A Generalisation of Classically Intractable Learning Problems that are Solvable with Quantum Machine Learning

Saleh Naghdi

A thesis in fulfilment of the requirements for the degree of

Bachelor of Science (Honours)

School of Physics

Faculty of Engineering

The University of New South Wales

2022

Contents

Chapter 1

Literature Review

1.1 Introduction

It was a machine learning (ML) algorithm that, in 2020, finally cracked the famous problem of predicting the 3D structure of proteins [1]. Unsurprisingly, in the era of big data, machine learning has become of great utility for being able to leverage data to build statistical models that make accurate predictions [2]. Paralleling this growing investment in machine learning is the nascent field of quantum computing which can promise boosts in efficiency and performance of algorithms that are fundamentally inaccessible by classical computing [3]. The possibility for this *quantum advantage* has ushered in a great incentive to develop quantum analogues for otherwise classical machine learning algorithms, the process of which may colloquially be referred to as quantisation [4].

The gamut of QML models in the literature spans widely but it can ultimately be divided into two main categories depending on the input model used for loading the classical data into the quantum computer. These include QRAM- (quantum memory) and kernel-based methods [5]. The main distinction between these two is that QRAM is a hypothetical input model analogous to classical RAM that is proposed for future large-scale quantum computers, whereas kernel methods can be employed on today's Noisy Intermediate-Scale Quantum (NISQ) computers.

Most notable among those who pioneered the QRAM-based QML methods are Lloyd *et al.*, who in 2014 introduced an exponentially faster, *quantum* version of Principal Component Analysis (PCA) — a widely used feature extraction technique that finds the orthogonal axes along which the dataset displays maximum variance [6]. qPCA was the first algorithm to have provided an efficient circuit implementation for the matrix exponential operator $U = e^{i\rho t}$ of a density matrix ρ, using it to exponentiate the covariance matrix Σ of a dataset which was later proved possible to encode as a density matrix ρ provided a QRAM input model [16]. Prior to qPCA, many fundamental algorithms such as Quantum Phase Estimation would require that the matrix exponential U already be provided as a black box or *quantum oracle* [17]. Lloyd *et al.*'s work led to the subsequent proliferation of QRAM-based QML models (Table 1.1) which employed this breakthrough finding, such as the Quantum Support Vector Machine (QSVM) which required exponentiating the kernel matrix K of a dataset, or quantum Discriminant Analysis (qDA)

Model[†]	Complexity		QRAM	Objective
	Classical	Quantum		
qPCA [6]	$\mathcal{O}(d^2N + d^3)$	$\mathcal{O}(\epsilon^{-3}\log(d)$	Yes	Obtain the k largest eigenvectors of the covariance matrix
qSVM [7]	$\mathcal{O}(\log\left(\frac{1}{\epsilon}\right)\text{poly}(N, d))$	$\mathcal{O}(\log(N, d))$	Yes	Find the hyperplane that optimally separates two classes.
qDA [8]	$\mathcal{O}(Nd^2)$	$\mathcal{O}(\epsilon^{-3}\log(Nd))$	Yes	Invert the covariance matrix of the dataset
qSFA [9]	$\mathcal{O}(\min(N^2d, d^2N))$	$\mathcal{O}(\text{polylog}(N, d))$	Yes	Linearly transform dataset so that intraclass (interclass) distance is minimised (maximised)
qKPCA [10]	$\mathcal{O}(N^3)$	$\mathcal{O}(\epsilon^{-1}\text{polylog}(Nd))$	Yes	Project dataset onto space spanned by k largest eigenvectors of kernel matrix
qDM [11]	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2\text{polylog}(N))$	Yes	Obtain the eigendecomposition of the transition matrix induced by the dataset's adjacency matrix
qNN [12]	Incommensurable		Optional	Optimise the angular parameters of a parameterised circuit to reduce classification error
qCNN [13]			No	
qKE-SVM [14, 15]	Provable classical intractability		No	Obtain kernel matrix of classically intractable but expressive feature map

[†] In descending row order, the full names of the models go as follows: Principal Component Analysis, Support Vector Machine, Discriminant Analysis, Slow Feature Analysis, Kernel Principal Component Analysis, Diffusion Model, Neural Network, Convolutional Neural Network, Quantum Kernel Estimation Support Vector Machine.

Table 1.1: **A comparison of the range of QML models found in the literature as distinguished by their dependence on the input model QRAM and their proposed quantum speedup.** We see that for QRAM-based models, there is a consistent trend of promising exponential speedup for classical ML algorithms that all involve linear algebra routines characterised by some central matrix (e.g. covariance matrix for qPCA; adjacency matrix for qDM). The last three rows on the other hand indicate non-QRAM-based methods, which include variational (parameterisable) methods used by Quantum Neural Networks (qNNs) and feature-map methods by qKE-SVM, both of which fall under the general category of kernel-based methods. qNNs offer no speedup and are different in design and use-case from their classical counterparts. Thus, it is important to note that the only *true* result indicating quantum supremacy in this table is that by QKE-SVM [15], as the speedup afforded by QRAM-based models depends on the existence of the as-yet speculative input model of QRAM.

which likewise required the exponential of the Linear Discriminant Analysis matrix [7, 8]. In later years, more general QRAM-based subroutines arose for exponentiating matrices that could not be stored as a density matrix by having them instead be stored as a block-diagonal unitary operator [18]. With this development came another explosion in QRAM-based QML models that required the exponential of more exotic matrices, including the derivative covariance matrix in quantum Slow Feature Analysis (qSFA), the non-linear kernel matrix in quantum kernel PCA (qKPCA), and transition matrix in the quantum Diffusion Model (qDM) [9–11]. Indeed, the emergence of QRAM-based methods is a story about the latest developments in the quantum algorithms for matrix exponentiation.

Kernel methods arose in 2018 in response to the over-reliance of QML methods on QRAM — which fundamentally remained speculative — as they could instead be applied on NISQ computers [14]. Then, in 2021, Liu *et al.* showcased the first instance of a learning problem that was not only efficiently solvable by a kernel-based QML model, but also *classically intractable* [15]. This immediately brought kernel-based methods into the limelight, as no other QRAM-based model had previously solved a classical intractable learning problem; at best they provided exponential speed-up on classical models that already solved tractable[1] problems. As a first of

[1]A problem is considered tractable if it can be computed by an algorithm in subexponential time O(poly(n))

its kind, this work revealed that in principle, there exist learning tasks that simply cannot be learned by a classical model but which can be by a quantum model — and a kernel-based one, at that.

Liu *et al.*'s learning problem was based on the Discrete Logarithm Problem (DLP) which is conjectured to be classically difficult to solve [17]. The quantum model that was able to learn it did not require QRAM but instead used classically nonsimulatable quantum *kernel* methods to reveal patterns in the dataset that are undetectable by a classical computer. From the perspective of computational learning theory, two important questions arise concerning this breakthrough result's relation to quantum complexity theory [19, 20]:

(i) Is there a more general class of classically intractable learning problems of which Liu *et al.'s* result is a special case? And if so,

(ii) Is there a quantum ML model based on kernel or QRAM methods that can learn these problems?

We investigate these questions by probing a set of classically intractable problems which the DLP belongs to called the Hidden Subgroup Problems [21]. We also take a look at the general design of QML models to date as informed by practices involving the use of QRAM and kernel methods.

Structure— This literature review is divided into three sections. In Section 1.2, we delve into the most relevant preliminary results in machine learning and quantum computing disciplines separately. With the background content established, in Section 1.3 we explore the ecosystem of QRAM-based quantised algorithms by beginning with QPCA and then moving on to more technical models and findings. Finally, in Section 1.4 we describe in detail the classically intractable DLP learning problem proposed by Liu *et al.* as well as the kernel-based ML model devised to solve it, with a look towards recontextualising the findings around a more general framework based on the Hidden Subgroup Problem.

1.2 Background material

To address both aims (i) and (ii), it is crucial to understand what current QML models look like. As QML is the combination of machine learning (ML) and quantum computing (QC), this section lays out the groundwork in each of those respective fields necessary for understanding the key QML models that will be discussed in later sections.

In addressing aim (i) specifically, we must analyse the types of problems that are solved by classical ML models and more efficiently by their quantised counterparts in order to understand what other learning problems can receive the same treatment. As for aim (ii), we must look at the *design* of QML models at the fundamental level of the key quantum subroutines they commonly use.

To this end, we first look at the common classical ML problem of feature extraction and a popular classical model used to solve it called Principal Component Analysis (PCA) which has also already been quantised. Then, we touch on two fundamental quantum algorithms that are used widely by QML models and that we expect a quantum PCA to use.

1.2.1 Machine learning principles for feature extraction [20]

Consider the case of classifying whether an image is that of a cat or dog. Mathematically, an *instance* of an image can be represented as a $d-$dimensional vector $\vec{x_i} \in \mathcal{X} = \mathbb{R}^d$ whose entries correspond to the d pixel values of the image, generally referred to as its *attributes.* One can consider there being some oracle function $f : \mathcal{X} \to \{\texttt{`cat'}, \texttt{`dog'}\}$ which correctly relates an image instance to its respective class $y_i = f(\vec{x_i})$. A set of such labelled image instances $\{(\vec{x_i}, y_i)|i \in [N]\}$ is then what comprises the *dataset*, which is often collected into a dataset matrix $X = (-\vec{x}_i-) \in \mathbb{R}^{N\times d}$ and labels $\vec{y}$. A machine learning model corresponds to a function $\hat{f}_{\vec{\theta}} : \mathcal{X} \to \{\texttt{`cat'}, \texttt{`dog'}\}$ parameterised by parameters $\vec{\theta}$. *Training* the model on a labelled dataset $(X, \vec{y})$ thus refers to obtaining the optimal parameters $\vec{\theta}_{opt}$ that best approximate the model to the inaccessible oracle $\hat{f}_{\vec{\theta}} \approx f$ by, for instance, minimising the least squared error:

$$\vec{\theta}_{opt} = \underset{\theta}{\operatorname{argmin}} \sum_{k=1}^{N} (f_\theta(\vec{x}_i) - y_i)^2 \,. \tag{1.1}$$

In the real world, it is often the case that the dataset under consideration is highly dimensional ($d \gg 1$), whereby data points becomes increasingly sparse with respect to the space that they inhabit. This makes the dataset less reliable to work with as the model on which it's trained is said to suffer from the *curse of dimensionality.* To remedy this, there are plenty of dimensionality reduction techniques which attempt to reduce the d raw attributes (e.g. pixel values) describing an instance to $k \ll d$ more expressive *features* (e.g. presence of whiskers) while preserving some property of the raw data. This process of identifying features from raw attributes is referred to as *feature extraction.* One feature extraction method which has seen recent developments in the quantum setting is that of Principal Component Analysis, which we will now discuss below.

