Approximating a Wavelet Kernel Using a Quantum Computer

Rivan Rughubar
Supervisor: Assoc Prof. Jonathan Shock

Department of Physics
University of Cape Town
South Africa
February 2023

Abstract

Machine learning and quantum computing are both fields which have gained a significant amount of popularity and attention in recent years. The intersection of these two fields, quantum machine learning, looks at whether quantum computers can aid or improve classical machine learning methods, or whether quantum computers can perform machine learning tasks which classical computers cannot. In this thesis we explore different implementations of quantum machine learning algorithms on near term quantum computers, and the limits of these systems. We focus on support vector machines and kernel methods, which are a form of supervised machine learning. We examine whether using quantum kernels to search for a quantum advantage over classical computers is suitable, and why it may be wise to search for quantum advantages using other methods. Lastly, we construct a quantum circuit which can approximate a wavelet kernel with a mean squared error over sample plots of 9.09×10^{-9} by estimating the Fourier coefficients of the kernel. We hope that this can be used as a starting point for performing wavelet analysis on quantum computers.

Plagiarism Declaration

Declaration:

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the ……Bibtex…….. convention for citation and referencing. Each contribution to, and quotation in, this essay/report/project/ ……Thesis…….. from the work(s) of other people has been attributed, and has been cited and referenced. Any section taken from an internet source has been referenced to that source.
3. This essay/report/project/……Thesis…… is my own work, and is in my own words (except where I have attributed it to others).
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
5. I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.

Signature Signed by candidate

Contents

Chapter 1

Introduction

1.1 Machine Learning

Over the past few decades the field of computing has made many advances. These have come through developments in both hardware and software. On the hardware front computer processors have gotten smaller and continue to shrink, allowing us to fit more transistors on chips and enabling us to compute at faster speeds while also being more power efficient [1]. Advancements in graphics card technologies have allowed us to perform parallel computing at new speeds. We have also developed specialised chips such as tensor cores and ray tracing cores, which excel at highly specialised tasks [2].

Computer algorithms and software have also improved greatly. One of the most prominent advancements in recent times is machine learning. Machine learning refers to many different algorithms, all of which aim to learn from data in some way. In a broad sense they are algorithms which often aim to find relationships between inputs and outputs in data, or to find patterns in data. Machine learning can find patterns much faster than humans and can parse though much larger volumes of data. Due to these factors and the increasingly large amounts of data being generated by humans, machine learning has become a popular method for analysing data [3].

Machine learning can be broken down into three broad categories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithms are given information about the inputs and the outputs of the data and the algorithm is tasked with finding a suitable relationship between them [4]. In unsupervised learning, information about the data is not given. The algorithms usually group or cluster data points based on similarity in order to form a prediction about the output [5]. Reinforcement learning algorithms learn by constantly making decisions and queries so that they can learn from their mistakes [6]. In this thesis we will focus on supervised learning, particularly, support vector machines and kernel methods.

1.2 Quantum Computing

With the number of advancements in computing and the rate at which they have occurred, one might think that, given enough time, computers can do anything. However, there are still limitations. At some point we will not be able to make transistors any

smaller, and that will put a cap on the current model of CPU development [1]. There are also certain tasks that our computers cannot perform in a feasible time period. Given these factors, it is reasonable to ask if we can improve computers in any other way, and if so, how?

One of the most important theories of physics from the last century is quantum mechanics. Quantum mechanics aims to model how the world works at microscopic scales. It has changed the way we look at the world. People asked if we could use the results and properties of quantum mechanics to improve how we perform computation. One of the first people to make this proposition was Richard Feynman, at a talk in 1982, where he proposed using quantum computers to simulate quantum systems which were too difficult to simulate using classical computing methods [7]. This was the beginning of the field known as quantum computing [8].

In the circuit model of quantum computing, we replace the binary system of bits consisting of 1's and 0's - used by classical computing with qubits. Qubit is short for quantum bit, refers to a quantum bit of information. Qubits are quantum objects which can be put into a state of superposition and can then be used to encode quantum data. Computation is performed inside a quantum computer by performing unitary operations on qubits. We are able to exploit the quantum properties of qubits such as superposition, entanglement and interference in order to perform some of the calculations which are not feasible on classical computers. Some examples of these calculations include Shor's algorithm and Grover's algorithm [9–11]. Shor's algorithm enables us to factorise numbers in polynomial time. Gover's algorithm is an unstructured search algorithm which provides a quadratic speedup over all known classical algorithms for unstructured search. Quantum computers can also, in theory, perform all tasks which a classical computer can perform.

Quantum computers work well in theory but are often hard to realise physically. Research to improve physical implementations of quantum computers is ongoing. Some implementations include; superconducting quantum computers, trapped ion quantum computers, photonic based systems, nuclear magnetic resonance quantum computers, and many more [12–14]. These quantum computers use different objects as qubits, photonic based systems may use photon packets or light beams as qubits, whereas other systems may use electrons or atoms. Due to the field being in relative infancy, the class of quantum computers we deal with in the real world is usually referred to as noisy intermediate scale quantum computers. (NISQ) [15]. Noisy refers to the noise inside quantum circuits. This, is a result of quantum circuits not being closed systems and often having decoherence and dissipation. Intermediate scale refers to the number of qubits in the quantum computer, usually up to 100 qubits [16].

Quantum supremacy is the point at which a quantum computer surpasses a classical computer at a specific task. Companies like Google have claimed to have already reached quantum supremacy with their Sycamore processor, which was said to perform calculations 3 million times faster than the Summit supercomputer [17]. As quantum computers grow larger and have more qubits on their processors, and with the refinement and discovery of quantum computing techniques we can expect more cases of quantum supremacy to appear over time.

1.3 Quantum Machine Learning

Quantum computing and machine learning are both rapidly developing fields, and it is therefore very interesting to examine the intersection of the two. The main questions that are asked are: what benefits can quantum computers provide to the field of machine learning? Can quantum computers enable us to run machine learning algorithms faster than classical computers? Can quantum computers enable us to learn functions and give us access to algorithms which are inaccessible on a classical computer?

None of these are straightforward questions. Quantum algorithms running faster than classical ones is referred to as speedup and can be broken down into different categories. In this thesis I will follow the categories laid out by Ronnow et al [18] so that mentions of speedup in the thesis can be kept consistent. Provable quantum speedup requires a proof that no classical algorithm can perform as well or better than the quantum algorithm. Strong quantum speedup compares the quantum algorithm to the best known classical algorithms. Potential quantum speedup compares two specific algorithms. Limited quantum speedup compares two conceptually equivalent algorithms.

The transition from classical to quantum machine learning (QML) can be broken down into two approaches, as seen in figure 1.1 [19]. The figure shows two axes: data processing and data generation. Along these axes we can define four quadrants. The first quadrant is classical classical denoted by CC, with the first classical referring to the type of data being processed and the second one referring to the type of computation used to process the data. This represents classical machine learning, and most machine learning that goes on today falls into this category. The next is classical quantum, CQ, where classical data is being processed by a quantum computer. The CQ implementation of quantum machine learning usually takes the form of parameterised quantum circuits. This will be one of the main focuses of this thesis and will be explained in detail in a later section.

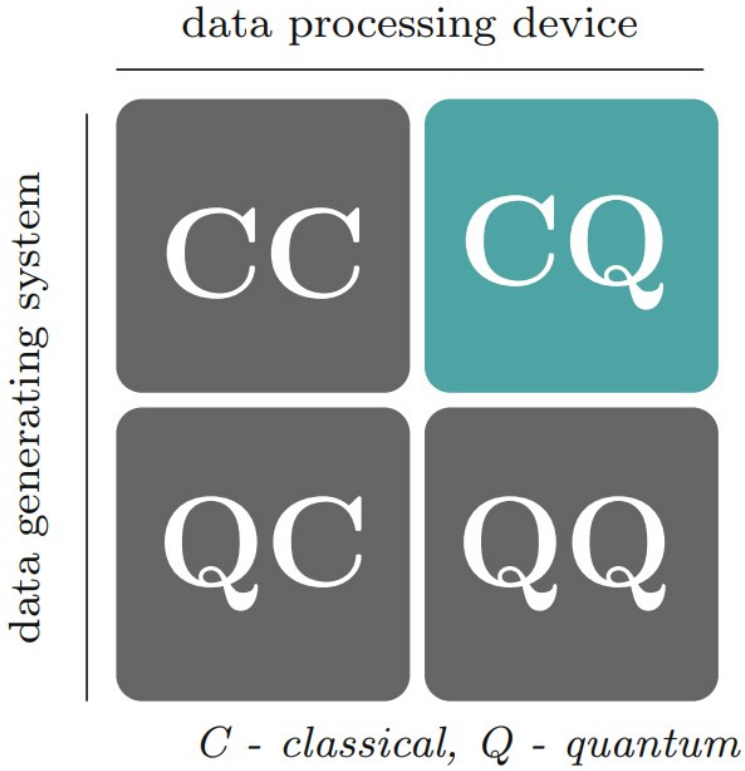


Figure 1.1: The four approaches to quantum machine learning. Image taken from "Supervised Learning with Quantum Computers" [20].

Next is quantum classical, QC, which is when quantum data is processed by a classical algorithm. This can happen when an experiment creates quantum data, such as photon manipulation experiments, and the data is processed using classical techniques. Last is quantum quantum, QQ, where quantum data is processed by a quantum algorithm. This is the hardest of the four approaches to implement at present but it is still being actively researched.

This thesis will focus mainly on the kernel method, which falls under the CQ quadrant. The kernel method refers to a class of machine learning algorithms used mainly for pattern analysis such as classification, correlation and clustering. One of the most well known implementations of kernel methods are support vector machines. Support vector machines and kernel methods will be discussed in detail in later sections.

In CQ machine learning we need to convert our classical data into quantum data. There are many ways to do this and the process of doing so is known as encoding. Each method of encoding has its own pros and cons, and some methods are more suitable for certain problems than others. Some methods of encoding will be explained in depth in a later section, as well as their effects on quantum machine learning algorithms.

1.4 Research Questions

The first set of questions on which this thesis will try to shed some light is why it's so hard to find a quantum advantage. This question will be answered through the lens of quantum kernel methods. We will ask what quantum advantages have been found in the field of quantum kernel methods. We will also ask how they are found and what researchers suggest as reasons for why it is so hard to find quantum advantage. We will finally ask whether quantum kernel methods are the best way to search for quantum advantage in general.

The next set of questions will deal with approximating kernels on near term quantum computers. We will first ask what kinds of kernels can be approximated and how it can be done. We will then ask whether the current methodology can be adapted in order to approximate a wavelet kernel and finally ask what the implications of this could be. The approximation of the wavelet kernel on a quantum computer has not been attempted before, and if successful could be used in the future to perform wavelet analysis on a quantum computer.

Chapter 2

Literature Review

The aim of this background section is to provide a strong mathematical basis for the concepts that will be used throughout the thesis. It will cover a broad range of topics from many fields, but should go over everything needed to understand the work in the main body. We begin by going over the general definitions and workings of support vector machines, kernels, Fourier analysis and wavelet analysis. We then summarise quantum circuits and their notation so we can go into quantum machine learning and parametrised quantum circuits. Next we look at practical examples of parametrised quantum circuits and then look at quantum encoding. Lastly we look at quantum kernels and how kernel methods are related to supervised machine learning models.

2.1 Support Vector Machines

Support vector machines (SVMs) are a form of supervised machine learning. They are mainly used to do classification and sometimes regression analysis. They do this by finding a function which defines a hyperplane or set of hyperplanes which optimally separate the given data, the hyperplane is then used to classify new data points. In general the optimal hyperplane is the one which creates the largest separation between the data groups [21]. This process can be visualised easily and seen in figure 2.3. We will start by examining a support vector machine that is given data which is linearly separable, and then move on to the non-linear case.

2.1.1 Linearly Separable Support Vector Machines

Given a linearly separable data set containing n data points of the form

$$(\boldsymbol{x}, y_1), ..., (\boldsymbol{x}, y_n) \tag{2.1}$$

Where each $\boldsymbol{x}_i$ is a p-dimensional vector and y_i has a value of either 1 or -1. The data points being linearly separable means that a hyperplane can be drawn which divides that data such that all the values labelled 1 appear on the opposite side of the hyperplane to the values labelled -1. The hyperplane dividing these data classes can be written using the normal vector to the hyperplane, $\boldsymbol{w}$, with $x \in \mathbb{R}^p$as:

$$\boldsymbol{w}^T \boldsymbol{x}_i - b = 0 \tag{2.2}$$

We now select two parallel hyperplanes which separate the data classes such that the distance between them is as large as possible. The region created between these two hyperplanes is called the margin, and the SVM looks for the maximum-margin hyperplane, which is the hyperplane that lies in the middle of the other hyperplanes. The data set is also normalised so that we may write the following equations:

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x} - b &= 1 \\ \boldsymbol{w}^T\boldsymbol{x} - b &= -1 \end{aligned} \tag{2.3}$$

This step allows us to classify the points. All points which appear above the hyper plane given by $\boldsymbol{w}^T\boldsymbol{x} - b = 1$ are classified as "1", and all points which appear belowe the hyperplane $\boldsymbol{w}^T\boldsymbol{x} - b = -1$ are classified as "-1". We also add the constraint:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i - b) \geq 1 \text{ for all } 1 \leq i \leq n \tag{2.4}$$

This constraint ensures that each data point will lie on the correct side of the margin. The distance between the two hyperplanes which define the margin is given by:

$$\frac{2}{|\boldsymbol{w}|}$$

The problem of the SVM then becomes to minimise $|\boldsymbol{w}|$ subject to $y_i(\boldsymbol{w}^T\boldsymbol{x}_i - b) \geq 1$ for all data points i. The maximum-margin hyperplane is completely determined by the data points, $\boldsymbol{x}_i$, which lie closest to it. All of the data points are checked against the condition given by equation 2.4, which ensures that the correct data points are chosen. These data points are known as the support vectors, and it is possible to have multiple support vectors as long as they have the same distance from the margin. A visualisation of this process can be seen in figure 2.3.

The minimisation which needs to be performed is usually done using the Lagrange method. The Lagrangian which will need to be minimised in this case is:

$$L = \frac{1}{2}|\boldsymbol{w}|^2 - \sum_i \alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i - b) - 1) \tag{2.5}$$

This is the Lagrangian dual of the SVM problem. The α_i is the Lagrange multiplier. The -1 comes from taking the 1 over to the other side in equation 2.4. To solve this equation we set:

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0, \frac{\partial \mathrm{L}}{\partial \alpha} = 0, \frac{\partial \mathrm{L}}{\partial b} = 0 \tag{2.6}$$

Plugging in these expressions will leave:

$$L = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_j \cdot \boldsymbol{x}_j \tag{2.7}$$

What we have done is create an expression which has variables which depend implicitly on $\boldsymbol{w}$. Using this equation allows us to simplify the computation of the support vector machine greatly. The maximisation of the margin will require only the computation of dot products of pairs of support vectors. This computation can be done fairly quickly on a computer and is sped up further if we have any information on which data points can be used as a support vector.

2.1.2 Non-Linear Support Vector Machines

Separating data clusters with a margin is fairly easy when the data is linearly separable, however, data is often non-linearly separable. To deal with this, the support vector machine must be adapted in order to create margins which can divide this data. In order to do this we employ the kernel trick. Data points are mapped into a higher dimensional space using feature maps and are then separated in the higher dimension before being projected back down. A feature map is a function which maps data into a new space called the feature space. The family of functions which could be considered feature maps is broad, but in general they are used to try and map data to a space where certain features are easier to identify.

A visualisation of the mapping process can be seen in figure 2.1. The projection sends the data to a new space and can allow us to more easily separate data clusters. A margin drawn in this space may look like a surface in the higher dimension, but when projected back down into the original space it can take on a more complex form and therefore allows us to create non-linear margins.

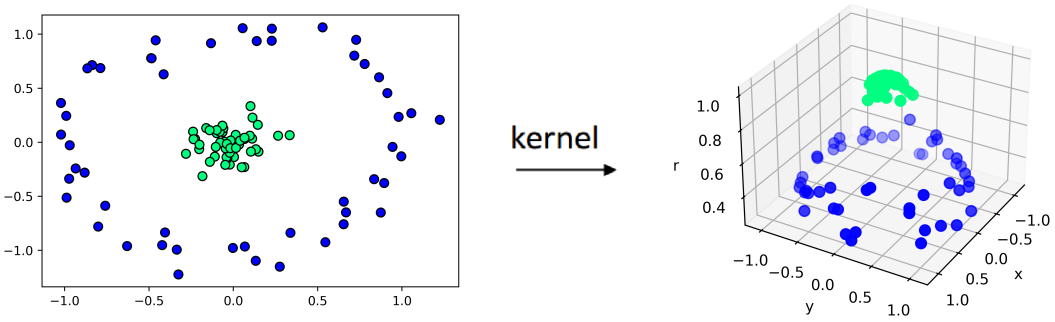


Figure 2.1: A visualisation of projecting a 2-dimensional problem into a 3-dimensional problem in order to create a non-linear support vector machine.

The lift causes the Lagrangian in equation 2.7 to become:

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \tag{2.8}$$

Where ϕ is a feature map which $\phi : \mathcal{X} \to \mathcal{F}$ and $\phi(\boldsymbol{x}_j) \cdot \phi(\boldsymbol{x}_j)$ is the inner product between two vectors which have been mapped into a new feature space. We then define a kernel function K:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \tag{2.9}$$

The exact nature of the kernel will be explained in depth in the next section, but for now using the kernel trick we are now able to calculate the inner product of the mapped data without having to perform the actual mapping and only using the sample data set. The kernel trick allows us to not need to specify the feature map explicitly and to instead just consider a kernel function which we know there is implicitly a feature space. This allows margins to be drawn between data sets which are not linearly separable. This helps us to classify a larger family of data sets without the need for intense computation [22].

This is very useful as it allows us to use computational power more efficiently and process larger volumes of data. Different kernels project the data into different spaces and therefore some kernels are more suited to certain problems and data sets than others. Some popular kernels include:

- Polynomial $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j)^d$
- Inhomogeneous Polynomial $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + 1)^d$
- Radial Basis Function $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$

The Gaussian kernel is used in many cases as it works well when there is no prior knowledge of the data [23].

Example: Support vector machine

Now that the theoretical framework of support vector machines has been discussed, it is useful to see an example to see how they work in practice. We will start by looking at a data set which is linearly separable, and then look at a data set which is not linearly separable. The linearly separable data set we are using can be seen in figure 2.2. The data was generated randomly using the scikit-learn module in python.

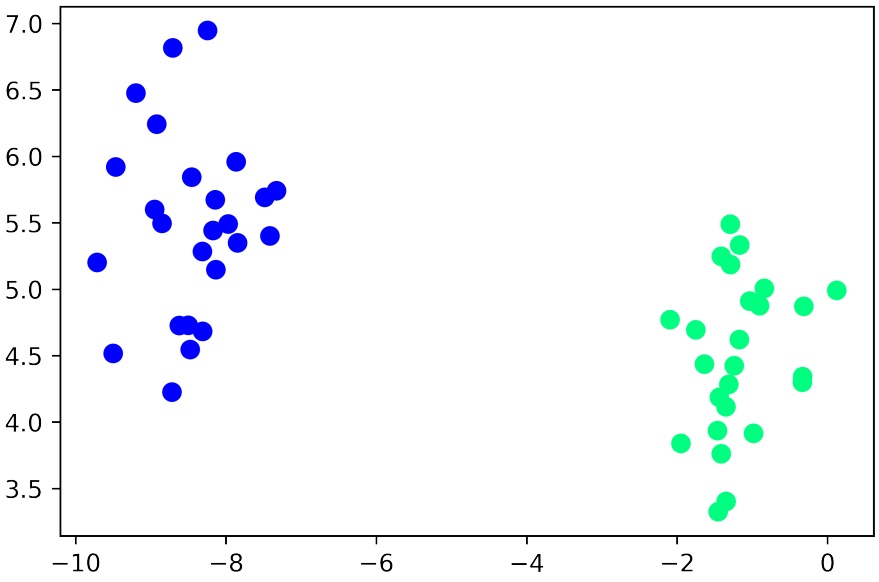


Figure 2.2: A randomly generated linearly separable data set.

By applying the methods above, we are able to identify the support vectors and create the maximum-margin-hyperplane between the green and blue data clusters. The reults of this can be seen in figure 2.3. The solid line is the maximum-margin-hyperplane and the data points along the dashed lines are the support vectors. It is relatively easy for a person to identify the support vectors visually and separate the two data sets by drawing a margin. However, drawing the maximum-margin is much harder to do visually.

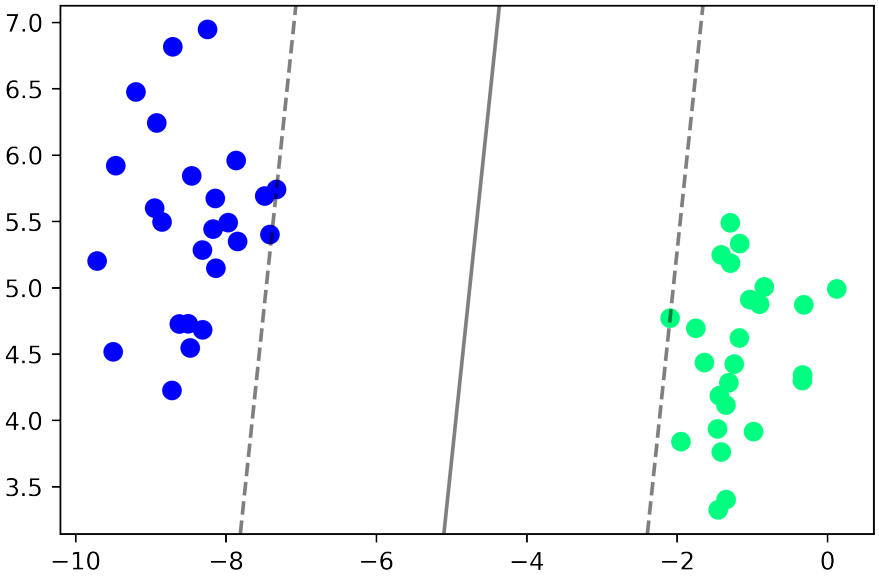


Figure 2.3: The maximum-margin-hyperplane drawn between the two data sets and the support vectors used to draw the margin.

We now move on to the non-linearly separable data set which can be seen in figure 2.4. This data set was also randomly generated using scikit-learn.

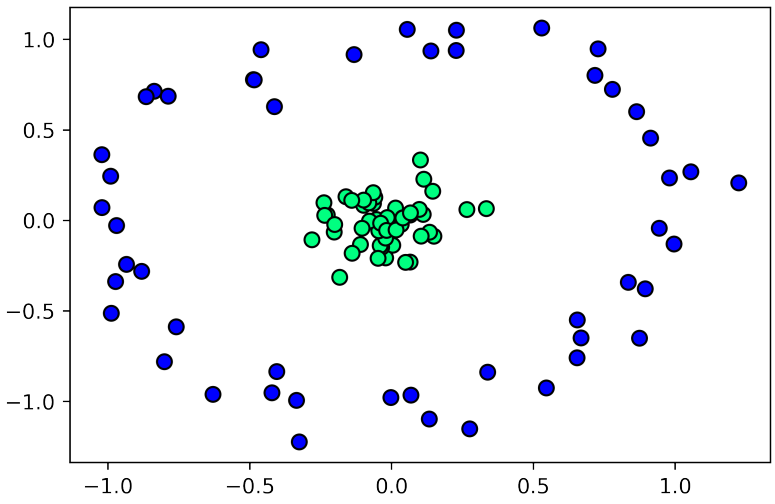


Figure 2.4: A randomly generated non-linearly separable data set.

Applying the same SVM algorithm used to generate figure 2.3, we obtain the results seen in figure 2.5. The margin drawn does not separate the two data sets in a way that would reliably classify new data points into the correct categories.

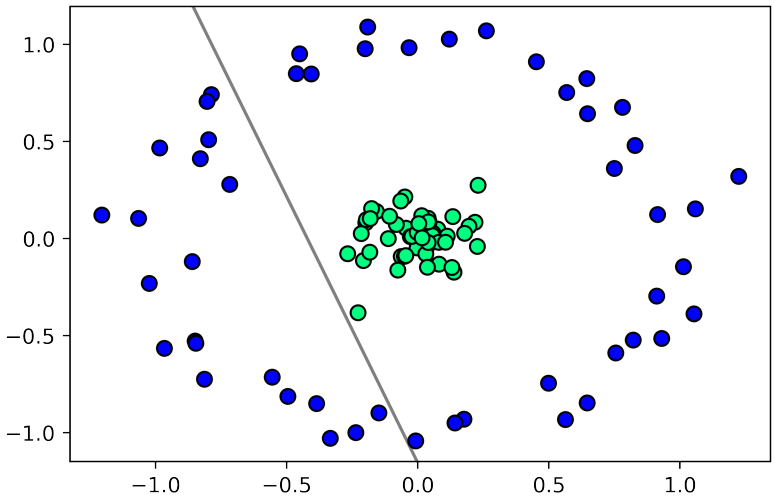


Figure 2.5: Applying the algorithm for a linear support vector machine to a non-linearly separable data set.

We now use a kernel to project the data into a feature space where the two data sets can be separated by a linear margin. For this data set we do so using the Radial Basis Function (RBF) kernel, and the projected data set can be seen in figure 2.6.

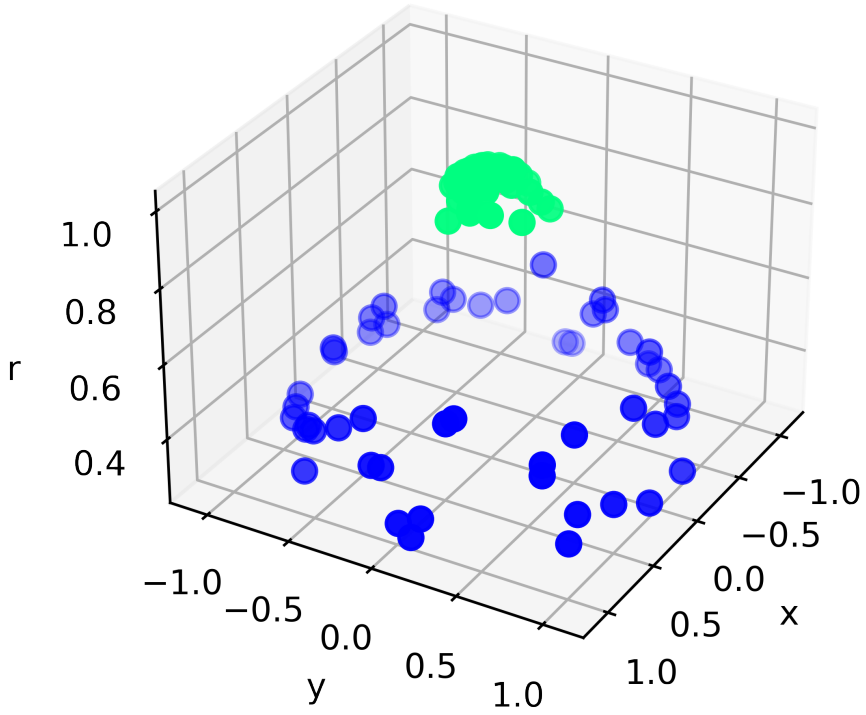


Figure 2.6: Using the Radial Basis Function kernel to map the non-linearly separable data set into a feature space where it is linearly separable.

Once we have mapped the data into the RBF feature space we can apply the support vector machine algorithm in order to find the support vectors and draw the maximum-margin-hyperplane. We then map the data back into the original space and draw a margin which is non-linear and separeates the two data sets. The results of this can be seen in figure 2.7. The solid line shows the maximum margin and the data points which lie on the dashed lines are the support vectors.

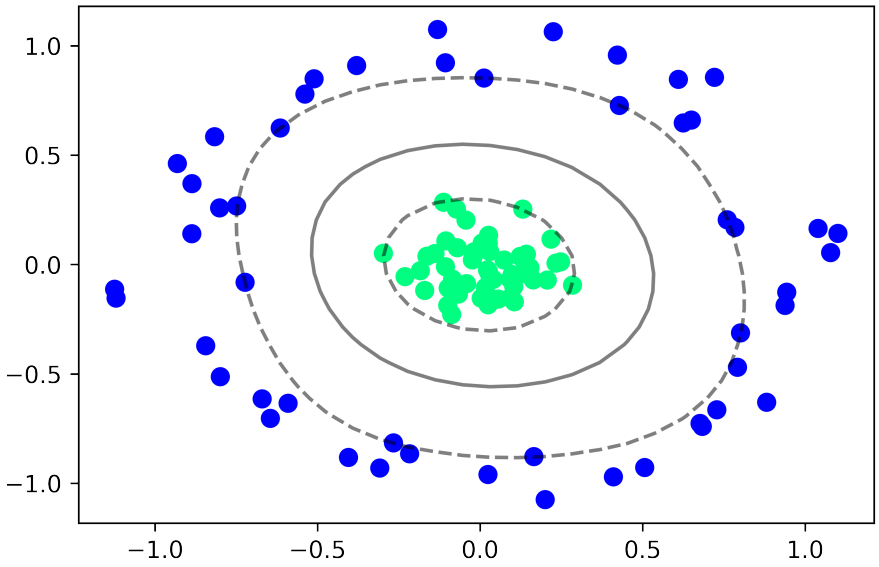


Figure 2.7: Using the Radial Basis Function kernel to map the non-linearly separable data set into a feature space where it is linearly separable.

2.2 Kernels

Kernels can play an integral role in support vector machines and, as will be shown in a later section, kernel methods are also closely linked to supervised quantum machine learning models. In this section we will go through the mathematical definition of kernels and some of their properties as well as some special kernels, following Chapter 3 of "Kernel Methods for Pattern Analysis" by J Shawe-Taylor and N Christiani [24].

