampling access, we can create an approximate matrix T with $\text{rank}(T) \leq k$, for some threshold k; provided that the low-rank assumption is true for sparse and high-rank P, something already assumed for QRA. Specifically, the matrix T takes the form of

$$T = P(S^\dagger U\Sigma^{-1})(S^\dagger U\Sigma^{-1})^\dagger, \tag{3.1}$$

where $S \in \mathbb{C}^{l\times n}$ is a subset of rows in P, and U, Σ are from the singular value decomposition $P = U\Sigma V^T$. The matrix T is "close" to the original matrix P up to an error factor of ε. The Modified FKV algorithm runs in $O(\text{poly}(k, \frac{1}{\varepsilon}))$.

In the context of recommendation systems, we can now build the quantum-inspired classical algorithm based on the above dequantized results. Consider a preference matrix $P \in \mathbb{R}^{m\times n}$, where we have m users and n products, similar to Table 1. To generate good recommendations, first we apply the Modified FKV algorithm to obtain a low-rank approximation of T. Since we have sampling and query access to the preference matrix, we can sample from each row of the approximation

$$T_i = P_i S^T M S, \tag{3.2}$$

where P_i is a row in the preference matrix, S is a subset of rows from the preference matrix, and $M = U\Sigma^{-1}(\Sigma^{-1})^T U^T$, with U, Σ as defined before.

With this form, we can now approximate the inner product $P_i S^T$ with the procedure described above, and matrix multiply it with M. Finally, we have $P_i S^T U U^T S$ which we can sample for new recommendations using equation (3.2). With this result we can get recommendations in $O(\text{poly}(k)\log(mn))$ time, where the additional $O(\log(mn))$ comes in overhead from the implementation of the required data structure. This classical algorithm is not only polynomially slower than QRA, but is exponentially faster than the previous best classical recommendation algorithms.

4 Feature Mapping

Tang's dequantization result demonstrates that when given similar sampling access to quantum data, classical algorithms can perform many machine learning tasks in runtimes similar to quantum

algorithms. While there exists QML algorithms that require only classical access to data, until recently, it was not clear if these algorithms could achieve exponential speed-ups over their classical counterparts.

Recent work exploring supervised machine learning using quantum feature mapping has demonstrated a *provable* exponential speed-up for a particular learning problem, through the generation of correlations that are hard to compute classically. In particular, support vector machines are shown to efficiently classify classical data mapped onto a quantum feature space for the discrete logarithm problem. The use of these quantum enhanced feature spaces was first proposed and experimentally implemented by Havlíček et al. [13], as described in Section 4.1. In 2020, Liu et al. [3] further developed the method to achieve this exponential quantum speed-up for the discrete logarithm problem, as described in Section 4.2.

4.1 Supervised Machine Learning in Quantum Feature Spaces

Consider the following supervised machine learning task. A dataset with labels is split into a training set T and a test set S. An ML algorithm attempts to predict the labels of S by studying and learning from the data in T. More formally, the labels for the data points provide a map $m : T \cup S \to \{+1, -1\}$. The classifier algorithm is only provided with the labels of $\vec{x} \in T$, and must learn an approximate map $\tilde{m}(s) \approx m(s)$ for a test set data point $\vec{s} \in S$.

A common supervised machine learning model for such a task is the support vector machine (SVM). In SVM, the goal is to draw a separating hyperplane boundary between the two classes $\{+1, -1\}$ in feature space that maximizes the margin between the boundary and any nearby data points, the so-called *support vector*. This is often done using the *kernel method*, in which inner products between feature vectors are performed using a non-linear kernel function. Calculations using kernel functions are not only computationally cheaper, but they generalize the calculation to higher dimensional feature spaces. However, as the feature space becomes larger, the kernel functions become expensive to calculate. Quantum computers can estimate quantum kernels that might be infeasible to estimate classically via a procedure called quantum kernel estimation (QKE).