1.2.1.1 Principal Component Analysis: a model for feature extraction

In the spirit of extracting useful features from the raw attributes of a data point, Principal Component Analysis (PCA) assumes that the most important features are those that show the highest level of variance across different instances of the dataset (Figure 1.1a) [22]. If one makes the assumption that these features can be constructed as a linear combination of the raw attributes, then the following theorem holds:

Theorem 1. *The unit vector $\vec{w}_1$ along which the variance of the dataset $X \in \mathbb{R}^{N\times d}$ is maximum is the largest eigenvector of its covariance matrix $\Sigma = \frac{1}{N}X^{\intercal}X$. $\vec{w}_1$ is called the first principal component of X.*

Proof. Assume the dataset is centred. Let a feature be given by the projection of an instance $\vec{x}_i$ along a unit vector $\vec{w}_1$:

$$t_i = \vec{w}\cdot\vec{x}_i$$

The variance of this feature across the instances is thus given by:

$$\text{Var}(t) = \sum_{i=1}^{N}(t_i)^2 \ = (\vec{w}\cdot\vec{x}_i)^2 = ||X\vec{w}||^2 = \vec{w}^{\intercal}X^{\intercal}X\vec{w} = \frac{\vec{w}^{\intercal}X^{\intercal}X\vec{w}}{\vec{w}^{\intercal}\vec{w}} \tag{1.2}$$

The optimal unit vector, $\vec{w}_1$ is that which maximises the variance of the feature across the instances, that is,

$$\vec{w}_1 = \underset{\vec{w}}{\text{argmax}}\{\text{Var}(t)\} = \underset{\vec{w}}{\text{argmax}}\{\frac{\vec{w}^{\intercal}X^{\intercal}X\vec{w}}{\vec{w}^{\intercal}\vec{w}}\} \tag{1.3}$$

This is the famous Rayleigh quotient, whose solution is the largest eigenvector of the central matrix $X^{\intercal}X$ which is the correlation matrix Σ up to a scalar fact. Since eigenvectors of a matrix are the same up to scalar multiplication of the matrix, we have that $\vec{w}_1$ corresponds to the normalised eigenvector of Σ. □

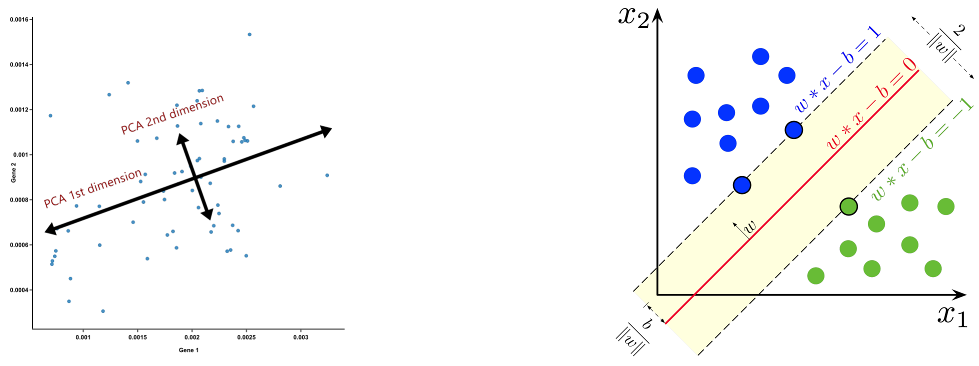
Corollary 1.1. *It can be shown by induction that the kth principal component $\vec{w}_k$ of a dataset corresponds to the kth largest eigenvector of its covariance matrix [22].*

Theorem 1 is one of the many instances in ML where a learning problem (finding features of maximum variance) actually reduces to a well-known linear algebra problem (the eigenvalue problem for the covariance matrix of the dataset). The main takeaway from this is that a quantum PCA would require a quantum subroutine that can solve the eigenvalue problem exponentially faster. The next section covers one such quantum subroutine, and another that solves the related linear algebra task of inverting a matrix.

1.2.2 Common quantum subroutines used in quantum machine learning

1.2.2.1 Quantum Phase Estimation (QPE)

One of the most common quantum subroutines used in QML is the Quantum Phase Estimation (QPE) algorithm which solves the famous eigenvalue problem [17]. Suppose a unitary operator



(a) (b)

Figure 1.1: **Illustration of Principal Component Analysis and Support Vector Machine for 2D datasets.** In 1.1a, PCA is able to find the pair of orthogonal axes along which the dataset has the largest variances; notice how the spread of the distribution is widest along the first principal component, followed by the second. These correspond to the semi-major and semi-minor axes of the ellipse of best fit to the dataset. In this way, PCA is thought of as fitting an ellipsoid to the dataset. In 1.1b, we see SVM has found a *large-margin* hyperplane with parameters $(\vec{w}, b)$ partitioning the binary labeled dataset. An unseen data point is classified blue (green) if it lies to the left (right) of the hyperplane. Note that the large-margin criterion insists the optimal hyperplane have its margin $2/\|\vec{w}\|$ maximised.

U has an eigenvector $|u\rangle$ with corresponding eigenvalue $e^{2\pi i\varphi}$[2]. Then the QPE algorithm is able to compute φ provided the eigenvector $|u\rangle$. That is,

$$|0\rangle\,|u\rangle \xrightarrow{QPE} |\varphi\rangle\,|u\rangle \tag{1.4}$$

where $|\varphi\rangle^l$ is an $l-$bit representation of the eigenvalue φ for some choice of l.

The QPE circuit relies on having access to $C\text{-}U^j$ (i.e. controlled-U^j) operators, which will be used to construct $\sum_{k=0}^{l-1} |k\rangle\langle k| \otimes U^k$. However, it suffices to know the circuit for U, from which both controlled-U and thus controlled-U^j can be implemented. So, with U treated as a readily available oracle, the QPE algorithm is described as follows:

Algorithm 1 Quantum Phase Estimation (QPE)

```
    **Input** An oracle for performing $C\text{-}U^j$ operations for any integer $j$, an eigenvector of $U$ $|u\rangle$ with eigenvalue $e^{2\pi i\varphi}$, and a register of $l$ qubits $|0\rangle$.
    **Output** An $l$-bit approximation of $\varphi$ stored as $|2^l\varphi\rangle = |\varphi_1\varphi_2...\varphi_l\rangle$ in $\mathcal{O}(l^2)$.
1: $\xrightarrow{H^{\otimes l}\otimes I} \frac{1}{2^{l/2}}\sum_{k=0}^{2^l-1} |k\rangle\,|u\rangle$
2: $\xrightarrow{\sum_{k'=0}^{2^l-1}|k'\rangle\langle k'|\otimes U^{k'}} \frac{1}{2^{l/2}}\sum_{k'=0}^{2^l-1} e^{2\pi i\varphi k'}\,|k'\rangle\,|u\rangle$
3: $\xrightarrow{QFT^{\dagger}_{2^l}\otimes I} |2^l\varphi\rangle\,|u\rangle$
```

The circuit for this diagram is displayed in Figure 1.2. To see how the central operator corre-

[2]recall that the eigenvalues of a unitary matrix are all of the form $e^{2\pi i\theta}$ for $\theta \in [0, 1)$

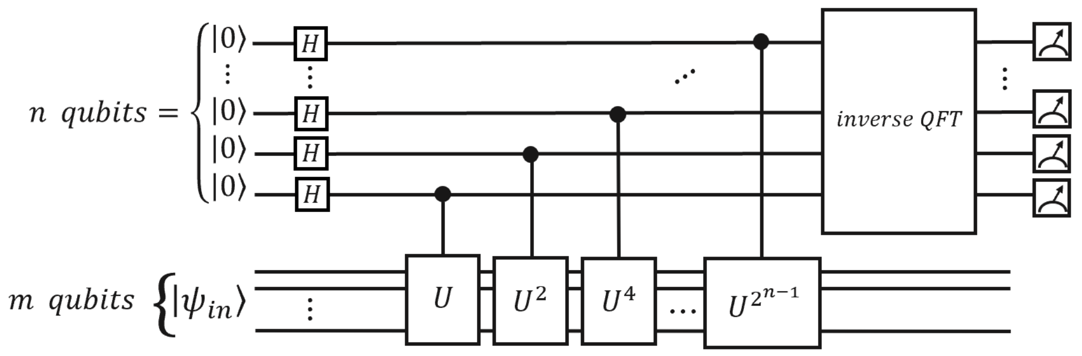


Figure 1.2: **The circuit implementation for Quantum Phase Estimation performed on an eigenvector $|\psi_{in}\rangle$ ($|u\rangle$ in main text) of a provided quantum oracle U with eigenvalue** φ The first register of n qubits (l in the main text) is the eigenvalue register which will store the n-qubit representation of the eigenvalue $\varphi = 0.\varphi_1\varphi_2...\varphi_l$, and the second register stores the eigenvector input. The algorithm is divided into three steps: making a maximum superposition ($H^{\otimes n}$), phase-kickback to produce $e^{2\pi i\varphi}$ (CU^j), and encoding the phase into the state $|2^n\varphi\rangle$ ($QFT^\intercal$).