Definition 2.2.1 (Kernel)**.** Let X be a non-empty set. A function $X \times X \to \mathbb{R}$ is a kernel if there exists a Hilbert space, the feature space, and a map $\phi : X \to \mathcal{F}$ such that for all $x, x' \in X$

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}'), \phi(\mathbf{x}) \rangle_{\mathcal{F}} \tag{2.10}$$

ϕ is known as a feature map. $\mathcal{H}$ is the Hilbert space, which is a real or complex inner-product space that is also a complete metric space with respect to the distance function induced by the inner product. Hilbert spaces can be infinte dimensional. The notation used here is slightly different to the notation used in equation 2.9. Here we have defined the kernel using the inner product instead of a dot product. The dot product is an inner product in some spaces, but defining the kernel in this way gives us a more general definition.

The feature map is a function which can take data set and map it into a feature space. This is very useful for support vector machines, as they can allow us to take data which is not linearly separable and project into a space where it is - as was demonstrated in the previous section. The kernel is a function which allows us to evaluate inner products in the feature space We will now define the Gram matrix and Mercer's Theorem. These concepts are important as they allow us to extend kernels into infinite-dimensional input spaces and allow us to define the Reproducing Kernel Hilbert Space.

Definition 2.2.2 (Gram matrix)**.** Given a data set $\{\boldsymbol{x_i} | i \in [n]\}$ the Gram matrix is the matrix $G \in \mathbb{R}^{n \times n}$ with entries $G_{ij} = \langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle$

Definition 2.2.3 (Kernel matrix)**.** Given a kernel, $k : X \times X \to \mathbb{R}$ and a dataset $\{\boldsymbol{x_i} | i \in [n]\}$, the kernel matrix is the matrix $K \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = k(\boldsymbol{x_i}, \boldsymbol{x_j})$

Theorem 2.2.1 (Mercer's Theorem)**.** A symmetric function $k(.,.)$ is a kernel iff for any finite sample X the kernel matrix for X is positive semi-definite.

As mentioned before, each feature map maps data to a unique feature space. Feature maps are specifically chosen to try and exploit underlying patterns in the data and the kernels associated with these feature maps are used to compute the inner products. The polynomial kernel functions mentioned above are often used in image processing problems. Sigmoid and hyperbolic tangent kernel functions are often used in neural networks. The graph kernel, which is used to compute the inner product on graphs, is often used in fields like bioinformatics and the string kernel is often used to text processing.

Kernel Properties

Let k_1 and k_2 be kernels over $X \times X, X \subseteq \mathbb{R}^n$, with $a \in \mathbb{R}^+$, $f(\cdot)$ a real valued function on X, $\phi : X \longrightarrow \mathcal{F}$, with a kernel k_3 over $\mathbb{R}^n \times \mathbb{R}^n$. Then the following functions are kernels

$$\begin{aligned} (i) \quad & k_1(x,z) + k_2(x,z) \\ (ii) \quad & ak_1(x,z) \\ (iii) \quad & k_1(x,z)k_2(x,z) \\ (iv) \quad & f(x)f(z) \\ (v) \quad & k_3(\phi(x), \phi(z)) \end{aligned}$$

For Support Vector Machines, a special type of kernel is used which maps us into a space known as a Reproducing Kernel Hilbert Space.

Definition 2.2.4 (Reproducing Kernel Hilbert Space)**.** Let X be a non-empty set and $\mathcal{H}$ be a Hilbert space consisting of functions $f : X \to \mathbb{R}$.

- A function $k : X \times X \to \mathbb{R}$ is called a reproducing kernel of $\mathcal{H}$, with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ if for all $x \in X$ and the reproducing property.

$$f(x) = \langle f, k(\cdot, x) \rangle$$

- The space H is called a reproducing kernel Hilbert space over X if for all $x \in X$ the Dirac functional $\delta_x : H \to \mathbb{R}$ defined by:

$$\delta_x(f) := f(x), \qquad f \in H$$

  is continuous.

The RKHS allows us to ensure the existence of an inner product and the ability to evaluate each function in the space at every point in the domain. Every kernel has a unique RKHS and every RKHS has a unique kernel. From this definition we can say that if two functions inside an RKHS are close in norm, then each of the functions are pointwise close. i.e. for two functions, f and g, if $||f - g||$ is small then $|f(x) - g(x)|$ is small for all x.

A visual representation of how basis the functions of an RKHS can be used to recreate functions can be seen in figure 2.8. Each time we have a data point we pass it to the kernel function, which in this case is a Gaussian kernel. These Gaussians produced by the kernel function are then summed together in order to create a new function, f. We can think of the kernel as assigning a distance measure at each point. When we give the kernel the data points x and x', we are taking a distance measure between these two points. The functions inside of an RKHS can therefore be thought of as linear combinations of distance measures with the kernel regulating the resolution of the distance measure.

Every reproducing kernel is also positive-definite and each positive-definite kernel defines a unique RKHS. Feature maps are also linked to RHKS. Every feature map defines a kernel via:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

K is symmetric, and the positive definite property follows from the properties of the inner product and therefore has the properties of an RKHS. Conversely, every positive definite function and corresponding reproducing kernel Hilbert space has infinitely many associated feature maps. These properties also make support vector machines, and kernel methods in general very attractive in quantum machine learning. The reasons as to why will be explored later.

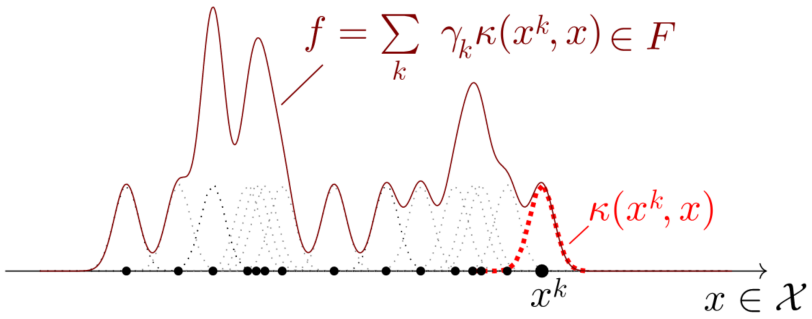


Figure 2.8: A visualisation showing how a general function can be constructed within an RKHS. Image taken from "Supervised quantum machine learning models are kernel methods" by Maria Schuld [25]

.

2.3 Fourier Analysis

Fourier analysis is a widely used tool in modern computing and is named after the French mathematician Jean Baptiste Fourier. It is primarily used for complex signal analysis, and allows us to break down waves into their fundamental components. This has uses in everything from telecommunications to breaking down wave functions in quantum mechanics, and is useful for many aspects of quantum computing. We will first define the Fourier series and give some of its properties, and then go on to discuss the limitations of Fourier analysis.

The Fourier series tells us that if a function is periodic then it can be written as a discrete sum of trigonometric functions with specific frequencies. Consider a function $f(x)$ which is periodic on the interval $0 \leq x \leq L$. Fourier's theorem tells us that any periodic function can be written as:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2\pi n x}{L}\right) + b_n \sin\left(\frac{2\pi n x}{L}\right) \right] \tag{2.11}$$

The coefficients a_n and b_n are defined by:

$$a_n = \frac{2}{L} \int_0^L f(x) \cos\left(\frac{2\pi n x}{L}\right) dx \tag{2.12}$$

and

$$b_n = \frac{2}{L} \int_0^L f(x) \sin\left(\frac{2\pi n x}{L}\right) dx \tag{2.13}$$

This can require an infinite sum to reproduce the original function. Often in practice there are certain terms of the Fourier series that dominate the sum and have a larger weight in determining the shape of the function. These terms will have higher coefficient values and in practical use cases it is usually enough to only calculate a small number of leading terms in the Fourier series in order to have a good representation of the function [26].

An interesting property of the sine and cosine functions that make up a Fourier series is that they are orthogonal.

$$\int_0^L \cos(mx) cos(nx) dx = \frac{1}{2} \int_0^L \cos((n-m)x) + \cos((n+m)x) dx = \pi \delta_{nm}$$

similarly,

$$\int_0^L \sin(mx) \sin(nx) dx = \pi \delta_{nm}$$

and

$$\int_0^L \sin(mx) \cos(nx) dx = 0$$

The sine and cosine functions which make up the Fourier series form an orthonormal basis in a Hilbert space and can also be written in terms of exponentials.

$$\cos(z) = \frac{e^{iz} + e^{-iz}}{2} \qquad \sin(z) = \frac{e^{iz} - e^{-iz}}{2i} \tag{2.14}$$

This allows us to rewrite equation 2.11 as:

$$f(x) = \sum_{n=-\infty}^{\infty} C_n e^{\frac{i2\pi nx}{L}} \tag{2.15}$$

with

$$C_n = \frac{1}{L}\int_0^L f(x) e^{\frac{-i2\pi nx}{L}} dx \tag{2.16}$$

In practice, the sum of the Fourier series goes from $-N$ to N as it speeds up computation drastically. The higher the value of N the more frequencies we have access to and therefore we usually obtain a more accurate approximation of the function.

The Fourier transform can also be applied to functions which are not periodic. To do this we replace the discrete sum with a continuous integral.

$$f(x) = \int_{-\infty}^{\infty} C(k) e^{ikx} dk \tag{2.17}$$

where,

$$C(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x) e^{-ikx} dx \tag{2.18}$$

The Fourier transform is often used to switch between the time and frequency representations of functions. The same technique is used in quantum mechanics to switch between position and momentum space. To more clearly illustrate the time and frequency space, as well as the Fourier transform in general, we can look at the Fourier transform of a square wave.

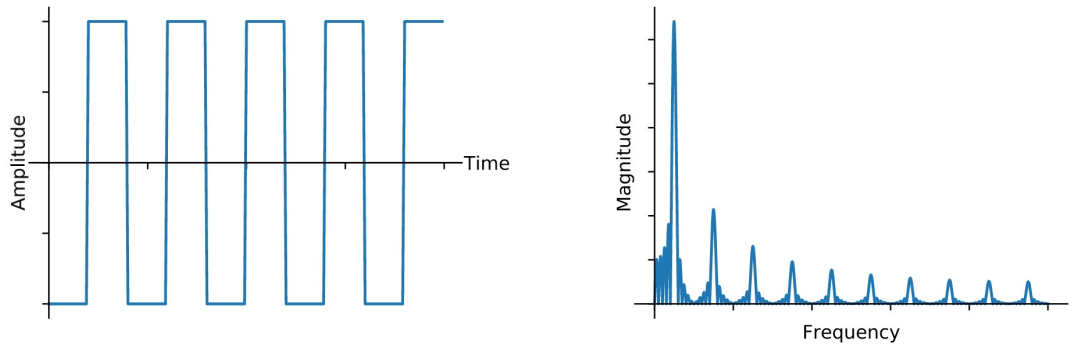


Figure 2.9: The frequency and time representations of a square wave function. The square wave function is shown on the left and the Fourier coefficients of the square wave are shown on the right.

The example we will look at is the square wave, also known as the periodic odd step function, pictured on the left in figure 2.9. The function has a value of 1 from $0 < x \leq \pi$ and a value of 0 for $\pi < x \leq 2\pi$ with this pattern repeating. We know that the function is odd and therefore we know that only the sine terms will contribute to the Fourier series.

We therefore only have to calculate the b_n coefficients.

$$\begin{aligned} b_n &= \frac{2}{L}\int_{\frac{-L}{2}}^{\frac{L}{2}} f(x)\sin\left(\frac{2\pi nx}{L}\right)dx \\ &= 2\cdot\frac{2}{L}\int_0^{\pi}\sin\left(\frac{2\pi nx}{L}\right)dx \\ &= \frac{2}{\pi n}(1-\cos(\pi n)) \end{aligned} \tag{2.19}$$

This gives us a Fourier series for the square wave of

$$\begin{aligned} f(x) &= \sum_{n=1,odd}^{\infty}\frac{4}{\pi}\sin\left(\frac{2\pi nx}{L}\right) \\ &= \sum_{n=0}^{\infty}\frac{4}{\pi}\sin\left(\frac{2\pi(2n+1)x}{L}\right) \end{aligned} \tag{2.20}$$

We can now visualise the results by plotting the approximation for the square wave with an increasing number of terms as seen in figure 2.10. We can also plot the frequency and time domains of the square wave function.

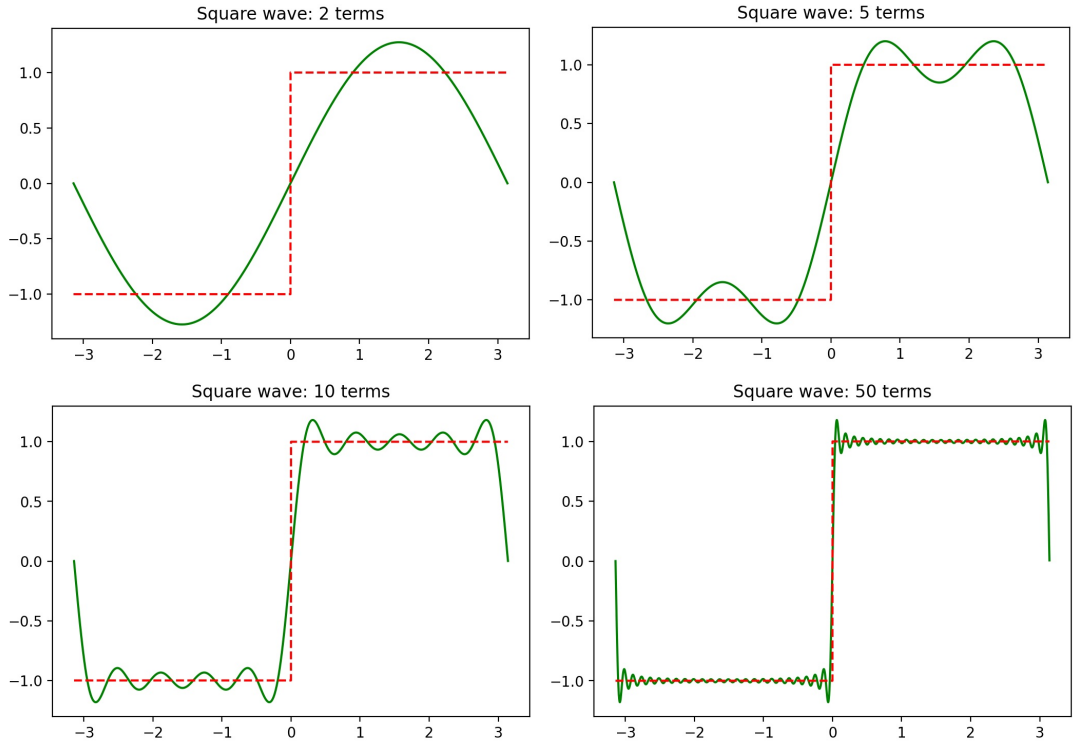


Figure 2.10: We have the plots of four different Fourier approximations of the square wave. Each plot has more terms of the Fourier series than the previous plot with the number of terms appearing above each plot. An ideal square wave is plotted with a red dotted line in each plot so we can compare our approximation to what we are trying to approximate. We can see that as more terms are included the function more closely resembles the square wave function.

While Fourier analysis has many uses, it is not always the best tool to use when approximating functions. As has been shown, any function can be approximated, but the approximations are not always true to the original function. It can often be computationally expensive to make the series accurate enough for every use case. The other main weakness is that real world data is often not periodic and can contain transients. Fourier analysis is not able to accurately account for these things when the number of terms in the Fourier series is constrained and therefore other methods must be used. One of these methods is wavelet analysis which will be discussed next.

2.4 Wavelet Analysis

Real world data and signals are often not periodic, and that can make performing Fourier transforms more computationally expensive. Data often has transients, which are sharp spikes in data. This can be seen in things like stock market data and complex audio waves. Images also often have large smooth regions which are interrupted by edges or sharp changes in contrast. Fourier analysis can't reliably account for all of these factors when computational or time resources are constrained. One of the main reasons for this is that the waves used are not localised in time [27].

The basis functions of Fourier analysis are sine and cosine functions which are periodic across the entire spatial domain, or for all of time. This means that a singular spike cannot be easily represented.

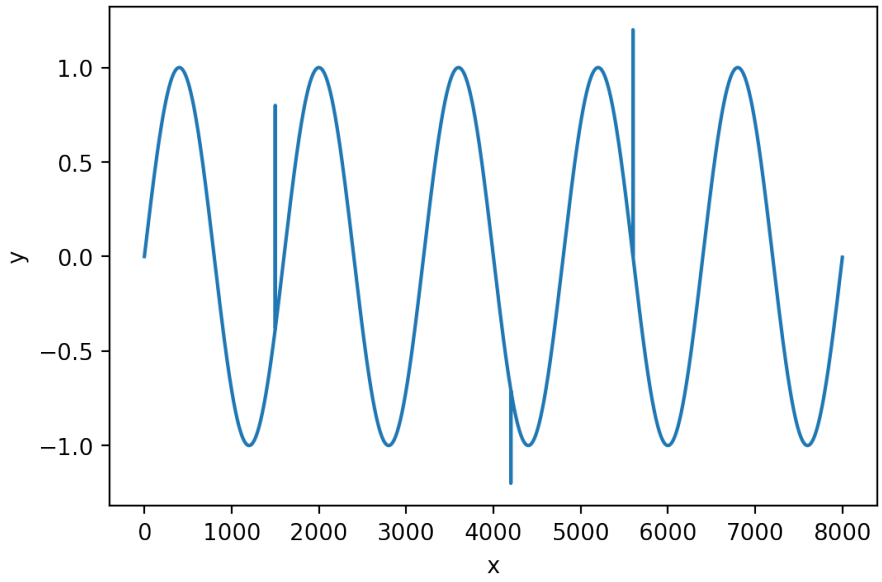


Figure 2.11: An example of transients within periodic data which are hard to approximate using Fourier analysis. Here we have a sin function which has 3 transients in the data causing a quick jump up or down. The transients have no discernable period to them.

To accurately analyse signals with abrupt changes, we need a new class of functions which are localised in time and frequency. These functions will act as a new basis for the Hilbert space. The functions which are usually used for this are wavelets, which are rapidly decaying wavelike oscillations that have zero mean. Wavelets can have many

shapes and different wavelets are suited for different problems. Some popular wavelets include Daubechies, Biorthogonal, Morlet and the Mexican hat wavelet.

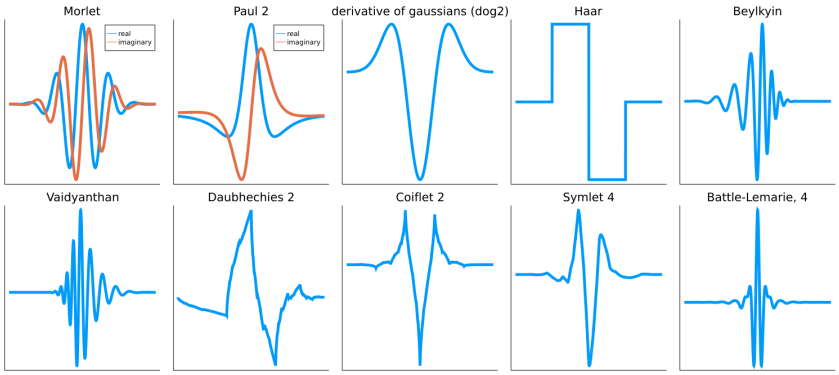


Figure 2.12: Plots of various mother wavelet functions [28].

The basis functions of wavelet analysis are known as mother wavelets, Ψ

$$\Psi_{(a,b)}(t) = \frac{1}{\sqrt{a}}\Psi\left(\frac{t-b}{a}\right) \tag{2.21}$$

Here the variable, b, can be viewed as a factor which moves the wavelet around in time and a would increase or decrease the frequency of the wavelet. Using this we can express the wavelet transform as the inner product between the mother wavelets and a function f:

$$W_\Psi(f)(a,b) = \left\langle f(t), \Psi_{(a,b)}(t) \right\rangle \tag{2.22}$$

By varying the values of a and b we create functions which are orthogonal to each other. These orthogonal functions can then be used to create an orthogonal basis. By taking the inner products of each wavelet with the function we wish to decompose, we can break down any function and represent it in the basis of the orthogonal wavelets. This is essentially the same process as the Fourier transform, but we now have frequency and time data about a function instead of just one or the other [29].

There are two types of wavelet transforms: discrete and continuous transforms. The continuous transform is used for frequency analysis and filtering out time localised frequency components from data. Discrete wavelet transforms are often used for de-noising and the compression of data.

This brings to a close the discussion of the necessary concepts used in classical computing which will be needed in this thesis. We now move on to the quantum concepts which will be needed. We start by going over some of the notation used in quantum mechanics and laying out the postulates of quantum mechanics. We then move on to quantum computing and quantum machine learning.

2.5 Quantum Mechanics

In order to understand quantum computing, we must lay out the parts of quantum mechanics which are used for quantum computing. We start by looking at Dirac notation, which is used throughout quantum mechanics and quantum computing. We then give an overview of the postulates of quantum mechanics before going over density states, which are used in the formulation of quantum kernels. We then look at the quantum Fourier transform and finally look at Shor's's algorithm, which is an implementation of the quantum Fourier transform that gives us an advantage over classical computing.

2.5.1 Dirac Notation

Dirac notation is also known as bra-ket notation and is used throughout quantum mechanics to represent quantum states. Kets denote vectors in abstract complex vector spaces. For most of our cases, the vector space will be a Hilbert space. The ket representation of a vector $\boldsymbol{\alpha}$ with coefficients α_i is:

$$|\alpha\rangle = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} \tag{2.23}$$

The adjoint of a ket is called a bra. The bra of the vector $\boldsymbol{\alpha}$ is denoted by:

$$\langle\alpha| = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix} \tag{2.24}$$

Dirac notation is useful because it provides a compact way of expressing large vectors and performing operations on and with them. Using Dirac notation we can compactly represent the inner product between two vectors. If we take two vectors, $|\alpha\rangle$ and $|\beta\rangle$, from a Hilbert space with an orthonormal basis, the inner product using Dirac notaion is:

$$(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \langle\alpha|\beta\rangle = \sum_{i=1}^{\infty} \alpha_i^* \beta_i \tag{2.25}$$

If we place a number in front of a bra or ket this has the effect of multiplying each element contained in the bra or ket by the integer.

$$x\,|\alpha\rangle = x \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} x \cdot \alpha_1 \\ x \cdot \alpha_2 \\ \vdots \\ x \cdot \alpha_n \end{bmatrix} \tag{2.26}$$

Lastly we can multiply kets with matrices, and this is used in quantum mechanics for applying measurement operators to systems.

2.5.2 Postulates of Quantum Mechanics

In this section we will go over 4 postulates of quantum mechanics as laid out in Nielsen and Chuang [8]. These are statements which we make and assume to be true when beginning to formulate a theory or build a model. We then derive the properties of the theory or model from these postulates.

Postulate 1:
Any isolated physical system has a state space. A state space is a Hilbert space with inner product that is completely described by its state vectors. A state vector is a unit vector in the system's state space.

Postulate 2:
The evolution of a closed quantum system is described by a unitary transformation. The state, $|\psi\rangle$, of the system at time, t_1, is related to the state $|\psi'\rangle$ at time, t_2 by a unitary operator which depends only on time.

$$|\psi'\rangle = U(t_1, t_2)\,|\psi\rangle \tag{2.27}$$

The time evolution of the state of a closed quantum system is described by the Schrödinger equation:

$$i\hbar\frac{d\,|\psi\rangle}{dt} = H\,|\psi\rangle \tag{2.28}$$

Where $\hbar$ is the reduced Planck constant and H is the Hamiltonian, a measurement operator which gives the total energy of a system.

Postulate 3:
Quantum measurements are described by measurement operators, $\{M_m\}$. The different outcomes of the measurements are represented by the subscript, m. Measurement operators act on the state space of the system being measured. If the state of the quantum system is $|(\rangle\,\psi)$ immediately before the measurement then the probability that the results m occurs is:

$$p(m) = \langle\psi|\,M_m^\dagger M_m\,|\psi\rangle\,. \tag{2.29}$$

The state of the system after the measurement is given by:

$$\frac{M_m|\psi\rangle}{\sqrt{\left\langle\psi\left|M_m^\dagger M_m\right|\psi\right\rangle}}. \tag{2.30}$$

The measurement must also satisfy the completeness equation given by:

$$\sum_m M_m^\dagger M_m = I. \tag{2.31}$$

The satisfaction of the completeness equation indicates that the sum of the probabilities will be one.

$$1 = \sum_m p(m) = \sum_m \left\langle\psi\left|M_m^\dagger M_m\right|\psi\right\rangle \tag{2.32}$$

Postulate 4:
The state space of a compostie physical system is the tensor product of the state spaces of the composite physical systems. For a system of n states, numbered 1 through n, where a system i is prepared in the state $|\phi_i\rangle$ the total system is given by:

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle\,. \tag{2.33}$$

2.5.3 The Density Operator

The postulates of quantum mechanics laid out above are described using state space and state vectors. This formalism is used throughout quantum computing. However, it is also useful to view quantum mechanics through the alternate formalism of the density operator and density matrix. The mathematics is equivalent, but conceptually it can allow us to more easily tackle certain questions within quantum computing.

The density operator allows us to more easily describe quantum systems with states that are not completely known. It is a generalisation of state vectors which allows us to represent mixed states. Mixed states occur when there are systems which are entangled together or when the preparation of the system is not entirely known. The density operator of a system is defined as:

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right| \tag{2.34}$$

Where $|\psi_i\rangle$ is one of the possible states of a quantum system, and p_i are the probabilities of these states. The ensemble of pure states is labeled $\{p_i |\psi_i\rangle\}$. An operator can only be the density operator of some ensemble $\{p_i |\psi_i\rangle\}$ if and only if $tr(\rho) = 1$ and ρ is a positive operator. An operator, T, in a Hilbert space, $\mathcal{H}$ is a positive operator if:

$$\begin{aligned} T &= T^*, \\ \langle v|Tv\rangle &\geq 0, \quad v \in \mathcal{H} \end{aligned} \tag{2.35}$$

The density operator must be positive and have a unit trace, this is because we need to associate a probabilistic interpretation to the density operator. Using this formalism, we can rewrite the postulates stated above as:

Postulate 1:
Associated with any isolated physical system is a Hilbert space, which is known as the state space of the system. The system is completely described by its density operator, which is a positive operator, with trace 1, acting on the state space of the system. If a quantum system is in the state ρ_i, then the density operator for the system is, $\sum_i p_i \rho_i$.

Postulate 2:
The evolution of a closed quantum system is described by a unitary transformation. The state, ρ, of the system at time, t_1, is related to the state ρ' at time, t_2, by a unitary operator which depends only on the time.

$$\rho' = U(t_1, t_2)\rho U(t_1, t_2)^\dagger \tag{2.36}$$

Postulate 3:
Quantum measurements are described by measurement operators, $\{M_m\}$. The different outcomes of the measurements are represented by the subscript, m. Measurement operators act on the state space of the system being measured. The only possible result of a measurement of an observable, $\hat{M}$, is one of the eigenvalues of the corresponding operator

$\hat{M}$. If the state of the quantum system is ρ immediately before the measurement then the probability that the result m occurs is given by:

$$p(m) = \text{tr}\left(M_m^\dagger M_m \rho\right). \tag{2.37}$$

The state of the system after the measurement is given by:

$$\frac{M_m \rho M_m^\dagger}{\text{tr}\left(M_m^\dagger M_m \rho\right)}. \tag{2.38}$$

The measurement must also satisfy the completeness equation given by:

$$\sum_m M_m^\dagger M_m = I. \tag{2.39}$$

Postulate 4:
The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. If we have states numbered 1 through n, and the system number, i, is prepared in the state ρ_i, then the joint state of the total system is $\rho_1 \otimes \rho_2 \otimes cdots \otimes \rho_n$.

One of the most useful forms of the density operator is the reduced density operator. It is used as a descriptive tool for sub-systems of a composite quantum system. Let A and B be physical systems whose state is described by the density operator, ρ^{AB}. The reduced density operator for system A is:

$$\rho^A = \text{Tr}_B\left(\rho^{AB}\right) \tag{2.40}$$

Where tr_B is the partial trace over system B. The partial trace is linear and is defined as [8, 30]:

$$\text{Tr}_B\left(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|\right) = |a_1\rangle\langle a_2|\,\text{Tr}\left(|b_1\rangle\langle b_2|\right) \tag{2.41}$$

Where $|a_1\rangle$ and $|a_2\rangle$ are any two vectors from the state space of system A and $|b_1\rangle$ and $|b_2\rangle$ are any two vectors from the state space of system B. The trace operation on the right hand side is given by:

$$\text{Tr}(|b_1\rangle\langle b_2|) = \text{Tr}\,\langle b_2|b_1\rangle \tag{2.42}$$

We now have a very simple overview of the concepts from quantum mechanics which we will need in order to explore the field of quantum computing.

2.6 Quantum Computing

2.6.1 Qubits

Quantum bits, or qubits, are the basic unit of information in quantum computing. A qubit is any quantum mechanical system in the Hilbert space $\mathbb{C}^2$. They are typically denoted by the Dirac vectors, $|0\rangle$ and $|1\rangle$, which form the orthonormal basis of a 2 dimensional Hilbert space known as the computational basis. They may also be represented in vector and a geometric representation depending on the problem. The states correspond directly to 0 and 1, the two states a bit can assume in classical computing [8]. The most general form of a qubit arises from representing it as a superposition of the two basis vectors.

$$|\psi\rangle = \alpha_1 |0\rangle + \alpha_2 |1\rangle \tag{2.43}$$

where $\alpha_1, \alpha_2 \in \mathbb{C}$. The coefficients must also be normalised, $|\alpha_1|^2 + |\alpha_2|^2 = 1$ The vector notation representation of a qubit is:

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} \tag{2.44}$$

The basis states can also be represented in the vector notation as:

$$\begin{aligned} |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned} \tag{2.45}$$

And the geometric representation is:

$$|\psi\rangle = e^{i\gamma} \left(\cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle \right) \tag{2.46}$$

Where, $\theta, \gamma, \phi, \in \mathbb{R}$ with, $0 \leq \theta \leq \pi$ and $\leq \phi \leq 2\pi$. γ is a global phase factor.