4.1.1 Quantum Kernel Estimation

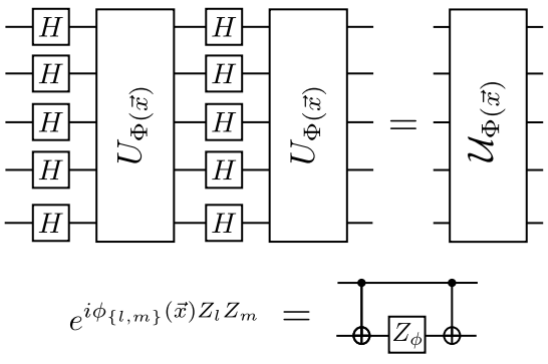


Figure 1: The circuit used in [13] for encoding classical data onto a quantum state. This circuit was used to map artificial data that was classified using a quantum computer. The experimental implementation equation (4.1) for two qubits is shown as well.

The first step to the method described by Havlíček et al. [13] is to map the classical data onto a quantum state. It is important that such a mapping is not too simple; for example, a feature map that only gives product states can easily be classified classically. The feature map used was $\mathcal{U}_\Phi(\vec{x}) = U_{\Phi(\vec{x})}H^{\otimes n}U_{\Phi(\vec{x})}H^{\otimes n}$, where H is the Hadamard gate and

$$U_{\Phi(\vec{x})} = \exp\left(i\sum_{S\subset[n]}\phi_S(\vec{x})\prod_{i\in S}Z_i\right), \tag{4.1}$$

where the coefficients $\phi_S(\vec{x})$ are used to encode the data, and Z_i is the Pauli Z measurement on the i^{th} qubit. This circuit is shown in Figure 1, and will be applied onto an initial $|0\rangle^{\otimes n}$ state. In general, $U_{\Phi(\vec{x})}$ can be any diagonal and unitary gate; this particular form was chosen for experimental convenience.

With the data mapped onto a quantum state, we now want to use SVM to create a classifier for data in $T = \{\vec{x}_1, \cdots, \vec{x}_t\}$, with corresponding labels $y_i = m(\vec{x}_i) \in \{+1, -1\}$. The maximization of the margin can be formulated as Lagrangian multipliers using a kernel function, the solution of which will give the parameters $\vec{\alpha} = (\alpha_1, \cdots, \alpha_t)$ and b the bias for a separating hyperplane boundary.

For any data point in the test set $s \in S$, the approximate classifying map is

$$\tilde{m}(\vec{s}) = \text{sign}\left(\sum_{i=1}^{t} y_i\alpha_i^* K(\vec{x}_i, \vec{s}) + b\right), \tag{4.2}$$

where $K(\vec{x}_i, \vec{s})$ is the kernel matrix that we can estimate through QKE. Since we are working with states in the quantum feature space, the inner product between feature vectors can be estimated from the transition probability between quantum state

$$K(\vec{x}, \vec{s}) = \| \langle\Phi(\vec{x})|\Phi(\vec{s})\rangle \|^2 = \| \langle 0|^{\otimes n}\mathcal{U}^\dagger_{\Phi(\vec{x})}\mathcal{U}_{\Phi(\vec{s})}|0\rangle^{\otimes n} \|^2. \tag{4.3}$$

Computationally, QKE works by running the quantum circuit $\mathcal{U}^\dagger_{\Phi(\vec{x})}\mathcal{U}_{\Phi(\vec{s})}$ on the initial state. We can then measure the output probability of $\langle 0|^{\otimes n}$ with $R = O(\varepsilon^{-2}t^4)$ shots of measurements. The resulting estimate $\tilde{K}$ will be close to the exact kernel matrix $\|K - \tilde{K}\| \leq \varepsilon$. The QKE procedure will be done twice: once during training on T and again for validation on S. Finally, we can implement the classifier as specified in equation (4.2).

4.2 First Provable Exponential QML Advantage

In this section we describe a recently proposed quantum algorithm that achieves an exponential speed-up over any classical algorithm, under the widely-held assumption of the hardness of the discrete logarithm problem (DLP) [3]. This method avoids any objections about its advantage arising from quantum state preparation assumptions by taking only classical input data. It then maps the data non-linearly to a high dimensional Hilbert space where it is linearly separable and can be classified by a hard-margin support vector machine (SVM). The exponential speed-up is achieved by estimating the SVM kernel matrix using a QKE program on a fault-tolerant quantum computer. This is, in fact, the only quantum sub-task in the process, as once the kernel matrix is calculated the resulting SVM program is entirely classical. We now describe the algorithm in detail.