sponds to the desired operation $\sum_{k=0}^{l-1} |k\rangle\langle k| \otimes U^k$, observe that

$$\begin{aligned}
|j\rangle|\psi\rangle &\xrightarrow{C_l - U^{2^0}} |j\rangle U^{j_l 2^0} |\psi\rangle \\
&\xrightarrow{C_{l-1} - U^{2^1}} |j\rangle U^{j_l 2^0 + j_{l-1}2^1} |\psi\rangle \\
&\quad\vdots \\
&\xrightarrow{C_1 - U^{2^{l-1}}} |j\rangle U^{j_l 2^0 + j_{l-1}2^1 + \dots j_1 2^{l-1}} |\psi\rangle = |j\rangle U^j |\psi\rangle .
\end{aligned}$$

The power of QPE is in its ability to compute the eigenvalues of any Hermitian matrix provided the circuit for its matrix exponential can be efficiently implemented (which we show in Section 1.3 is often achievable). Suppose A is a Hermitian matrix whose eigenvectors $|u_j\rangle$ are known but not their corresponding eigenvalues λ_j (assumed to be > 0 for simplicity). Then substituting the matrix exponential $U = e^{2\pi i A/\lambda_{max}}$ as the oracle QPE enables the eigenvalues λ_j to be computed:

$$|0\rangle|u_j\rangle \xrightarrow{QPE} |\lambda_j\rangle|u_j\rangle \tag{1.5}$$

Without QPE, it is difficult to imagine any learning problems (i) based on the eigenvalue problem to be efficiently solvable by a QML model (ii).

1.2.2.2 The Harrow-Hassidim-Lloyd algorithm (HHL)

With a small alteration to QPE, we can construct the HHL algorithm which performs matrix inversion with exponential speedup [23]. As many QML models depend on solving linear equations, it is no wonder the HHL subroutine is used widely across QML [24].

Suppose that at the end of the QPE algorithm, an ancilla qubit initialised to $|0\rangle$ is added. Now using rotation gates conditioned on the eigenvalue register, we may produce the following state:

$$|2^l \varphi\rangle |u\rangle |0\rangle \to |2^l \varphi\rangle |u\rangle \left(\sqrt{1 - \frac{1}{\varphi^2}} |0\rangle + \frac{1}{\varphi} |1\rangle \right)$$

Performing inverse QPE on the eigenvalue register disentangles it from the eigenvector and allows us to discard it:

$$\xrightarrow{QPE^\dagger} |u\rangle \left(\sqrt{1 - \frac{1}{\varphi^2}} |0\rangle + \frac{1}{\varphi} |1\rangle \right) \tag{1.6}$$

How is this result even related to matrix inversion? Suppose we wanted to solve a linear system of equations $A|x\rangle = |b\rangle$, where $A \in \mathbb{R}^{n\times n}$ is a Hermitian matrix with eigenvectors $|u_j\rangle$ and corresponding eigenvalues λ_j. We have that

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=1}^{N} \beta_j A^{-1} |u_j\rangle = \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} |u_j\rangle, \tag{1.7}$$

where we've expressed $|b\rangle$ in the eigenbasis of $\mathbf{A}$. If $U = e^{2\pi i A/\lambda_{max}}$ is efficiently implementable, then from 1.5, running QPE and then 1.6 with input $|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$ produces the state:

$$\to \sum_{j=1}^{N} \beta_j |u_j\rangle \left(\sqrt{1 - \frac{1^2}{\lambda_j^2}} |0\rangle + \frac{1}{\lambda_j} |1\rangle \right)$$

which measured on outcome $|1\rangle$ of the ancilla qubit yields

$$\to \frac{1}{\sqrt{\sum_{k=1}^{N} |\beta_k/\lambda_k|^2}} \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} |u_j\rangle .$$

But from 1.7, this is precisely our solution $|x\rangle$ up to a normalisation factor.[3]

However, the question remains as to how one would efficiently prepare a circuit implementation for the matrix exponential $U_t = e^{iAt}$. In Section 1.3, we will see that by introducing the concept of quantum memory, we can efficiently construct U_t and more.

[3]Should $\|x\|$ ever be needed, it can be computed from this circuit alone as the square root of the probability of obtaining $|1\rangle$ on the ancilla qubit.

1.3 Quantisation based on quantum memory

In this section, we investigate the QML models that depend on a *quantum Random Access Memory* or QRAM [5]. The range of QML methods found in the literature is divided into two groups: QRAM- and kernel-based models. The dependency of certain models on QRAM stems from their requirement that classical data, such as the entries of a data point $\vec{x}_i$, be loaded into the quantum computer *in superposition.* Loading data in such a way is a requirement for many powerful quantum linear algebra subroutines on which QML methods often depend (such as matrix exponentiation) [23]. Because they constitute the majority of QML models in the literature, QRAM-based models need to be understood in order to help design novel QML models for new learning problems (ii).

In the following, we define QRAM by comparing it with classical RAM and discuss some of its immediate implications. Then, we describe two QRAM-based QML models that efficiently perform matrix exponentiation, an important subroutine. We conclude by discussing the limitations of QRAM, namely that it is still a theoretical architecture with no scalable physical realisation.

1.3.1 QRAM

The central tension that lies at the core of QML is how machine learning — a discipline involving huge datasets — can be discussed in the context of quantum computing which presently has no means for storing quantum states for later retrieval. The (hypothetical) solution to this issue is a speculative architecture that is analogous to classical memory known as quantum RAM (QRAM) [5].

All types of memory have three basic components: an array of memory cells, an address (input) register, and an output register. In classical RAM, a memory address is given as input and the data stored in that memory cell is returned to the output register:

$$\underbrace{[j]_a}_{address} \xrightarrow{RAM} [j]_a \underbrace{[D_j]_d}_{data} . \tag{1.8}$$

On the other hand, QRAM is able to take a *superposition* of addresses and efficiently return the data at each of those cells such that they are entangled with their addresses:

$$\underbrace{\textstyle\sum_j \lambda_j |j\rangle_a}_{\text{addresses}} \xrightarrow{QRAM} \textstyle\sum_j \lambda_j |j\rangle_a \underbrace{|D_j\rangle_d}_{\text{data}} . \tag{1.9}$$

In this way, QRAM is memory that is *addressable in superposition.* The bit string D_j stored at address j can represent any sort of classical data, such as the jth entry x^j of a numerical data point $\vec{x} \in \mathbb{R}^d$, in which case 1.9 gives a *basis encoding* of the vector $\vec{x}$:

$$|x\rangle_{basis} := \frac{1}{\sqrt{d}} \sum_{j=0}^{d-1} |j\rangle |x^j\rangle . \tag{1.10}$$

In fact, it can be shown that if the memory cells are arranged in a binary tree data structure [25], then even the more sophisticated *amplitude encoding* of $\vec{x}$ can be prepared in time $\mathcal{O}(\log d)$ using QRAM:

$$|x\rangle_{amplitude} := \frac{1}{\|\vec{x}\|} \sum_{j=0}^{d-1} x^j |j\rangle . \tag{1.11}$$

The amplitude-encoded form of a vector is so useful that from now onwards we will suppress its subscript, $|x\rangle := |x\rangle_{amplitude}$, and consider it to be the standard way of storing a data point as a quantum state.[4] Moreover, for a dataset of instances $\{\vec{x}_i\}$, it is also possible to show that their amplitude encodings are addressable in superposition too, enabling the operation $\sum_i |i\rangle |0\rangle \to \sum_i |i\rangle |x_i\rangle$ [18]. [5]

With this theoretical tool in our arsenal, we are positioned to quantise a variety of ML algorithms that are based on linear algebra techniques.

1.3.2 Quantum machine learning methods using quantum memory

1.3.2.1 Quantum Principal Component Analysis (qPCA)

In 2014, Lloyd *et al.* put forward the concept of quantum feature extraction algorithms with their design of a quantum version of Principal Component Analysis [6]. By assuming QRAM, Lloyd *et al.* showed the promise of an exponential speedup over classical PCA detailed in section 1.3. In addition to demonstrating a quantum speedup for a widely used algorithm, qPCA is also celebrated for providing an efficient circuit implementation for the matrix exponential $U_t = e^{i\rho t}$ of a density matrix ρ. The matrix exponential is an input to many quantum linear algebra subroutines like quantum matrix inversion (as discussed in Section 1.2) that we are likely to encounter in designing a model for a learning problem (i), as was the case for the QRAM-based models in Table 1.1.

In its most distilled form, the task of performing PCA on a dataset is just one of computing the eigenvectors of the covariance matrix with the largest eigenvalues (recall Theorem 1). qPCA is tasked with this same problem, but with the added burden of promising an exponential speedup. The crucial link that allows qPCA to attain just this, is realising that both the covariance matrix of a dataset (a classical construction) and the density matrix representing a mixed quantum state (a quantum construction) are symmetric positive definite matrices. Hence, up to normalisation, a covariance matrix Σ can be encoded as the density matrix of some ensemble of quantum states $\{\lambda_j, |p_j\rangle\}$, or adopting density matrix notation, $\rho = \sum_{j=1}^{N} \lambda_j |p_j\rangle\langle p_j|$ where $|p_j\rangle$ is an eigenvector with corresponding eigenvalue λ_j. Before diving into explaining the algorithm, it is worthwhile

[4]Indeed, we implicitly used this convention when introducing the state $|b\rangle$ in the HHL algorithm and $|u\rangle$ in the QPE algorithm. And while previously we assumed there to be an oracle for preparing $|b\rangle$, now we can insert QRAM and particularly 1.11 as the method of state preparation

[5]Be cautioned, this isn't just a rehash of 1.9. Here, $|x_i\rangle$ is an *amplitude encoding* of $\vec{x_i}$ as in definition 1.11 and not merely a bit string $|D\rangle_d$ encoding classical data.

