



TECHNISCHE UNIVERSITÄT BERLIN

Analyzing Hyperspectral EO-Images with Quantum Computers

vorgelegt von

Antonius Benedikt Anani Scherer

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

MASTER OF SCIENCE

genehmigte Masterarbeit

Prüfende: Prof. Dr.-Ing. Olaf Hellwich, TU Berlin
Prof. Dr. Begüm Demir, TU Berlin
Dr. Tobias Guggemos, Deutsches Zentrum für Luft- und Raumfahrt

Berlin 08.09.2022

Abstract

With the advancement of machine learning, the analysis of earth observation data gained popularity in recent years. Especially the particularly difficult task of classifying hyperspectral images showed significant improvements. A hyperspectral image is a 3-dimensional data cube, where two dimensions have spatial information and one dimension has spectral information about the electromagnetic wavelengths. Its high information density allows for a much more detailed analysis of the sensed data compared to RGB images. Through classification, we assign hyperspectral images' pixels to a set of classes and can identify materials or objects of interest. In this thesis, we explore the application of a novel form of Machine Learning, Quantum Machine Learning, on the Pavia University hyperspectral image dataset. Quantum Machine Learning uses quantum algorithms to either speed up existing machine learning subroutines or to develop methods that have a higher capacity and expressibility than classical machine learning methods. We explore several forms of classical image representations on a quantum computer and design three quantum machine learning algorithms to perform pixel-wise hyperspectral image classification. Two of the three methods use quantum kernels to implement a quantum version of the support vector machine and the third is a quantum neural network. We implement these methods using the PennyLane framework and show that they classify with similar accuracy as a classical support vector machine benchmark. Since we use quantum simulators, the results have no immediate implications on the models' performance but rather demonstrate that they are feasible.

Zusammenfassung

Mit den Fortschritten des maschinellen Lernens hat die Analyse von Erdbeobachtungsdaten in den letzten Jahren an Popularität gewonnen. Vor allem bei der besonders schwierigen Aufgabe der Klassifizierung von Hyperspektralbildern wurden deutliche Verbesserungen erzielt. Ein Hyperspektralbild ist ein dreidimensionaler Datenwürfel, bei dem zwei Dimensionen räumliche Informationen und eine Dimension spektrale Informationen in Form von elektromagnetischen Wellenlängen enthalten. Seine hohe Informationsdichte ermöglicht im Vergleich zu RGB-Bildern eine viel detailliertere Analyse der erfassten Daten. Durch Klassifizierung ordnen wir die Pixel von Hyperspektralbildern einer Reihe von Klassen zu und können so Materialien oder Objekte von Interesse identifizieren. In dieser Arbeit untersuchen wir die Anwendung einer neuartigen Form des maschinellen Lernens, des Quanten-Maschinellenlernens, auf den Hyperspektralbilddatensatz der Universität Pavia. Quantum Machine Learning nutzt Quantenalgorithmen, um entweder bestehende Subroutinen des maschinellen Lernens zu beschleunigen oder um Methoden zu entwickeln, die eine höhere Kapazität und Ausdrucksfähigkeit haben als klassische maschinelle Lernmethoden. Wir erforschen verschiedene Formen klassischer Bilddarstellungen auf einem Quantencomputer und entwickeln drei Quantenalgorithmen für maschinelles Lernen, um eine pixelweise Klassifizierung hyperspektraler Bilder durchzuführen. Zwei der drei Methoden verwenden Quantenkernel, um eine Quantenversion der Support-Vektor-Maschine zu implementieren, und die dritte ist ein neuronales Quantennetzwerk. Wir implementieren diese Methoden mit Hilfe des PennyLane-Frameworks und zeigen, dass sie mit ähnlicher Genauigkeit klassifizieren wie eine klassische Support-Vektor-Maschine als Benchmark-Methode. Da wir Quantensimulatoren verwenden, haben die Ergebnisse keine unmittelbaren Auswirkungen auf die Leistung der Modelle, sondern zeigen vielmehr, dass sie funktionieren.

Ich versichere an Eides statt, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle wörtlich oder inhaltlich übernommenen Stellen habe ich als solche gekennzeichnet. Die vorliegende Arbeit wurde weder in der vorliegenden noch einer modifizierten Fassung einer dritten, in- oder ausländischen Fakultät als Prüfungsleistung, oder zum Erlangen eines akademischen Grades vorgelegt.