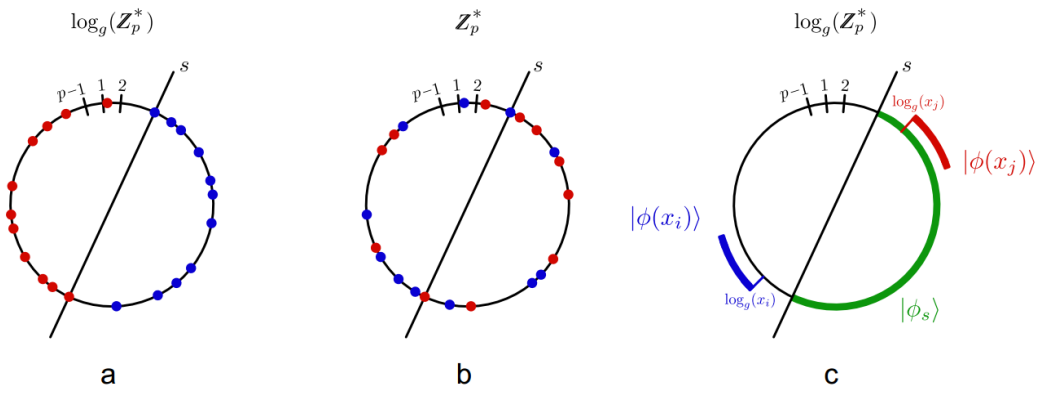


Figure 2: The data looks linearly separable to a quantum computer after it has taken a discrete log (**a**), while looking completely random to any classical algorithm (**b**). Inspired by this insight, we can use a quantum feature map that maps $x \in \mathbb{Z}_p^*$ to a quantum state $|\phi(x)\rangle$ to yield data that is linearly separable with a large margin (**c**). This can be seen by observing that the -1 labeled blue interval has no overlap with the green separating hyperplane, while the +1 red interval does.

4.2.1 DLP

The discrete logarithm problem as defined in [3] is

Definition 1 *Given a large prime number p and a generator g of $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, find $\log_g x$ on input $x \in \mathbb{Z}_p^*$, in time polynomial in $n = \lceil \log_2(p) \rceil$, the number of bits needed to represent p.*

This problem is believed to be infeasible for any classical algorithm, while it has been proven that Shor's quantum algorithm can solve it. The input data for the QML algorithm of Liu et al. [3] is based on a variation of DLP that has been proven to be just as hard as DLP, wherein the labels are created by first fixing some choice of $s \in \mathbb{Z}_p^*$ (which is not known to the algorithm) as well as fixing values for g and p (each choice leading to a different data set), and then choosing the label $f_s(x)$ of the data point x from

$$f_s(x) = \begin{cases} +1, & \text{if } \log_g x \in [s, s + \frac{p-3}{2}], \\ -1, & \text{else.} \end{cases} \tag{4.4}$$

These labels look random to any classical algorithm, and the authors prove that you could not achieve an accuracy that is inverse-polynomially better than simply guessing randomly with a classical approach, as shown in Figure 2. A quantum computer, on the other hand, can calculate $\log_g x$ and then find the boundary that linearly separates the classes by learning s, which is an easy machine learning task for a support vector machine. This is the insight that allows for a quantum exponential advantage, albeit for a somewhat contrived example that does not hold much practical relevance.

4.2.2 Quantum Feature Map

We are now ready to describe the quantum feature map that harnesses the power of the DLP classical-quantum separation to achieve an exponential speed-up. The authors prove that SVM-

QKE can classify the data of (4.4) with high accuracy by using the following predefined feature map

$$x \mapsto |\phi(x)\rangle = \frac{1}{\sqrt{2^k}} \sum_{i=0}^{2^k-1} |x \cdot g^i\rangle \,. \tag{4.5}$$

It has been proven that these quantum states can be efficiently constructed on a fault-tolerant quantum computer by use of Shor's algorithm. The author's show that this feature map results in a linearly separable feature space with a large margin, meaning that an SVM hyperplane could correctly classify every data point. However, QKE outputs a noisy kernel with variance defined by finite sampling statistics of the R measurement shots. We need to ensure that this error does not degrade the accuracy of the SVM to the point of it being unusable. The authors prove that because of the large margin property, we can achieve a test accuracy of atleast 0.99 using this method with probability 2/3, using results from soft-margin classifiers and robustness of the bounding hyperplane under additive noise.

This result is an encouraging rebound from the excitement of the exponential QML algorithms that were dulled down by Tang. Researchers now continue to search for similar methods with practical motivation, and perhaps applications that would be feasible for near-term quantum computers.