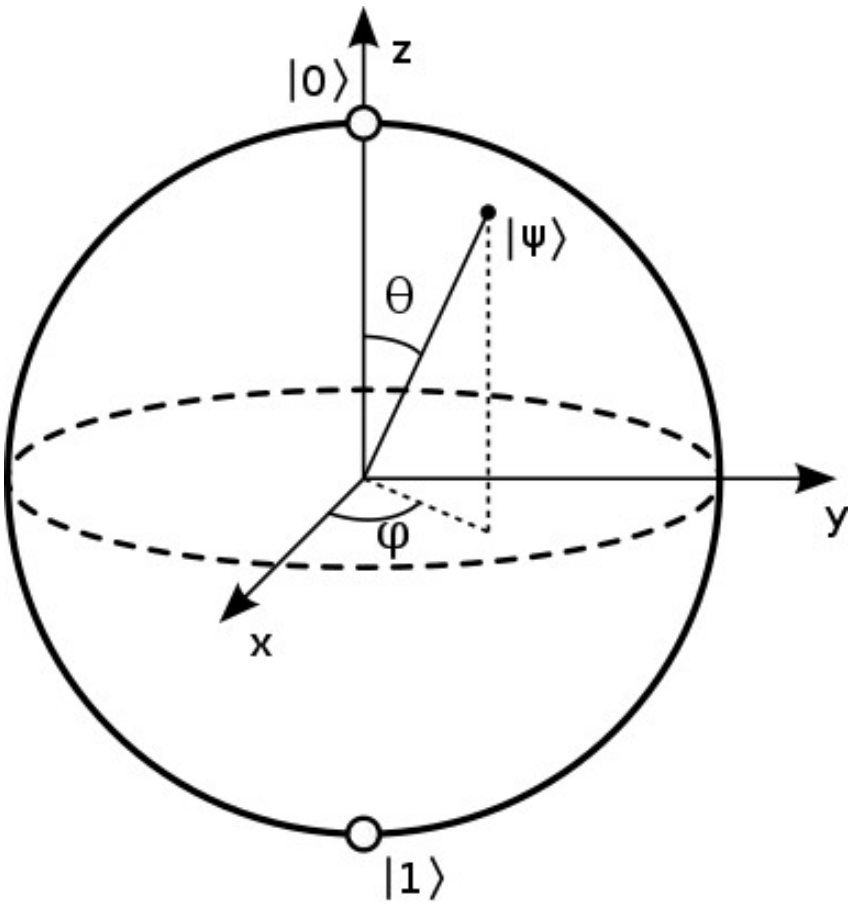


Figure 2.13: The geometric representation of a qubit, also called a Bloch Sphere.

Dirac notation is the most common way of representing qubits as, for one, it allows us to compute inner products easily. Vector notation is useful when trying to understand the effect of quantum gates and building quantum circuits. The geometric representation of a qubit is known as a Bloch sphere, and can be seen in figure 2.13. A Bloch sphere is a unit 2-sphere with the North and South poles corresponding to the $|0\rangle$ and $|1\rangle$ states, respectively. The points on the surface of a Bloch sphere correspond to the pure states of the system and the interior points correspond to mixed states. This notation is often used when doing rotations and encodings.

Given two qubits:

$$|\psi_1\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$$
$$|\psi_2\rangle = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

The inner product of two qubits is given by:

$$\langle\psi_1|\psi_2\rangle = \alpha_1^*\beta_1 + \alpha_2^*\beta_2 \tag{2.47}$$

and the outer product is:

$$|\psi_1\rangle \langle\psi_2| = \begin{pmatrix} \alpha_1\beta_1^* & \alpha_1\beta_2^* \\ \alpha_2\beta_1^* & \alpha_2\beta_1^* \end{pmatrix} \tag{2.48}$$

We can also describe systems with multiple qubits.

Definition 2.6.1 (Separable state)**.** Let $\{|a_i\rangle\}_{i=1}^n \subset \mathcal{H}_1$ and $\{|b_j\rangle\}_{j=1}^n \subset \mathcal{H}_2$ be orthonormal bases for H_1 and H_2, a pure or separable state of the composite system can be written as:

$$\begin{aligned} |\psi\rangle &= \sum_{i,j} c_{i,j} \left(|a_i\rangle \otimes |b_j\rangle\right) \\ &= \sum_{i,j} c_{i,j} |a_i b_j\rangle \end{aligned} \tag{2.49}$$

If the qubits are entangled then the states of the individual qubits are no longer separable. To represent this we rewrite the tensor product between two qubits, $|0\rangle$ and $|1\rangle$, as $|01\rangle$. The entangled state then becomes:

$$|\psi\rangle = \alpha_1 |0\cdots 00\rangle + \alpha_2 |0\cdots 01\rangle + \cdots + \alpha_{2^n} |1\cdots 11\rangle \tag{2.50}$$

Writing out states like this can be very tedious and therefore a different notation is used in order to compress the expressions.

$$|\psi\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle \tag{2.51}$$

The state $|i\rangle$ refers to the i^{th} component of the computational basis state in the basis. The binary integers in the state correspond to $i-1$. For example, $|5\rangle = |100\rangle$ as 100 is the binary representation of the number 5-1. We can also use this notation to represent a general density matrix with mixed states:

$$\rho = \sum_{i,j=1}^{2^n} \alpha_{ij} |i\rangle \langle j| \tag{2.52}$$

2.6.2 Quantum Gates

Quantum gates perform two operations that are central to quantum computing, the implementation of quantum logic gates and measurement operations. Quantum logic gates are used to perform operations on one or more qubits in order to perform computations. The measurement operations project the qubits into the computational basis, which allows us to see the results of a quantum circuit after it has been run, and interpret the results.

Quantum logic gates are represented by unitary transformations. On a single qubit this takes the form of a 2x2 unitary matrix. The operations must be unitary so that they preserve the normalisation condition on the state vector. A set of quantum gates that allows us to approximate any quantum gate to any desired precision is called a universal gate set. Using the Solovay-Kitaev theorem we can also approximate any single qubit gate to arbitrary precision using a set of fixed, single qubit gates as long as the set generates a dense subset in $SU(2)$ [31].

One example of a universal set of gates is the CNOT gate, which can be represented in matrix form as:

$$U_{CN} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.53}$$

The CNOT gate uses a controlled qubit: if the control qubit is set to 1 then the target qubit is flipped; if the controlled qubit is 0 then the target qubit is left alone. A CNOT gate can be seen as a generalisation of the classical XOR gate. The possible state outcomes of a two qubit system interacting with a CNOT gate can be seen in the table below.

Before		After	
Control	Target	Control	Target
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Some of the most common single qubit gates are the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.54}$$

The effect of the Pauli gates can be visualised as rotating a qubit by π degrees on a Bloch sphere, with the rotation occurring along the axis of Pauli matrix. For example, the Pauli-X matrix can be used to perform a bit flip operation by rotating the qubit π degrees along the x axis:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{2.55}$$

There is also the Hadamard gate, which is used to entangle qubits, the phase gate, denoted by S, and the $\frac{\pi}{8}$ or T gate. The matrix representations of the aforementioned gates are given by:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{2}} \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \tag{2.56}$$

These gates are important as they allow us to manipulate the states of our qubits and by doing so in specific ways we are able to perform computations. The Hadamard gate is of particular interest as it allows us put a qubit into an equal superposition of the basis states:

$$\begin{aligned} |0\rangle &\mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle &\mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned} \tag{2.57}$$

The phase gate is used to rotate a qubit by a quarter turn around the Bloch sphere. It is also interesting to note that unlike the Pauli gates the phase gate is not its own

inverse. Phase gates can be adjusted to rotate qubits to any position within a Bloch sphere and this is useful for encoding classical information in to qubit, something which will be explored in depth in a later section.

In classical computing multiple bit gates are an integral part of computation. The NAND, *not-and*, gate is known as the universal gate because any function on bits can be computed using just NAND gates. When it comes to quantum computing, it is natural to think that we should try and implement the same gates on quantum computers because if a NAND gate can be implemented on a quantum computer then we would be able to perform the function of any classical gate.

However, it turns out that the NAND gate cannot be implemented on a quantum computer directly. The reason for this is that the NAND gate, as well as some of the other classical gates, are irreversible and non-invertible. Quantum gates are required to be unitary to preserve the normalisation condition and, unitary gates are also reversible [8].

2.6.3 Quantum Circuits

Quantum circuits are theoretical objects made up of gates which are connected by wires and look quite similar to the circuit representations used in classical computing. Each wire corresponds to a qubit and shows the evolution of the qubit through the circuit in time. The qubits are changed and can interact with each other via gates placed throughout a circuit. A key difference between quantum and classical circuit diagrams is that in classical circuits, the wires represent the flow of information, whereas in a quantum circuit the wires show how the qubits are being changed over time.

The circuit representation of some of the gates we have talked about can be seen in figure 2.14. The circuit representation of the CNOT gate can be seen at the bottom of figure 2.14. The black circle on the wire identifies that wire is the controlled qubit and the $\oplus$ represents addition modulo 2.

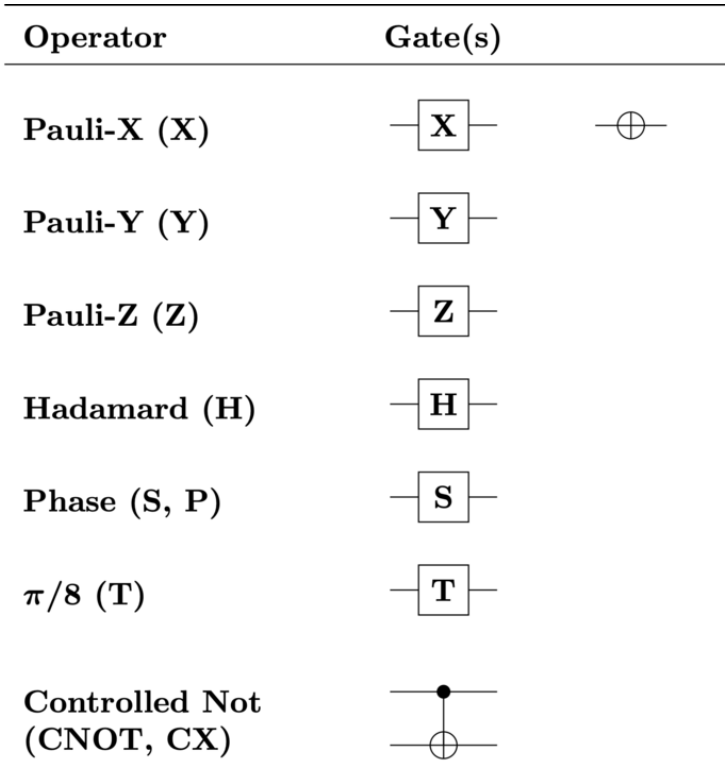


Figure 2.14: The circuit representation of a several different quantum circuit gates. Image taken from [32].

There are also certain elements of classical circuits which cannot be emulated with quantum circuits. For example, a classical circuit operation which merges two or more bits and outputs a single bit known as FANIN, cannot be performed on a quantum computer because it is not reversible. Quantum circuits are also acyclic. This means that wires cannot loop back to previous points in the circuit. This is because looping back in a quantum circuit would signify sending information back in time which, is not possible. Gates can be repeated multiple times in a quantum circuit but we do not visually represent that with a loop as we might in a classical circuit diagram.

While specific circuit elements cannot always be implemented, the circuit itself can always be simulated efficiently by a quantum circuit. This means that although we may not be able to create one-to-one copies of classical circuits on a quantum computer, the computation of the classical circuit is always possible on a quantum computer using appropriate transforms.

Although quantum circuits are theoretical objects they, can be implemented physically. This can be done in many different ways as long as the physical implementations are able to perform all the functions laid out by the quantum circuit. According to DiVincenzo, there are five characteristics that we should look for when looking to create physical implementations of quantum circuits [33].

The first is that the qubits should be well characterised and scalable. This means that the physical parameters of the qubits should be well known and that we should be able

to build on the physical system in order to implement more complex quantum circuits. The second characteristic is the ability to initialise the qubits into a base state reliably. This is useful because it allows us to minimise variance between runs of the same circuit and helps with error correction.

The third characteristic is that decoherence times should be much longer relative to the gate operation times. Physical implementations of qubits are often prone to noise and outside interference. We would prefer systems which minimise this interference so we can be more confident that the operations performed on the qubit by the gates are the only interactions that the qubit has had. If this is true we can perform quantum computations more reliably and have less need for quantum error correction. The fourth characteristic is that the gates should be universal. This means that we would like to have the ability to perform arbitrary single and multi-qubit operations thereby allowing us to to implement any quantum circuit on the physical system. The last characteristic is the ability to measure individual qubits. This is needed so that we can accurately take measurements once the circuit has run and get the results of what the quantum circuit has computed.

The above points characterise an ideal quantum computer. However, it is impossible to physically realise an ideal quantum computer. Current implementations of quantum computers are prone to noise, and decoherence times are not always as long as we would like. To help get around this, we use quantum error correction. Quantum error correction is a sub-field of quantum computing with a lot depth and research however, it is outside of the scope of this thesis.

2.6.4 Quantum Parallelism

At this point we have all the tools needed to implement a quantum algorithm. The first algorithm we will explore allows us to extract the global properties of a function. A global property is true for the whole space, when viewed as a single neighborhood. We can achieve this by using superposition to evaluate many values of x in parallel, hence the name quantum parallelism.

The first example will not give any advantage over a classical computer, but it is useful to understand how a quantum computer operates. In the next section we will alter the algorithm in order to obtain a quantum advantage. The general idea is to prepare an equal superposition over all basis states. Once we have this state we pass it through a unitary operation which evaluates a function in parallel on all inputs by virtue of the linearity of unitary transforms.

Consider a function $f(x)$, $f : \{0, 1\} \to \{0, 1\}$. This is a function that takes a bit as an input and outputs a bit. We want to construct a unitary transformation which uses two qubits, x and y:

$$U_f : (x, y) \to (x, y \oplus f(x)) \tag{2.58}$$

Here $\oplus$ is addition modulo 2. The possible outcomes of adding two bits modulo 2 are, $0 \oplus 0 = 1$, $0 \oplus 1 = 1$, and $1 \oplus 1 = 0$. The reason why we use two qubits instead of one

is because the unitary operations must be reversible and if we use one qubuit this is no longer the case. We can express the unitary transform in operator notation as:

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle \tag{2.59}$$

At this point we set the qubit, y, to 0 so that after the transform the second qubit is only the value of $f(x)$:

$$U_f : (x, 0) \to (x, 0 \oplus f(x)) = (x, f(x)) \tag{2.60}$$

Consider the circuit pictured in figure 2.15. We start by applying the Hadamard gate to the $|0\rangle$ qubit.

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.61}$$

Applying the unitary transform from equation 2.59 to the qubits x and y both in the state, $|0\rangle$ results in a state, $|\psi\rangle$:

$$\begin{aligned} |\psi\rangle &= U_f(H|0\rangle \otimes |1\rangle) \\ &= U_f \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle \otimes |f(0)\rangle + |1\rangle \otimes |f(1)\rangle) \end{aligned} \tag{2.62}$$

The end result of this circuit is a state which contains information about both $f(1)$ and $f(0)$, however when we take a measurement in the computational basis we are only able to extract information about either $f(1)$ or $f(0)$.

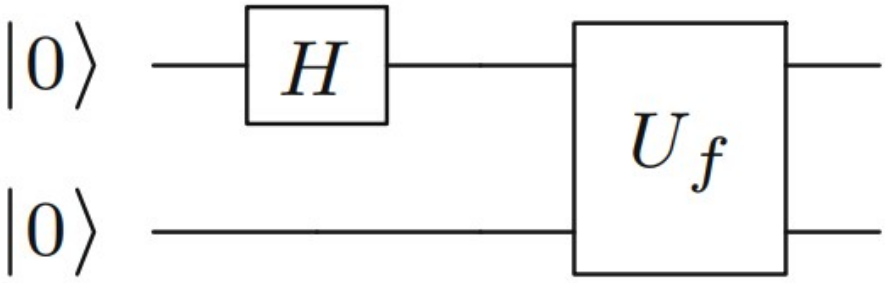


Figure 2.15: A simple quantum circuit to demonstrate quantum parallelism.

2.6.5 The Deutsch Algorithm

Binary functions that take in a single bit and output a single bit, such as the one used above, can either be constant or balanced. In order to know which, we would usually have to compute the function for both possible inputs. Using Deutsch's algorithm allows us to find this information by running a quantum circuit once. The entire circuit can be described by:

$$|\psi_3\rangle = (H \otimes \mathbb{1})U_f(H \otimes H)|01\rangle \tag{2.63}$$

The circuit can be seen in figure 2.16. The state of the qubits in the circuit initially is labelled $|\psi_0\rangle$ and the index is increased by one each time a new gate is applied. This allows

us to differentiate between the different states that occur in the circuit as it evolves. At the start of the circuit the two qubits are in the state $|10\rangle$ and each qubit passes through a Hadamard gate, giving us:

$$\begin{aligned} |\psi_1\rangle &= (H \otimes H)\,|01\rangle \\ &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\ &= \frac{1}{2}\left(\sum_{x=0}^{1}|x\rangle\right) \otimes (|0\rangle - |1\rangle) \end{aligned} \tag{2.64}$$

We now evaluate the action of U_f on $|\psi_1\rangle$, to do this we have to consider the case where $f(x) = 1$ and $f(x) = 0$ separately. For $f(x) = 0$:

$$\begin{aligned} U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) &= |x, 0 \oplus f(x)\rangle - |x, 1 \oplus f(x)\rangle \\ &= |x, 0 \oplus 0\rangle - |x, 1 \oplus 0\rangle \\ &= |x\rangle\,(|0\rangle - |1\rangle) \end{aligned} \tag{2.65}$$

and for the case of $f(x) = 1$:

$$\begin{aligned} U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) &= |x, 0 \oplus f(x)\rangle - |x, 1 \oplus f(x)\rangle \\ &= |x, 0 \oplus 1\rangle - |x, 1 \oplus 1\rangle \\ &= -\,|x\rangle\,(|0\rangle - |1\rangle) \end{aligned} \tag{2.66}$$

The state $|\psi_2\rangle$ can then be expressed as:

$$|\psi_2\rangle = \frac{1}{2}\left(\sum_{x=0}^{1}(-1)^{f(x)}\,|x\rangle\right) \otimes (|0\rangle - |1\rangle) \tag{2.67}$$

Lastly, we apply a Hadamard gate to the first qubit:

$$|\psi_3\rangle = (H \otimes \mathbb{1})\left(\frac{1}{2}\left(\sum_{x=0}^{1}(-1)^{f(x)}\,|x\rangle\right) \otimes (|0\rangle - |1\rangle)\right) \tag{2.68}$$

The effect of this is:

$$H\left[\frac{1}{2}\left(\sum_{x=0}^{1}(-1)^{f(x)}\,|x\rangle\right)\right] = \frac{1}{2}((-1)^{f(0)} + (-1)^{f(1)})\,|0\rangle + \frac{1}{2}((-1)^{f(0)} - (-1)^{f(1)})\,|1\rangle \tag{2.69}$$

We can now simplify the final state to:

$$|\psi_3\rangle = \pm\,|f(0) \oplus f(1)\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right] \tag{2.70}$$

Now, by measuring the first qubit we can determine whether the function is constant or not. If the first qubit is measured to be in the state $|0\rangle$, then $f(1) = f(0)$ and the function is constant. If it is measured in the state $|1\rangle$ then we know that the function is balanced. This means that we have successfully determined a global property of the

function by taking a single measurement, something which cannot be done on a classical computer.

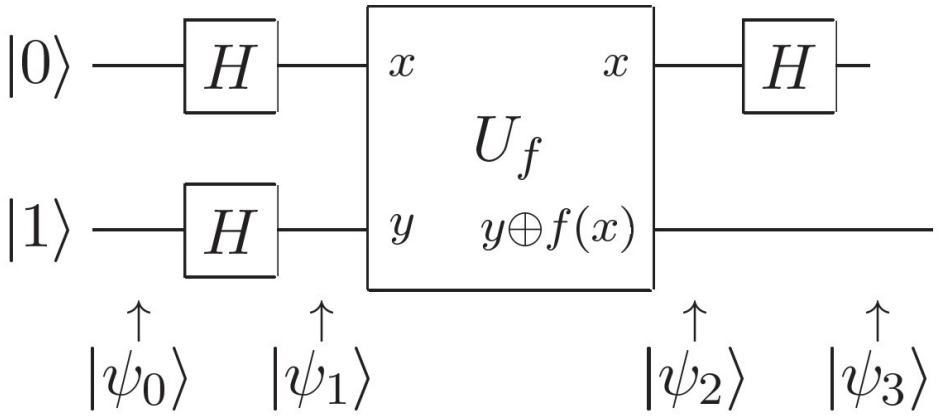


Figure 2.16: A quantum circuit to implement the Deutsch Algorithm.

2.6.6 Quantum Fourier Transform

As mentioned in section 2.3, the Fourier transform is an important tool in many fields. The quantum Fourier transform is an implementation of the discrete Fourier transform, described by equation 2.15 over the amplitudes of a wavefunction. It is used in a multitude of quantum algorithms, most notably Shor's algorithm.

The classical Fourier transform takes a periodic function and gives frequency information, in other words, we go from a time domain to a frequency domain. The quantum Fourier transform takes us from the computational basis into the Fourier state basis. In the computational basis, numbers are stored using binary representations. The quantum Fourier transform encodes the same information by rotating about the z-axis. This can be seen in figure 2.17 where we see the state of the electrons both storing the number 5.

The quantum Fourier transform is useful because it allows us to transform our data into a new basis. This can help us solve certain problems such as phase estimation or the hidden subgroup problem, which will be explored in the next section when looking at Shor's algorithm. Viewing the data in a different basis allows us to see patterns or perform computations that were previously not feasible with the given resources. In the quantum Fourier transform we go from the computational basis to rotating the qubits around the z axis. This puts the qubits into a phase space and makes performing certain computations related to the phase of the qubits easier.

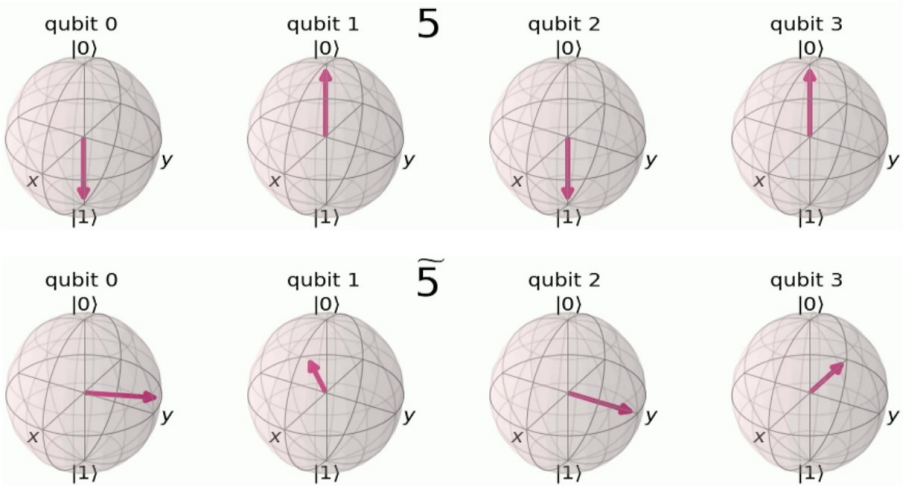


Figure 2.17: The first four qubits are in the z basis and are used to implement the binary representation of 5. $\tilde{5}$ shows the qubits in the Fourier basis and demonstrates how the quantum Fourier transform can be used to embed the same information into a new basis on the same 4 qubits. Image taken from [34].

The quantum Fourier transform performs the mapping of a vector of complex numbers, $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})$ to a vector of complex numbers, $\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})$:

$$QFT : |X\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \rightarrow |Y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \tag{2.71}$$

The formula for the coefficients, y_k is given by:

$$\begin{aligned} y_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{i2\pi \frac{jk}{N}} \\ &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk} \end{aligned} \tag{2.72}$$

Where we have introduced the variable $\omega_N = e^{\frac{i2\pi}{N}}$ to simplify notation.

There are some differences to the classical discrete Fourier transform described in equation 2.15. We now sum from 0 to $N-1$ instead of summing over all numbers. L has been relabelled to N to show the number of qubits instead of the period. The variable, j, is used to describe a state instead of n to avoid confusion with the number of qubits in the system, which is usually labelled n.

We can also represent equation 2.72 as a map of the qubit state as:

$$|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle \tag{2.73}$$

and as a unitary matrix as:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle \langle j| \tag{2.74}$$

We can now express the quantum Fourier transform acting on a state, $|x\rangle = |x_1, x_2, \cdots, x_n\rangle$, composed of $N = 2^n$ qubits:

$$\begin{aligned} \mathrm{QFT}_N |x\rangle &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i xy/2^n} |y\rangle \\ &= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} \left(|0\rangle + e^{2\pi i x/2^k} |1\rangle \right) \end{aligned} \tag{2.75}$$

We now write $|y\rangle$ in fractional binary notation, $|y\rangle = |y_1 \cdots y_n\rangle$

$$\begin{aligned} \mathrm{QFT}_N |x\rangle &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i \left(\sum_{k=1}^{n} y_k/2^k\right) x} |y_1 \ldots y_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=1}^{n} e^{2\pi i x y_k/2^k} |y_1 \ldots y_n\rangle \end{aligned} \tag{2.76}$$

We then rearrange the sum and products and then expand, $\sum_{y=0}^{N=1} = \sum_{y_1 0}^{1} \sum_{y_2=0}^{1} \cdots \sum_{y_n=0}^{1}$

$$\begin{aligned} \mathrm{QFT}_N |x\rangle = \frac{1}{\sqrt{N}} &\left(|0\rangle + e^{\frac{2\pi i}{2} x} |1\rangle \right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}} |1\rangle \right) \otimes \ldots \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}} x} |1\rangle \right) \\ &\otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n} x} |1\rangle \right) \end{aligned} \tag{2.77}$$

Figure 2.18 shows a quantum circuit which is able to implement the quantum Fourier transform on n qubits. The n qubits are labelled $|x_1 x_2 \cdots x_n\rangle$. The circuit uses only two types of gates: the Hadamard gate and the controlled rotation gate. The general idea of the algorithm is to apply a Hadamard gate to a qubit and then apply controlled rotation gates to the qubit for every other qubit below it in the circuit. The process then repeats for all the following qubits, first applying a Hadamard gate and then applying controlled rotations for all the qubits below them. The CROT gate is a two qubit controlled rotation and its action on a two qubit in state $|x_l x_j\rangle$ is:

$$CROT_k |1 x_j\rangle = e^{\left(\frac{2\pi i x_j}{2^k}\right)} |1 x_j\rangle \tag{2.78}$$

Here the first qubit is in a state 1 and therefore the rotation is applied. If the first qubit was in the state 0, the second qubit would be left the same. The CROT gate can also be expressed as:

$$CROT_k = \begin{pmatrix} I_{2\times 2} & 0 \\ 0 & UROT_k \end{pmatrix} \tag{2.79}$$

with

$$UROT_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^i}} \end{pmatrix} \tag{2.80}$$

The Hadamard gate acting on a qubit in state $|x_k\rangle$ is:

$$H |x_k\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi i}{2} x_k} |1\rangle \right) \tag{2.81}$$

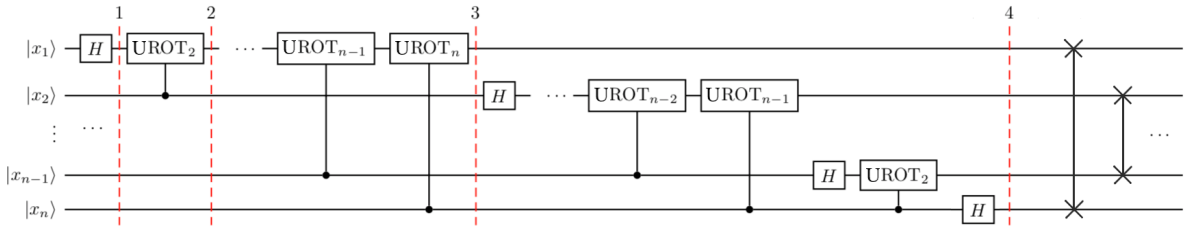


Figure 2.18: A quantum circuit used to implement the quantum Fourier transform. Image taken from [34].

Following the circuits, we start by applying a Hadamard gate, H_1, to the first qubit which leaves the circuit in a state:

$$H_1 |x_1 x_2 \cdots x_n\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi i x_1}{2}} |1\rangle \right) \otimes |x_2 x_3 \cdots x_n\rangle \tag{2.82}$$

After this the $UROT_2$ gate is applied to the first qubit using the second qubit as the control qubit leaving the circuit in the state:

$$\frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^2} x_2 + \frac{2\pi i}{2} x_1\right)} |1\rangle \right] \otimes |x_2 x_3 \ldots x_n\rangle \tag{2.83}$$

After this the rest of the controlled rotations are applied and after the final rotation, $CROT_n$, is applied the circuit is left in the state.

$$\frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^n} x_n + \frac{2\pi i}{2^{n-1}} x_{n-1} + \ldots + \frac{2\pi i}{2^2} x_2 + \frac{2\pi i}{2} x_1\right)} |1\rangle \right] \otimes |x_2 x_3 \ldots x_n\rangle \tag{2.84}$$

We can use the base 2 notation in order to rewrite an integer, x, as:

$$x = 2^{n-1} x_1 + 2^{n-2} x_2 + \ldots + 2^1 x_{n-1} + 2^0 x_n \tag{2.85}$$

This lets us simplify equation 2.84 to:

$$\frac{1}{\sqrt{2}} \left[|0\rangle + \exp\left(\frac{2\pi i}{2^n} x\right) |1\rangle \right] \otimes |x_2 x_3 \ldots x_n\rangle \tag{2.86}$$

We then repeat this process on all the remaining qubits which leaves us with the final state:

$$\begin{aligned} &\frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^n} x\right)} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^{n-1}} x\right)} |1\rangle \right] \otimes \ldots \\ &\otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^2} x\right)} |1\rangle \right] \otimes \frac{1}{\sqrt{2}} \left[|0\rangle + e^{\left(\frac{2\pi i}{2^1} x\right)} |1\rangle \right] \end{aligned} \tag{2.87}$$

We now have the quantum Fourier transform of the input state with the qubits in reverse order.

2.6.7 Shor's Algorithm

Shor's algorithm allows us to perform prime factorisation, a problem which is near impossible to do on a classical computer using any known method due to the large amount of time needed. No known classical algorithm has been published can perform prime factorisation in polynomial time. The problem is so time-consuming for classical computers that a large portion of public key cryptography is based on multiplying two large prime numbers together as it is unlikely that anyone would be able to determine these two numbers.