08.09.2022

Datum

Antonius Benedikt Anani Scherer



Contents

1	Introduction	1
2	Application Domain	3
2.1	Remote Sensing	3
2.2	Hyperspectral Image	3
2.3	Machine Learning	5
2.3.1	Machine Learning Methods	6
2.3.2	Applications	8
3	Background	9
3.1	Elements of Quantum Theory	9
3.2	Elements of Quantum Computation	11
3.2.1	Quantum states and observables	12
3.2.2	Transforming Quantum States and Measurement	15
3.2.3	Quantum Processing Unit	18
3.3	Quantum Embedding of Data	19
3.3.1	Classical and Quantum Data	19
3.3.2	Universal Embedding Techniques	19
3.4	Quantum Computational Complexity	22
4	Quantum Machine Learning	25
4.1	Variational Quantum Algorithms	25
4.2	Quantum Kernel methods	28
4.2.1	Theoretical Foundation of Kernel Methods	28
4.2.2	Quantum Kernels	32
4.3	Quantum Image Representation	33
5	Methodology	37
5.1	Quantum Neural Networks	37

5.2	Quantum Support Vector Machine	43
6	Implementation	51
6.1	Pavia University Dataset	51
6.1.1	Dimensionality Reduction	51
6.1.2	Framework	53
6.2	Spectral classification methods	54
6.3	Experiments	55
7	Conclusion	57
	Bibliography	59
	List of Figures	65
	List of Tables	68

Chapter 1

Introduction

The search for patterns in data has long gone hand in hand with technological advances. Although scientists and especially mathematicians have developed methods to describe data long before computational devices were discovered, recent years have shown stepwise improvements in the development of such methods stemming from advances in hardware. Shortly after von Neumann and Turing designed and built the first computer, the field of Machine Learning was born [1], [2], [3], [4]. Machine Learning, a subfield of Artificial Intelligence, is the study of algorithms that allow programs to automatically improve through experience and by analyzing patterns in data [5]. In the last 70 years, the hardware development followed Moore's law and the field of Machine Learning caught up with increasing computational power. The same happened in the beginning of quantum computing, where machine learning was already seen as one of the potential applications. Especially the theoretical work of Seth Lloyd laid down the fundamentals of what is now known as the first wave of quantum computing [6], [7], [8]. During that time the theorists focused on methods that would speed up existing machine learning subroutines. A notable example is the quantum basic linear algebra subroutines (QBLAS) as proposed in [8]. QBLAS would speed up solving linear equations, finding eigenvalues and -vectors, or performing Fourier transforms up to exponentially faster compared to their classical counterparts. While it delivers a quantum speed-up theoretically, it heavily depends on large fault-tolerant devices and hardware components such as QRAM, which by the time of writing is beyond the horizon.¹ Thus the devices available today are too small and too noisy. So one of the leading figures in the field, John Preskill, coined them 'noisy intermediate scale quantum' devices, short NISQ, [9] and opened up the field to develop specific NISQ era algorithms. Recently, IBM, Google, and other firms developed NISQ-era devices and platforms, enabling researchers to develop and test their ideas. Quantum computing reached a peak when [10]

¹Fault-tolerant means they either work with a fidelity that ensures secure, lossless computation or come with error-correcting algorithms. QRAM is a not yet realized concept of random access memory of quantum data.

presented the first experiment that showed a quantum advantage. Guided by this success many fields like finance [11], chemistry [12], and image recognition [13] are now being explored for potential quantum machine learning use-cases. The classification of remotely sensed data in Earth Observation is an especially interesting use case for quantum machine learning [14] [15]. Particularly a subset of EO-data, hyperspectral images. They can be viewed as a 3D data cube, where two dimensions have spatial information and one dimension has spectral information in the form of intensity values for given electromagnetic wavelengths'. Hyperspectral images have a particularly high spectral resolution which can be used to precisely classify the composition of an object and developing accurate classification models for hyperspectral images would have a huge impact on many sciences [16].