4.3 Power of Data in Quantum Machine Learning

In this last section, we discuss a recent paper on the "Power of Data in Quantum Machine Learning" [4], which presents promising results for QML. Supervised learning deals with the problem of approximating some function f if we are given a dataset of the form $\{(x_i, f(x_i))\}_{i=1}^n$. A naive approach to obtaining quantum advantages is to simply look at functions f that are hard to compute classically. If classical algorithms have trouble computing f, and our machine learning algorithm is a classical algorithm, would it not also have trouble dealing with f? The answer is surprisingly no. The presence of *data* changes everything.

4.3.1 How Data Changes Complexity Theoretic Considerations

Let us look at a trivial example that the authors Huang et al. [4] used to provide intuition. Consider some arbitrary d dimensional data $\{x_i\}_{i=1}^n$, and use it to define a function that is hard to compute classically. First, we will encode each d dimensional vector x_i into quantum state vectors. There are many ways to do this, such as using amplitude encoding, where we encode x_i into the amplitudes of a quantum state $|x_i\rangle$.

After we have encoded all of our data points, we define a unitary circuit U_{QNN} that corresponds to a time evolution under a many-body Hamiltonian. This U_{QNN}, along with our encoded data vectors, can be used to define a function f that is, in general, hard to compute classically. We define f by feeding each encoded data vector $|x_i\rangle$ to U_{QNN} and then measuring an observable O to obtain $f(x_i) = \langle x_i | U_{QNN}^\dagger O U_{QNN} | x_i \rangle$. As mentioned previously, this f is, in general, difficult to compute classically. But is it difficult to learn if we are given a dataset $\{(x_i, f(x_i))\}_{i=1}^n$? The answer is no. First, let us expand the expectation value given by $f(x_i)$, keeping in mind that we are using an

amplitude encoding,

$$\begin{aligned}\langle x_i| U_{QNN}^{\dagger} O U_{QNN} |x_i\rangle &= \left(\sum_{k=1}^{d} x_i^{k*} \langle k|\right) U_{QNN}^{\dagger} O U_{QNN} \left(\sum_{l=1}^{d} x_i^{l} |l\rangle\right) \\ &= \sum_{k=1}^{n}\sum_{l=1}^{n} x_i^{k*} x_i^{l} B_{kl},\end{aligned} \tag{4.6}$$

where B_{kl} is defined to be $\langle k| U_{QNN}^{\dagger} O U_{QNN} |l\rangle$. This is a function that is quadratic in the x_i^{k*} and x_i^l, and thus, it is easy to fit if we have enough training data. While f itself may be difficult to compute, the presence of data allows us to transform the problem to a simple learning problem. The rigorous separation between the complexity classes of QML and classical machine learning algorithms with data is described in the Appendix.

So the blows just keep coming and coming for quantum machine learning. First, the hype around quantum linear algebra subroutines died down with Tang's dequantization results. Then, we hoped that methods like the quantum kernel methods could obtain quantum advantages because they leverage circuits that are difficult to deal with classically, but we still had to see if these circuits are actually useful in practice and on real world datasets. And now, we have shown that an additional complication arises because the presence of data can elevate a classical model to the level of a quantum model! However, this paper still discusses positive results, both theoretical and empirical, that show that QML can provide substantial advantages over classical machine learning.