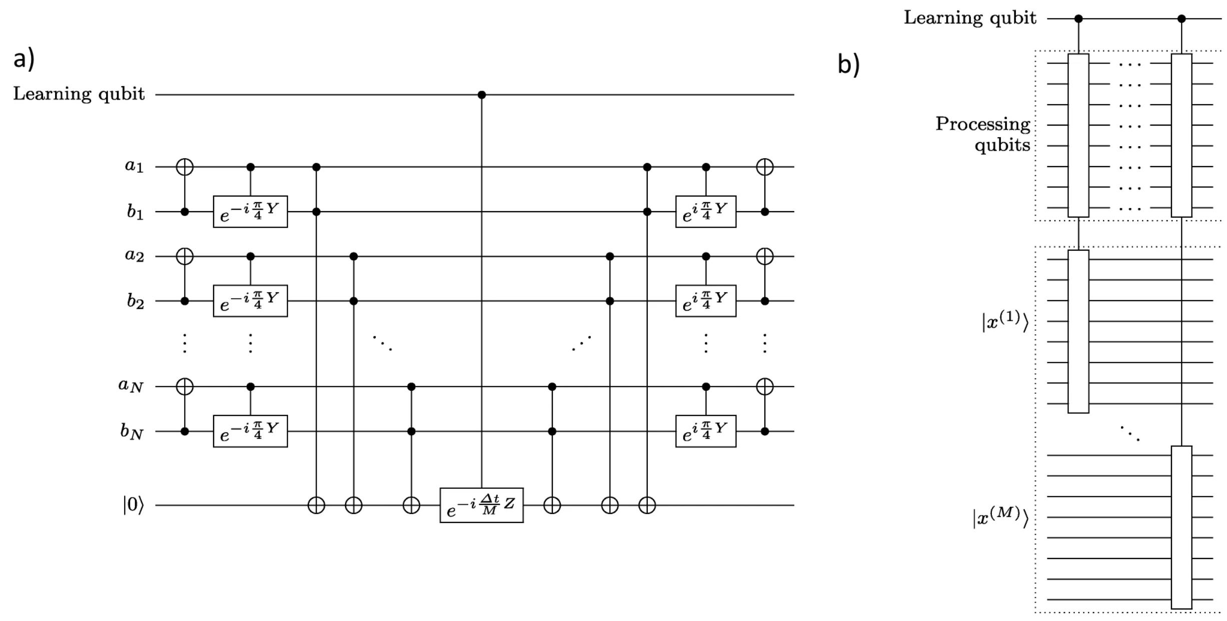


Figure 1.3: **Circuit diagrams illustrating (a) a single swap-rotation operation and (b) the overall density matrix exponentiation circuit $C - e^{i\rho t}$ used for the Quantum Principle Component Analysis (qPCA) algorithm outlined by Bromley and Rebentrost [26].**

first to compare the inputs and outputs of qPCA:

$$\underbrace{\sum_{j=1}^{N} \lambda_j |p_j\rangle\langle p_j|}_{\rho} \otimes |0\rangle\langle 0| \xrightarrow{QPCA} \sum_{j=1}^{N} \lambda_j |p_j\rangle\langle p_j| \otimes \left|\tilde{\lambda}_j\right\rangle\left\langle\tilde{\lambda}_j\right| . \tag{1.12}$$

The upshot of this output state is that measuring the eigenvalue register in the computational basis returns the state $|p_j\rangle\langle p_j| \otimes \left|\tilde{\lambda}_j\right\rangle\left\langle\tilde{\lambda}_j\right|$ with probability λ_j^2, such that eigenvectors with the largest eigenvalues are proportionally more likely to be obtained upon measurement. Indeed, this is *exactly* what is required by PCA.

The algorithm. The qPCA algorithm is just one application of QPE. It is not difficult to see that the output state 1.12 can be obtained if one performs QPE with the eigenvector register set to the density matrix $\rho = \sum_{j=1}^{N} \lambda_j |p_j\rangle\langle p_j|$ encoding the covariance matrix, and the oracle set to $U = e^{i\rho t}$.[6]

As mentioned earlier, the main contribution of the qPCA algorithm is its efficient circuit implementation of the matrix exponential U_t, which QPE otherwise treats as a black box. Lloyd *et al.* show that to simulate $e^{i\rho t}$ to accuracy ϵ, it is sufficient to perform $N = O(t^2\epsilon^{-1})$ two-qubit operations between N copies of ρ and the eigenvector register in QPE, which in our case is also initialised to ρ. A schematic of this implementation is illustrated in figure 1.3.

[6]If further convincing is required, consider how this is in exact analogy to setting $U_t = e^{iAt}$ in the HHL algorithm where A is a Hermitian matrix – and indeed, ρ is one such matrix.

But how are these $N+1$ copies of ρ produced? This is where the assumption of QRAM enters the picture [16]. Recall that an equivalent representation of the covariance matrix is $\Sigma = \frac{1}{N}X^{\intercal}X = \frac{1}{N}\sum_{j=1}^{N}\mathbf{x}_j\mathbf{x}_j^T$ so that as a density matrix, it is just as valid to load ρ as an ensemble $\{1/N, |x_j\rangle\}$ instead:

$$\rho = \frac{1}{N}\sum_{j=1}^{N}|x_j\rangle\langle x_j| \tag{1.13}$$

where we remind the reader that $|x_j\rangle$ is the amplitude encoding of data point $\vec{x}_j$. This mixed state can be obtained by first preparing the following pure state using QRAM,

$$|\psi\rangle = \frac{1}{\sqrt{N}}\sum_{j=1}^{N}|j\rangle\,|x_j\rangle\,, \tag{1.14}$$

and then tracing out (i.e. discarding) the first register:

$$\mathrm{Tr}_1[|\psi\rangle\langle\psi|] = \frac{1}{N}\sum_{i,j=1}^{N}\langle j|i\rangle\,|x_j\rangle\langle x_i| = \frac{1}{N}\sum_{j=1}^{N}|x_j\rangle\langle x_j| = \rho. \tag{1.15}$$

In summary, we obtain $e^{i\rho t}$ by performing operations on copies of ρ which in turn are produced by using QRAM. This general technique can be applied also to density matrices ρ encoding data other than the covariance matrix Σ. For example, if we were to trace out the second register in 1.14 instead, we would get a very closely related matrix called the kernel matrix K in its normalised form

$$\mathrm{Tr}_2[|\psi\rangle\langle\psi|] = \frac{1}{N}\sum_{i,j=1}^{N}\langle x_j|x_i\rangle\,|j\rangle\langle i| = \frac{K}{\mathrm{Tr}[K]} \tag{1.16}$$

which we can exponentiate via the same method as above. The versatility of this matrix exponentiation algorithm makes it a promising tool for the design of new QML models that involve unique matrices, such as the kernel matrix which the next section reveals leads to another quantised algorithm.

1.3.3 qSVM

The main insight of QPCA on how exponentiate matrices able to be encoded as a density matrix has led to many advancements in QML models. A prime example of this is the quantum Support Vector Machine (qSVM) which harnesses matrix exponentials to perform a necessary matrix inversion step with exponential speedup [7].

The Support Vector Machine (SVM) is a binary classifier trained on binary datasets of the form $\{(\vec{x}_i, y_i) : \vec{x}\in\mathbb{R}^d,\ y_i\in\{+1,-1\}\}$ with the objective of finding a *large-margin* hyperplane in $\mathbb{R}^d$ that best separates the +1-labelled training instances from those labeled with -1 (see Figure 1.1b). Since a general hyperplane is defined by its normal vector $\mathbf{w}$ and offset $b/\|\mathbf{w}\|$, the task of training a SVM on a binary labelled dataset reduces to obtaining hyperplane parameters $(\mathbf{w}, b)$. Classifying an unseen data point $\vec{x}$ with the trained SVM corresponds to seeing on which side of the hyperplane it falls:

$$y(\vec{x}) = \mathrm{sign}(\vec{w}\cdot\vec{x} + b). \tag{1.17}$$

While training a classical SVM takes time $\mathcal{O}(\text{poly}(N,d))$, Rebentrost *et al.*'s quantum SVM (qSVM) attains an exponential speedup of $\mathcal{O}(\text{polylog}(N,d))$ by performing quantum matrix inversion.

The algorithm. The exponential speedup in training a qSVM leverages the fact that the SVM problem can be reformulated as the linear equation

$$\underbrace{\begin{bmatrix} 0 & \vec{1}^T \\ \vec{1} & K+\gamma^{-1}\mathbb{1} \end{bmatrix}}_{F} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{y} \end{bmatrix}. \tag{1.18}$$

Here, $\vec{\alpha}$ are *multipliers* that recover $\vec{w} = \sum_{j=1}^{N} \alpha_j \vec{x}_j$, and $K \in \mathbb{R}^{N\times N}$ is the inner product or *kernel*[7] matrix with entries $K_{ij} = \vec{x}_i \cdot \vec{x}_j$. Additionally, γ is a user-specified term that relaxes the condition that the hyperplane fully partition the class labels, in exchange for improved generalisability on test results.