In 1995 Peter Shor developed a quantum algorithm which can factor a number N in $\mathcal{O}(logN)$ time and in 2001 IBM was able to implement this algorithm on a 7 qubit quantum computer [9,35]. The algorithm is a polynomial time algorithm as the input size is the length of the binary representation of N, and not the number N itself. While the limitations of near term quantum computers prevent us from implementing the algorithm on large numbers, the algorithm is still very interesting to study and could have major implications on the way we perform cryptography in the future.

Shor's algorithm starts by reducing the problem of finding factors of a number to an order finding problem. This part of the algorithm can be done classically and is usually done using Euclid's algorithm. Euclid's algorithm is used to compute the greatest common divisor of two integers. The quantum part of the algorithm uses the quantum Fourier transform to find a period, and from there we are able to solve for factors of the original number.

Euclid's Algorithm:
Let a and b be integers where $a > b$ and let r be the remainder when b divides a. If the remainder is not 0 then, $gcd(a, b) = gcd(b, r)$. This tells us that a number that divides a and b must also divide r. If $r = 0$ then the $gcd(a, b) = b$. Euclid's algorithm uses this property repeatedly until a remainder of 0 is found.

We start by calculating the remainder of $a \div b$, if the remainder is 0 then b is the gcd and the problem is solved. If the remainder is not 0 then we calculate $b \div r = r_n$. This process repeats n times until $r_n = 0$ telling us that we have found the gcd which is also the greatest common divisor (gcd) of a and b.

Going from Factoring to Order Finding

The aim of this algorithm is to factor a number, N. We assume that N is not even and that we can find a non-trivial solution to the equation:

$$\begin{aligned} x^2 &= 1 \bmod N \\ (x+1)(x-1) &= 0 \bmod N \end{aligned} \tag{2.88}$$

Therefore N must have a common factor with either $(x-1)$ or $(x+1)$. We now need to find a factor of N that is the greatest common divisor of either of these terms and N.

Using Euclid's algorithm we can guarantee that if we can find $(x+1)(x-1) = 0 \bmod N$ then we can factor N. If we select a random number, y, between 1 and $N-1$, then we can check whether $gcd(y, N) = 1$. If it is equal to 1 then we have found that y is coprime with N and if it is not equal to 1 then y is a factor of N. The order of y is the smallest integer, r, such that:

$$y^r = 1 \bmod N \tag{2.89}$$

If y is even then we have found a solution to equation 2.88. If the probability of a random coprime number having an even order is high, then the problem has been reduced from factoring to finding the order of a number.

Let N be a number which is a product of powers of an prime, $N = p_1^{a_1} \cdots p_s^{a_s}$. The probability that a random number, x, is coprime to N does not satisfy $x^{\frac{r}{2}} = -1 \bmod N$ is:

$$P \geq 1 - \frac{1}{2^{s-1}} \tag{2.90}$$

From this we can say that if the order finding problem can be solved then a factor can be found.

Order Finding

The implementation of Shor's algorithm on a quantum computer requires two registers. The control register, which consists of n qubits, and a work register, which contains $m = \log_2 N$ qubits. Taking measurements of the control register gives a probability distribution which peaks at $\frac{2^n s}{r}$, where r is the order. The algorithm consists of 5 key steps.

We begin by preparing the control and work registers in the ground state. We then apply Hadamard gates to the control register and encode x onto the m^{th} qubit of the work register:

$$|0\rangle^{\otimes n}|0\rangle^{\otimes m} \rightarrow \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|1\rangle \tag{2.91}$$

The next step is to implement the modular exponentiation function. To do this we continually apply a unitary operation, U, which implements $y^x \bmod n$ on the work register when the control register is in the state $|x\rangle$.

$$\begin{aligned} \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|1\rangle &\rightarrow \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle \, |y^x \bmod N\rangle \\ &= \frac{1}{\sqrt{r2^n}} \sum_{s=0}^{r-1} \sum_{x=0}^{2^n-1} e^{2\pi i s x/r} |x\rangle \, |u_s\rangle \end{aligned} \tag{2.92}$$

Here $|u_s\rangle$ is the eigenstate of the operator U, $U|u_s\rangle = e^{2\pi i s/r}|u_s\rangle$. We also have that $\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle$.

The next step is to apply the inverse quantum Fourier transform to the control register, which does the following transform.

$$\frac{1}{\sqrt{r2^n}} \sum_{s=0}^{r-1} \sum_{x=0}^{2^n-1} e^{2\pi i s x/r} |x\rangle \, |u_s\rangle \rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\varphi_s\rangle \, |u_s\rangle \tag{2.93}$$

After performing the inverse Fourier transform we take measurements on the control register. This has probability peaks for states where $\rho_s = \frac{2^n s}{r}$. The last step of the algorithm is to use continued fractions in order to extract r from ρ_s. The process of doing so is explained in the appendix of "Demonstration of Shor's factoring algorithm for $N = 21$ on IBM quantum processors" by Skosana and Tame, and laid out below [35].

Given an integer, L, a $2L+1$ bit rational number, φ, is said to have a continued fraction expansion if it can be written as:

$$\varphi \equiv [a_0, a_1, \ldots, a_n] \equiv a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_n}}}} \tag{2.94}$$

Where n is a finite integer and $a_i \in \mathbb{Z}$. If $\varphi < 1$ then $a_o = 0$. The convergents of the continued fraction expansion are the rationals:

$$a_0, a_0 + \frac{1}{a_0}, a_0 + \cfrac{1}{a_0 + \frac{1}{a_2}}, \cdots \tag{2.95}$$

If a rational number, $\frac{s}{r}$ satisfies:

$$\|\frac{s}{r} - \varphi\| \leq \frac{1}{2r^2} \tag{2.96}$$

The $\frac{s}{r}$ will appear as convergent in the continued fraction expansion of φ. If φ is an approximation of $\frac{s}{r}$ accurate to $2L+1$ bits then we have:

$$\|\frac{s}{r} - \varphi\| \leq \frac{1}{2^{2L+1}} \tag{2.97}$$

For $r \leq N \leq 2^L$, we have $\frac{1}{2^{2L+1}} \leq \frac{1}{2r^2}$. Since the inequality holds for the approximation, φ, there is a classical algorithm that can compute the convergents of φ and produce integers r', s' such that $gcd(r', s') = 1$ [8].

Running Shor's algorithm guarantees us that we can find a factor of an integer, N. It has been implemented on near term quantum computers but due to the noise nature of those devices and the probabilistic nature of the algorithm it is still hard to physically realise this algorithm on large numbers. The largest number which has been run through Shor's algorithm on a physical quantum computer is 21 [35]. The algorithm does however demonstrate that quantum computers are capable of solving certain problems that classical computers cannot and has lead many researchers to look for other quantum algorithms which can perform tasks previously thought to be unsolvable due to time constraints using classical algorithms.

2.7 Quantum Machine Learning

Quantum machine learning is a large field in scope and can be approached from many different angles. One can look for new algorithms which quantum computing has given us access to and try to use them to learn functions which are hard to learn on a classical computer [36, 37]. We can also look for ways that quantum computers can speed up the run times of classical machine learning algorithms [19, 38–43]. Lastly, we can look at the hardware implementations of quantum computing and see whether they lend themselves to a specific type of machine learning or how the hardware of the machine can affect the algorithms which run on it [44].

Quantum algorithms that implement supervised machine learning are often referred to as quantum models within literature [25, 45]. Quantum models are made up of a model class that needs a quantum computer to be evaluated, and an optimization method. On near term quantum computers quantum models usually consist of three parts, data encoding, a parameterised circuit block, and a measurement. The data encoding maps classical input data to a quantum state and is usually represented as follows, $x \rightarrow |\phi(x)\rangle$. The methods performing this mapping and the choices of mappings will be explored in more detail in section2.11. The parameterised circuit block contains variables which are controlled by an optimization algorithm. The measurement of one or many qubits after the circuit which implements the quantum model has been run is the output of the model. Due to the noisy nature of most quantum model implementations they are often performed multiple times and a statistical analysis of the results is used as the output of the model [25, 46].

Most current methods of quantum machine learning fall under the CQ segment of figure 1.1. The data fed into the learning algorithms is classical and the computation is performed using a mix of different quantum and classical algorithms. Quantum algorithms often perform linear algebra operations and are used to solve linear systems of equations used in least-squares linear regression and the least-squares version of the support vector machines [38]. There are also algorithms which use Grover's search algorithm to perform machine learning tasks which translate into unstructured search tasks [47]. Quantum annealing is used to find the local minima and maxima of functions, something which is used for optimisation in many parts of machine learning [48–50].

Most quantum machine learning algorithms are currently implemented in python using libraries such as PennyLane and Qiskit [34, 51, 52]. These libraries make it easy for people with coding backgrounds to implement code on quantum computers. They also provide tool which allow one to simulate a quantum computer on a classical computer which allows for certain quantum algorithms to be tested without using a physical quantum computer. This is very useful as quantum computers are expensive to build, noisy and most people don't have easy access to a quantum computer. The calculations performed in this thesis were run on a simulated quantum computer using the PennyLane library.

It is important to ask what improvements quantum computing can offer to classical machine learning and it is a question which garners a lot of attention due to the implications of the answer. However, one could also argue that at this point in time it is more important to figure out how to implement the methods of classical computing on

NISQ computers. Implementing algorithms which are already well understood on classical computers can allow us to better understand the nature of quantum computers and could help us to find quantum advantages further down the road.

2.8 Parameterised Quantum Circuits

Parameterised circuits, also known as adaptable quantum circuits, variational circuits, or quantum neural networks have become a go to for testing near term quantum computers. Due to their structure, we can test a quantum model, such as a quantum support vector machine, while still using a classical computer to perform the optimization. This is beneficial as it allows us to test what current quantum computers are capable of doing while also probing for new things that they can allow us to do and exploring their properties. Parameterised circuits also allow us to test multiple forms of machine learning on near term quantum computers and can allow us to find which methods are more suited to quantum machine learning [53–62].

A parameterised circuit consists of 3 parts, the preparation of an initial state, a quantum circuit parametrised by free variables, also called free parameters, and the measurement. Data and the free parameters are encoded into the qubit from an initial state. The initial state of a qubit is usually one which is simple to prepare and in most practical implementations it is set to $|0\rangle$. The quantum circuit then runs and a measurement is taken. In this thesis we focus on supervised machine learning which uses labelled data points. The data points take the form, $(\boldsymbol{x}, y)$, where $\boldsymbol{x}$ is a vector and y is the label. When training a circuit we use a set of training data points which are used to build the model. One or more test data sets are also used in order to evaluate the model.

The results of the measurement are fed into a classical optimisation algorithm which adjusts the free parameters. The qubits are then set back to the initial state before the data is then encoded again and the free parameters are updated. The process repeats until circuit meets its designated number of runs or until some other condition is met, such as the threshold of a loss function being reached. The general structure of a parameterised quantum circuit can be seen in the diagram below.

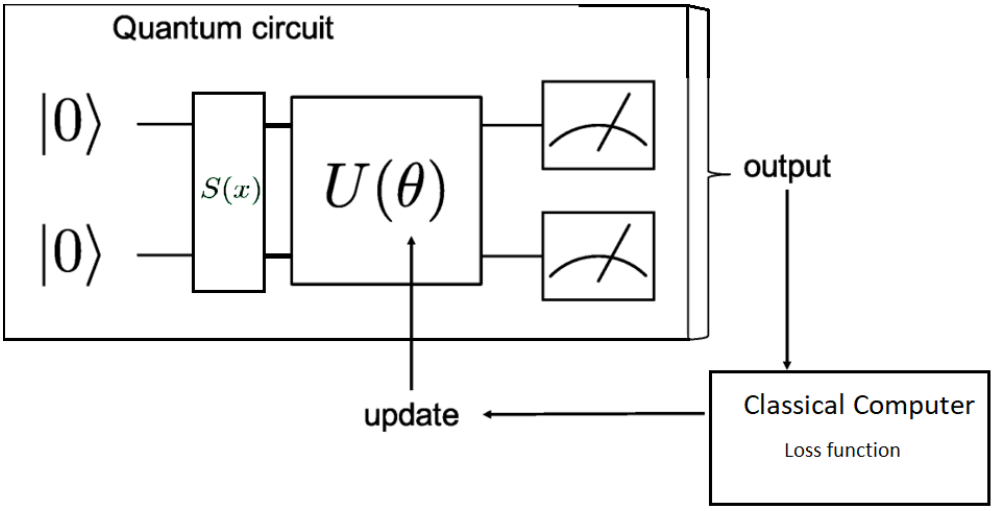


Figure 2.19: A diagram showing the layout of a parameterised quantum circuit.

The circuit begins by setting all qubits to the ground state. Classical data is then mapped into aa quantum state using the desired map. This process is known as quantum encoding and is something which will be focused on in this thesis. The qubits are then acted upon by a unitary operator which has free parameters that have been set by the classical optimisation algorithm. It is also common for the initial value of the free variables to be randomised before the first pass to the classical optimisation algorithm. This unitary operation transforms the initial state and leaves us with an output state which can be measured. These parameters are altered with each run of the circuit in order to change the output state of the circuit. The tuning of the free parameters is done by a classical computer which updates the parameters of the quantum circuit. A chosen classical optimisation is used in order to see how close the guess of the quantum part of the circuit is to the test data.

Once the classical optimisation algorithm has reached its endpoint we are left with a quantum circuit that models a function based on the data set it was trained on. We are then able to feed a data point, $\boldsymbol{x}$ into the circuit and run it. The measurement of the circuit will give us an output.

Although the general structure of a parameterised circuit remains the same the specific structure of the circuit can vary and can have a large effect on the learning process. There are many proposed ansatz for the structure of a circuit all with their own pros and cons. There are three main architectures which are used in practice, the layered gate architecture ansatz, the alternating operator ansatz and the tensor network ansatz [63–67].

In the layered gate architecture ansatz a sequence of gates forms a layer and these layers can be repeated if needed in order to form a deeper circuit. These layers are usually broken up into the encoding and unitary blocks as described above. The alternating operator

ansatz works in a similar fashion to the layered gate ansatz but the blocks which make up the circuit are defined by a time evolving Hamiltonian. Lastly the tensor network ansatz is defined by a tensor network which is then compiled into a quantum circuit.

The structure of the parameterised circuit allows us to implement many types of machine learning algorithms, but it also some drawbacks. The first is the use of the circuit ansatz because we cannot guarantee that the circuit ansatz used will provide the optimal results for the given data set. Due to the nature of machine learning we cannot guarantee that we know anything about the structure of the data set before any learning is performed on it and we therefore need to pick a circuit ansatz which is optimal for the data set but also general enough to apply the results to other unseen data sets.

Another problem that parameterised circuits face is that of barren plateaus during training [68–70]. This refers to a problem in the optimisation of variational circuits. For a wide class of variational circuits, the probability that the gradient of the optimisation is large enough to move the training algorithm forward is exponentially small as a function of the number of qubits. This can cause the optimisation algorithm to not know which direction to travel in in order to optimise the circuit, and causes the optimisation to get stuck. This can lead to large wastes in computational power and the circuit learning incorrect or not meaningful functions.

The last big problem with parameterised circuits deals with the number of parameters needed to learn from data. To train a circuit that can express any quantum model would require parameters for all $\mathcal{O}(2^{2n})$ degrees of freedom, where n is the number of qubits in the quantum model we wish to learn [25]. For any practical uses this becomes intractable. There are still however models which can be learned by variational circuits, we just cannot guarantee that they are the optimal solution.

We will now go through two examples of parameterised circuits to see how the work in practice. The first example shows us how a quantum model can be created and implemented in order to learn functions in a variational circuit. The second example shows us how a support vector machine can be implemented on a quantum computer. Both of these examples will give us insights into how the embedding of classical data can have a large effect on the learning process. It has also been shown that the training of variational quantum eigensolvers is NP-hard [71].

2.9 Practical Examples of PQCs

2.9.1 Quantum Models as a Partial Fourier Series

The first example comes from a paper written by Schuld, Sweke and Meyer [57]. In this paper quantum models are expressed in terms of a Fourier series and a parameterised quantum circuit is used in order to learn a function.

A univariate quantum model can be defined as the expectation value of an arbitrary observable with respect to a state prepared by a parameterised quantum circuit.

$$f_\theta(\boldsymbol{x}) = \langle 0| U^\dagger(\boldsymbol{x}, \boldsymbol{\theta}) M U(\boldsymbol{x}, \boldsymbol{\theta}) |0\rangle \tag{2.98}$$

$|0\rangle$ represents the initial state that we begin in and $U(\boldsymbol{x}, \boldsymbol{\theta})$ is a quantum circuit which takes in data and has tunable parameters, $\boldsymbol{\theta}$. The univariate data, $\boldsymbol{x}$, is data that consists of observations on only a single characteristic or attribute, $x \in \mathbb{R}$. A univariate quantum model is a quantum model which is built using only univariate data. M is an arbitrary measurement operator.

The circuit in this example is comprised of 2 blocks, a data encoding block and a trainable circuit block. The circuit can also have multiple layers each containing a data encoding block, $S(x)$, and trainable circuit block, $W(\theta)$, seen in figure 2.20 below. The circuit was able to run in series or parallel, if there are multiple qubits used the circuit was run in parallel. If the number of layers, L, is greater than 1 then the circuit is in series. When the circuit is in series only one qubit is used. The layers or number of qubits used in parallel is set before running the circuit, a maximum of 5 qubits or 5 layers was used in the paper.

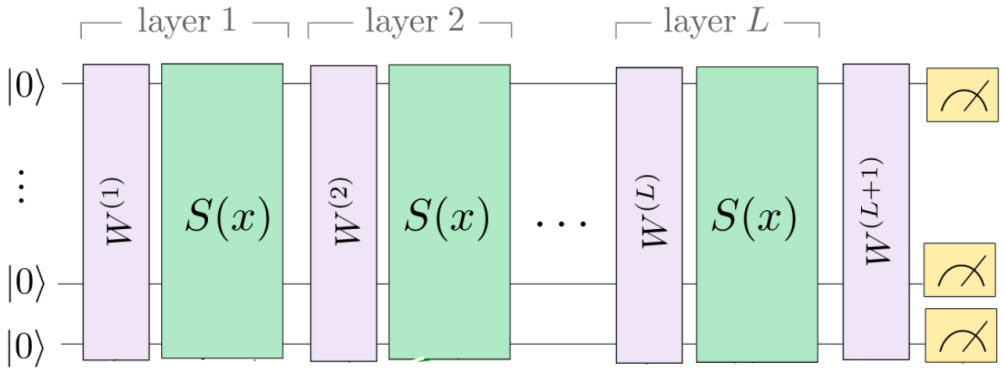


Figure 2.20: A diagram showing the general structure of the circuits used in this paper.

The encoding block is always the same an composed of gates of the form e^{-ixH}. The H is a Hamiltonian which generates the time evolution which encodes the data. In this paper a Pauli rotation was used as the Hamiltonian as researchers wanted to analyse the effects that Pauli encoding could have on learned models. The trainable circuit blocks are viewed as arbitrary unitary operations so that no extra assumptions had to be made about them so that the researchers could focus on analysing the effects of the chosen encoding method.

We can now examine the path of a qubit through this circuit. The qubit first starts in a $|0\rangle$ state and first goes into the parameterised block where it is acted upon by a unitary function. The qubit then enters the encoding block and is acted upon by the Hamiltonian and finally it is measured. The data is then fed into a classical computer which makes adjustments to the free parameters within the parameterised blocks, W. These changes are dictated by a classical optimisation algorithm. A new qubit is then prepared into the 0 state and goes through the updated parameterised block and the process repeated until the classical loss function has reached its tolerance or the entire process repeated the maximum number of times. In this paper a mean squared error loss function was used. The quantum computer is noisy so measurements are taken multiple times and

a statistical analysis performed before passing the data to the classical loss function in order to minimise the effect of noise.

Now that we have laid out the general structure of the circuit it is useful to examine the circuit in more detail and analyse what is happening. The goal of this circuit is to be able to express a function learned from this circuit as a truncated Fourier series. We will use this to demonstrate how the encoding method affects the frequencies which make up the Fourier series.

$$f(x) = \sum_{n \in \Omega} c_n e^{inx} \tag{2.99}$$

Here, Ω are the integer valued frequencies which we can use to build our Fourier series. The eigenvalue decomposition of the generator Hamiltonian can always be found. We can express the Hamiltonian, H, in terms of a diagonal operator, Σ, containing the eigenvalues of H on its diagonal.

$$H = V^{\dagger} \Sigma V \tag{2.100}$$

It was stated that the parameterised blocks which are on either side of the encoding block are arbitrary. This means that we can absorb the operators, V and $V^{\dagger}$, into W without loss of generality. The encoding block can now always be expressed in terms of the diagonal matrix of eigenvalues.

$$S(x) = e^{-i\Sigma x} \tag{2.101}$$

Using this the researchers were able to determine that the frequency spectrum of the Fourier series is directly determined by the eigenvalues of the Hamiltonian. The largest frequency which is accessible is termed the degree of the spectrum. The coefficient were determined by the arbitrary gates in the parameterised blocks.

When looking at circuits with one layer it was found that the circuit could almost perfectly, but only, learn sine functions. An image of the results is presented in figure 2.21. It was also found that when more circuit layers were added the circuit was able to learn more complicated functions. These multiple layers could be added in series or parallel but the results remained similar. The number of layers seems to correspond directly to the the number of terms in the Fourier series of the learned function. A proof of why a single layer circuit can only learn a sine function and in depth discussion of these results will be discussed further on.

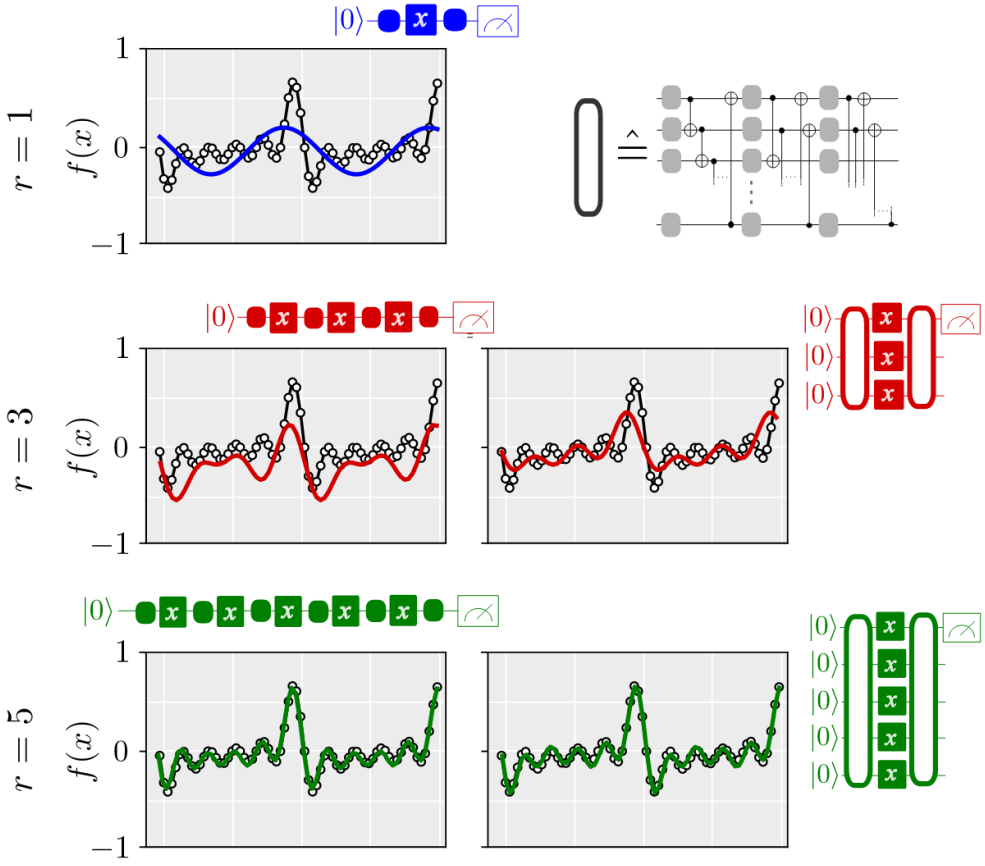


Figure 2.21: The main results from this paper. We can see that as the number of layers, r, is increased the circuit is able to more accurately approximate the function. This trend occurred in both the sequential and parallel versions of the circuit with the sequential circuits appearing on the left hand side of the image and the parallel circuits appearing on the right.

2.9.2 Supervised Learning with Quantum Enhanced Feature Spaces

This section will go through the methods and results of Supervised Learning with Enhanced Feature Spaces by Havlicek et al [56]. The main goals of this paper were to implement a quantum analogy to a support vector machine and the other was to create a quantum circuit which could be used as a kernel estimator. These experiments were performed on a superconducting processor and therefore are representative of what we can do with the quantum computers we currently have.

In this work the support vector machine was used as a classifier. The data given to the support vector machine was labelled and a test set was used to asses the model generated. The job of the model was to learn from the data in order to be able to sort new data points into one of these two groups. The first experiment used a variational circuit in order to construct the hyperplane of the SVM and used that hyperplane to classify the data. The second experiment used a quantum computer in order to estimate the kernel matrix of all the data entries.

The authors of this paper also noted that the only time this method can yield a quantum advantage is when the kernels used cannot be calculated classically. The kernels in this paper could be calculate classically but the purpose of the paper was not to search for a robust quantum advantage, but rather to show that these methods could work on a real quantum computer.

The circuit used for the quantum variable classifier in this paper closely resembles the ciruit used in the previous example. The qubits start in a zero state $|0\rangle^n$ and the data is encoded using a unitary gate. The circuit then goes through a variable unitary gate and finally a measurement is taken in the Z-basis. The results of the measurements is turned into a bit string and then mapped to a label. The circuit is also run multiple times for each data point.

The algorithms for the training phase and classification phase are given in pseudo code in the paper and shown below.

Algorithm 1 Quantum variational classification: the training phase

Input Labeled training samples $T = \{\vec{x} \in \Omega \subset \mathbb{R}^n\} \times \{y \in C\}$, Optimization routine,
Parameters Number of measurement shots R, and initial parameter $\vec{\theta_0}$.
Calibrate the quantum Hardware to generate short depth trial circuits.
Set initial values of the variational parameters $\vec{\theta} = \vec{\theta_0}$ for the short-depth circuit $W(\vec{\theta})$
while Optimization (e.g. SPSA) of $R_{\text{emp}}(\vec{\theta})$ has not converged **do**
    for $i = 1$ `to` $|T|$ **do**
        Set the counter $\mathrm{r}_y = 0$ for every $y \in C$.
        for *shot* $= 1$ `to` R **do**
            Use $\mathcal{U}_{\Phi(\vec{x}_i)}$ to prepare initial feature-map state $|\Phi(\vec{x}_i)\rangle\langle\Phi(\vec{x}_i)|$
            Apply discriminator circuit $W(\vec{\theta})$ to the initial feature-map state .
            Apply $|C|$ - outcome measurement $\{M_y\}_{y \in C}$
            Record measurement outcome label y by setting $\mathrm{r}_y \to \mathrm{r}_y + 1$
        end for
        Construct empirical distribution $\hat{p}_y(\vec{x}_i) = \mathrm{r}_y R^{-1}$.
        Evaluate $\Pr(\tilde{m}(\vec{x}_i) \neq y_i | m(\vec{x}) = y_i)$ with $\hat{p}_y(\vec{x}_i)$ and y_i
        Add contribution $\Pr(\tilde{m}(\vec{x}_i) \neq y_i | m(\vec{x}) = y_i)$ to cost function $R_{\text{emp}}(\vec{\theta})$.
    end for
    Use optimization routine to propose new $\vec{\theta}$ with information from $R_{\text{emp}}(\vec{\theta})$
end while
return the final parameter $\vec{\theta}^*$ and value of the cost function $R_{\text{emp}}(\theta^*)$

Figure 2.22: The algorithm used for the training phase of the quantum variational classifier as laid out by the researchers [56]

In Algorithm 1 laid out in figure 2.22, T represents the training data, with $\vec{x}$ being the data vector and y being the label. Initial parameter $\vec{\theta_0}$ refers to the variables in the parameterised circuit block of the circuit, $W(\vec{\theta_0})$. $R_{emp}(\vec{\theta_0})$ refers to the empirical risk which is given by the equation:

$$R_{\text{emp}}(\vec{\theta}) = \frac{1}{|T|} \sum_{\vec{x} \in T} \Pr(\tilde{m}(\vec{x}) \neq m(\vec{x})) \tag{2.102}$$

Where $m(\vec{x})$ is the output of the model and $\tilde{m}(\vec{x})$ is the label of the test data. $U_\phi(\vec{x_i})$ is the unitary operation which is used to map the data into the desired feature space.

Algorithm 2 Quantum variational classification: the classification phase

Input An unlabeled sample from the test set $\vec{s} \in S$, optimal parameters $\vec{\theta}^*$ for the discriminator circuit.
Parameters Number of measurement shots R
Calibrate the quantum Hardware to generate short depth trial circuits.
Set the counter $\mathrm{r}_y = 0$ for every $y \in C$.
for $shot = 1$ `to` R **do**
    Use $\mathcal{U}_{\Phi(\vec{s})}$ to prepare initial feature-map state $|\,\Phi(\vec{s})\rangle\langle\Phi(\vec{s})\,|$
    Apply optimal discriminator circuit $W(\vec{\theta^*})$ to the initial feature-map state .
    Apply $|C|$ - outcome measurement $\{M_y\}_{y \in C}$
    Record measurement outcome label y by setting $\mathrm{r}_y \to \mathrm{r}_y + 1$
end for
Construct empirical distribution $\hat{p}_y(\vec{s}) = \mathrm{r}_y R^{-1}$.
Set label = $\mathrm{argmax}_y\{\hat{p}_y(\vec{s})\}$
return label

Figure 2.23: The algorithm used for the classification phase of the quantum variational classifier as laid out by the researchers [56].

For this paper artificial data was generated which could be fully separated by the chosen feature map. When using a single layer circuit the cost function converges quickly, but not to a low value. When the number of layers was increased to 4 however the cost function converged at a slower rate but to a better value, indicating a better fit.