Scope of thesis In this thesis, we explore several methods to represent classical hyperspectral images on a quantum computer and design three NISQ-era algorithms for pixel-wise classification. Two of them will use quantum kernel methods and a classical support vector machine, while the third is a quantum neural network. We implement them using the PennyLane framework, which we also optimize for our use case. The goal of this thesis is to show that quantum kernel methods and a quantum neural network are indeed implementable for hyperspectral data and that classification of a benchmark dataset delivers comparable results to basic classical machine learning methods. Since we use quantum simulators to run our circuits, we are not interested in the exact accuracy of our classification and any potential implications for a quantum advantage.

Structure of the thesis We will first explain the application domain, namely image processing for remote sensing and hyperspectral images, in Chapter 1. We then lay out the fundamentals of quantum computing in Chapter 2, followed by an introduction to quantum machine learning and its application within image processing in Chapter 3. Chapter 4 describes the core methods we use to perform pixel-wise classification for hyperspectral images and Chapter 5 shows their implementation. Chapter 6 summarizes the thesis and gives an outlook on future research in the field.

Chapter 2

Application Domain

The following chapter defines the application domain of this thesis. Since we classify remotely sensed hyperspectral images with quantum computing, we first define the term remote sensing, explain hyperspectral images and then give some background on classical machine learning.

2.1 Remote Sensing

The data acquisition of an object without making physical contact is known as remote sensing. It is mostly used to gather information about our and other planets. Remotely sensed data, which is usually a measurement of certain electromagnetic waves, is acquired through either space- or aircraft. Here, we distinguish between active and passive methods, as seen in Figure 2.1. While passive remote sensors solely rely on measuring electromagnetic waves emitted or reflected from an object, active sensors have their own source of energy to illuminate the object. A common camera for example is a good example of a passive sensor, while radar is a typical active sensor method. In general, there is a vast variety of remote sensors. It ranges from active sensors like LiDar, synthetic aperture radar or radar to passive sensors which mostly concern classical imaging, multi- or hyperspectral imaging. With recent advances in optical sensor technologies, hyperspectral imaging has become more and more important. Its application ranges from land cover classification to medical imaging, providing more information which also leads to new challenges.

2.2 Hyperspectral Image

A hyperspectral image is a 3D data cube that holds 2-dimensional spatial information and 1-dimensional electromagnetic spectral information as pictured in Figure 2.2a. To produce the image, the respective sensor measures the intensity of the radiant flux. More precisely, it



Figure 2.1. (a) Passive sensors detect the reflection of the sunlight from the object. (b) Active sensors have their own source of energy to illuminate the object.

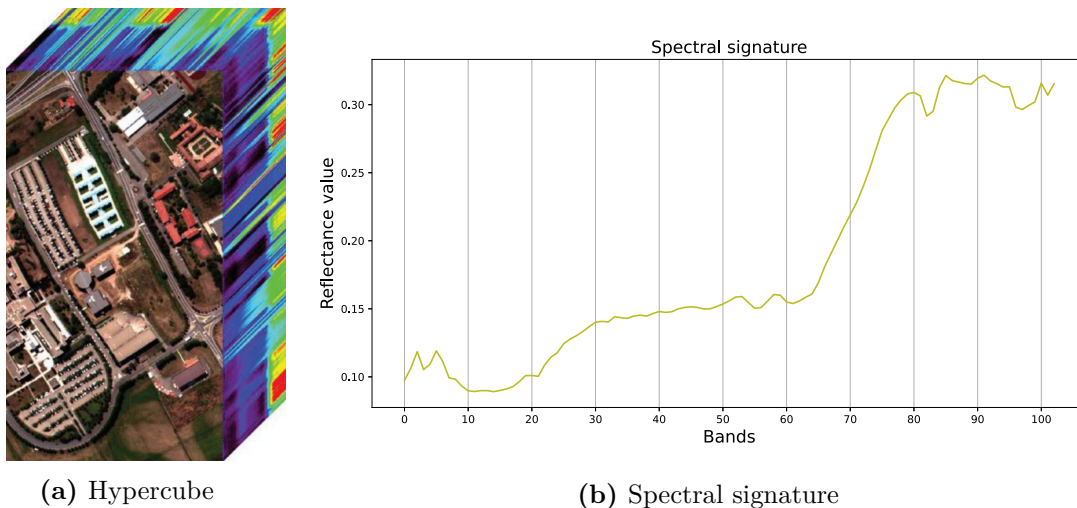


Figure 2.2. (a) 3-dimensional hypercube representation of the Pavia University dataset showing the RGB image and its spectra (b) The spectral signature shows the normed reflectance intensities of the respective bands. This signature was randomly sampled from the class meadows within the Pavia University dataset.