The task is to solve the equation $\hat{F}^{-1}|\tilde{y}\rangle$ where $\hat{F} = \frac{F}{\text{Tr}[F]}$. From the HHL algorithm, this can be achieved if the matrix exponential $U_t = e^{it\hat{F}}$ is efficiently implementable. By recognising that $F = J + \gamma^{-1}\mathbb{1} + K$ [8] and absorbing $\Delta t \leftarrow \text{Tr}[F]\Delta t$, we have by the Lie product formula

$$e^{i\Delta t\hat{F}} \simeq e^{i\Delta tJ} e^{i\Delta t\gamma^{-1}\mathbb{1}} e^{i\Delta tK} + \mathcal{O}(\Delta t^2). \tag{1.19}$$

The first two terms already have known efficient circuit implementations so that only $e^{i\Delta tK}$ remains [7]. But Equation 1.16 indicates that K can be encoded as a density matrix up to normalisation, allowing it to be efficiently exponentiated just as the covariance matrix was in qPCA. Putting it all together, we are able to produce the state

$$|b,\vec{\alpha}\rangle = \hat{F}^{-1}|\tilde{y}\rangle = \frac{1}{\sqrt{C}}(b|0\rangle + \sum_{j=1}^{N} \alpha_j |j\rangle). \tag{1.20}$$

Note that the computational basis states go up to $|N+1\rangle$ in this case. This concludes the training stage.

Now we perform classification on a test data point $\vec{x}$. Let us assume all data points are normalised for the sake of argument. By addressing our training dataset in superposition with $|b,\vec{\alpha}\rangle$, we get $|\tilde{u}\rangle = \frac{1}{\sqrt{C}}(b|0\rangle|0\rangle + \sum_{j=1}^{N}\alpha_j|j\rangle|x_j\rangle)$. Using QRAM, we can also generate state $|\tilde{x}\rangle = \frac{1}{\sqrt{D}}(b|0\rangle|0\rangle + \sum_{j=1}^{N}|j\rangle|x\rangle)$. Then using what is called a Swap Test protocol we can directly measure the value of $\langle\tilde{u}|\tilde{x}\rangle$ [27]. But observe that

$$\langle\tilde{u}|\tilde{x}\rangle \propto \sum_{j=1}^{N} \alpha_j \langle x_j|x\rangle + b = \vec{w}\cdot\vec{x} + b \tag{1.21}$$

So the sign of 1.21 directly gives us the classification 1.17: $y(\vec{x}) = \text{sign}(\langle\tilde{u}|\tilde{x}\rangle)$.

[7]this matrix will be making a reappearance in the next section as well, so keep a watchful eye on it!

[8]with a slight abuse of notation where the latter two terms have been overloaded to mean their natural extension to N+1 dimensions

1.3.4 More recent findings

There are several shortcomings to assuming QRAM as an input model. The most obvious issue is the fact that QRAM presently remains a speculated architecture that is yet to experimentally realised.

However, even if QRAM is physically realisable, lingering behind the claim of quantum advantage for algorithms such as qPCA and qSVM remains a suspicion that the source of these speedups lies not in clever circuit implementation, but rather a dependence on an input model that is over promising. Indeed, in 2019, Ewin Tang provided compelling evidence that by assuming a classical input model that was equivalent to qRAM, it may be possible to classically gain as much of a computational speedup for many QRAM-based QML models. Tang formulated a series of quantum-inspired classical algorithms which effectively *dequantised* the earlier works by Kerenedis who pioneered qSFA and quantum recommendation systems [4,28,29]. This result may just be rescued by the fact that the dequantised classical models, while polynomial in time, nonetheless from high orders compared to those typically found in their quantised counterparts. Given the currently dubious footing of the literature on QRAM-based methods, it is worthwhile to also consider kernel-based methods.

1.4 Quantisation based on kernel methods

We now depart from depending on a hypothetical input model such as qRAM to consider an approach that can be immediately applied on currently available NISQ computers. Recall that the qRAM input model performs what is known as amplitude encoding to encode a data point $\vec{x} \in \mathbb{R}^d$ described by d attributes as a quantum state,

$$\begin{aligned} |0\rangle^{\otimes \lceil \log_2 d \rceil} &\xrightarrow{qRAM(\vec{x})} \sum_{i=1}^{d} x_i |i\rangle & (1.22) \\ &:= |\vec{x}\rangle_{qRAM} & (1.23) \end{aligned}$$

However, encoding the entries of a vector as the amplitudes of a quantum state is only one possible encoding technique. A more general input model can be considered to be an arbitrary parametrisable unitary gate, or *variational circuit* $V(\vec{\omega})$, which acts on m qubits to produce the following map [30]:

$$\begin{aligned} |0\rangle^{\otimes m} &\xrightarrow{V(\vec{x})} V(\vec{x}) |0\rangle & (1.24) \\ &:= \ |V(x)\rangle \,. \end{aligned}$$

Note that in general, $|V(x)\rangle \neq |\vec{x}\rangle_{qRAM}$ and further, $m \neq \lceil \log_2 d \rceil$ which would otherwise be required for amplitude encoding.

The map in Eq. 1.24 is known as a feature map as it takes a classical vector $\vec{x}$ and embeds it into a Hilbert space $\mathbb{C}^{2^m}$ known as a feature space:

$$\vec{x} \in \mathbb{R}^d \mapsto |V(x)\rangle \in \{\, |V(\vec{\omega})\rangle \;|\, \forall \vec{\omega} \,\} \subset \mathbb{C}^{2^m}. \tag{1.25}$$

A judicious choice for a feature map can embed the classical dataset $\{\vec{x}_i\}$ into a feature space that better partitions the instances with respect to their class labels, allowing for improved classification. In this sense, the feature space parses the raw attributes of the data into more informative *features*.

From a mathematical perspective, it is often the case that the similarity of two data points $\vec{x}_1$ and $\vec{x}_2$ is measured by their inner product $\langle \vec{x}_1, \vec{x}_2 \rangle_{\mathbb{R}^d} = \sum_{i=1}^{d} x_{1,i} x_{2,i}$. In the quantum feature space, the inner product is defined by the Hilbert-Schmidt operator:

$$\langle |\vec{x}_1\rangle \,, |\vec{x}_2\rangle \rangle_{\mathcal{V}} = |\, \langle V(\vec{x}_1) | V(\vec{x}_2) \rangle \,|^2 \tag{1.26}$$

Thus, a good feature space would be one in which the inner product $\langle |\vec{x}_1\rangle \,, |\vec{x}_2\rangle \rangle_{\mathcal{V}}$ is even more discriminative of similarity. Because it is often difficult to calculate inner products in a feature space, many models which make use of it require a *kernel function* $K(\vec{x}_1, \vec{x}_2) = \langle |\vec{x}_1\rangle \,, |\vec{x}_2\rangle \rangle_{\mathcal{V}}$ which takes any two raw data points as input and immediately calculates their inner products without explicitly having to map the data points to the feature space first. This saves the cost of mapping a low-dimensional raw data point to a high- (and potentially infinite) dimensional

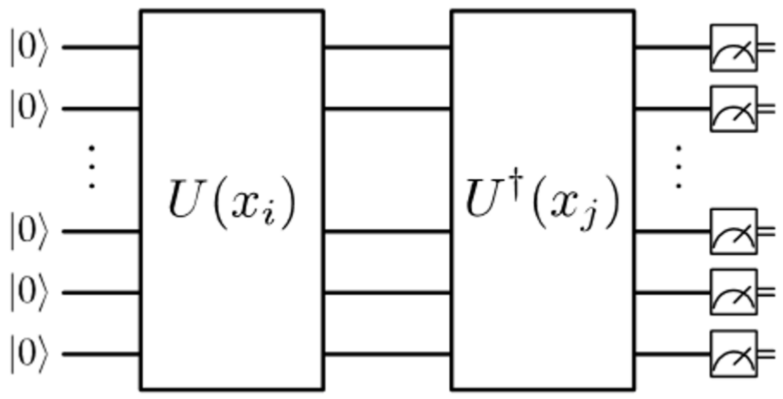


Figure 1.4: **General design of a quantum circuit serving as the kernel function of the map $\mathbf{x} \mapsto |\mathbf{x}\rangle_{\mathbf{U}}$ as defined in 1.29**. It is sufficient to measure the probability of measuring $|0\rangle$ to obtain the inner product between pairs of data points in the feature space.

space first before calculating the inner product. Calculating the feature-space inner product between all pairs of points in the dataset then yields the kernel matrix K,

$$\begin{aligned} K &:= (K_{ij}) && (1.27) \\ &= (K(\vec{x}_i, \vec{x}_j))\,, && (1.28) \end{aligned}$$

which is the main input to all kernel-based machine learning models.

The efficiency and practicality of mapping a data point to a more expressive feature space thus hinges on having access to a kernel function that is able to efficiently compute inner products in the feature space. This explains why quantum kernel methods have enjoyed a growth in popularity: kernel functions for $\mathcal{V}$ (Equation 1.25) have a canonical definition,

$$|\langle V(\vec{x}_1)|V(\vec{x}_2)\rangle|^2 = |\langle 0|V^\dagger(\vec{x}_1)V(\vec{x}_2)|0\rangle|^2\,, \tag{1.29}$$

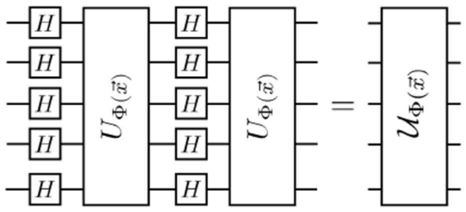
which simply corresponds to evaluating the probability of measuring $|0\rangle$ on the state $V^\dagger(\vec{x}_1)V(\vec{x}_2)|0\rangle$ whose circuit representation is shown in Figure 1.4 [14].

The next section demonstrates how a quantum feature map can be used to solve a classification problem to perfect accuracy.

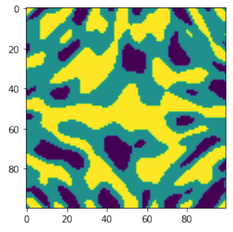
1.4.1 Results demonstrating quantum supremacy

1.4.1.1 Proof of Concept

To motivate the concept of a feature map, we consider a QML model that makes use of such a principle [14]. Note that this method will just be a proof of concept as there is no proven exponential speedup.