4.3.2 Geometric Kernel Distances and the Projected Quantum Kernel

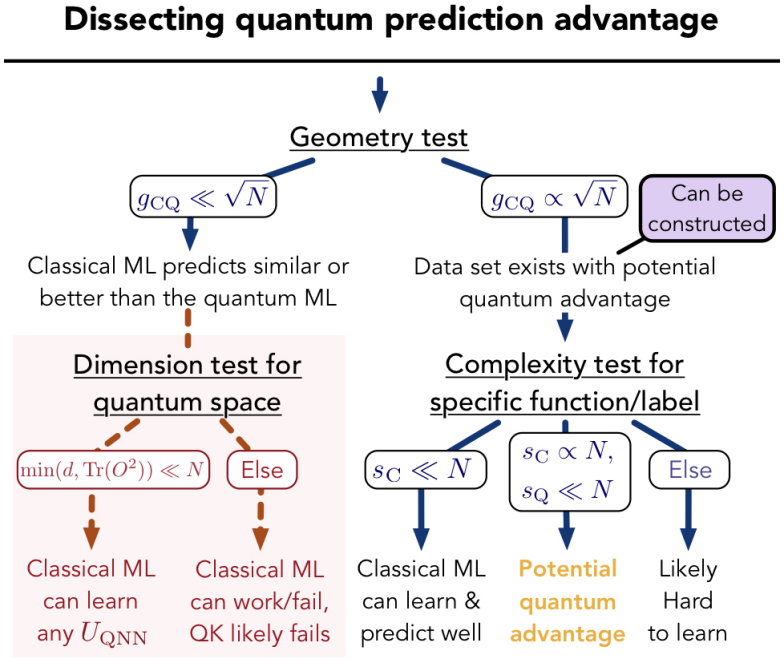
We saw in the previous section that one has to be careful when making claims of speed-ups in quantum machine learning, because the presence of data can easily put classical algorithms on equal footing with their quantum counterparts. So how do we check if our QML algorithm is actually giving us an advantage? Well, we can always rely on good old empirical evidence. We simply run our quantum algorithm on some tough to classify dataset, and compare it against a suite of properly optimized state of the art machine learning algorithms. This is the "deep learning research" way of doing things. Sure, the early work in neural nets had a lot of motivation, such as the flexibility and biological plausibility of distributed representations over the inherently local representations of symbolic learning algorithms, but the algorithms themselves did not have any deep mathematical theory guaranteeing properties like robustness or the ability of gradient descent to find minima that generalizes well in the optimization landscape. Deep learning only came to the forefront of Machine Learning when a team led by University of Toronto students and faculty obliterated the rest of the teams in the 2012 ImageNet competition by using a convolutional neural network. Empirical results and specific motivations are what brought fame to deep learning.

We could put all of our eggs in that basket, and rely on heuristics and empirical evidence to check if our algorithm is doing well. Admittedly, this is not very satisfying, for two reasons. First, there is the annoying feeling that we do not understand why our algorithm is performs well, and second, a lack of rigorous theoretical structure prevents us from reasoning *a priori* about a new algorithm. It would be nice if we had some sort of formula or test that could determine if an algorithm is a efficient without ever running it on actual computers. The second major contribution of Huang et al. [4] is exactly that, defining a flowchart can be used to test for potential quantum advantages.

More precisely, this paper focuses on kernel methods. We have already seen that kernel methods are well motivated in machine learning because of their rich theory and competitive empirical performance. The setting is as follows: given some classically difficult quantum model, we want to study the ability of classical and quantum machine learning algorithms to fit the data. As before, our quantum model is defined by

$$f(x) = \langle x | U_{QNN}^{\dagger} O U_{QNN} | x \rangle = Tr[O^U \rho(x)], \tag{4.7}$$

where $O^U = U_{QNN}^{\dagger} O U_{QNN}$. However, we are not restricted to amplitude encoding; any quantum encoding is possible. The flowchart proposed is shown here:



Let's define these variables and motivate them, alluding to a few theorems proved in the paper along the way. The first undefined variable, starting from the root, is g_{CQ}. However, the motivation for this variable is contingent on first understanding s_K, which appears at the second level of the flowchart in the form of the variables s_C and s_Q, so let's talk about that first. The main equation involving s_K is

$$\mathbb{E}_x |h(x) - \mathrm{Tr}[O^U \rho(x)| \leq c\sqrt{\frac{s_K}{N}}, \tag{4.8}$$

for $c > 0$, where $s_K = \sum_{i=1}^{n} \sum_{j=1}^{n} K_{i,j}^{-1} \mathrm{Tr}[O^U \rho(x_i)] \mathrm{Tr}[O^U \rho(x_j)]$. Here, $K_{i,j} = k(x_i, x_j)$, where k is the kernel function, and h is the function defined by our *trained* kernel algorithm.