A quantum computer is also used to calculate a kernel matrix of all the data points. The resultant matrix is then fed to a classical computer which performs the classification. The idea is to use Hilbert space as a feature space, which is only possible because one can evaluate inner products in Hilbert space on a quantum computer. We hope that the high dimensionality of the feature space results in the creation of a hyperplane which might not exist in a lower dimensional feature space. Once this matrix is obtained it is straightforward to use an SVM algorithm to find the hyperplane which separates the data points optimally. Calculating the kernel matrix can be very time consuming and resource intensive, especially with large data sets. The entries of the kernel matrix computed by the quantum computer were the transition amplitudes of the input vectors, $|\langle\phi(\vec{x})|\phi(\vec{z})\rangle|$. In the training of the circuit $\mathcal{O}(T^2)$ amplitudes have to be calculated.

There can be no quantum advantage for this method unless the kernel is difficult to calculate classically, but this paper proves that the calculation of a kernel matrix can be performed on a current day quantum computer and allows us to search for new kernels which might provide a quantum advantage.

Significance of Repeat Encodings

Both of these examples take a slightly different approach to repeat encodings but both find experimentally that they increase the accuracy of the function learned by a variational circuit but the reason as to why this happens is not clear. These papers do however tell us that the process of encoding plays a more crucial role in quantum machine learning than we initially thought. In section 2.12 we will look at explanations of why the repetition of encoding in these two examples resulted in the degree of the Fourier series increasing.

2.10 Viewing Supervised Quantum Machine Learning Models as Kernel Methods

Now that we have a practical understanding of how quantum machine learning working in practice we can look at it through a different lens in order to further our theoretical understanding. In this section we will go through the argument presented by Schuld that supervised quantum machine learning models are kernel methods [25]. We will start by defining some terms before getting into the argument.

If we represent quantum states as vectors in $\mathcal{C}^{2n}$, then the corresponding density matrix is given by the outer product of a vector with itself. The density matrix corresponding to a state vector, $|\psi\rangle$, is:

$$\rho = |\psi\rangle \langle\psi| \tag{2.103}$$

The density matrix contains all the observable information of $|\psi\rangle$. It is useful to make probability distributions, $\{p_k\}$ over multiple quantum states as mixed states:

$$\rho = \sum_k p_k |\psi_k\rangle \langle\psi_k| \tag{2.104}$$

Definition 2.10.1 (Data encoding feature map)**.** Given an n qubit quantum system with states $|\psi\rangle$. Let $\mathcal{F}$ be the space of complex valued $2^n \times 2^n$ dimension matrices equipped with the Hilbert-Schmidt inner product $\langle\rho, \sigma\rangle_{\mathcal{F}} = tr\{\rho^\dagger \sigma\}$ for $\rho, \sigma \in \mathcal{F}$. The data encoding feature map is the transformation:

$$\begin{aligned} &\phi : \chi \to \mathcal{F} \\ &\phi(x) = |\phi(x)\rangle \langle\phi(x)| = \rho(x) \end{aligned} \tag{2.105}$$

and can be implemented by a data encoding quantum circuit U(x).

Definition 2.10.2 (Quantum kernel)**.** Let ϕ be a data encoding feature map over a domain χ. A quantum kernel is the inner product between two data encoding feature vectors $\rho(x)$ and $\rho(x')$ with $x, x' \in \chi$

$$k(x, x') = tr[\rho(x'), \rho(x)] = |\langle\phi(x')|\phi(x)\rangle|^2 \tag{2.106}$$

Quantum kernels are positive semi-definite.

Quantum kernels can also be visualised and implemented with an alternate formulation which can be helpful when doing practical examples.

$$\begin{aligned} \phi_\upsilon : &\to \mathcal{F}_\upsilon \subset \mathcal{H} \otimes \mathcal{H}^* \\ \phi_\upsilon &= |\phi(x)\rangle \otimes |\phi^*(x)\rangle \end{aligned} \tag{2.107}$$

Where $|\phi^*(x)\rangle$ is the quantum state created by applying the complex conjugate of the encoding unitary matrix and $\mathcal{F}_\upsilon$ is the space of tensor products of a data encoding Dirac vector with its complex conjugate. The inner product in this feature space is the absolute square of the inner product in the Hilbert space of the quantum states.

$$\langle\psi|\varphi\rangle_{\mathcal{F}_\upsilon} = |\langle\psi|\varphi\rangle_{\mathcal{H}}|^2 \tag{2.108}$$

What all of this tells us is that the encoding block of a quantum circuit can simply be viewed as a feature map which maps classical data into a Hilbert space, which can be viewed as a feature space. The rest of the circuit then performs linear algebra operations on the data in this new space and we are then able to learn from the data. This is exactly the same logic which is employed by classical kernel methods.

From all this we can also say that each encoding method of quantum computing corresponds to a specific kernel and we can use this knowledge in order to say how different methods of encoding affect the circuit and the functions which can be learnt from the circuit. We are also able to use our knowledge of kernel methods to see how we can use this way of looking at supervised learning to search for quantum speed up.

Non-linearity in quantum kernels is thought to be a factor in what leads to speed-up over classical algorithms, as if the mapping is simple enough a classical computer also can perform it and give us access to the same space. If we are able to find spaces which can not be reached by classical algorithms, as was done by Liu et al, there is a chance that that space can yield a quantum advantage [36]. The data set used in this paper was artificially constructed to demonstrate ideal conditions. When using real world data sets we cannot guarantee that any patterns which give a quantum advantage are present in the data set before we process it and therefore the problem of finding practical examples of quantum speed-up is very hard.

Using the knowledge that each method of quantum encoding directly corresponds to a kernel we can now look at the kernels linked to the most popular encoding methods. We can use this information to see how the method of encoding can bias the results of a quantum circuit and use it to help explain the experimental results obtained in the practical examples.

2.11 Quantum Encoding

Quantum computers need qubits in order to run, however most data we have is classical. This means that we need ways to convert classical information into qubits, this process is known as quantum encoding or quantum embedding. There are several ways to perform quantum encoding and each has its own pros and cons. Usually the methods of quantum encoding used is dependant on the problem being solved and the architecture of the quantum computer. The conversion of classical bits to qubits is done through mapping the classical data into quantum states which are then used as qubits.

2.11.1 Basis encoding

Basis encoding is in some sense the most straightforward way of mapping classical bits to qubits and because of this it is also one of the most popular methods of encoding. In basis encoding each bit of classical information is represented by a quantum subsystem. Given classical data in the form of a binary string, the quantum state will be a bitwise translation of the string. For example the 3 bit string 101 would be represented by the 3 qubit state $|101\rangle$.

Given a classical data set, χ, with M entries composed of N-bit binary strings, x^m. The data set can be represented in superpositions of the computational basis state as:

$$|M\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle \tag{2.109}$$

Where x^m is an N-bit binary string. Using basis encoding for N bits there are 2^N possible basis states. The data encoding feature map of basis encoding is:

$$\phi : x \to |i_x\rangle \langle i_x| \tag{2.110}$$

And the quantum kernel it gives is:

$$k(x, x') = |\langle i_x | i'_x \rangle|^2 = \delta_{x,x'} \tag{2.111}$$

2.11.2 Amplitude Encoding

Amplitude encoding requires that the data inputs are normalised. It associates each input with a quantum state whose amplitudes in the computational basis are the elements of the input vector. Given an N dimensional data point, x, it can be represented by the amplitudes of an n qubit quantum state as:

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle \tag{2.112}$$

With $N = 2^n$. The normalisation condition that must be followed is, $\sum_i |x_i|^2 = 1$. The feature map for amplitude encoding is given by:

$$\phi : \mathbf{x} \to |\mathbf{x}\rangle \langle \mathbf{x}| = \sum_{i,j=1}^{N} x_i x_j^* |i\rangle \langle j| \tag{2.113}$$

And the quantum kernel is given by:

$$k(\mathbf{x}, \mathbf{x}') = |\langle \mathbf{x}' | \mathbf{x} \rangle|^2 = |\mathbf{x}^\dagger \mathbf{x}'|^2 \tag{2.114}$$

The main advantage of amplitude encoding is that it requires $log(NM)$ qubits to encode a data set of M inputs with N features and is often associated with Grover's quantum search algorithm. We can also perform repeated amplitude encoding which has a map and kernel which look quite similar to that of amplitude encoding.

2.11.3 Hamiltonian Encoding

Instead of explicitly encoding data into quantum states, Hamiltonian encoding tries to do so implicitly. Instead of directly preparing a quantum state we instead prepare an evolution applied to a state. To do this we create a Hamiltonian which evolves a given state. The Hamiltonian encodes a matrix that contains the data we wish to use. This can be done by creating a matrix which has the feature vectors as rows. We start with an initial quantum state $|\psi\rangle$ and we use a Hamiltonian to evolve to a final state $|\psi'\rangle$ which now contains all the relevant information.

$$|\psi'\rangle = e^{-iHt} |\psi\rangle \tag{2.115}$$

This process can be done in polynomial time [20].

We start by considering a Hamiltonian, H, that can be decomposed into the sum of elementary Hamiltonians, H_k which are easy to simulate. We cannot use the factorisation rule for scalar exponentials unless the Hamiltonians commute, so instead we use the first order Suzuki-Trotter formula.

$$e^{-i\sum_k H_k t} = \Pi_k e^{-iH_k t} + \mathcal{O}\left(t^2\right) \tag{2.116}$$

This states that for small time steps, δt the factorisation rule is approximately valid and we can write,

$$\mathrm{e}^{-iHt} = \left(\mathrm{e}^{-iH\Delta t}\right)^{\frac{t}{\Delta t}} \tag{2.117}$$

This means that with small enough time steps the error becomes negligible. We also have to repeat the process many times the smaller the time step.

Rotational encoding is a type of Hamiltonian encoding. In rotational encoding a fixed Hamiltonian is evolved and the data is used to parameterise the time via which the Hamiltonian is evolved. Using what we have defined we are now able to examine how rotational embedding affects the RKHS of a quantum circuit and therefore what functions can be learnt from it. Every kernel has an associated RKHS and circuits can be used to define kernels. We can therefore associate a quantum circuit with a RKHS and use that to try and understand the functions which a given circuit can learn.

2.12 The Fourier Spectrum of PQCs That Use Rotational Embedding

The papers explored in section 2.9 both found that repeating a rotation encoding gate in a parameterised quantum circuit had an effect of the functions which could be learned by the circuit. In this section we will begin by looking at how using a rotational encoding gate affects the RKHS. Following that, we will show that using a single rotational encoding gate allows a circuit to only learn a sine function. Finally, we will examine ways in which family of functions learned by a PQC using rotational encoding can be extended.

2.12.1 How Rotational Embedding Affects the RKHS

Start by considering a quantum embedding implemented by the Pauli-X rotation:

$$R_x(x) = e^{-i\frac{x}{2}\sigma_x} = \begin{pmatrix} \cos\left(\frac{x}{2}\right) & -i\sin\left(\frac{x}{2}\right) \\ -i\sin\left(\frac{x}{2}\right) & \cos\left(\frac{x}{2}\right) \end{pmatrix} \tag{2.118}$$

The data encoding feature map is then given by:

$$\phi : x \to \rho(x) \tag{2.119}$$

where

$$\rho(x) = \begin{pmatrix} \cos^2\left(\frac{x}{2}\right) & -i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \\ i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) & \cos^2\left(\frac{x}{2}\right) \end{pmatrix} \tag{2.120}$$

The quantum kernel is then given by:

$$\begin{aligned} k(x, x') &= \left|\cos\left(\frac{x}{2}\right)\cos\left(\frac{x'}{2}\right) + \sin\left(\frac{x}{2}\right)\sin\left(\frac{x'}{2}\right)\right|^2 \\ &= \cos\left(\frac{x - x'}{2}\right)^2 \end{aligned} \tag{2.121}$$

This feature map can also be represented using the form in equation 2.107 as:

$$\begin{aligned}\phi_v(x) &= \left(\cos\left(\frac{x}{2}\right)|0\rangle - i\sin\left(\frac{x}{2}\right)|1\rangle\right) \otimes \left(\cos\left(\frac{x}{2}\right)|0\rangle + i\sin\left(\frac{x}{2}\right)|1\rangle\right) \\ &= \begin{pmatrix} \cos^2\left(\frac{x}{2}\right) \\ i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \\ -i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \\ \sin^2\left(\frac{x}{2}\right) \end{pmatrix}\end{aligned} \tag{2.122}$$

We wish to extract the eigenvalues and eigenfunctions of this matrix in order to see how the functions which make up the RKHS are affected. In turn, this will allow us to understand how learning process can be biased by this encoding method. To do so we need to introduce the integral operator defined by:

$$(Kf)(x) = \int k(x, x')f(x')\mu(dx') \tag{2.123}$$

Where μ is a marginal distribution. A marginal distribution is used here because often when training circuits the full data set is divided into a training set and one or multiple test sets. By using a marginal distribution we can use the integral operator on any subset created from the full data set. Mercer's Theorem ensures that there exists a spectral decomposition of K with eigenvalues, γ, and eigenfunctions, ϕ_i. Models will be biased towards learning functions that correspond to the eigenfunctions. Functions which have features which are not able to be recreated using the eigenfunctions will lose these features in the model that is generated. However, if a function, or a close enough approximation of the function, can be reconstructed by linear combinations of eigenfunctions then the model can learn the function more accurately.

If M is a Hermitian operator, then the RKHS consists of all functions $f \in \mathcal{F}$ which can be written as:

$$f(x) = Tr[\varphi(x)M] \tag{2.124}$$

The action of the integral operator on a function f is:

$$\begin{aligned}(Kf)(x) &= \int k(x, y)f(y)\mu(dy) \\ &= \int Tr(\varphi(y)M)Tr(\varphi(y)\varphi(x))\mu(dy) \\ &= \int Tr\left[(M \otimes \varphi(x))(\varphi(y) \otimes \varphi(x))\right]\mu(dy) \\ &= Tr\left[(M \otimes \varphi(x))\int(\varphi(y) \otimes \varphi(x))\mu(dy)\right]\end{aligned} \tag{2.125}$$

To analyse the integral operator we need the matrix elements of a linear map, T.

$$\begin{aligned}T(M)_{ij} &= \int \varphi(x)_{ij}\varphi(x)_{kl}M_{lk}\mu(dx) \\ &= \int \varphi(x)_{ij}\varphi^*(x)_{lk}M_{lk}\mu(dx)\end{aligned} \tag{2.126}$$

We now apply a vectorisation which converts the matrix into a column vector:

$$\begin{aligned}\text{Vec}(T(M)) &= \int \text{Vec}(\varphi(x))\text{Vec}(\varphi(x))^T \mu(dx)\text{Vec}(M) \\ &= A_\mu \text{Vec}(M)\end{aligned} \tag{2.127}$$

What we are left with after the vectorisation is a column vector, Vec(M), and a vector which correspond to the linear map used to perform the encoding as well as the transpose of this matrix. We then label the matrix that would result from perform the multiplication of the two encoding vectors as A_μ. We calculate A_μ by using the quantum kernel given by equation 2.121 which corresponds to rotational encoding with a Pauli-X gate.

$$\begin{aligned}
A_\mu &= \frac{1}{\pi}\int_0^\pi \begin{pmatrix} \cos^2\left(\frac{x}{2}\right) \\ -i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \\ i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \\ \sin^2\left(\frac{x}{2}\right) \end{pmatrix} \times \\
&\left(\cos^2\left(\frac{x}{2}\right) \quad i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \quad -i\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) \quad \sin^2\left(\frac{x}{2}\right)\right) dx \\
&= \frac{1}{\pi}\int_0^\pi \begin{pmatrix} \cos^4\left(\frac{x}{2}\right) & i\cos^3\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) & -i\cos^3\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) & \cos^2\left(\frac{y}{2}\right)\sin^2\left(\frac{x}{2}\right) \\ -i\cos^3\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) & \cos^2\left(\frac{x}{2}\right)\sin^2\left(\frac{x}{2}\right) & -\cos^2\left(\frac{x}{2}\right)\sin^2\left(\frac{x}{2}\right) & -i\cos\left(\frac{x}{2}\right)\sin^3\left(\frac{x}{2}\right) \\ i\cos^3\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) & -\cos^2\left(\frac{x}{2}\right)\sin^2\left(\frac{x}{2}\right) & \cos^2\left(\frac{x}{2}\right)\sin^2\left(\frac{x}{2}\right) & i\cos\left(\frac{x}{2}\right)\sin^3\left(\frac{x}{2}\right) \\ \cos^2\left(\frac{x}{2}\right)\sin^2\left(\frac{x}{2}\right) & i\cos\left(\frac{x}{2}\right)\sin^3\left(\frac{x}{2}\right) & -i\cos\left(\frac{x}{2}\right)\sin^3\left(\frac{x}{2}\right) & \sin^4\left(\frac{x}{2}\right) \end{pmatrix} dx \\
&= \frac{1}{8}\begin{pmatrix} 3 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 3 \end{pmatrix}
\end{aligned}$$

This produces the eigenvalues $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$ and 0. and produces the matrices:

$$\begin{aligned}
H_1 &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad H_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \\
H_3 &= \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix}, \quad H_4 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}
\end{aligned} \tag{2.128}$$

The eigenvectors can be used to obtain the eigenfunctions via equation 2.124.

$$\begin{aligned}
&f_1(x) = 1, && f_2(x) = \cos^2\left(\frac{x}{2}\right) - \sin^2\left(\frac{x}{2}\right) = \cos(x) \\
&f_3(x) = 2\cos\left(\frac{x}{2}\right)\sin\left(\frac{x}{2}\right) = \sin(x), && f_4(x) = 0
\end{aligned} \tag{2.129}$$

The eigenfucntions of the RKHS are sine and cosine functions. This means that all the functions which are contained in this space can be made by shifting and scaling a sine or cosine function. Cosine functions can be expressed as shifted sine functions therefore we can say that functions in the RKHS are parametrised by:

$$x \to a\cos(x+b) + c \tag{2.130}$$

This can be more clearly demonstrated and visualised by looking at how this concept works in an actual circuit. By going back to the previous examples of parametrised circuits we can show that single layer circuits are only able to learn sine functions.

2.12.2 Single Layer Circuits on One Qubit

Going back to the example of quantum models as a partial Fourier series we can now show that single layer qubit circuits can only have expectation values with a sinusoidal form.

Consider circuits $U = U_D...U_1$ where each gate is either fixed or parameterised. Parameterised gates have the form:

$$U_d = e^{(-i\frac{\theta_d}{2}H_d)} \tag{2.131}$$

Where H_d is the generator and a Hermitian, unitary matrix with the property $H_d^2 = I$ and $\theta \in (-\pi, \pi]$

We now take the exponential definition of the gate:

$$\begin{aligned} U_d &= \sum_{k=0}^{\infty} \frac{(-i)^k(\frac{\theta_d}{2})^k(H_d)^k}{k!} \\ &= \sum_{k=0}^{\infty} \frac{(-i)^{2k}(\frac{\theta_d}{2})^{2k}(H_d)^{2k}}{2k!} + \sum_{k=0}^{\infty} \frac{(-i)^{2k+1}(\frac{\theta_d}{2})^{2k+1}(H_d)^{2k+1}}{(2k+1)!} \\ &= \sum_{k=0}^{\infty} \frac{(-i)^{2k}(\frac{\theta_d}{2})^{2k}I}{2k!} + \sum_{k=0}^{\infty} \frac{(-i)^{k}(\frac{\theta_d}{2})^{(2k+1)}H_d}{(2k+1)!} \\ &= \cos\left(\frac{\theta_d}{2}\right)I - i\sin\left(\frac{\theta_d}{2}\right)H_d \end{aligned} \tag{2.132}$$

We now apply the circuit to the state $\bar{\rho}$ and then measure a Hermitian operator $\bar{M}$. The expectation value can then be written as:

$$\langle \bar{M} \rangle = tr(\bar{M}U_D...U_d...U_1\bar{\rho}U_i^\dagger...U_d^\dagger...U_D^\dagger) \tag{2.133}$$

We can simplify this expression by analysing the expectation value as a function of a single parameter, θ_d. We can also simplify the notation by absorbing all gates before U_d into the density operator and all gates after U_d into the measurement operator.

$$\begin{aligned} \rho &= U_{d-1}...U_d\bar{\rho}U_1^\dagger...U_{d-1}^\dagger \\ M &= U_{d+1}^\dagger...U_D^\dagger\bar{M}U_D...U_{d+1} \end{aligned} \tag{2.134}$$

We are able to do this because unitary transformations preserve the Hermitian properties of the operators. We now also drop the d indexes and replace them with the parameter value θ. The expectation value now becomes:

$$\begin{aligned} \langle M \rangle_\theta &= tr(MU_\theta\rho U_\theta^\dagger) \\ &= tr\left(M\left(\cos\left(\frac{\theta}{2}\right)I - i\sin\left(\frac{\theta}{2}\right)\right)\rho\left(\cos\left(\frac{\theta}{2}\right)I + i\sin\left(\frac{\theta}{2}\right)\right)H\right) \\ &= \cos^2\left(\frac{\theta}{2}\right)tr(M\rho) + i\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right)tr(M[\rho, H]) + \sin^2\left(\frac{\theta}{2}\right)tr(MH\rho H) \end{aligned} \tag{2.135}$$

When we examine the terms of the last line we see that; $tr(M\rho)$ is the expectation value when $\theta = 0$

$$\langle M \rangle_0 = tr(M\rho)$$

The trace in the second term can be written as the difference between 2 expectation values:

$$\begin{aligned}
\langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}} &= tr(MU_{\frac{\pi}{2}}\rho U_{\frac{\pi}{2}}^\dagger) - tr(MU_{\frac{-\pi}{2}}\rho U_{-\frac{\pi}{2}}^\dagger) \\
&= \frac{1}{2}\left(tr(M(I - iH)\rho(I + iH)) - tr(M(I + iH)\rho(I - iH))\right) \\
&= -itr(MH\rho) + itr(M\rho H) \\
&= itr(M[\rho, H])
\end{aligned}$$

Where in the second line we have used $U_{\pm\frac{\pi}{2}} = \frac{1}{\sqrt{2}}(I \mp iH)$. Finally we note that $tr(MH\rho H)$ is the expectation value obtained from evaluating the circuit at $\theta = \pi$

Putting all of this together allows us to write the expectation value as:

$$\begin{aligned}
\langle M \rangle_\theta &= \cos^2\left(\frac{\theta}{2}\right)\langle M \rangle_0 + \sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right)\left(\langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}}\right) + \sin^2\left(\frac{\theta}{2}\right)\langle M \rangle_\pi \\
&= \frac{1 + \cos(\theta)}{2}\langle M \rangle_0 + \frac{\sin(\theta)}{2}\left(\langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}}\right) + \frac{1 - \cos(\theta)}{2}\left(\langle M \rangle_0 + \langle M \rangle_\pi\right) \\
&= \frac{\cos(\theta)}{2}\left(\langle M \rangle_0 - \langle M \rangle_\pi\right) + \frac{\sin(\theta)}{2}\left(\langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}}\right) + \frac{1}{2}\left(\langle M \rangle_0 + \langle M \rangle_\pi\right)
\end{aligned} \tag{2.136}$$

At this point we employ the trig substitution, $a\cos(x) + b\sin(x) = \sqrt{a^2 + b^2}\sin\big(x + \arctan\left(\frac{a}{b}\right)\big)$. Let:

$$\begin{aligned}
A &= \frac{1}{2}\sqrt{\left(\langle M \rangle_0 - \langle M \rangle_\pi\right)^2 + \left(\langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}}\right)^2} \\
B &= \arctan 2\left(\langle M \rangle_0 - \langle M \rangle_\pi, \langle M \rangle_{\frac{\pi}{2}} - \langle M \rangle_{\frac{-\pi}{2}}\right) \\
C &= \frac{1}{2}\left(= A\sin(\theta + B) + C + \langle M \rangle_\pi\right)
\end{aligned} \tag{2.137}$$

Using these substitutions we are able to write the expectation value as:

$$\langle M \rangle_\theta = A\sin(\theta + B) + C \tag{2.138}$$

This shows that single qubit gates can only produce sine functions, or in the context of machine learning, we have shown that single qubit gates will only be able to learn sine functions. This is very limiting as only being able to learn sine functions gives us a very small family of functions which the quantum models can work with. We therefore need ways around it.

As was mentioned in section 2.9, increasing the number of times the encoding block of a circuit was repeated also increased the complexity of the functions it was able to learn. We will now look at how the frequency spectrum of the Fourier series can be extended in a parametrized circuit in order to give the quantum model access to a larger family of functions.

2.12.3 Extending the Fourier Spectrum

Following the paper mentioned in section 2.9.1 we can now examine circuits with multiple layers and see they are able to express more complex functions than single layer circuits. We start by considering single qubit Pauli rotations in parallel as seen in figure 2.21. This gives us an encoding gate of:

$$\begin{aligned} S(x) &= e^{-i\frac{x}{2}\sigma_r} \otimes \ldots \otimes e^{-i\frac{x}{2}\sigma_1} \\ &:= e^{-ixH} \end{aligned} \tag{2.139}$$

Where σ are the Pauli matrices with equations given by equation 2.54. These rotations are occurring in parallel on different qubits and therefore the rotation gates commute with each other. This allows us to diagonalise the matrix, H, and rewrite the encoding matrix as:

$$\begin{aligned} S(x) &= V_r e^{-i\frac{x}{2}\sigma_z} V_r^\dagger \otimes \ldots \otimes V_1 e^{-i\frac{x}{2}\sigma_z} V_1^\dagger \\ &= V e^{\left(-i\frac{x}{2}\sum_{i=1}^{r}\sigma_z^{(i)}\right)} V^\dagger \\ &:= V e^{-ix\Sigma} V^\dagger \end{aligned} \tag{2.140}$$

The matrix, Σ, will have $r+1$ unique entries along the diagonal which are the eigenvalues and can be written as:

$$\Sigma = diag(\lambda_1, \lambda_2, \cdots, \lambda_{2^r}) \tag{2.141}$$

Where the eigenvalues are given by:

$$\lambda_p = \left(\frac{p}{2} - \frac{r-p}{2}\right) = p - \frac{r}{2}, \quad p \in \{0, \ldots, r\} \tag{2.142}$$

The frequency spectrum for a single layer parallel circuit is determined by the differences of the eigenvalues and is given by:

$$\begin{aligned} \Omega_{\mathrm{m}} &= \{\lambda_{k_1} - \lambda_{j_1} \mid k_1, j_1 \in \{1, \ldots, 2^r\}\} \\ &= \left\{\left(p - \frac{r}{2}\right) - \left(p' - \frac{r}{2}\right) \mid p, p' \in \{0, \ldots, r\}\right\} \\ &= \{p - p' \mid p, p' \in \{0, \ldots, r\}\} \\ &= \{-r, -(r-1), \ldots, 0, \ldots, r-1, r\} \end{aligned} \tag{2.143}$$

We have now successfully increased the degree of the Fourier series from 1 to r by increasing the number of parallel encodings which are performed. This tells us that with enough qubits we can approximate the Fourier series of any function. We can also perform the encoding in series on a single qubit L times and it turns out that the spectrum of frequencies which the model has access to is the same as a circuit with L qubits in parallel. This allows us to use the same method on near term quantum computers which do not posses many qubits.

Using this method we are able to approximate any function which can be decomposed into a Fourier series given enough resources. However it is not the most resource efficient method. In the next section we will look at an circuit which gives us access to 3^n frequencies of the Fourier spectrum with just n qubits.

2.12.4 Exponentially Extending the Fourier Spectrum

We will now examine an encoding method proposed in "Exponential data encoding for quantum supervised learning" by Shin, Teo and Jeong which exponentially increases the number of Fourier frequencies we can access with n qubits [72].

We start with the unitary operation:

$$U_{\boldsymbol{x};\vec{\theta}} = W_2\left(\vec{\theta}_2\right) S(\boldsymbol{x}) W_1\left(\boldsymbol{\theta}_1\right) \tag{2.144}$$

Where $\mathrm{S}(\vec{x})$ is the encoding block and $W_i(\vec{\theta_i})$ is the parameterised block. This unitary operation is placed into a circuit where it acts upon the ground state and a measurement is made in the Z-basis at the end of the circuit. The trainable parameters are labelled θ and the data points are labelled $\vec{x}$. The encoding gate, $\mathrm{S}(\vec{x})$ given by:

$$\begin{aligned} S(\boldsymbol{x}) &= \bigotimes_{m=1}^{M} \bigotimes_{n=1}^{N} \mathrm{e}^{-\mathrm{i}\beta_{mn} x_m Z/2} \\ &= \bigotimes_{m=1}^{M} \sum_{\boldsymbol{k}_m \in \{0,1\}^N} |\boldsymbol{k}_m\rangle \, \mathrm{e}^{-\mathrm{i}\lambda_{\boldsymbol{k}_m} x_m} \, \langle \boldsymbol{k}_m| \end{aligned} \tag{2.145}$$

$\mathrm{S}(x)$ is a diagonal encoding unitary in the N qubit basis, $|k_m\rangle = |k\rangle$. The eigenvalues of $\sum_{n=1}^{N} \beta_{mn} \frac{Z_n}{2}$ are given by λ_{k_m}. A function, f, expressible by this quantum model is given by:

$$\begin{aligned} f(\boldsymbol{x}) &= \left\langle \mathbf{0} \left| U^{\dagger}_{\boldsymbol{x};\boldsymbol{\theta}} Z U_{\boldsymbol{x};\boldsymbol{\theta}} \right| \mathbf{0} \right\rangle \\ &= \bigotimes_{m=1}^{M} \sum_{\boldsymbol{k}_m, \boldsymbol{k}'_m \in \{0,1\}^N} \mathrm{e}^{-\mathrm{i}\left(\lambda_{\boldsymbol{k}_m} - \lambda_{\boldsymbol{k}'_m}\right) x_m} \left\langle \boldsymbol{k}_m \left| D_1^{\dagger} W_2^{\dagger} Z W_2 D_1 \right| \boldsymbol{k}'_m \right\rangle \\ &\equiv \sum_{n_1 \in \Omega_1} \sum_{n_2 \in \Omega_2} \cdots \sum_{n_M \in \Omega_M} n_{n_1, n_2, \ldots, n_M} \mathrm{e}^{-\mathrm{i}\boldsymbol{n}\cdot\boldsymbol{x}} \end{aligned} \tag{2.146}$$

where

$$D_1 = \sum_{\boldsymbol{k}'} |\boldsymbol{k}'\rangle \, \langle \boldsymbol{k}' \, | W_1 | \, \mathbf{0}\rangle \, \langle \boldsymbol{k}'| \tag{2.147}$$

The Fourier frequency spectrum is composed of the set of all possible integral differences between the eigenvalues and is labelled, Ω_m. As with previous examples of parametrized quantum circuits the circuit can be layered to increase the complexity and by repeating the layers enough times the Fourier series of any function can be learned to an arbitrary accuracy. Sometimes this Fourier approximation has to be rescaled in amplitude and period to accurately reflect the function which is being approximated.