(a)



(b)

Figure 1.5: **Circuitry and visualisation of an artificially generated dataset used to showcase the power of feature maps**. b) Visualisation of the artificially generated dataset displaying the linearly inseparable distribution of the data. Yellow regions correspond to data points with -1 labels, and vice-a-versa for navy blue regions and +1 labels. a) A parameterised circuit used in the generation of the artificial dataset that is later recognised as a feature map.

The model will be trained on an artificially generated dataset which consists of two-dimensional vectors $\vec{x} \in [0, 2\pi]^2$ each of which is labeled either $+1$ or -1 according to the following hidden rule:

$$g(\vec{x}) = \langle 0| U^\dagger(\vec{x}) W^\dagger (Z_1 \otimes Z_2) W U(\vec{x}) |0\rangle \tag{1.30}$$

$$f(\vec{x}) = \begin{cases} +1, & \text{if } g(x) \geq 0.3 \\ -1, & g(x) \leq -0.3 \end{cases} \tag{1.31}$$

In this definition, $V \in SU(4)$ is a random unitary gate and $U(\vec{x})$ a parameterised circuit that takes a data point as input and is defined in Figure 1.5a. A visualisation of this dataset can be seen in Figure 1.5b where it is immediately clear that the data points are distributed in such a way that they are not linearly separable with respect to class label, that is, able to be separated by a linear boundary. This indicates that the model to be used must be more complex than a simple linear model.

In much the same way that the dataset was generated, one classifier for this toy dataset can be of the following form:

$$\tilde{g}(\vec{x}, \vec{\theta}) = \langle 0| \tilde{W}^\dagger(\vec{x}, \vec{\theta}) (Z_1 \otimes Z_2) \tilde{W}(\vec{x}, \vec{\theta}) |0\rangle \tag{1.32}$$

$$f_\theta(\vec{x}) = \begin{cases} +1, & g(x, \theta) \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{1.33}$$

where $\tilde{W}(\vec{x}, \vec{\theta})$ is a parameterisable (i.e. variational) circuit which takes both a data point $\vec{x}$ and variational parameters $\vec{\theta}$ as input. This classifier can be made more accurate if it is additionally assumed that the circuit $U(\vec{x})$ which was used to create the dataset is known a priori; note, however, that we do not assume the same for the random unitary V. As a result of this assumption, we make the substitution $\tilde{W}(\vec{x}, \vec{\theta}) \to \tilde{W}(\vec{\theta}) U(\vec{x})$, and the classifier becomes:

$$\tilde{g}(\vec{x}, \vec{\theta}) = \langle 0| U^\dagger(\vec{x}) \tilde{W}^\dagger(\vec{\theta}) (Z_1 \otimes Z_2) \tilde{W}(\vec{\theta}) U(\vec{x}) |0\rangle \tag{1.34}$$

The task of training this dataset then involves finding the optimum variational parameters $\vec{\theta_{opt}}$

Model	Accuracy
Linear SVC	0.58
Nonlinear SVC	0.63
Logistic Regression	0.58
Neural Network	0.63
Random Forest	0.70
QKE-SVM (this paper)	**1.0**

Table 1.2: **Various ML models on an artificial dataset being outperformed by the quantum SVC discussed in this section.**

that minimise the error of the classifier on the training dataset:

$$\vec{\theta}_{opt} = \min_{\vec{\theta}} \left(\frac{1}{|T|} \sum_{\vec{x} \in T} I\Big(\tilde{f}_\theta(\vec{x}) \neq f(\vec{x})\Big) \right). \tag{1.35}$$

where I is the indicator function.

When applied to this dataset, Table 1.2 showcases that this model outperforms classical models such as the Support Vector Machine by a considerable degree – even when trained on fewer data points. This is, of course, attributable to the contrived nature of the dataset being based on expectation values of quantum observables which is in fact what this quantum classifier is based on as well.

The classifier can actually be reformulated in a way that truly captures its connection with feature maps. Consider the space $\mathcal{V} = \mathbb{C}^d \times \mathbb{C}^d$ equipped with the trace inner product $\langle A, B \rangle_{\mathcal{V}} := Tr[AB]$ and a choice of orthogonal basis $\mathcal{B} = \{P_\alpha\}$ which consist of d-dimensional Pauli matrices satisfying $\langle P_\alpha, P_\beta \rangle_{\mathcal{V}} = d\delta_{\alpha,\beta}$. Any vector $A \in \mathcal{V}$ has the following representation with respect to the Pauli basis:

$$A = \frac{1}{d} \sum_\alpha Tr[AP_\alpha] P_\alpha := \frac{1}{d} \sum_\alpha A_\alpha P_\alpha \tag{1.36}$$

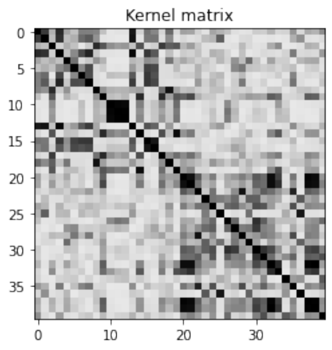


Figure 1.6: **The kernel matrix of the feature map $x \mapsto |U(x)\rangle$ revealing two clusters each corresponding to one of the two class labels**. This indicates that the feature space space has the capacity to discriminate between different data points.

in which case, A can be uniquely identified with its basis coefficients (A_α). In this vector-space representation, the classifier f takes up the following definition:

$$\begin{aligned}
\tilde{f}_\theta(x) &= \text{sign}(\langle 0| \tilde{W}^\dagger(\vec{x},\vec{\theta})(Z_1 \otimes Z_2)\tilde{W}(\vec{x},\vec{\theta})|0\rangle) \\
&= \text{sign}(Tr[\langle 0| \tilde{W}^\dagger(\vec{x},\vec{\theta})(Z_1 \otimes Z_2)\tilde{W}(\vec{x},\vec{\theta})|0\rangle]) \\
&= \text{sign}(Tr[\Big(\tilde{W}^\dagger(\vec{\theta})(Z_1 \otimes Z_2)\tilde{W}(\vec{\theta})\Big)\Big(U(\vec{x})|0\rangle\langle 0| U^\dagger(\vec{x})\Big)]) \\
&= \text{sign}(Tr[\frac{1}{d}\sum_\alpha W_\alpha(\theta)P_\alpha \frac{1}{d}\sum_\beta U_\beta(x)P_\beta]) \\
&= \text{sign}(\frac{1}{d^2}\sum_{\alpha,\beta} W_\alpha(\theta)U_\beta(x)Tr[P_\alpha P_\beta]) \\
&= \text{sign}(\frac{1}{d}\sum_{\alpha,\beta} W_\alpha(\theta)U_\beta(x)\delta_{\alpha,\beta}]) \\
&= \text{sign}(\frac{1}{d}\sum_{\alpha} W_\alpha(\theta)U_\alpha(x)]) \qquad (1.37)
\end{aligned}$$

Equation 1.37 has a very geometric interpretation: $(W_\alpha(\theta_{opt}))$ represents the normal vector to a hyperplane separating $\mathcal{V}$ into two halves, on one side of which lie vectors $\{(U_\alpha(x)|f(x) = +1\}$, and on the other, $\{(U_\alpha(x)|f(x) = -1\}$. That there exists such a hyperplane in $\mathcal{V}$ implies that the map $x \mapsto (U_\alpha(x))$ which enabled the dataset to become linearly separable is in fact, not only a feature map by definition, but a discriminative one at that!

The discriminative quality of the feature map can be visualised by looking at the kernel matrix generated from it (figure 1.6), which reveals two clear clusters representing the two class labels. It should be noted that since a good feature map has been found, performing the optimisation stage of finding θ_{opt} on a quantum computer can instead be replaced with computing the kernel matrix K to feed as an input to a given classical kernel-based machine learning model.

1.4.1.2 A quantum feature map achieving quantum advantage

While the quantum feature map in the previous example is capable of learning a provided dataset to perfect accuracy, no formal proof has been given that forbids the existence of some equivalent classical learner that is able to perform just as well. This is hinted at by the range of accuracies attained by the classical classifiers in Table 1.3, the best of which obtains an accuracy as high as 70%.

In the spirit of achieving quantum advantage, what is needed is a quantum feature map that *provably* outperforms all classical learners on some dataset. This is precisely what was demonstrated by Liu *et al* in Ref. [15] for a different adhoc dataset based on the Discrete Logarithm Problem (DLP) which is widely conjectured to be classically intractable. The authors prove that if there were a classical classifier able to learn their constructed dataset, or more technically, *concept class*, then that classifier could also solve the DLP, thus contradicting its conjectured intractability. By producing a quantum feature map which they prove has a good max-margin hyperplane, they can consequently claim that their quantum classifier outperforms all classical learners. A high-level summary of this finding is shown in Figure 1.7b.

As the only current instance of end-to-end quantum machine learning achieving quantum advantage in the literature, the work of Liu *et al.* deserves a closer exploration which we will now cover in the following section.

1.4.1.3 A DLP-based dataset

The Discrete Log Problem (DLP) is formulated as follows [21]. Let p be a prime number and $\mathbb{Z}_p^* = \{1, ..., p-1\}$ the integers under multiplication modulo p. If g is a generator of $\mathbb{Z}_p^*$, then given a $y \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_p^*$ such that $y = g^x$ or equivalently, evaluate the discrete logarithm $\log_g(y) = x$. In summary,

DLP(p, g)
Input: A prime number p, a generator g of the set $\mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$, and any element $y \in \mathbb{Z}_p^*$
Question: What is $\log_g(y)$?