Equation (4.8) is extremely important for understanding the intuition behind the results in this paper. Fundamentally, it means that our expected prediction error is bounded by two things: N, which is the size of our dataset (once again emphasizing the importance of data), and s_K. This s_K is a measure of model complexity. A more intuitive way to understand s_K is as follows: a trained kernel model h can always be written as $h(x) = w^{\dagger}\phi(x)$, where $\phi(x)$ is the associated feature function. Then $s_K = ||w||^2$. If s_Q is the model complexity of a particular quantum model, and s_C is the model complexity of the best classical kernel algorithm, then the potential advantage from our quantum algorithm depends on the separation between s_Q and s_C. For small separation,

there is no advantage in using a quantum algorithm. This is what the "Complexity test for specific function/label" part of the flowchart is describing. If s_C is proportional to N and s_Q is much smaller, then a potential quantum advantage exists. If s_C is much smaller than N, we do not need to consider using a quantum model because the classical model likely already works well. If both s_C and s_K are very large, the dataset is likely difficult to learn.

But notice that s_C and s_Q are giving us information about potential quantum advantages for a specific dataset. They are dependent on the actual function values that we are trying to approximate. What if for a particular function it is not worth using a quantum model, but for another function, it is? If there are *some* functions which the quantum model can approximate better than the classical model, then it is worth studying. This is what the geometric difference test in the first layer of the flowchart seeks to determine.

The asymmetric geometric difference between two kernels K^1 and K^2 is given by

$$g_{12} = \sqrt{||\sqrt{K^2}(K^1)^{-1}\sqrt{K^2}||_\infty}. \tag{4.9}$$

The importance of this distance arises from the following inequality

$$s_{K^1} \leq g_{12}^2 s_{K^2}. \tag{4.10}$$

Thus, the geometric difference gives us an idea about the separation of the model complexity, without referencing a specific function. This is exactly what we need. If the geometric difference g_{CQ} between the quantum kernel and the classical kernel is big enough, then there exists some dataset with a potential quantum advantage, and the quantum technique being used is worth considering. Otherwise, we can safely say that classical ML is the better choice.

The red part of the flowchart corresponds to additional tests we can run if we are using the quantum kernel method $K_{ij}^Q = k^Q(x_i, x_j) = \mathrm{Tr}(\rho(x_i)\rho(x_j))$. Let d be the rank of K^Q (d is also called the effective dimension). If $\min(d, \mathrm{Tr}(O^2)) \gg N$, then no matter the choice of U_{QNN}, a classical ML algorithm can learn the desired function. Hence, we can deduce that the problem here is that the encoding being used is classically easy and not worth considering.

The analysis conducted by the authors revealed some problems in QML algorithms, such as the fact that when the effective dimension d is large, the quantum kernel $\mathrm{Tr}[\rho(x_i)\rho(x_j)]$ will result in a small geometric difference, in which case a classical ML algorithm is preferable. This lead them to propose a new family of projected quantum kernels which work by projecting quantum states to some approximate classical representation. The idea is that even if d is large, the projection reduces the problem to a low dimensional space. In addition, by going through the quantum space in between the original space and the reduced space, these kernels are difficult to compute classically. For example, we can define a projected quantum kernel by using reduced physical observables. If we measure the one particle reduced density matrix on all qubits of the encoded state, $\rho_k(x_i) = Tr_{j!=k}[\rho(x_i)]$, we can define a projected quantum kernel as $k^{PQ}(x_i, x_j) = \exp(-\gamma \sum_k ||\rho_k(x_i) - \rho_k(x_j)||_F^2)$.

In their experiments, the authors find that projected quantum kernels increase the geometric difference, which is a positive thing as per our previous discussion. Interestingly, the authors show that projected quantum kernels can solve the discrete logarithm learning problem we discussed in a previous section, thus showing that projected quantum kernel lead to a rigorous quantum speed up for this problem. Let us now briefly discuss the experiments that the authors ran to validate their methods. We note that these are the largest QML experiments and simulations to date, reaching an impressive 1.1 quadrillion flops.

Robust empirical evidence of a quantum advantage in machine learning

Huang et al. empirically validated their ideas on the classical Fashion-MNIST dataset, three datasets with function values that come from a quantum neural network, and a suite of datasets that are specifically engineered to test the formulae relating the geometric difference, model complexity and prediction accuracy. There are 3 possible quantum encodings, three standard quantum kernel models (SVMs that estimate the kernel using quantum computers) that use the quantum kernel $k(x_i, x_j) = Tr[\rho(x_i)\rho(x_j)]$ (one model for each encoding), and three projected quantum kernel models that use the kernel function

$$k^{PQ}(x_i, x_j) = \exp\left(-\gamma \sum_k \sum_{P\in\{X,Y,Z\}} (Tr[P\rho(x_i)_k] - Tr[P\rho(x_j)_k])^2\right). \tag{4.11}$$

where γ is a hyper-parameter which is tuned to maximize the prediction accuracy, and X, Y and Z are the Pauli matrices. The three possible quantum encodings are as follows:

1. A qubit rotation circuit, with $|x_i\rangle = \bigotimes_{i=1}^{n} e^{-iX_j x_{ij}} |0^n\rangle$, where x_{ij} is the j-th entry of the data point x_i, and X_j is the Pauli X operator acting on the j-th qubit.
2. An IQP style encoding circuit, where $|x_i\rangle = U_Z(x_i)H^{\otimes n}U_Z(x_i)H^{\otimes n}|0^n\rangle$, where $U_Z(x_i) = \exp(\sum_{i=1}^{n} x_{ij}Z_j + \sum_{j=1}^{n}\sum_{j'=1}^{n} x_{ij}x_{ij'}Z_jZ_{j'})$, where Z_j is the Pauli Z operator acting on the j-th qubit.
3. A Hamiltonian Evolution ansatz, where $|x_i\rangle = (\prod_{i=1}^{n} \exp(-i\frac{t}{T}x_{ij}(X_jX_{j+1} + Y_jY_{j+1} + Z_jZ_{j+1}))) \bigotimes_{j=1}^{n+1} |\psi_j\rangle$. T is arbitrary (the authors choose $T = 20$), and t is an arbitrary variable proportional to the number of qubits (the authors choose $t = \frac{n}{3}$. $|\psi_j\rangle$ is a Haar-random single qubit quantum state.

The list of classical algorithms (implemented in scikit-learn) that the authors considered for the comparison are as follows: neural networks, linear kernels, Gaussian kernels, random forests, gradient boosting, and AdaBoost. All classical and quantum machine learning algorithms had their hyper-parameters optimized by a grid search. We note that some methods, such as random forest, do not have an associated kernel, but the authors still included them in the comparison for prediction accuracy.

First, the authors studied the relationship between the effective dimension d and the geometric difference g_{CQ} for each quantum model. In theory, we would have to consider all possible classical models and take the one with the smallest geometric distance to our quantum model, but in practice, we only look at the geometric distance of the finite suite of well optimized machine learning models such as Gaussian and Linear SVMs. The authors find that, as expected, when d becomes large, g_{CQ} decreases rapidly for the quantum kernel, which is in contrast to projected quantum kernels which can maintain a sizable geometric difference even as d becomes large. If we recall from our discussion of geometric difference, the formulae indicated that a large geometric difference is necessary for a quantum model to outperform a classical one. The empirical results validate this: classical machine learning outperforms or matches quantum models with a small geometric difference, while projected quantum kernels with a large geometric difference can outperform classical machine learning models on some of the datasets where function values come from a quantum neural network. For example, on one such dataset, the projected quantum kernel with the Hamiltonian evolution embedding achieved a prediction error of slightly less than 10%, while the best classical ML algorithm had a prediction error of around 13%.

Finally, the authors tested their algorithms on the engineered datasets that are explicitly designed to saturate inequality (4.10) between the classical and projected quantum models. In other words, these datasets are specifically designed to study the effect of the geometric difference between projected quantum kernels and classical models. We will not discuss how these datasets were constructed, but the algorithm is given in Appendix F of [4]. The authors find that as this geometric difference increases, the difference in performance between the projected quantum kernel and the standard quantum kernel and classical models increases as well, culminating in a difference of more than 20% in prediction error for the dataset with the largest geometric distance! This is an extremely significant result, as it is the first evidence of such a large separation in prediction accuracy between quantum and classical models.

5 Conclusion

The Huang et al. [4] result, along with the other manuscripts surveyed in this report, give us a roadmap towards obtaining practical advantages with quantum machine learning. The quantum recommendation algorithm and dequantization results showed us that as a general rule of thumb, relying on exotic data access models as the source of exponential advantages does not work; the algorithm itself has to be the source of the advantage. We then saw that kernel methods that don't assume any unconventional data models are a good candidate for quantum advantages if the dataset contains correlations that are hard to learn classically. Section 4.1 presents heuristic methods and motivation towards that end, while Section 4.2 gives a rigorous proof of an exponential advantage for the DLP problem, even though the problem is not practically motivated. Finally, Section 4.3 showed us that we have to be wary of the power of data in machine learning, as that can quickly transform a classically hard problem into an easy one.