We will now look at what happens to the frequency spectrum if we make β_{mn} positive in equation 2.145. We can write the eigenvalues as:

$$\begin{aligned} \lambda_{mn} &= \sum_{n=1}^{N} \frac{\pm\beta_{mn}}{2} \\ &= \sum_{n=1}^{N} a_n^{k_m} \end{aligned} \tag{2.148}$$

This means that there are 2^N eigenvalues if there are degenerate eigenvalues. We can therefore write the frequency spectrum as:

$$\Omega_m = \left\{ \lambda_{k_m} - \lambda'_{k'_m} \mid k_m, k'_m \in \{0, 1\} \right\} \tag{2.149}$$

The frequency spectrum can contain at most $4^N - 2^N + 1$ frequencies. If $\beta_{mn} = 1$ then there are $2^N + 1$ frequencies and the degree of the Fourier series would be equal to the number of qubits, N. The spectrum can however be extended by making sure that the β_{mn} values satisfy the recurrence relation:

$$\beta_{mn} = 2 \sum_{j=1}^{n-1} \beta_{mj} + 1 \tag{2.150}$$

Solving this gives us $\beta_{mn} = 3^{n-1}$. This also tells us that the degree of the Fourier series, d, is related to the number of qubits by:

$$N = \log_3(2d + 1) \tag{2.151}$$

2.13 Chapter Conclusion

This brings the literature review section of this thesis to a close. We have gone over support vector machines and kernel methods, as well as looked at Fourier and wavelet analysis. We also looked at the basic notation needed for quantum mechanics and the postulates from which the properties of quantum mechanics are derived. We then looked at how that led to the invention of quantum computing and the basic structure of a quantum computer. We demonstrated that in theory a quantum computer can out perform a classical computer in certain tasks. From there we moved to quantum machine learning and gave a brief overview of where the field is currently. We then looked at parameterised quantum circuits and examined how different methods of encoding could affect the functions which they could learn.

Given all of this information we are now ready to discuss whether using kernels is a good place to search for quantum advantage. Following that discussion we will layout the methodology for how to create a quantum circuit which can approximate a wavelet kernel. We will then go implement this circuit and examine the results.

Chapter 3

Using Kernels to Find Quantum Advantage

3.1 Quantum Advantage

We briefly described quantum speedup in section 1.3. Here we will go over the definitions again and give some examples so that we can examine the advantages and disadvantages of quantum computers in the context of kernel methods. We will then argue that quantum kernel methods should not be used in order to search for new quantum advantages but that kernel methods should still exploit and adapt quantum advantages when they are found elsewhere to improve the performance of quantum kernel methods.

We will break quantum advantage into 4 categories [18]. The terms speed-up and advantage are also used interchangeably in this thesis. The first category is provable quantum speed up. Algorithms in this category prove that there is no existing classical algorithm which can solve the problem more efficiently. An example of provable quantum speedup is Grover's search algorithm. There is an intricacy to this definition, it claims the best possible classical algorithm which is not the same as the best known classical algorithm. To claim that an algorithm is the best possible algorithm the upper and lower bounds of the problem itself need to be thoroughly looked at and in the case of many problems in computer science the best possible algorithm is unknown. Many problems involving a provable quantum advantage also require the use of an oracle to get the solution.

We therefore create a category for speed-ups over the best known classical algorithms and term it a strong quantum speedup. An example of strong quantum speed up is Shor's algorithm. A problem that occurs is that in the field of quantum computing is that some papers claim to have provided a quantum advantage but do not specify the type of quantum advantage which was achieved explicitly [73, 74]. However, as research continues this problem seems to be addressed more frequently.

The next category of quantum speedup is limited quantum speedup. This category refers to algorithms where the quantum advantage is obtained through quantum effects which are not replicable on classical computers. These algorithms usually use the same or a similar algorithmic approach as their classical counterparts and usually rely on

the quantum properties of the computer to gain an advantage. These algorithms are much more dependant on the architecture being used to find their advantages than the algorithms mentioned in the two previous categories. Examples of algorithms which fall into limited quantum speedup include adiabatic quantum optimization, quantum annealing or simulated quantum annealing [48, 75–78].

Lastly we have potential quantum speedup. This refers to when we directly compare a classical algorithm to a quantum algorithm and see how the quantum algorithm provides a speedup. This is usually done to try and show how quantum computers could improve computing performance for more common use cases and works towards quantum supremacy by ensuring that algorithms which are currently performed on classical computers are able to run on quantum computers as well [79, 80].

3.2 Advantages Offered by Quantum Kernels

In kernel methods of machine learning the function of the kernel is to evaluate the inner product in a feature space. Usually the spaces which we map the data to allow us to identify features within the data which can be hard to extract in the original space. As was explained in section 2.10 quantum models themselves can also be viewed as kernel methods and using this we are able to study quantum models in a different light.

By examining the kernel function and the space which it maps to we can try to understand the features which are accentuated within that feature space and what sorts of data sets would benefit from being processed in that space. We can also do this without necessarily having to run any data sets through a machine learning model which can save computational resources, especially when processing large amounts of data.

In the context of quantum computing the question then becomes: how do we know whether a kernel provides quantum advantage? As mentioned previously, quantum kernels can provide a quantum advantage classical kernels when there is non-linearity. The non-linearity can manifest when the mapping requires a quantum property such a superposition to be performed. If the mapping is linear then a classical computer could map us into the same space with some overhead. The factors which result in quantum advantage are still being actively researched and we will likely find more contributing factors as more research is done.

An example of a quantum kernel which offers this speed up can be seen in Liu et al [36]. The kernel used in this paper uses a kernel based on the discrete logarithm problem in order to classify data using supervised machine learning. The discrete logarithm problem is computationally intractable [81]. This means that it can be solved on a classical computer, however there is no known classical polynomial time algorithm for the discrete logarithm problem. This means that when large enough numbers are used it becomes impossible to solve within a reasonable time frame. The discrete logarithm problem can be adapted solved using a modified version of Shor's algorithm [82].

The data sets used in the paper are classical and constructed, so that ideal conditions for the kernel could be met. The researchers were able to construct a quantum classifier which could correctly classify the data and showed that a classical computer could not classify the data inverse-polynomially better than random guessing. This paper shows that finding quantum rigorous quantum speed-up is possible and give us a way to compare the classical methods to quantum kernels.

The tactic of the paper by Liu et al. was to start with a problem which had a quantum solution that offered a strong quantum speedup. They then analysed the mathematics behind the problem in order to identify properties about the space which the problem is mapped to before it is solved. Using this information they were able to construct a data set which had patterns that were identifiable in this space, but that could not be mapped to using a known classical feature map. They were then able to build a classifier based on a kernel which mapped them to this space where the data was exploitable and classify the data better than a classical computer would be able to. This general tactic could possibly be used in the future if we are able to find new spaces which are hard to map to using a quantum computer and are also able to show us underlying properties in data that aren't visible in other spaces.

Quantum kernels can also offer an advantage over classical methods when the data itself is quantum [45]. This speed-up has to make assumptions about the data as real world quantum data is rare and experiments are hard to perform due to the noisy nature of NISQ computers. When simulations and experiments are performed that involve quantum data or particles the data is usually converted to classical data before it is processed and interpreted. This happens because most researchers are using classical computers when processing data and because there is no way to easily and reliably store quantum data at this point [83, 84]. Some hope that quantum computers will be able to manipulate and interpret this data before it is mapped into a classical space which could affect the results of the data [85]. However, more research needs to be done in this field before anything concrete can be said.

3.3 The Downsides of Searching for Quantum Advantage in Kernel Methods

Although there are possible speed-ups provided through the use of kernel methods in quantum computing this does not mean that quantum kernel methods should be used to search for new quantum advantages. A quantum kernel might yield a speed up if it facilitates the efficient evaluation of inner products in some feature space, which cannot be efficiently evaluated using a classical computer.

Quantum kernel methods are a form of machine learning and in machine learning you often do not know anything about the underlying structures of the data. We generally employ machine learning methods to try and uncover these underlying structures so that we can build more robust models and understand the structures. If one were to try and search for a kernel which mapped us to a space which gave a quantum advantage using quantum kernel methods they would have to essentially try generating random kernels and applying them to a data set to see whether the algorithm was able to find patterns

in the data when it is mapped to the new space. The problem with this approach is that it is not an efficient method of searching.

The algorithm would just be taking shots in the dark over and over again and is not guaranteed to ever produce what we are looking for. Even if a kernel was found that gave us a quantum advantage, we would not be able to guarantee that a data set had the features with which the advantage could manifest. The only way to guarantee that the data set contained these features would be to artificially construct them and to artificially construct them we would need to know what the kernel was in the first place.

In most examples of quantum speed-up in kernel methods the quantum advantage has come from another area of quantum computing and then been adapted to offer its advantage in the area of machine learning. The discrete logarithm kernel was adapted from Shor's algorithm as it is arguably the most prominent example of quantum advantage [36, 82]. In most examples of quantum advantage in kernel methods the advantage is demonstrated using synthetic data sets and is usually used as proof of concept [56,62,73]. The methods of quantum kernels are usually hard to generalise to larger families of data sets. There has also been work done which shows that by preparing data correctly classical models can be competitive with quantum models [45].

A better path towards finding quantum advantages would be to create well defined problems where a quantum algorithm can be compared to classical algorithms to establish what, if any, quantum advantage the quantum algorithm can offer. If a quantum advantage is found we can then look for ways to adapt the algorithm into machine learning and kernel methods to utilise its speed-up. If a quantum algorithm is found to not have a quantum advantage this, while not as exciting, still gives us useful information and could help us better determine in the future which properties give rise to quantum advantage and which do not.

Chapter 4

Approximating a Wavelet Kernel on a Quantum Computer

In this chapter we will follow the methods laid out in a tutorial on how to approximate clasical kernel using a quantum computer, which is available on PennyLane [86]. We will be using it in order to approximate a wavelet kernel. This kernel comes from the paper by Zhang et al. where the wavelet kernel was used in a classical support vector machine to approximate an arbitrary non-linear function [87].

As outlined in section 2.4, wavelet analysis is a useful tool with many applications in modern computing, including noise reduction and compression. The quantum wavelet transform is not something that has been researched thoroughly and very little information is available about the topic. There have been quantum circuits created which can implement a Haar and Daubechies wavelet transform [88, 89], but they have not been tested on near term quantum computers.

As has been shown in section 2.12, the functions which can be learnt by parameterised quantum circuits using rotational embedding can be described using a Fourier series. The frequency values that make up the Fourier series are given by the eigenvalues obtained by treating the embedding method as a kernel following the method used in section 2.12. We also showed in section 2.4 that wavelet analysis can be viewed as an extension of Fourier analysis. It is therefore interesting to examine whether we can use quantum kernels to begin implementing a form of wavelet anlaysis on NISQ quantum computers.

We will be focusing on using a quantum computer to recreate the mother wavelet function. The methodology laid out in the tutorial [86] was released in March 2022 making it quite a recent development. We have adapted the code and method used in the tutorial in order to approximate a wavelet kernel. This kernel had not been successfully approximated using this method previously.

4.1 Wavelet Kernel

The wavelet kernel is a shift invariant kernel defined by:

$$k(x, y) = \prod_{i=1}^{d} h\left(\frac{x_i - y_i}{a}\right) \tag{4.1}$$

Where d is the number of dimensions, in our case $d = 1$. The mother wavelet function, h, we will use is:

$$h(x) = \cos(1.75x)e^{\left(-\frac{1}{2}x^2\right)} \tag{4.2}$$

Here the factor of 1.75 inside of the cosine function is used to adjust the frequency of the function. This value is chosen because it makes the period of the function in the region of interest π.

4.2 Method

We will now go over the method used to approximate the wavelet kernel. The method will be kept as general as possible so that it can also be used to approximate other kernels. This method was implemented in python using the numpy and PennyLane libraries but the logic of the methods will be outlined, and the code used will be attached as an appendix.

Algorithm 1 Approximating a kernel

1: Choose a kernel, $k(x)$.
2: Adapt k(x) such that, $k(0) = 1$ and $k(x) > 0$.
3: Periodically extend $k(x)$.
4: Find the Fourier Spectrum.
5: Determine number of coefficients needed to accurately approximate the kernel.
6: Construct quantum circuit.
7: Create test amplitudes.
8: Predict the Fourier spectrum from test amplitude.
9: Optimise amplitudes to match Fourier coefficients using Newton's method
10: Send test amplitudes to quantum circuit.
11: Compare Fourier coefficients of the classical kernel to the approximated coefficients.
12: Scale and shift the approximated kernel to match $k(x)$.

Algorithm 1 gives a very general outline of the process we will use to approximate the wavelet kernel on a quantum computer. We will now go through each step of algorithm 1 and provide more detail.

4.2.1 Preparing the kernel function

To simplify the problem we will consider 1-dimensional data. Steps 1 and 2 of algorithm 1 deal with the selection of the kernel and making sure that the quantum circuit we will use can approximate it. The kernel function can be any function as long as it meets the requirements to be a kernel as laid out in section 2.2. The quantum circuit we use requires that the kernel function have $k(\mathbf{x}) > 0$ for all values of $\mathbf{x}$. It also requires that $k(0) = 1$. The second requirement is because the kernels are used as similarity measures and therefore need to be normalised. Finally, the kernel must be shift invariant i.e.

$$k(\mathbf{x}_1, \mathbf{x}_2) = k(\mathbf{x}_1 + \delta, \mathbf{x}_2 + \delta), \quad \delta \in \mathbb{R} \tag{4.3}$$

We will also take the kernel as a function of just one variable, x. This can be done because of the shift invariant property. The kernel only cares about the difference between the two data points fed into equation 4.3. Therefore, we relabel the difference between these two variables, x for simplicity. In the case of the wavelet kernel given by equation 4.2, the function meets the requirement of $k(0) = 1$. However, the function is not positive across its entire range, so we must first shift it until this condition is fulfilled. Once the function is shifted it must then be scaled so that $k(0) = 1$.

Because we are performing the approximation computationally, we must choose a domain for the function and the number of data points in the domain. We have elected to use 100 data points equally spaced between $-\pi$ and π. This will be a high enough resolution of x values to approxmiate the wavelet kernel in the domain we have specified.

After step 2 we periodically extend the kernel so that we can apply the Fourier series to it. We have multiplied the x values inside the cosine part of the wavelet kernel so that its features of interest to us appear from $-\pi$ to π. We therefore set the period of the extension to be 2π. To perform the periodic extension we simply repeat the values of the kernel in each interval. For example, the output that occurs at $k(\pi)$ would be the same data point as $k(-\pi)$ instead of its original value. We periodically extended the kernel 8 times for a total of 9 periods. This number was chosen to hopefully make the the extraction of the Fourier series more accurate.

4.2.2 Determining the Fourier Spectrum

After periodically extending the function we can now determine its Fourier spectrum using steps 4 and 5. To do this we use the *coefficients* function from the PennyLane qml.fourier library. This function computes the first $2d + 1$ Fourier coefficients of a 2π periodic function, where d is the highest degree of the Fourier spectrum.

At this point we need to determine how many terms of the Fourier series will be needed to effectively approximate the kernel we have chosen. The number of terms is also directly related to the number qubits that are needed. If n qubits are used then we will have access to 2^n terms of the Fourier series [72]. In this thesis we will do the approximation using a 3 qubit and 5 qubit version of the circuit so that we can compare the results.

To determine how many coefficients are necessary to effectively approximate the kernel we look at the magnitude of the coefficients. The higher the magnitude, the more the frequency associated with that coefficient contributes to the Fourier series. A threshold value should be chosen for the magnitude of the coefficients where the contributions can be seen as negligible. Terms of the Fourier series should be included until the magnitude of subsequent terms become 2-3 orders of magnitude smaller than the leading terms. It is good to plot the Fourier spectrum to visually see where this point lies before performing a closer inspection of the numbers. If there are extra computational resources available it is advisable to have a threshold value which includes more terms of the Fourier series.

4.2.3 Constructing the Quantum Circuit

The quantum circuit used to approximate the classical kernel follows the structure of a parameterised quantum circuit as laid out in section 2.8. The circuit contains a data encoding block and a parameterised circuit block. The structure of the circuit can be seen in figure 4.1. The circuit ansatz is adapted from the circuit proposed by Shin et al [72]. In this paper the circuit used was able to express 3^n terms of the Fourier series, where n is the number of qubits. The circuit used for this thesis, seen in figure 4.1 is able to express 2^n terms of the Fourier series. Although the cirucit used in this thesis can express fewer terms of the Fourier series per qubit, with the resources available we are still able to create a circuit which approximates over 99% of the Fourier series.

The circuit was implemented using the the PennyLane library and a quantum computer was simulated on a classical computer. The same methods and code that will be outlined should also work on a real quantum computer as long as it can implement the circuit needed.

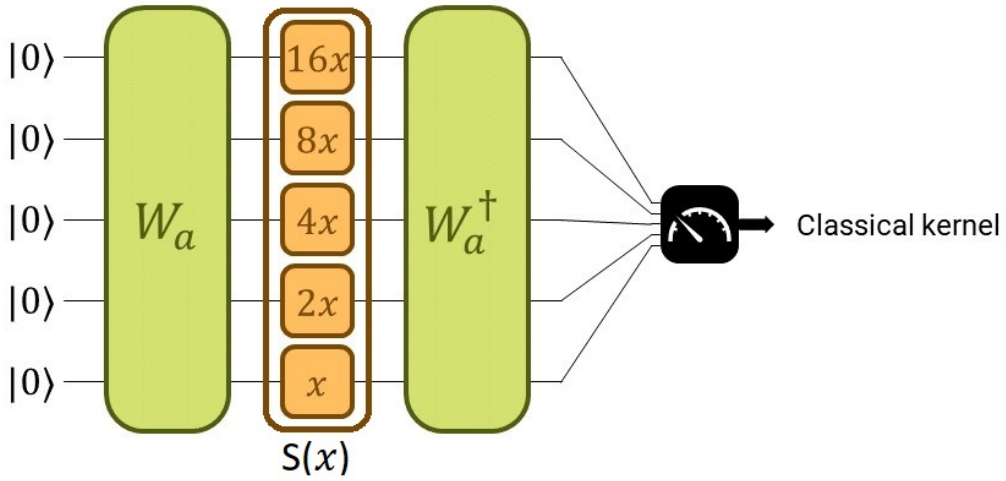


Figure 4.1: Form of the quantum kernel used to produce most of the results in this thesis. The parameterised block is pictured in green and labelled W_a with its conjugate transpose appearing before the measurement operation. The orange blocks between the parameterised blocks are the encoding gates. The numbers appearing in front of the x values are the θ_i values described by equation 4.4. This circuit uses 5 qubits but the number of qubits can be adjusted in the source code and the circuit is adapted accordingly. Image from PennyLane [86].

To implement the data encoding gates, S(x), pictured in orange in figure 4.1, we apply a Pauli Z rotation to each qubit. The rotation parameter is defined by x multiplied by a constant, labelled θ_i, where i indicates the i^{th} qubit. The θ_i values need to be chosen so that all the elements of the frequency spectrum are integer valued, this is required by the Fourier series expansion of a function with period 2π. The θ_i values must equal:

$$\theta_i = 2^{n-i} \tag{4.4}$$

Which leaves us with the data values all being multiplied by powers of 2 as can be seen in figure 4.1

The W_a gate is implemented using the *MottonenStatePreparation* function from PennyLane. According to the PennyLane documentation [51] this function prepares an arbitrary state via a series of controlled rotations. It is a form of amplitude encoding. The unitary transformation associated with this template maps the state $|0\rangle$ into a state $|a\rangle$ via:

$$U_m |0\rangle = |a\rangle = \sum_j a_j |j\rangle \tag{4.5}$$

The amplitudes, a_j, are given by $a = (a_o, a_1, \cdots, a_{2^n-1})$ and the amplitudes must satisfy the normalisation condition.
The quantum circuit constructed then implements the feature map:

$$|x_a\rangle = S(x)W_a |0\rangle \tag{4.6}$$

And the kernel is described by:

$$\begin{aligned} k_a(x_1, x_2) &= |\langle 0| W_a^\dagger S^\dagger(x_1) S(x_2) W_a |0\rangle|^2 \\ &= |\langle 0| W_a^\dagger S(x_2 - x_1) W_a |0\rangle|^2 \\ &= |\langle 0| W_a^\dagger S(x) W_a |0\rangle|^2 \end{aligned} \tag{4.7}$$

Where the shift invariant property of the kernel has been used to replace x_1 and x_2 with a single variable, x. The last step needed to construct the quantum circuit is to pick an initial set of amplitudes and ensure that they are normalised. The initial amplitudes are given by:

$$a_i = \frac{1}{1+i} \quad i = 0, ..., 2^n \tag{4.8}$$

These amplitudes are chosen so that when they are run through the circuit they will create the graph of $\frac{1}{|x|}$. This is chosen as a starting point because it fulfils the conditions needed for this circuit of, $k(0) = 1$, and $k(x) > 0$ for all values of x. We also expect that we will be able to adjust the value of the amplitudes from this point in order to get closer to a kernel which we are trying to approximate.

4.2.4 Approximating the Fourier Spectrum

The next step is to optimise the amplitudes so that they better approximate the classical kernel we are looking for. Traditionally in a parametrised quantum circuit we would use an iterative process and adjust the parameters of the circuit until the loss function calculated using the quantum kernel approximated, and the classical kernel reached the predetermined threshold, or till the run of runs in the algorithm was reached. This process is not always transparent, so even when the model produced does well in testing we do not know explicitly how it got to its result [90]. This is known as the black box problem. We use a classical computer to optimise the amplitudes to avoid the black box problem because we have more information about the parameterised variables and how they were derived instead of leaving the process up to an optimisation algorithm.

In order to construct an analytical solution we introduce a new variable, p_j, where p_j is the probability of measuring amplitude, a_j.

$$p_j = |a_j|^2. \quad \sum_j p_j = 1 \tag{4.9}$$

By expanding the matrix product $W_a^\dagger S(x) W_a$ we are able to get a formula for shift invariant kernels which links the probabilities to the Fourier coefficients, given by the entries of the matrix on the right hand side in equation 4.10.

$$\begin{pmatrix} p_o \\ p_1 \\ p_2 \\ \vdots \\ p_{2^n-1} \end{pmatrix} \mapsto \begin{pmatrix} \sum_{j=0}^{2^n-1} p_j^2 \\ \sum_{j=1}^{2^n-1} p_j p_{j-1} \\ \sum_{j=2}^{2^n-1} p_j p_{j-2} \\ \vdots \\ p_{2^n-1} p_0 \end{pmatrix} \tag{4.10}$$

This relation can be used in order to find the probabilities which produce the Fourier coefficients of any stationary kernel. This set of Fourier coefficients is also called its spectrum denoted, $s_0, s_1, s_2, ..., s_{2^n-1}$. If we subtract the spectrum of the kernel we want to approximate from the the kernel produced by the test probabilities we can gauge how far we are from the kernel we are trying to approximate.

We use a map, F_s, which relates the probabilities to the difference between the probabilities and the spectrum. This is used to gauge how far our current probabilities are from the Fourier spectrum. We can then use this as an error norm and choose a method to minimise it and get probabilities which are closer to the Fourier spectrum we are trying to approximate. The purpose of this is not to solve the whole problem but rather to give our quantum computer the best initial parameters that we can so that we can understand the process that the quantum computer uses in order to approximate the kernel. The error norms were calculated using the *linalg.norm* function from numpy.

$$F_s : \begin{pmatrix} p_o \\ p_1 \\ p_2 \\ \vdots \\ p_{2^n-1} \end{pmatrix} \mapsto \begin{pmatrix} \sum_{j=0}^{2^n-1} p_j^2 - s_0 \\ \sum_{j=1}^{2^n-1} p_j p_{j-1} - s_1 \\ \sum_{j=2}^{2^n-1} p_j p_{j-2} - s_2 \\ \vdots \\ p_{2^n-1} p_0 - s_{s^n-1} \end{pmatrix} \tag{4.11}$$

From this point the problem can be solved numerically and is done using Newton's method as the formula is quadratic and is therefore a convex function [91]. Newton's method was run with a maximum of 200 steps and a cut off tolerance of 1×10^{-20}. In the original tutorial the method was run for 100 steps and the tolerance was kept the same. We doubled the maximum number of steps because we believed the wavelet kernel might be harder for the model to learn and therefore wanted to allow it more steps to do so. The threshold was set to 1×10^{-20} because if the difference between the Fourier coefficients and the probabilities were reduced to this magnitude they could be considered close enough for most practical settings.

4.2.5 Running the Quantum Circuit

Once the amplitudes have been optimised we can run the quantum circuit again using the new amplitudes. At this time we can compare the Fourier spectrums of the quantum kernel once it has run on the quantum computer, the quantum kernel predicted by the amplitudes, and the Fourier spectrum of the classical kernel we are trying to approximate.

This is done so that we can compare the three Fourier series to each other and the values of the Fourier series are close to each other then we can expect that the functions they create will be close to each other. The method by which we evaluate this closeness is explained in the next subsection.

4.2.6 Evaluate the Accuracy of the Approximation

The next step is to shift and rescale the kernels if they were shifted and scaled in the second step of algorithm 1. The kernels should be shifted by a constant first and then multiplied by a second constant to ensure that $k(0) = 1$. The approximated quantum kernel can then be plotted alongside the classical kernel and the two can be compared in order to see how well the classical kernel was approximated.

To evaluate how close the approximated kernel is to the wavelet kernel we will use a mean squared error between the two functions:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(k(\mathbf{x}_i) - \hat{k}(\mathbf{x})_i\right)^2 \tag{4.12}$$

Where $k(\mathbf{x}_i)$ is the wavelet kernel and $\hat{k}(\mathbf{x})_i$ is the approximated kernel. By comparing the two functions at each point we can have a measure of how close the two functions are to each other.

This concludes the method section of this thesis. We will now move on to the results chapter where we show the figures and plots generated by following this method using the code provided in the appendix.

Chapter 5

Results

In what follows are the results obtained by implementing the method in chapter 4.2 to approximate the wavelet kernel defined in equation 4.2. The full code used to generate the plots and figures in this section is included in appendix A. We will display the results obtained in each subsection of section 4.2 and in the next chapter we will discuss these results and their implications. We will also compare the results obtained using this method to the results obtained in the original tutorial on PennyLane [86] when approximating a Gaussian kernel. This is done to see whether the results of the approximation of the wavelet kernel are in line with the results of the original tutorial.

5.1 Preparing the Wavelet Kernel

The kernel plotted between $-\pi$ and π can be seen in figure 5.1. The kernel was shifted upwards by a constant value of 0.289 and then scaled by a factor of $\frac{1}{1.287}$. The constant value to shift the kernel by was obtained by searching for the lowest value of the function, l, between π and $-\pi$. The magnitude of this value was then added to equation 4.2. To work out the scaling factor we took the maximum value of the kernel after the shift, m, and then multiplied the whole function by its inverse. The shifted and scaled version of the wavelet kernel is then given by the equation:

$$h(x) = (\cos(1.75x)e^{\left(-\frac{1}{2}x^2\right)} + l) \times \frac{1}{m} \tag{5.1}$$

The periodic extension of the kernel can be seen in figure 5.2. The kernel was repeated 8 times for a total of 9 periods.

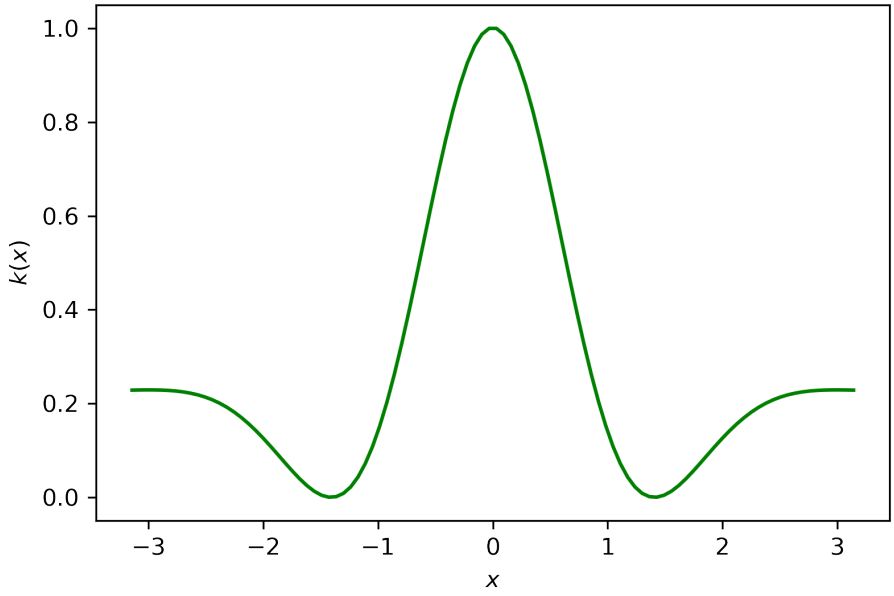


Figure 5.1: Wavelet kernel defined by equation 5.1.