It is widely conjectured that there is no classical algorithm capable of efficiently evaluating $\log_g(y)$, whereas a quantum DLP solver does exist that is based on Shor's algorithm. This places the DLP within the Bounded-error Quantum Polynomial time (BQP) complexity class [31].

Given this definition of the DLP, Liu *et al.* define a dataset over the elements of $\mathbb{Z}_p^*$ which are labelled according to the following hidden binary labelling rule:

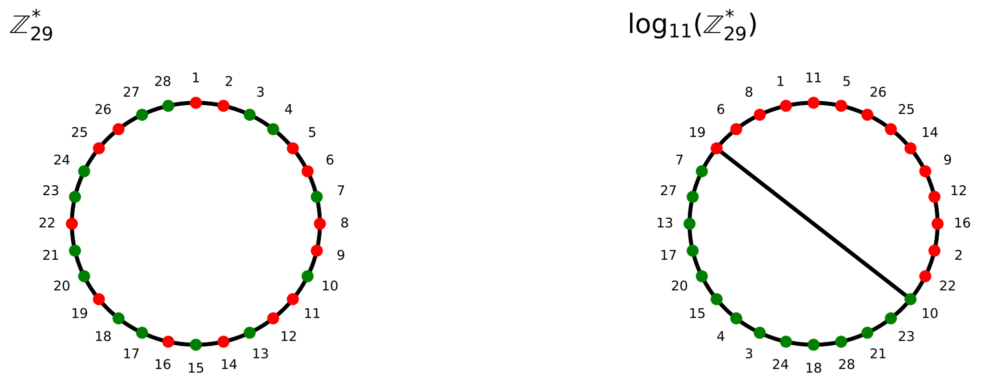
$$f_s(x) = \begin{cases} +1, & \text{if } \log_g(x) \in [s, s + \frac{p-3}{2}] \\ -1, & \text{else.} \end{cases} \tag{1.38}$$

for a choice of $s \in \mathbb{Z}_p^*$. Note that the interval in the definition sweeps half of the elements.[9]

To illustrate what this labelling rule corresponds to, it is instructive to first imagine the $p-1$ many elements of $\mathbb{Z}_p^*$ as being distributed evenly along a circle. While in this representation the class labels seem to be distributed randomly (Figure 1.7a) – and indeed, they are to a classical classifier – mapping each point to its discrete logarithm $x \mapsto \log_g(x)$ reveals the underlying classification rule, whereby the two labels are separated by a diameter line intercepting the point s, with $+1$ (-1) data points lying clockwise (anticlockwise) from s (Figure 1.7b).

The existence of this diameter partition already hints at the possibility of there being a hyperplane in some feature space that differentiates between the two class labels, and indeed, the next section shows that there is one such feature map. Importantly, however, it is that the feature map must necessarily be quantum which distinguishes this case from the previous.

[9]The addition in the interval is a slight abuse of notation as addition is not a native operation of $\mathbb{Z}_p^*$, so we have (p-1) + 1 = 1



(a) (b)

Figure 1.7: **Distribution of the class labels (red and blue) using the classification rule in 1.38 for $(\mathbf{p}, \mathbf{g}, \mathbf{s}) = (\mathbf{29}, \mathbf{11}, \mathbf{11})$ where p is a prime number, g is a generator of $\mathbb{Z}_p^*$ and s is a user-specified value related to the classification rule**

1.4.1.4 An expressive quantum feature map

Let $n = \lceil \log_2(p) \rceil$ be the minimum number of bits required to represent prime number p and define (y, g, s) as before. The quantum feature map is defined as

$$|y\rangle \mapsto \sum_{i=0}^{2^k-1} |y \cdot g^i \mod p\rangle := |C_{y,k}\rangle \,. \tag{1.39}$$

for a choice of a constant $k \leq n$. Unsurprisingly, the circuit for this feature map will make use of the DLP-solver which was shown in the previous section to organise the dataset into well-separated class clusters.

As a first step towards building the desired quantum feature map, consider the following function:

$$C_{y,k} : \{0,1\}^k \to \{0,1\}^n$$
$$C_{y,k}(i) = y \cdot g^i \mod p \,.$$

By using elementary multiplication modulo p – the circuit for which is well understood and efficient[10]– it is easy to implement the reversible operator

$$\hat{C}_{y,k} : \mathcal{H}^{\otimes 2n} \to \mathcal{H}^{\otimes 2n}$$
$$\hat{C}_{y,k} |i\rangle |0\rangle^{\otimes n} = |i\rangle |C_{y,k}(i)\rangle$$

where $|i\rangle$ uses the k least significant bits from the first register. Now using the DLP circuit from Ref. [17] the following operator can be reproduced:

$$U_y |C_{y,k}(i)\rangle |0\rangle^{\otimes k} = |i\rangle |C_{y,k}(i)\rangle \tag{1.40}$$

[10]Using the multiplication operator $U_{k,p} |l\rangle = |k \cdot l \mod p\rangle$ and exponentiation operator $V_{k,p} |l\rangle = |k^l \mod p\rangle$, one can at once see that $|C_{y,k}(i)\rangle = U_{y,p}^i V_{g,p} |i\rangle$ [17]

Finally, this enables the following superposition state to be obtained up to the addition and removal of auxiliary qubits:

$$|0\rangle \xrightarrow{H^{\otimes k}} \sum_{i=0}^{2^k-1} |i\rangle \xrightarrow{\hat{C}_{y,k}} \sum_{i=0}^{2^k-1} |i\rangle\, |C_{y,k}(i)\rangle \xrightarrow{U_y^\dagger} \sum_{i=0}^{2^k-1} |C_{y,k}(i)\rangle := |C_{y,k}\rangle$$

To see why this feature map is useful for learning the dataset shown in Figure 1.7a, we begin by showing there exists a hyperplane in the respective feature space capable of separating the classes with a large margin.

For a concept class f_s, the superposition state,

$$|\phi_s\rangle = \sum_{i=0}^{(p-1)/2-1} |s \cdot g^i \mod p\rangle \ , \tag{1.41}$$

consists of basis states that sweep the data points lying along one semicircle, as shown for the green data points in Figure 1.7b. This state represents a discriminative half plane since taking its inner product with a feature vector 1.39 gives a different value depending on its class label:

- $|\langle \phi_s | C_{y,k} \rangle|^2 = \Delta$ for $(1 - \Delta)\%$ of $\{y | f_s(y) = +1\}$
- $|\langle \phi_s | C_{y,k} \rangle|^2 = 0$ for $(1 - \Delta)\%$ of $\{y | f_s(y) = -1\}$

where $\Delta = \frac{2^k}{(\frac{(p-1)}{2})} = \frac{2^{k+1}}{p-1}$ is the ratio of the arc length spanned by the feature vector to that of the feature halfspace.

To clarify the sense in which this $|\phi_s\rangle\langle\phi_s|$ is a hyperplane, we can alternatively represent both it and additionally any $|C_{y,k}\rangle\langle C_{y,k}|$ with respect to the 4^n-dimensional Pauli basis $\{P_\alpha\}$, identifying them with their Fourier coefficients, $\vec{w}_s = (w_{s,\alpha})$ and $\vec{y} = (y_\alpha)$ respectively as defined using the trace inner product from 1.26. By the exact same procedure as in 1.37, the Hilbert-Schmidt inner product of the space $\mathbb{C}^{2^n \times 2^n}$ is thus equivalent to the Euclidean space $\mathbb{R}^{4^n}$:

$$\langle |\phi_s\rangle\langle\phi_s| \, , |C_{y,k}\rangle\langle C_{y,k}| \rangle_{HS} = |\langle \phi_s | C_{y,k} \rangle|^2 = \sum_{\alpha=0}^{4^n-1} w_{s,\alpha} y_\alpha = \langle \vec{w}_s, \vec{y} \rangle_{\mathbb{R}} \, .$$

Taking $b = \frac{\Delta}{2}$, the inner products can be recast as Euclidean vectors:

- $\vec{w}_s \cdot \vec{y} - b = \frac{\Delta}{2}$ for $(1 - \Delta)\%$ of $\{y | f_s(y) = +1\}$
- $\vec{w}_s \cdot \vec{y} - b = -\frac{\Delta}{2}$ for $(1 - \Delta)\%$ of $\{y | f_s(y) = -1\}$.

where $\vec{w}_s$ and b can naturally be interpreted as the normal vector to a hyperplane and offset term respectively; at least $(1 - \Delta)\%$ of the data points labelled +1 (-1) lie above (below) the hyperplane. Thus, this hyperplane separates the classes with a soft margin of $\Delta / \|w_s\|$.

Classical hardness of learning the DLP dataset Not only is the concept class described above solvable for a quantum classifier, but it is provably unsolvable on a classical computer. This was shown in Ref. [15] by proving that if a classical algorithm achieves at least $1/2 + O(n^{-t})$ test accuracy on the concept class, then that same algorithm could solve the $DLP_{\frac{1}{2}}$ for more than $1/2 + O(n^{-t})$ of the inputs and thus the DLP. However, assuming the classical hardness of DLP, this leads to a contradiction, implying that there does not exist a classical algorithm capable of attaining a test accuracy better than $1/2 + O(n^{-t}) < 0.99$ and thus learning the concept class.

1.4.2 Kernel methods for data with group structure

Despite the DLP-based dataset discussed in the previous section having been artificially constructed, it was later shown that it formed part of a larger class of learning problems that may be suited to being solved by a Quantum Kernel Estimator (QKE) all of which have the unique property that the domain of their dataset forms a group G [32]. However, while this to some extent generalises the nature of Liu *et al.*'s learning problem as being based on group theory, it does not mention the aspect of it that makes it classically intractable.