We also introduced a useful theoretical foundation for quantum kernel methods that can be expanded on and used for empirical evidence and validation of new results. The question of whether QML is worth applying in practice over classical ML is certainly not settled, but these promising results will hopefully make for an exciting and useful research topic over the coming years.

6 Appendix

6.1 Rigorous Separation between BPP, BQP, and Classical Algorithms that Learn from Data

A Complexity Class for Classical Algorithms that Learn from Data

Let's define a complexity class for classical algorithms that learn from data. This part is simple: the complexity class is just the bounded error probabilistic polynomial time (BPP) class with data available. More rigorously, a language L is in this complexity class if we have the following:
There exists a probabilistic Turing Machine that takes in an input x of size n along with a dataset $D = \{(x_i, y_i)\}_{i=1}^{m}$, where $y_i = 1$ if $x_i \in L$ and 0 otherwise, with $m = \text{poly}(n)$, and we require that this probabilistic turing machine runs in polynomial time on all inputs, that it outputs 1 on input x with probability $p > \frac{2}{3}$ if $x \in L$, and that it outputs 1 with $p < \frac{1}{3}$ if $x \notin L$. We also require that the training data in D be sampled independently according to some distribution χ and we assume that this sampling can be done in polynomial time on a classical Turing machine.

Proof of the Complexity Separation

The authors first give a simple proof that this new complexity class for classical algorithms that learn from data (which we will refer to as C_{ML} from now on) is included in or equal to P/poly. They then rigorously prove that there exists a separation between C_{ML} and BPP (and also BQP). The idea behind the proof is very simple, and we repeat it here because it is very illuminating. Consider some undecidable unary language $L_{\text{hard}} = \{1^n\}, n \in A$ for some subset A of the natural numbers. Let L_{easy} be some easy language in BPP, and assume that for every input size n, $\exists\ a_n \in L_{\text{easy}}$ and $\exists\ b_n \notin L_{\text{hard}}$. We define a new language L in the following way: for every n, if 1^n is in L_{hard}, add every $x \in L_{\text{easy}}$ with $|x| = n$. Otherwise, if $1^n \notin L_{\text{hard}}$, add every $x \notin L_{\text{easy}}$ with $|x| = n$. Is this language in BPP? A simple proof by contradiction shows that it is not. Assume that the problem of finding out whether $x \in L$ is in BPP. Then we can do the following. Take some given x with $|x| = n$, and check if it is in L. If it is, then check if is in L_{easy}. If it is, then by the definition of L, $1^n \in L_{\text{hard}}$, and if it is not, then $1^n \notin L_{\text{hard}}$. But this means that we can decide the undecidable language L_{hard}, a contradiction! So the problem of finding out where $x \in L$ is not in BPP. Since L_{hard} is undecidable, the same result holds for BQP. We will now show an algorithm that, given data, can decide whether $x \in L$ in C_{ML}.

Suppose that for every n, we are given a single data point of the form (x_0, y_0), where $|x_0| = n$ and y_0 indicates whether x_0 is in L or not. Then, if we are given some x and we want to check if it is in L, we simply look at the corresponding data tuple (x_0, y_0) where $|x| = |x_0|$ that we have. If $x_0 \in L$, then $x \in L$, by definition, and if not, $x \notin L$. Thus, we have a rigorous separation between C_{ML} and BPP and BQP.

7 References

[1] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016. 2, 3

[2] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 217–228, 2019. 2, 4

[3] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *arXiv preprint arXiv:2010.02174*, 2020. 2, 6, 7, 8

[4] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *arXiv preprint arXiv:2011.01938*, 2020. 2, 9, 10, 14

[5] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009. 2

[6] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015. 3

[7] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014. 4

[8] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013. 4

[9] Ewin Tang. Quantum-inspired classical algorithms for principal component analysis and supervised clustering. *arXiv preprint arXiv:1811.00414*, 2018. 4

[10] András Gilyén, Seth Lloyd, and Ewin Tang. Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension. *arXiv preprint arXiv:1811.04909*, 2018. 4

[11] Nai-Hui Chia, András Gilyén, Tongyang Li, Han-Hsuan Lin, Ewin Tang, and Chunhao Wang. Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 387–400, 2020. 4

[12] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004. 5

[13] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019. 6, 7