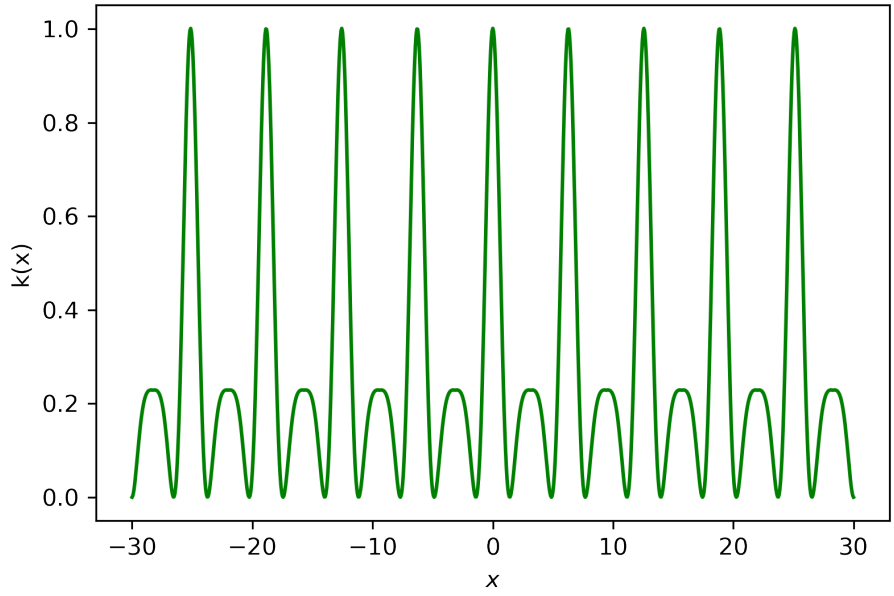


Figure 5.2: Periodic extension to the wavelet kernel

5.2 Determining the Fourier Spectrum

The first 32 terms of the Fourier spectrum of the periodically extended wavelet obtained using the PennyLane Fourier module can be seen in figure 5.3. The 4th Fourier coefficient is 0.071, the 5th is 0.012 and the 6th is 9.08×10^{-4}. Of the first 32 terms of the Fourier series the first 5 coefficients made up 99.864% of the spectrum. The circuit that used 3

qubits approximated 8 terms of the Fourier series, while the 5 qubit circuit approximated 32 terms of the Fourier series.

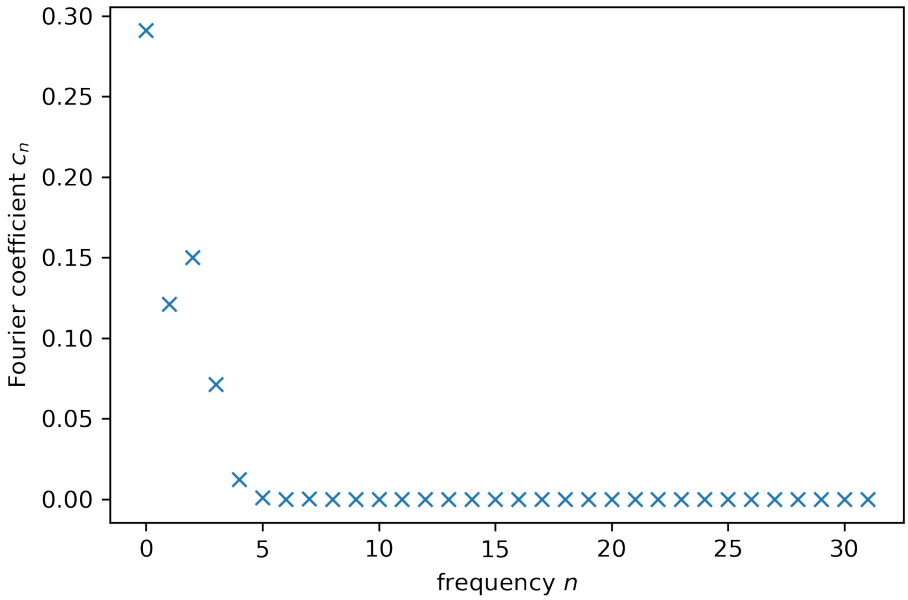
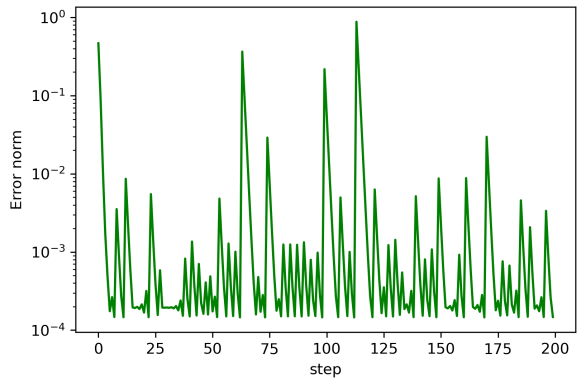


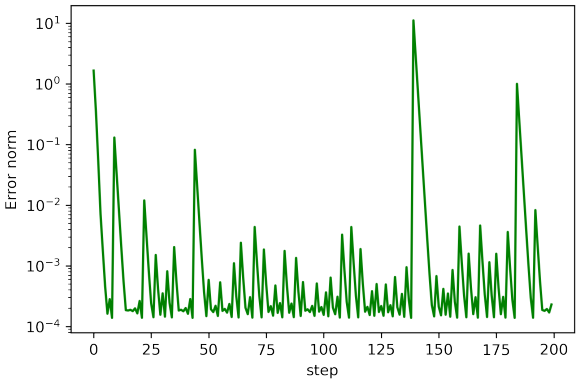
Figure 5.3: Fourier coefficients of the periodically extended wavelet kernel

5.3 Optimising the Amplitudes

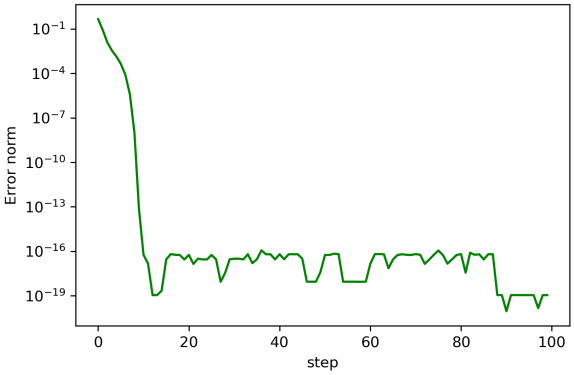
Using 200 iterations of Newton's method the amplitudes were optimised according to equation 4.11. The error norms are plotted below alongside error norms for the approximation of a Gaussian kernel using the same method.



(a) Error norms of wavelet kernel amplitudes from the quantum circuit using 5 qubits



(b) Error norms of wavelet kernel amplitudes from the quantum circuit using qubits

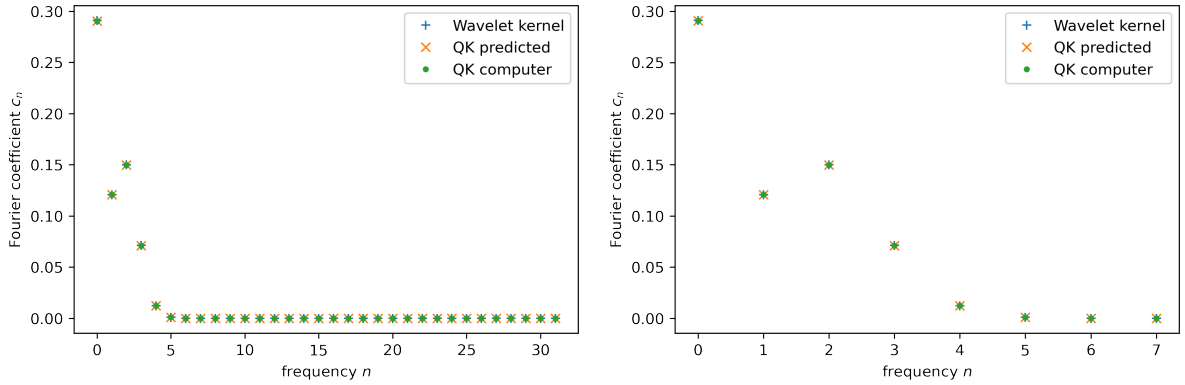


(c) Error norms of Gaussian kernel amplitudes

Figure 5.4: The error norms from using Newton's method in order to optimise the probabilities being fed into the quantum computer to better approximate a quantum kernel. The error norms of the wavelet kernel are much than the error norms of the Gaussian kernel implying that the amplitudes of the Gaussian kernel were better approximated.

5.4 Comparing the Fourier Coefficients

After running the quantum circuit we are able to obtain the Fourier coefficients of the approximated quantum kernel, the optimised amplitudes and the original wavelet kernel. These are all plotted in figure 5.5a



(a) 5 qubit circuit.

(b) 3 qubit circuit.

Figure 5.5: The Fourier spectrums of the wavelet kernel as well as the predicted spectrums and the spectrums from the quantum computer.

We then plotted the first 8 Fourier coefficients on a logarithmic scale, seen in figure 5.6, for the 5 qubit circuit. We also put the values of the first 8 Fourier terms of the wavelet kernel, the approximation and the difference between them into a table so that they could be examined.

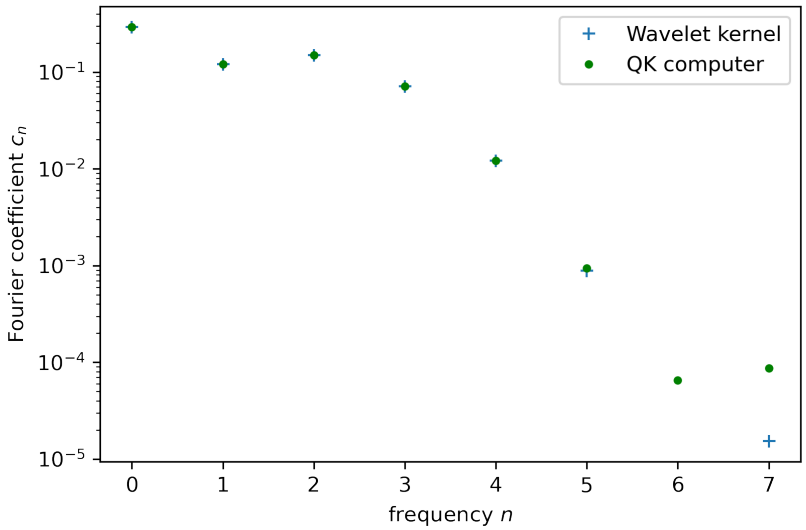
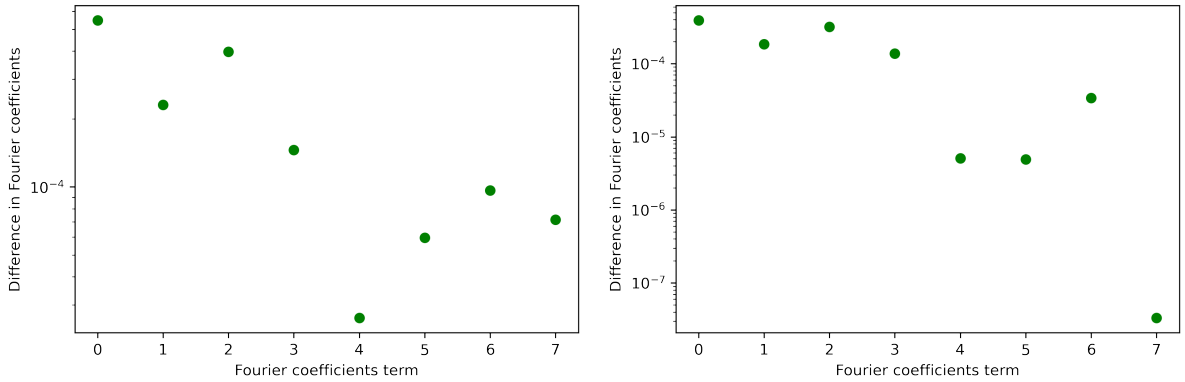


Figure 5.6: The first 8 terms of Fourier spectrum of the wavelet kernel and the spectrum from the quantum computer plotted with a logarithmic scale.



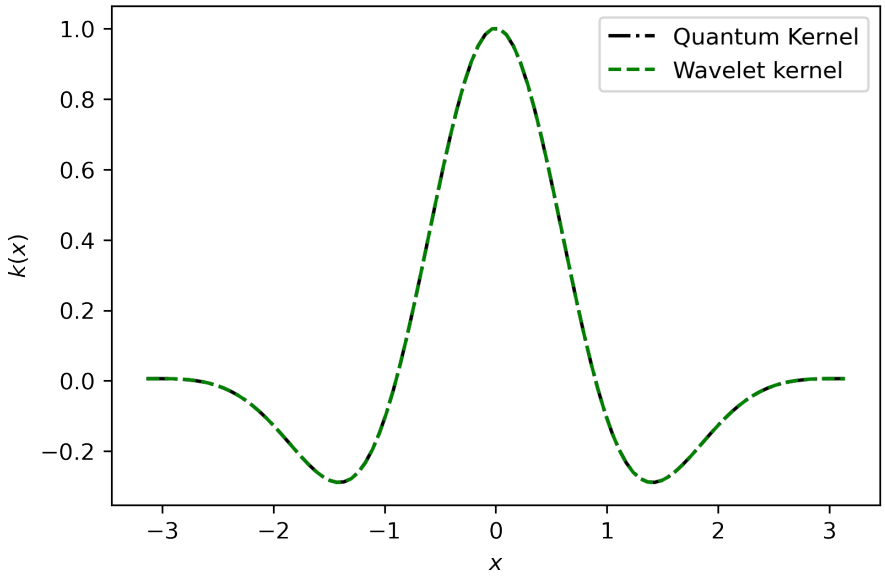
(a) 5 qubit circuit.

(b) 3 qubit circuit.

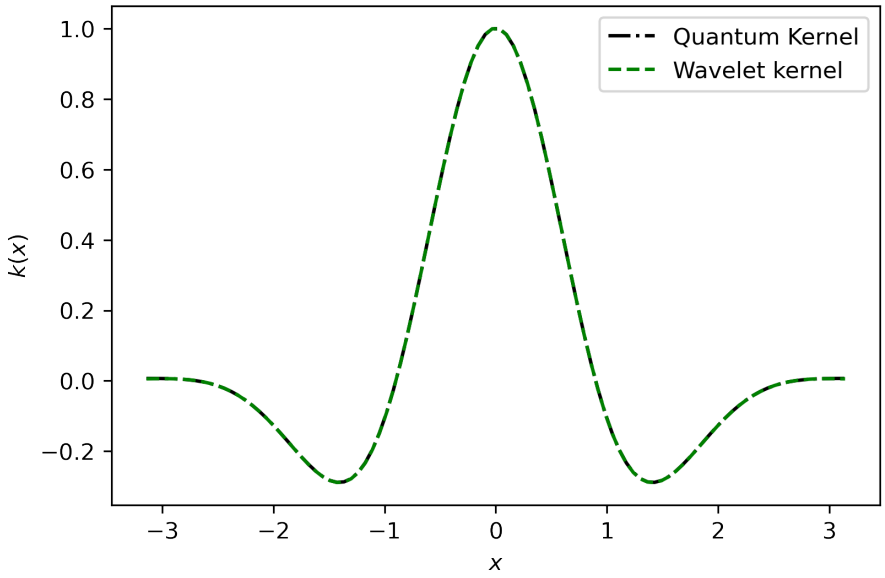
Figure 5.7: The differences betwen the first eight terms of the Fourier series of the wavelet kernel and the approximated Fourier spectrum.

5.5 Evaluate the Accuracy of the Approximation

The shifted and scaled wavelet kernel is plotted below with the kernel approximated by the quantum computer. This is done to see whether the quantum computer was able to approximate the key features of the wavelet kernel. The wavelet kernel is then multiplied by 1.289 and shifted by -0.289 to return it to its original form. The approximated quantum kernel is similarly scaled by 1.29089 and shifted by -0.289 in order to match the original wavelet kernel. The results of this can be seen in figure 5.8a.



(a) 5 qubit circuit.



(b) 3 qubit circuit.

Figure 5.8: Quantum kernel scaled up to match the wavelet kernel and shifted to match the original wavelet kernel function

This concludes the results section of this thesis. We were able to follow the method laid out in section 4.2 and create the plots and figures presented. The method could possibly be generate better reults by using a different optimisation method over Newton's method, but the results presented seem to allign with the goals laid out in the introduction in section 1.4. We will now move on to an in depth discussion of these results int he next chapter.

Chapter 6

Discussion

6.1 Analysing the Approximation of the Classical Wavelet Kernel

6.1.1 Preparing the kernel

The wavelet kernel had to be shifted upwards because the kernel was negative in the regions $-2.692 < x < 0.898$ and $0.898 < x < 2.692$. The shifting of the kernel also caused it to break the condition of $k(0) = 1$, which meant we had to scale it. This was an anticipated problem as wavelets typically have a total area integral of 0 and therefore have negative regions. The processes of shifting and scaling are reversible so we can guarantee that no information is lost.

The periodic extension of the wavelet kernel also preserved these features. The periodic extension also caused there to be smaller peaks in between the large peaks, however changing some features of a function is almost unavoidable when turning an aperiodic function into a periodic function. The periodic extension was successful in letting us determine the Fourier series of the wavelet kernel.

6.1.2 Determining the Fourier Spectrum

From the first 32 Fourier coefficients determined by the PennyLane library we were able to determine that the first 5 coefficients made up 99.9% of the Fourier spectrum. This tells us that the first 5 terms of the Fourier series dominate and that they contain the majority of the information needed to recreate the periodically extended wavelet kernel.

We elected to run two approximations, one using 3 qubits and one using 5 qubits. The quantum computer run using 5 qubits gave us 32 terms of the Fourier series. The quantum computer with 3 qubits gave the first 8 terms of the Fourier series. By having two different approximations we can compare them and see whether investing more computational resources creates a more accurate approximation. We expect that the 5 qubit approximation would be closer to the wavelet kernel as it has more terms in its Fourier series.

6.1.3 Optimising the Amplitudes

The test amplitudes created which made a kernel that looked like $\frac{1}{|x|}$ were able to be optimised using equation 4.11. After running Newton's method for 200 steps the error norms reached a value of 1.47×10^{-4} in the 5 qubit circuit. This value is much higher than the error norms attained in the demonstration when approximating a Gaussian kernel as seen in figure 5.4c. The 3 qubit quantum circuit reached an error norm of 1.47×10^{-4}, a value very close to that of the 5 qubit circuit. This tells us that when approximating a Gaussian kernel the algorithm was able to find amplitudes which corresponded more closely to the value of the Fourier series than for the wavelet kernel.

The finding of optimal parameters was done using Newton's method. Newton's method can break down when the initial guess is too far away from the root which you are trying to find with it. This can cause the algorithm to continue indefinitely without getting closer to the true solution. However, we have a convex optimisation problem therefore, it is unlikely that this is the case.

It could be useful to try a different optimisation method to find the Fourier coefficients and to compare them to the results achieved using Newton's method. This would allow us to more clearly whether the approximation of the wavelet kernel is only possible to this degree of accuracy using this method. Without trying an different optimisation method we must conclude that this method was not able to determine the Fourier coefficients of the wavelet kernel as well as it could for a Gaussian kernel.

Although the error norms are much higher than that of the Gaussian kernel we can still examine the Fourier coefficients that were found and see if they have been approximated well enough to reconstruct the original wavelet kernel.

6.1.4 Comparing the Fourier Coefficients

Figure 5.5 shows the Fourier coefficients determined from the PennyLane library as well as the predicted Fourier coefficients and the final Fourier coefficients from the quantum computer. In figure 5.5a, showing the results of the 5 qubit circuit, all three sets of coefficients seem to take on the same values and any differences in values are not visible on this scale. The values also appear to be the same at this scale for the 3 qubit circuit. We therefore look at figure 5.6 which shows us the Fourier coefficients on a logarithmic scale for the 5 qubit circuit.

In figure 5.6 it appears that at this scale the first 5 Fourier coefficients still seem to be close together. As we move to the 6^{th} Fourier coefficient and further we see that the difference between the approximated Fourier series and the Fourier series of the wavelet kernel seems to grow. When take the difference between the Fourier coefficients and the approximate Fourier coefficients we see that the difference for between the terms is of the order 10^{-4} for the first four terms and 10^{-5} for the next four terms.

We have plotted the differences between the Fourier series and the approximated Fourier series in figure 5.7. In figure 5.7a, which shows the 5 qubit circuit, the first

three terms of the Fourier series are of the order 10^{-1}. This means that the first three terms have been approximated more accurately than the terms that follow. The fourth and fifth term are of the order 10^{-2} and the fourth to eighth terms are of the order 10^{-5}. This means that the difference between the approximated Fourier coefficients and the Fourier coefficients of the wavelet kernel are not accurate past 6 terms of the Fourier series. The difference between the terms of the two Fourier series beyond this points is too large. The accuracy of the first four terms is not close to our original threshold.

In figure 5.7b we have the 3 qubit circuit. The first four terms have a difference on the order of 10^{-4} and the terms of the Fourier series are on the order of 10^{-4}. The difference in terms is smaller than the difference in terms on the 5 qubit circuit. This implies that the 3 qubit circuit was able to approximate the Fourier series of the wavelet kernel more accurately than the 5 qubit quantum circuit.

The 6th term of the wavelet kernel's Fourier series is -3.110^{-5}. Both the 3 and 5 qubit approximations approximated this value of the Fourier series as a positive number. This seems to be caused by the implementation of equation 4.11. The *MottonenStatePreparation* requires normalised amplitudes to be fed into the function in order to implement the W_a gates. In the function *probabilities_threshold_normalize* all probabilities that are below 1×10^{-10} are set to zero. This removes all the negative Fourier coefficients and coefficients which are very small, and as a consequence does not allow us to have negative Fourier coefficients.

This could be the reason that the error norms for the wavelet kernel approximations are higher. In order to rectify this problem we would likely need to choose a different gate for the W_a section of the circuit. This result also indicates that the family of functions which can be realistically realised using this circuit is more restricted than initially thought. Further testing would need to be done before any further conclusions are drawn from this.

We must now ask if these results will produce an approximated wavelet kernel which is acceptable. We discussed in section 4.2.2 that the leading terms of the Fourier series are the most dominant terms. In this case the leading terms of the Fourier series are more accurately approximated and make up 99% of the series. Before we write the approximation off completely, we must look at the function which the approximated Fourier series produces and see how far off it is from the original wavelet kernel.

6.1.5 Comparing the Approximated Kernel to the Wavelet Kernel

We begin by scaling and then shifting the wavelet kernel to return it from the form seen in equation 5.1 to its form in equation 4.2. We perform the same scaling and shifting to the approximated wavelet kernel to both the 5 and 3 qubit circuits. At this point plotting the two functions give us the image seen in figures 5.8a and 5.8b. At a glance it would appear that the quantum computer was able to approximate the wavelet kernel.

We must now compare the approximated wavelet kernel to the wavelet kernel at each point to see how well the kernel was approximated. We do this by taking the mean squared error of the two functions. A mean squared error of zero would indicate that the two functions were identical at each point, so the closer to zero out mean squared error the closer the functions are pointwise. Our approximation will also be close together as there is no finite sampling noise and no device noises since we are using a simulated, ideal quantum computer.

A mean squared error was calculated using the 100 values used to plot the wavelet kernel and approximated kernel is figure 5.8a. The calculation gives a mean squared error of 1.67×10^{-7} for the 5 qubit circuit. This number is six orders of magnitude smaller than the minimum value of the wavelet kernel. This means that the approximated wavelet kernel we have generated is accurate up to a scale of 10^{-6}. This is a scale which can offer practical use cases.

The 3 qubit quantum circuit produced a mean squared error of 9.09×10^{-8}. This is an order of magnitude smaller than the result produced from the 5 qubit circuit and indicates that it is a more accurate approximation.

We can also use the mean squared error test on the approximated Gaussian kernel for comparison. Here we find that the mean squared error is 4.009×10^{-8}. This value is an order of magnitude smaller than the mean squared error of the 5 qubit approximation of the wavelet kernel and the same magnitude as the 3 qubit approximation.

The mean squared error result reinforces that approximating the leading terms of the Fourier series are more important than approximating the later terms. We expected that the 5 qubit circuit would perform better than the 3 qubit circuit because it had more terms in its Fourier series and should therefore have been a more accurate approximation. It appears that the circuits inability to accurately approximate the later terms of the Fourier series is what caused the 3 qubit approximation to perform better than the 5 qubit one. We expect that if the approximation of the Fourier series could be improved then the 5 qubit approximation would perform better than the 3 qubit one.

In the end, we were successfully able to approximate a wavelet kernel and show how the demonstration on PennyLane could be adapted in order to include a larger family of kernels. Improvements could possibly be made to the quantum circuit in the future to remove the requirement of the kernel being positive at all values of x.

Now that we have successfully approximated the wavelet kernel we can move into the conclusion of this thesis where we summarise the main results of this thesis. We suggest ways that this method could be improved in the future, and how this could possibly lead to an implementation of wavelet analysis on a quantum computer.

Chapter 7

Conclusions

The first question this thesis aimed to tackle was, why it is so hard to find a quantum advantage. We aimed to do so using the lens of kernel methods. We have shed light on how quantum advantage is found within kernel methods in chapter 3. We also reviewed different types of quantum advantage and provided a framework with which we can compare different quantum algorithms and establish the type of quantum advantage which it provides in section 3.1. We showed examples of quantum kernel methods providing quantum speedups over classical algorithms and showed how it can arise from quantum advantages in other problems in section 3.2.

In section 3.3 we made the argument that quantum kernel methods should not be used to search for new forms of provable quantum advantage as the method of search is too inefficient. Using quantum kernel methods to search for provable quantum speed-ups has too many unknown factors. The choice of possible kernels is large and the data which is used to evaluate the kernels is not guaranteed to contain the properties which the kernel can exploit. We argued that quantum advantage in quantum kernel methods should be found by adapting other problems which have solutions with a quantum advantage and using them to speed up kernel methods by mapping the data into a space where it presents features which are not easily accessible classically.

Quantum computers are still in a relatively early stage of their development and as more research is done we are able to learn more about what quantum advantage is, how to characterise it and how to find new instances of it. Hopefully the discussions in this thesis will be able to help future researchers better define the problem of finding quantum advantage and allow them to more efficiently invest their resources.

The next question we looked at was how kernel methods are implemented on near term quantum computers. In section 2.8 we went over how parameterised quantum circuits were used to perform machine learning on near term quantum computers. We then looked at two papers which presented examples of parameterised quantum circuits which used rotational embedding in section 2.9. We then made the argument in section 2.10 that supervised quantum models could be viewed as kernel methods. We showed that each method of embedding classical information into a quantum data could be linked to a specific kernel and therefore could be used as a way to implement kernel methods on near term quantum computers.

The last question this thesis dealt with was whether we could adapt a method used for approximating a Gaussian kernel on a quantum computer to approximate a wavelet kernel. We were able to successfully adapt the code used to approximate a Gaussian kernel to approximate a wavelet kernel. The wavelet kernel had the first 32 terms of its Fourier series approximated with a mean squared error of 1.7×10^{-7}, and was done using a quantum computer with only 5 qubits. We were also able to use a 3 qubit circuit to approximate the first 8 terms of the Fourier series and have approximate the kernel with a mean squared error of 9.09×10^{-8}. The code was implemented in python using the PennyLane library to program the quantum circuits.

Improvements could be made to the methods used to approximate the wavelet kernel. A new circuit should be constructed and tested which uses different parameterised circuit gates. This could possibly avoid the inability of the circuit presented in this thesis to approximate negative Fourier coefficients. These results show that although the approximation of functions on near term quantum computers is possible, the current methods used restrict the family of functions more than initially thought. More research needs to be done before we can draw any concrete conclusions about the full family of functions which can be implemented on near term quantum computers.

7.1 Further implementation of Wavelet Analysis on Quantum Computers

We have successfully shown that a quantum computer can be constructed which approximates a wavelet kernel. It is now natural to ask what we can do with this kernel and how this could be used to perform wavelet analysis using a quantum computer. In section 2.4 we laid out how wavelet analysis works on classical computers, using this we can ask what needs to happen to the wavelet kernel in order to perform the same tasks. The two main equations used in wavelet analysis are equation 2.21 and equation 2.22 which are repeated again below.

$$\Psi_{(a,b)}(t) = \frac{1}{\sqrt{a}} \Psi \left(\frac{t-b}{a} \right)$$

$$W_{\Psi}(f)(a,b) = \left\langle f(t), \Psi_{(a,b)}(t) \right\rangle$$

Equation 2.21 tells us how the mother wavelet transforms and equation 2.22 tells us how to work out the contribution from each wavelet when reconstructing or approximating a function. We can take both these equations and see how they can be implemented using a quantum computer.

7.1.1 Transforming the Mother Wavelet

We need to be able to scale the mother wavelet function and to move it along a domain in order to perform the wavelet analysis. The most intuitive way to do this would be to do this on a classical computer and just change the values of the function as we did in section 5.5. This could be done fairly easily and the parameters could easily be adjusted to create new wavelets with minimal computation. This is how the process is performed on classical computers so we can be fairly certain that it will work.

We can however try to find quantum methods which implement the same idea. For example, we could possibly construct a time evolving Hamiltonian which scales as the wavelet function as it evolves in time. The parameters of the Hamiltonian would have to be adjusted to make sure that the wavelet kernel covered the correct time domain and had peaks at the desired heights. For the circuit we have used in this thesis the Hamiltonian would have to transform the values of the Fourier series in order to create the desired wavelet which is not necessarily a trivial task.

Due to the design of the circuit and its encoding method which requires the function to always be positive and have $k(0) = 1$ it would likely be easier to scale the mother wavelet on a classical computer at this time or design a new circuit which does not have these constraints.

7.1.2 Performing the Inner Product

Performing an inner product is a linear algebra operation, something which quantum computers are usually good at. In section 2.9.2 we showed an example of a quantum computer being used to measure fidelity between two states to measure the inner product. The first set of unitary encoding gates is applied using the data point x_1. The inverse unitary gates are then applied with the data point x_2 and then a measurement is made. If $x_1 = x_2$ the gates should have no effect on the qubit in the end and all the result of the qubit measurement would be the $|0\rangle$ state. If the values of x_1 and x_2 are different then the inverse unitary gates do not get all the qubits back to the ground state and the measurement can be used to measure how close together they are. The closer to zero that the final measurement is the closer the states are to each other.