Inspired by this group-theoretic approach, we now examine a generic class of group problems to which the DLP also belongs which *are* classically intractable but efficiently solvable on a quantum computer known as the Hidden Subgroup Problems (HSG) [21]. As was the case for the DLP, it is reasonable to expect that there exist other known problems within the HSG that can be used to create a classically intractable learning problem solvable by a quantum kernel machine, as we stated in (i). In what follows, we investigate the HSG in its most general form, and consolidate that understanding with one special case called Simon's problem.

1.4.2.1 The Hidden Subgroup Problem (HSP)

The Hidden Subgroup Problem
Input: A function $f : G \to Y$ mapping a group G to a set Y promised to be *constant* on cosets gH of some unknown subgroup $H \leqslant G$ but *distinct* across different cosets:

$$f(g_1H) = f(g_2H) \iff g_1H = g_2H \qquad \text{for} \quad g_1\,, g_2 \in G.$$

Question: What is the generating set S of H?

The HSP has many special cases such as the DLP, three of which are shown in Table 1.3. It can be shown that finding S for the special cases in which G is abelian is possible with a quantum algorithm, while remaining classically intractable REF. As a result, should a learning problem be constructed based on any one of them — like in the case of the DLP — there is hope that a QKE model with a suitable quantum feature map can solve it. We consolidate the definition above by looking at two special cases, and meditating how they can each inspire the construction of a learning problem.

Problem	G	Y	f	H
Simon's Problem	$\mathbb{Z}_2^n$	$\mathbb{Z}_2^n$	$f(\mathbf{x}) = f(\mathbf{x} \oplus \mathbf{h})$ for some hidden bit string $\mathbf{h}$	$\{0, \mathbf{h}\}$
Order finding	$\mathbb{Z}^+$	$\mathbb{Z}_{N_0}^+$	$f(x) = a^x \mod N_0$ $\gcd(a, N_0) = 1$	$r\mathbb{Z}$ where r is order of a
Discrete log problem	$\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$	$\mathbb{Z}_p$	$f(a, b) = g^a x^b \mod p$ $\langle g \rangle = \mathbb{Z}_p^\times$ & $\log_g(x) = y$	$(y, -1)\mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$

Table 1.3: **A summary of three special cases of the Hidden Subgroup Problem**. These special cases are characterised by a function f that fulfils some promise, as well as a group G on it acts

Simon's problem In Simon's problem, we are given a function $f : \{0,1\}^n \to Y$ that shares the same value between all pairs of bit strings whose difference evaluates to the hidden bit string $\mathbf{h}$. The goal in this case is to find $\mathbf{h}$. For example, for $n = 2$, assuming that $\mathbf{h} = 10$, then f by definition satisfies

$$\begin{aligned} f(00) &\overset{!}{=} f(00 + \mathbf{h}) = f(00 + 10) = f(10) \\ f(01) &\overset{!}{=} f(01 + \mathbf{h}) = f(01 + 10) = f(11). \end{aligned} \tag{1.42}$$

Notice the domains over which f is constant are partitioned into two: $\{00, 10\} = 00+H = 10+H$ and $\{01, 11\} = 01 + H = 11 + H$ where $H = \{0, \mathbf{h}\}$. In the language of 1.4.2.1, these sets are the cosets of the subgroup H generated by the set $S = \{\mathbf{h}\}$, and the goal is to find S i.e. $\mathbf{h}$. A quantum algorithm exists that finds $\mathbf{h}$ efficiently, while a classical algorithm does not..

Comparing Simon's problem with the DLP, we can see that the classical intractability of obtaining the hidden bit string $\mathbf{h}$ is analogous to that of obtaining the discrete logarithm problem $y = \log_g(x)$. This would suggest that a learning task based on Simon's problem would have to in some way incorporate the hardness of obtaining $\mathbf{h}$. With a look towards addressing (i), we would more generally expect that a learning problem based on the HSP would involve the generating set S of the hidden subgroup in some way.

References

[1] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[2] A. Brnabic and L. M. Hess, “Systematic literature review of machine learning methods used in the analysis of real-world data for patient-provider decision making,” *BMC medical informatics and decision making*, vol. 21, no. 1, pp. 1–19, 2021.

[3] K. Najafi, S. F. Yelin, and X. Gao, “The development of quantum machine learning,” 2022.

[4] E. Tang, “A quantum-inspired classical algorithm for recommendation systems,” in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 217–228.

[5] V. Giovannetti, S. Lloyd, and L. Maccone, “Quantum random access memory,” *Physical Review Letters*, vol. 100, no. 16, p. 160501, Apr. 2008, arXiv:0708.1879 [quant-ph]. [Online]. Available: http://arxiv.org/abs/0708.1879

[6] S. Lloyd, M. Mohseni, and P. Rebentrost, “Quantum principal component analysis,” *Nature Physics*, vol. 10, no. 9, pp. 631–633, Sep. 2014, arXiv:1307.0401 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1307.0401

[7] P. Rebentrost, M. Mohseni, and S. Lloyd, “Quantum support vector machine for big data classification,” *Physical Review Letters*, vol. 113, no. 13, p. 130503, Sep. 2014, arXiv:1307.0471 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1307.0471

[8] I. Cong and L. Duan, “Quantum Discriminant Analysis for Dimensionality Reduction and Classification,” *New Journal of Physics*, vol. 18, no. 7, p. 073011, Jul. 2016, arXiv:1510.00113 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1510.00113

[9] I. Kerenidis and A. Luongo, “Quantum classification of the MNIST dataset with Slow Feature Analysis,” *Physical Review A*, vol. 101, no. 6, p. 062327, Jun. 2020, arXiv:1805.08837 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1805.08837

[10] Y. Li, R.-G. Zhou, R. Xu, W. Hu, and P. Fan, “Quantum algorithm for the nonlinear dimensionality reduction with arbitrary kernel,” *Quantum Science and Technology*, vol. 6, no. 1, p. 014001, Jan. 2021. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-9565/abbe66

[11] A. Sornsaeng, N. Dangniam, P. Palittapongarnpim, and T. Chotibut, “Quantum diffusion map for nonlinear dimensionality reduction,” *Physical Review A*, vol. 104, no. 5, p. 052410, Nov. 2021, arXiv:2106.07302 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2106.07302

[12] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, “The power of quantum neural networks,” *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, Jun. 2021. [Online]. Available: http://www.nature.com/articles/s43588-021-00084-1

[13] I. Cong, S. Choi, and M. D. Lukin, “Quantum Convolutional Neural Networks,” *Nature Physics*, vol. 15, no. 12, pp. 1273–1278, Dec. 2019, arXiv:1810.03787 [cond-mat, physics:quant-ph]. [Online]. Available: http://arxiv.org/abs/1810.03787

[14] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, “Supervised learning with quantum-enhanced feature spaces,” *Nature*, vol. 567, no. 7747, pp. 209–212, Mar. 2019. [Online]. Available: http://www.nature.com/articles/s41586-019-0980-2

[15] Y. Liu, S. Arunachalam, and K. Temme, “A rigorous and robust quantum speed-up in supervised machine learning,” *Nature Physics*, vol. 17, no. 9, pp. 1013–1017, Sep. 2021. [Online]. Available: https://www.nature.com/articles/s41567-021-01287-z

[16] M. Gordon, M. Cerezo, L. Cincio, and P. Coles, “Covariance matrix preparation for quantum principal component analysis,” *arXiv:2204.03495*, 2022. [Online]. Available: https://arxiv.org/abs/2204.03495

[17] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[18] S. Chakraborty, A. Gilyén, and S. Jeffery, “The power of block-encoded matrix powers: improved regression techniques via faster Hamiltonian simulation,” p. 14 pages, 2019, arXiv:1804.01973 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1804.01973

[19] E. Bernstein and U. Vazirani, “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.

[20] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning.* MIT press, 2018.

[21] R. Jozsa, “Quantum factoring, discrete logarithms, and the hidden subgroup problem,” *Computing in science & engineering*, vol. 3, no. 2, pp. 34–43, 2001.

[22] I. T. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[23] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.

[24] O. Kyriienko and E. B. Magnusson, “Unsupervised quantum machine learning for fraud detection,” Aug. 2022, arXiv:2208.01203 [cond-mat, physics:quant-ph]. [Online]. Available: http://arxiv.org/abs/2208.01203

[25] I. Kerenidis and A. Prakash, "Quantum Recommendation Systems," Sep. 2016, arXiv:1603.08675 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1603.08675

[26] T. R. Bromley and P. Rebentrost, "Batched quantum state exponentiation and quantum hebbian learning," *Quantum Machine Intelligence*, vol. 1, no. 1, pp. 31–40, May 2019. [Online]. Available: https://doi.org/10.1007/s42484-019-00002-9

[27] C. Zoufal, A. Lucchi, and S. Woerner, "Variational quantum boltzmann machines," *Quantum Machine Intelligence*, vol. 3, no. 1, pp. 1–15, 2021.

[28] A. Gilyén, Z. Song, and E. Tang, "An improved quantum-inspired algorithm for linear regression," *Quantum*, vol. 6, p. 754, 2022.

[29] D. Chen, Y. Xu, B. Baheri, S. A. Stein, C. Bi, Y. Mao, Q. Quan, and S. Xu, "Quantum-inspired classical algorithm for slow feature analysis," *arXiv preprint arXiv:2012.00824*, 2020.

[30] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, "Quantum embeddings for machine learning," Feb. 2020, arXiv:2001.03622 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2001.03622

[31] G. Gentinetta, A. Thomsen, D. Sutter, and S. Woerner, "The complexity of quantum support vector machines," Feb. 2022, arXiv:2203.00031 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2203.00031

[32] J. R. Glick, T. P. Gujarati, A. D. Corcoles, Y. Kim, A. Kandala, J. M. Gambetta, and K. Temme, "Covariant quantum kernels for data with group structure," Mar. 2022, number: arXiv:2105.03406 arXiv:2105.03406 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2105.03406