We know that taking an inner product on a quantum computer is possible, the question now becomes how do we encode an arbitrary function into a feature space where an inner product can be taken between the wavelet kernel and the function. Usually the mapping to a feature space is performed by the encoding gate of a quantum circuit. It should be possible to use the encoding gate, S(x) described in section 4.2.3 which uses Pauli Z rotations to embed data points. Using the methods laid out by Havlicek et al [56]. we could then take the inner product between the arbitrary function and the wavelet kernel.

More research needs to be done to implement wavelet analysis on NISQ quantum computers. There are still many factors that need to be considered such as what the best circuit ansatz would be for wavelet analysis and how a quantum wavelet analysis compares to a classical wavelet analysis. Wavelet analysis is used frequently on modern day classical computers so if we want quantum computers to be at least as good as the classical computers we have today then we need them to be able to perform all such tasks which we use classical computers for.

The approximation of the wavelet kernel on a quantum computer opens up many doors for algorithms we can try and run using the wavelet kernel as the mother wavelet. We disused the ways in which the wavelet kernel presented in this thesis could be adapted in order to perform a wavelet analysis. The wavelet would have to be shifted and scaled in order to become a basis for wavelet analysis and we thought that at this point in time the best way to implement this on near term quantum computers would be to shift and scale

the values of the kernel on a classical computer. It was also thought that a new circuit anzats could be developed which placed less restrictions on the function of the wavelet kernel.

We also discussed how inner products could be performed on a quantum computer and how those could be used along with the wavelet kernel in order to perform the inner products necessary for wavelet analysis described by equation 2.22. The findings and discussions presented in this thesis should provide a good basis on how to start tackling and implementing wavelet transforms on near term quantum computers. This would be an important step in the evolution of quantum computers but more research and testing needs to be done to fully implement and evaluate quantum wavelet transforms.

This thesis highlights the role of kernels and kernel methods in quantum machine learning. We hope that these discussions can be used to better understand the inner workings of quantum machine learning algorithms so that they can be more effectively implemented on near term machines and so that we can develop new methods or algorithms further down the line. We also hope to be able to use this work as a starting block to begin implementing a form of wavelet analysis on quantum computers.

Bibliography

[1] R.R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997. Conference Name: IEEE Spectrum.

[2] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov. Parallel Computing Experiences with CUDA. *IEEE Micro*, 28(4):13–27, July 2008. Conference Name: IEEE Micro.

[3] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, July 2015. Publisher: American Association for the Advancement of Science.

[4] Vladimir Nasteski. An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62, December 2017.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, November 2018. Google-Books-ID: uWV0DwAAQBAJ.

[7] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1981.

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge ; New York, 10th anniversary ed edition, 2010.

[9] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[10] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[11] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[12] Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy Lloyd O'Brien. Quantum computers. *nature*, 464(7285):45–53, 2010.

[13] Carlos A Pérez-Delgado and Pieter Kok. Quantum computers: Definition and implementations. *Physical Review A*, 83(1):012303, 2011.

[14] Tirthak Patel, Abhay Potharaju, Baolin Li, Rohan Basu Roy, and Devesh Tiwari. Experimental evaluation of nisq quantum computers: error measurement, characterization, and implications. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[15] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. arXiv:1801.00862 [cond-mat, physics:quant-ph].

[16] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke, et al. Noisy intermediate-scale quantum (nisq) algorithms. *arXiv preprint arXiv:2101.08448*, 2021.

[17] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[18] Troels F Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V Isakov, David Wecker, John M Martinis, Daniel A Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, 345(6195):420–424, 2014.

[19] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.

[20] Maria Schuld and Francesco Petruccione. *Supervised learning with quantum computers*, volume 17. Springer, 2018.

[21] V. Kecman. Support Vector Machines – An Introduction. In Lipo Wang, editor, *Support Vector Machines: Theory and Applications*, Studies in Fuzziness and Soft Computing, pages 1–47. Springer, Berlin, Heidelberg, 2005.

[22] Bernhard Scholkopf, Christopher Burges, and Alexander Smola. Advances in Kernel Methods - Support Vector Learning. *MIT Press*, December 1998.

[23] DataFlair Team. Kernel Functions-Introduction to SVM Kernel & Examples, August 2017.

[24] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[25] Maria Schuld. Supervised quantum machine learning models are kernel methods, April 2021. arXiv:2101.11020 [quant-ph, stat].

[26] David Morin. *Fourier analysis*. 2009.

[27] Shawhin Talebi. The wavelet transform, September 2022. https://towardsdatascience.com/the-wavelet-transform-e9cfa85d7b34.

[28] Documenter.jl. Available wavelet families. https://docs.juliahub.com/ContinuousWavelets/JbYB7/

[29] Steve Brunton. Wavelets and multiresolution analysis. *Youtube*, June 2020. https://www.youtube.com/watch?v=y7KLbd7n75g.

[30] Noson S Yanofsky. AN INTRODUCTION TO QUANTUM COMPUTING. page 33.

[31] Christopher M. Dawson and Michael A. Nielsen. The solovay-kitaev algorithm, 2005.

[32] Quantum logic gate, September 2022. https://en.wikipedia.org/wiki/Quantum_logic_gate.

[33] David P DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics*, 48(9-11):771–783, 2000.

[34] MD SAJID ANIS, Abby-Mitchell, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Vishnu Ajith, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Anthony-Gandon, Israel F. Araujo, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHAMESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, M. Chandler Bennett, Daniel Bevenius, Dhruv Bhatnagar, Prakhar Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Kartik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, CisterMoke, Christian Claus, Christian Clauss, Caleb Clothier, Romilly Cocking, Ryan Cocuzzo, Jordan Connor, Filipe Correa, Zachary Crockett, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Navaneeth D, Sean Dague, Tareq El Dandachi, Animesh N Dangwal, Jonathan Daniel, Marcus Daniels, Matthieu Dartiailh, Abdón Rodríguez Davila, Faisal Debouni, Anton Dekusar, Amol Deshmukh, Mohit Deshpande, Delton Ding, Jun Doi, Eli M. Dow, Patrick Downing, Eric Drechsler, Marc Sanz Drudis, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Amir Ebrahimi, Pieter Eendebak, Daniel Egger, ElePT, Emilio, Alberto Espiricueta, Mark Everitt, Davide Facoetti, Farida, Paco Martín Fernández, Samuele Ferracin, Davide Ferrari, Axel Hernández Ferrera, Romain Fouilland, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, James R. Garrison, Jim Garrison, Tim Gates, Gian Gentinetta, Hristo Georgiev, Leron Gil, Austin Gilliam, Aditya Giridharan, Glen, Juan Gomez-Mosquera, Gonzalo, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, Dani Guijo, Guillermo-Mijares-Vilarino, John A. Gunnels, Harshit Gupta, Naman Gupta, Jakob M. Günther, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Kevin Hartman, Areeq Hasan, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Colin Hong, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Chih-Han Huang, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Ishwor, Raban Iten, Toshinari Itoko, Alexander Ivrii, Ali Javadi, Ali Javadi-Abhari, Wahaj

Javed, Qian Jianhua, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, John Judge, Tal Kachmann, Akshay Kale, Naoki Kanazawa, Jessica Kane, Kang-Bae, Annanay Kapila, Anton Karazeev, Paul Kassebaum, Tobias Kehrer, Josh Kelso, Scott Kelso, Hugo van Kemenade, Vismai Khanderao, Spencer King, Yuri Kobayashi, Kovi11Day, Arseny Kovyrshin, Rajiv Krishnakumar, Pradeep Krishnamurthy, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, Haggai Landa, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Tan Jun Liang, Jake Lishman, Dennis Liu, Peng Liu, Lolcroc, Abhishek K M, Liam Madden, Yunho Maeng, Saurav Maheshkar, Kahan Majmudar, Aleksei Malyshev, Mohamed El Mandouh, Joshua Manela, Manjula, Jakub Marecek, Manoel Marques, Kunal Marwaha, Dmitri Maslov, Paweł Maszota, Dolph Mathews, Atsushi Matsuo, Farai Mazhandu, Doug McClure, Maureen McElaney, Joseph McElroy, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Dekel Meirom, Corey Mendell, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Declan Millar, Daniel Miller, Hannah Miller, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Alejandro Montanez, Gabriel Monteiro, Michael Duane Mooring, Renier Morales, Niall Moran, David Morcuende, Seif Mostafa, Mario Motta, Romain Moyard, Prakash Murali, Daiki Murata, Jan Müggenburg, Tristan NEMOZ, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Aziz Ngoueya, Thien Nguyen, Johan Nicander, Nick-Singstock, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Tamiya Onodera, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Ashish Panigrahi, Vincent R. Pascuzzi, Simone Perriello, Eric Peterson, Anna Phan, Kuba Pilch, Francesco Piro, Marco Pistoia, Christophe Piveteau, Julia Plewa, Pierre Pocreau, Clemens Possel, Alejandro Pozas-Kerstjens, Rafał Pracht, Milos Prokop, Viktor Prutyanov, Sumit Puri, Daniel Puzzuoli, Pythonix, Jesús Pérez, Quant02, Quintiii, Rafey Iqbal Rahman, Arun Raja, Roshan Rajeev, Isha Rajput, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Oliver Reardon-Smith, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Matt Riedemann, Rietesh, Drew Risinger, Pedro Rivero, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Ben Rosand, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, Nahum Rosa Cruz Sa, Arijit Saha, Abdullah Ash-Saki, Sankalp Sanand, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Niko Savola, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Mark Schulterbrandt, Joachim Schwarm, James Seaward, Sergi, Ismael Faro Sertage, Kanav Setia, Freya Shah, Nathan Shammah, Will Shanks, Rohan Sharma, Polly Shaw, Yunong Shi, Jonathan Shoemaker, Adenilton Silva, Andrea Simonetto, Deeksha Singh, Divyanshu Singh, Parmeet Singh, Phattharaporn Singkanipa, Yukio Siraichi, Siri, Jesús Sistos, Iskandar Sitdikov, Seyon Sivarajah, Slavikmew, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, Vicente P. Soloviev, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Adrien Suau, Shaojun Sun, Kevin J. Sung, Makoto Suwama, Oskar Słowik, Rohit Taeja, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Kevin Tian, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Caroline Tornow, Enrique de la Torre, Juan Luis Sánchez Toural, Kenso Trabing, Matthew Treinish, Dimitar Trenev, Tr-

ishaPe, Felix Truger, Georgios Tsilimigkounakis, Davindra Tulsi, Doğukan Tuna, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Adish Vartak, Almudena Carrera Vazquez, Prajjwal Vijaywargiya, Victor Villar, Bhargav Vishnu, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, WinterSoldier, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Steve Wood, James Wootton, Matt Wright, Lucy Xing, Jintao YU, Yaiza, Bo Yang, Unchun Yang, Jimmy Yao, Daniyar Yeralin, Ryota Yonekura, David Yonge-Mallo, Ryuhei Yoshida, Richard Young, Jessie Yu, Lebin Yu, Yuma-Nakamura, Christopher Zachow, Laura Zdanski, Helena Zhang, Iulia Zidaru, Bastian Zimmermann, Christa Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, chetmurthy, choerst ibm, dalin27, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, ewinston, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hhyap, hykavitha, itoko, jeppevinkel, jessica angel7, jezerjojo14, jliu45, johannesgreiner, jscott2, kUmezawa, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico lgrs, nrhawkins, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, rithikaadiga, ryancocuzzo, saktar unr, saswati qiskit, septembrr, sethmerkel, sg495, shaashwat, smturro2, sternparky, strickroman, tigerjack, tsura crisaldo, upsideon, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, yuri@FreeBSD, and Mantas Čepulkovskis. Qiskit: An open-source framework for quantum computing, 2021.

[35] Unathi Skosana and Mark Tame. Demonstration of shor's factoring algorithm for N = 21 on ibm quantum processors. *Scientific Reports*, 11(1):16599, August 2021. Number: 1 Publisher: Nature Publishing Group.

[36] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9):1013–1017, September 2021. arXiv:2010.02174 [quant-ph].

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

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

[39] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[40] Fernando G. S. L. Brandão, Amir Kalev, Tongyang Li, Cedric Yen-Yu Lin, Krysta M. Svore, and Xiaodi Wu. Quantum SDP Solvers: Large Speed-Ups, Optimality, and Applications to Quantum Learning. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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

[42] Iris Cong and Luming Duan. Quantum discriminant analysis for dimensionality reduction and classification. *New Journal of Physics*, 18(7):073011, 2016.

[43] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[44] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du. Experimental realization of a quantum support vector machine. *Physical review letters*, 114(14):140504, 2015.

[45] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature communications*, 12(1):1–9, 2021.

[46] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum Circuit Learning. *Physical Review A*, 98(3):032309, September 2018. arXiv:1803.00745 [quant-ph].

[47] Bikram Khanal, Pablo Rivas, Javier Orduz, and Alibek Zhakubayev. Quantum machine learning: A case study of grover's algorithm. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 79–84. IEEE, 2021.

[48] Giuseppe E Santoro and Erio Tosatti. Optimization using quantum mechanics: quantum annealing through adiabatic evolution. *Journal of Physics A: Mathematical and General*, 39(36):R393, 2006.

[49] Itay Hen and Federico M Spedalieri. Quantum annealing for constrained optimization. *Physical Review Applied*, 5(3):034007, 2016.

[50] Arnab Das and Bikas K Chakrabarti. *Quantum annealing and related optimization methods*, volume 679. Springer Science & Business Media, 2005.

[51] Pennylane, September 2022. https://pennylane.ai/.

[52] Qiskit, September 2022. https://qiskit.org/.

[53] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[54] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[55] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[56] Vojtech Havlicek, 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, March 2019. arXiv:1804.11326 [quant-ph, stat].

[57] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. The effect of data encoding on the expressive power of variational quantum machine learning models. *Physical Review A*, 103(3):032430, March 2021. arXiv:2008.08605 [quant-ph, stat].

[58] Aram W Harrow and John C Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. *Physical Review Letters*, 126(14):140502, 2021.

[59] Maria Schuld, Alex Bocharov, Krysta M Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3):032308, 2020.

[60] Carsten Blank, Daniel K Park, June-Koo Kevin Rhee, and Francesco Petruccione. Quantum classifier with tailored quantum kernel. *npj Quantum Information*, 6(1):1–7, 2020.

[61] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020.

[62] Yong Liu, Dongyang Wang, Shichuan Xue, Anqi Huang, Xiang Fu, Xiaogang Qiang, Ping Xu, He-Liang Huang, Mingtang Deng, Chu Guo, et al. Variational quantum circuits for quantum state tomography. *Physical Review A*, 101(5):052316, 2020.

[63] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.

[64] Rupak Chatterjee and Ting Yu. Generalized coherent states, reproducing kernels, and quantum support vector machines. *arXiv preprint arXiv:1612.03713*, 2016.

[65] Guillaume Verdon, Trevor McCourt, Enxhell Luzhnica, Vikash Singh, Stefan Leichenauer, and Jack Hidary. Quantum graph neural networks. *arXiv preprint arXiv:1909.12264*, 2019.

[66] Reza Haghshenas, Johnnie Gray, Andrew C Potter, and Garnet Kin-Lic Chan. Variational power of quantum circuit tensor networks. *Physical Review X*, 12(1):011047, 2022.

[67] Pierre-Luc Dallaire-Demers and Nathan Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1):012324, 2018.

[68] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, December 2018.

[69] Junyu Liu, Zexi Lin, and Liang Jiang. Laziness, barren plateau, and noise in machine learning. *arXiv preprint arXiv:2206.09313*, 2022.

[70] M Cerezo, A Sone, T Volkoff, L Cincio, and PJ Coles. Cost-function-dependent barren plateaus in shallow quantum neural networks (2020). *arXiv preprint arXiv:2001.00550*, 2001.

[71] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is np-hard. *Physical review letters*, 127(12):120502, 2021.

[72] S Shin, YS Teo, and H Jeong. Exponential data encoding for quantum supervised learning. *Physical Review A*, 107(1):012422, 2023.

[73] Diego Riste, Marcus P Da Silva, Colm A Ryan, Andrew W Cross, Antonio D Córcoles, John A Smolin, Jay M Gambetta, Jerry M Chow, and Blake R Johnson. Demonstration of quantum advantage in machine learning. *npj Quantum Information*, 3(1):16, 2017.

[74] Andrew J Daley, Immanuel Bloch, Christian Kokail, Stuart Flannigan, Natalie Pearson, Matthias Troyer, and Peter Zoller. Practical quantum advantage in quantum simulation. *Nature*, 607(7920):667–676, 2022.

[75] Rolando D Somma, Daniel Nagaj, and Mária Kieferová. Quantum speedup by quantum annealing. *Physical review letters*, 109(5):050501, 2012.

[76] Tameem Albash and Daniel A Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1):015002, 2018.

[77] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

[78] Jennifer R Glick, Tanvi P Gujarati, Antonio D Corcoles, Youngseok Kim, Abhinav Kandala, Jay M Gambetta, and Kristan Temme. Covariant quantum kernels for data with group structure. *arXiv preprint arXiv:2105.03406*, 2021.

[79] Carlos Outeiral, Garrett M Morris, Jiye Shi, Martin Strahm, Simon C Benjamin, and Charlotte M Deane. Investigating the potential for a limited quantum speedup on protein lattice problems. *New Journal of Physics*, 23(10):103030, 2021.

[80] Salvatore Mandra and Helmut G Katzgraber. A deceptive step towards quantum speedup detection. *Quantum Science and Technology*, 3(4):04LT01, 2018.

[81] Neal Koblitz and Alfred Menezes. Intractable problems in cryptography. In *Proceedings of the 9th Conference on Finite Fields and Their Applications. Contemporary Mathematics*, volume 518, pages 279–300, 2010.

[82] Martin Ekerå. Modifying shor's algorithm to compute short discrete logarithms. *Cryptology ePrint Archive*, 2016.

[83] Christoph Clausen, Imam Usmani, Félix Bussieres, Nicolas Sangouard, Mikael Afzelius, Hugues de Riedmatten, and Nicolas Gisin. Quantum storage of photonic entanglement in a crystal. *Nature*, 469(7331):508–511, 2011.

[84] Ioana Craiciu, Mi Lei, Jake Rochman, Jonathan M Kindem, John G Bartholomew, Evan Miyazono, Tian Zhong, Neil Sinclair, and Andrei Faraon. Nanophotonic quantum storage at telecommunication wavelength. *Physical Review Applied*, 12(2):024062, 2019.

[85] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. Quantum advantage in learning from experiments. *Science*, 376(6598):1182–1186, 2022.

[86] How to approximate a classical kernel with a quantum computer, March 2022. https://pennylane.ai/qml/demos/tutorial_classical_kernels.html.

[87] Li Zhang, Weida Zhou, and Licheng Jiao. Wavelet support vector machine. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):34–39, 2004.

[88] Hai-Sheng Li, Ping Fan, Hai-ying Xia, and Shuxiang Song. Quantum multi-level wavelet transforms. *Information Sciences*, 504:113–135, 2019.

[89] Amir Fijany and Colin P Williams. Quantum wavelet transforms: Fast algorithms and complete circuits. In *Quantum Computing and Quantum Communications: First NASA International Conference, QCQC'98 Palm Springs, California, USA February 17–20, 1998 Selected Papers*, pages 10–33. Springer, 1999.

[90] Octavio Loyola-Gonzalez. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE access*, 7:154096–154113, 2019.

[91] Jorge J Moré and Danny C Sorensen. Newton's method. Technical report, Argonne National Lab., IL (USA), 1982.

Appendix A

Source Code

Below is the code used in order to approximate the wavelet kernel on with a quantum computer. Most of the code comes from the PennyLane tutorial and alterations have been made. The code is written in python3, to run it you will need the numpy and PennyLane packages installed

```
import pennylane as qml
from pennylane import numpy as np
import matplotlib.pyplot as plt
import math
from pennylane.fourier import coefficients
import pennylane.fourier.visualize as vis
import scipy as scipy
np.random.seed(53173)

plt.rcParams['figure.dpi'] = 300

#return the function for a wavelet kernel shifted upwards to be positive
def wavelet_kernel(x):
    return (math.cos(1.75*x)*math.exp(-0.5*x ** 2)
            +0.28879029981267096)/1.2867461305590557

# makes array of x values and then sends to mother wave function for y values
def make_data(n_samples, lower=-np.pi, higher=np.pi):
    x = np.linspace(lower, higher, n_samples)
    y = np.array([wavelet_kernel(x_) for x_ in x])
    return x,y

# create plot of wavelet function
X, Y_wavelet = make_data(100)

plt.plot(X, Y_wavelet,color='g')
plt.xlabel("$x$")
plt.ylabel("$k(x)$")
#plt.savefig("wavelet kernel.png",dpi=300)
plt.show();
print(max(Y_wavelet))
```

```
#periodically extend the wavelet
def periodic_wavelet(x, L=np.pi):
    # Send x to x_mod in the period around 0
    x_mod = np.mod(x+L, 2*L) - L
    return wavelet_kernel(x_mod)

x_func = np.linspace(-30, 30, 961)
y_func = [periodic_wavelet(x) for x in x_func]

plt.plot(x_func, y_func,color="g")
plt.xlabel("$x$")
plt.ylabel("k(x)")
#plt.savefig("periodic wavelet.png",dpi=300)
plt.show();

#fourier coefficients of the wavelet
def fourier_p(d):
    return np.real(coefficients(periodic_wavelet, 1, d-1)[:d])

N = [0]
for n in range(2,10):
    N.append(n)
    F = fourier_p(n)
    plt.plot(N, F, 'x', label='{}'.format(n))

plt.legend()
plt.xlabel("frequency $n$")
plt.ylabel("Fourier coefficient $c_n$")
plt.show();

plt.plot(range(8), fourier_p(8), 'x')
coeff= fourier_p(32)
plt.xlabel("frequency $n$")
plt.ylabel("Fourier coefficient $c_n$")
#plt.suptitle("Fourier spectrum of the wavelet kernel")
#plt.savefig("Fourier coeff wavelet.png",dpi=300)
plt.show();
total = 0
for i in range(8):
    total = total + coeff[i]
total5 = 0
for j in range(5):
    total5+=coeff[j]

percentage = (total5/total)*100
print("Percentage of Fourier spectrum contained in first 5 coefficients:",
        percentage,"%")
```

```
"""Quantum circuit construction"""
# apply a Pauli Z rotation to each wiire embedding the value x
def S(x, thetas, wires):
    for (i, wire) in enumerate(wires):
        qml.RZ(thetas[i] * x, wires = [wire])

# set thetas to get an interger valued spectrum
def make_thetas(n_wires):
    return [2 ** i for i in range(n_wires-1, -1, -1)]

def W(features, wires):
    qml.templates.state_preparations.MottonenStatePreparation(features, wires)

def ansatz(x1, x2, thetas, amplitudes, wires):
    W(amplitudes, wires)
    S(x1 - x2, thetas, wires)
    qml.adjoint(W)(amplitudes, wires)

# Set the number of qubits
#use for different number of qubits
#n_wires = 2
n_wires = 3
dev = qml.device("default.qubit", wires = n_wires, shots = None)
@qml.qnode(dev)

def QK_circuit(x1, x2, thetas, amplitudes):
    ansatz(x1, x2, thetas, amplitudes, wires = range(n_wires))
    return qml.probs(wires = range(n_wires))

def QK_2(x1, x2, thetas, amplitudes):
    return QK_circuit(x1, x2, thetas, amplitudes)[0]

def QK(delta, thetas, amplitudes):
    return QK_2(delta, 0, thetas, amplitudes)

def QK_on_dataset(deltas, thetas, amplitudes):
    y = np.array([QK(delta, thetas, amplitudes) for delta in deltas])
    return y

thetas = make_thetas(n_wires)
test_features = np.asarray([1./(1+i) for i in range(2 ** n_wires)])
test_amplitudes = test_features/ np.sqrt(np.sum(test_features ** 2))

Y_test = QK_on_dataset(X, thetas, test_amplitudes)

plt.plot(X, Y_test,color="g")
plt.xlabel("$x$")
```

```
plt.ylabel("$k(x)$")
#plt.savefig("test amp.png",dpi=300)
plt.show();


#Predeict the fourier spectrum instead of just randomly searching for it
def predict_spectrum(probabilities):
    d = len(probabilities)
    spectrum = []
    for s in range(d):
        s_ = 0

        for j in range(s, d):
            s_ += probabilities[j] * probabilities[j - s]

        spectrum.append(s_)

    # This is to make the output have the same format as
    # the output of pennylane.fourier.coefficients
    for s in range(1,d):
        spectrum.append(spectrum[d - s])

    return spectrum

def F(probabilities, spectrum):
    d = len(probabilities)
    return predict_spectrum(probabilities)[:d] - spectrum[:d]

def J_F(probabilities):
    d = len(probabilities)
    J = np.zeros(shape=(d,d))
    for i in range(d):
        for j in range(d):
            if (i + j < d):
                J[i][j] += probabilities[i + j]
            if(i - j <= 0):
                J[i][j] += probabilities[j - i]
    return J

def make_initial_probabilities(d):
    probabilities = np.ones(d)
    deg = np.array(range(1, d + 1))
    probabilities = probabilities / deg
    return probabilities

probabilities = make_initial_probabilities(2 ** n_wires)
spectrum = fourier_p(2 ** n_wires)
d = 2 ** n_wires
```

```
max_steps = 200
tol = 1.e-20
errors= np.zeros(200)



for step in range(max_steps):
    inc = np.linalg.solve(J_F(probabilities), -F(probabilities, spectrum))
    errors[step]=np.linalg.norm(F(probabilities,spectrum))
    probabilities = probabilities + inc
    if (step+1) % 10 == 0:
        print("Error norm at step {0:3}: {1}".format(step + 1,
                                                     np.linalg.norm(F(probabilities,
                                                                      spectrum))))
        if np.linalg.norm(F(probabilities, spectrum)) < tol:
            print("Tolerance trespassed! This is the end.")
            break


from matplotlib.ticker import FuncFormatter

plt.yscale('log')
plt.plot(errors,color="g")
plt.xlabel('step')
plt.ylabel('Error norm')
#plt.savefig("error norms.png",dpi=300)

plt.plot(range(d), probabilities, 'x')
plt.xlabel("array entry $j$")
plt.ylabel("probabilities $p_j$")
plt.show();


def probabilities_threshold_normalize(probabilities, thresh = 1.e-10):
    d = len(probabilities)
    p_t = probabilities.copy()
    for i in range(d):
        if np.abs(probabilities[i] < thresh):
            p_t[i] = 0.0

    p_t = p_t / np.sum(p_t)

    return p_t

probabilities = probabilities_threshold_normalize(probabilities)
amplitudes = np.sqrt(probabilities)

plt.plot(range(d), probabilities, '+', label = "probability $p_j = |a_j|^2$")
```

```
plt.plot(range(d), amplitudes, 'x', label = "amplitude $a_j$")
plt.xlabel("array entry $j$")
plt.legend()
#plt.savefig("test amp.png",dpi=1200)
plt.show();

plt.plot(range(d), fourier_p(d)[:d], '+', label = "Wavelet kernel")
plt.plot(range(d), predict_spectrum(probabilities)[:d], 'x', label = "QK predicted")
plt.xlabel("frequency $n$")
plt.ylabel("Fourier coefficient $c_n$")
#plt.suptitle("Fourier spectrum of the wavelet kernel and predicted spectrum")
plt.legend()
#plt.savefig("predicted spectrum.png",dpi=1200)
plt.show();

def fourier_q(d, thetas, amplitudes):
    return np.real(coefficients(lambda x: QK(x, thetas, amplitudes), 1, d-1))


plt.plot(range(d), fourier_p(d)[:d], '+', label = "Wavelet kernel")
plt.plot(range(d), predict_spectrum(probabilities)[:d], 'x', label="QK predicted")
plt.plot(range(d), fourier_q(d, thetas, amplitudes)[:d], '.', label = "QK computer")
plt.xlabel("frequency $n$")
plt.ylabel("Fourier coefficient $c_n$")
#plt.suptitle("Fourier spectrums of the wavelet kernel")
plt.legend()
#plt.savefig("qk spectrum.png",dpi=300)
plt.show();


print("Fourier Coefficients \n",fourier_p(8)[:8])
print("Approximated Coefficients \n", fourier_q(8, thetas, amplitudes)[:8])

Fourier7 = fourier_p(8)[:8]
QWavelet7 = fourier_q(8, thetas, amplitudes)[:8]

differences = np.abs(QWavelet7-Fourier7)

plt.yscale('log')

plt.xlabel("frequency $n$")
plt.ylabel("Fourier coefficient $c_n$")
plt.plot(range(8),Fourier7, '+', label = "Wavelet kernel")
plt.plot(range(8),QWavelet7,'.', label = "QK computer",color="g")
plt.legend()
#plt.savefig("first 7 fourier.png",dpi=300)
#plt.plot(range(7),differences)
print("Differences \n",differences)
```

```
Y_learned = QK_on_dataset(X, thetas, amplitudes)
Y_truth = [periodic_wavelet(x_) for x_ in X]

plt.plot(X, (Y_learned), '-.', label = "Quantum Kernel",color="k")
plt.plot(X, Y_truth, '--', label = "Wavelet kernel",color="g")
plt.xlabel("$x$")
plt.ylabel("$k(x)$")
plt.legend()
#plt.savefig("unscaled qk.png",dpi=300)
plt.show();
print(max(Y_truth))
print(max(Y_learned))

#shifts and scales
#Shift wavelet kernel back down

Y_truth = [periodic_wavelet(x_) for x_ in X]

Y_truth = [((periodic_wavelet(x_))*(1.28879029981267096))
            -0.28879029981267096 for x_ in X]
Y_learned = (Y_learned*1.29089)-0.28879029981267096

print(max(Y_truth))
print(max(Y_learned))


plt.plot(X, (Y_learned), '-.', label = "Quantum Kernel",color="k")
plt.plot(X, Y_truth, '--', label = "Wavelet kernel",color="g")
plt.xlabel("$x$")
plt.ylabel("$k(x)$")
plt.legend()
#plt.savefig("scaled qk.png",dpi=300)
plt.show();


sum=0
for i in range(100):
    diff = Y_truth[i]-Y_learned[i]
    sum+=(diff)**2

MSE = sum*(1/100)
print(MSE)

plt.ylabel("Difference in Fourier coefficients")
plt.xlabel("Fourier coefficients term")
plt.yscale('log')
```

```
plt.scatter(range(8),differences,color="g")
```