measures the emitted and reflected light (electromagnetic waves) of the object or surface in the pixel and belongs therefore to the passive remote sensing devices. The captured size of a pixel on the surface is called the spatial resolution, while the captured wavelength and the number of bands are considered the spectral resolution. Each holds continuous spectral bands, in contrast to multispectral or RGB imaging that measures spaced spectral bands. An example is given in Figure 2.2b. Since certain materials have specific spectral signatures, a hyperspectral image potentially carries information about the composition of each pixel. In hyperspectral image classification, we try to assign each pixel either a single class or a composition of classes. So in contrast to classical image classification, we do not look at the whole image, but rather at single pixels and find patterns. Hence the often used term pixel-wise classification. Nevertheless, the

spatial component of a hyperspectral image is getting more and more important, especially in the realm of convolutional neural networks. Even though the dimensions in the hyperspectral hypercube do not have the same physical displacements, we can perform a linear operation on a 3-dimensional subset. This is because all values within the hypercube are displayed by the same values [16]. Like most remotely sensed data, one also needs to deal with atmospheric disturbances and similar noise during the preprocessing phase. Fortunately, at least for the benchmark dataset we are using, this is already done and not part of this thesis.

From an image processing perspective, hyperspectral images are very interesting due to their high content of information. This allows for totally new applications, but also poses real challenges for nowadays technology. Especially the high dimensionality of the data requires a huge computational effort, but also spectral variability, mixing pixels and unbalanced or limited training data [17]. "Hughes phenomenon" or the curse of dimensionality are very good examples of a problem strongly connected to hyperspectral images [18]. It describes the decrease in classification accuracy when training data is scarce and the number of feature dimensions increases. Before the rise of deep learning and the subsequent success of convolutional neural networks [16], kernel methods [19], support vector machines [20] or bayesian methods [21] were models of choice to perform pixel-wise classification.

2.3 Machine Learning

In the previous chapter, we have seen one application domain of machine learning and several different model choices to perform a classification task for remotely sensed hyperspectral images. This chapter presents an overview of the fundamentals of classical machine learning. Machine learning is the use of methods or models that are able to learn by analysing patterns in data. Mostly depending on the type of data we want to learn from, there are three main branches of machine learning: supervised, unsupervised and reinforcement learning [22].

Supervised Learning A supervised classification task essentially learns a function $f(x)$ s.t. $f(x) = y$, where y are the labels for a given set of unlabeled data x . Sampling from the joint distribution of $P(X, Y)$, the goal is to infer the probability s.t. $P(Y = y|X = x)$.

Unsupervised Learning As the name suggests, in an unsupervised learning task, no labeled training data is given. By exploiting the underlying pattern of the given training samples $x \in X$, we want to approximate the real probability distribution $P(X)$.

Reinforcement Learning Reinforcement learning is a reward-based learning method that does not learn from given data or labels, but from self-generated data through a reward func-

tion. It is very prominent in fields like robotics, but not common in image processing.

What distinguishes machine learning from classical statistical methods like linear regression is its non-linearity. It enables an artificial neural network to approximate any physical function. So merging it with quantum computing might come as a surprise, as quantum computing itself is inherently linear. However, we can overcome those linear challenges and introduce methods that are at least as powerful as classical machine learning algorithms. In this thesis, we focus on quantum supervised learning.

2.3.1 Machine Learning Methods

The objective of supervised machine learning is to find a model which minimizes a cost function given some provided data and subsequently minimizes the same cost function on unseen data. Models can be used for classification and regression while being linear or non-linear. We can identify four different classes of machine learning models: linear models, artificial neural networks, graphical models, and kernel methods.

Linear models

Linear models form the theoretical foundation of machine learning and can mostly be found in the form of linear regression. It outputs continuous values and performs a regression on the given input data. To fit the model function, linear regression has usually some trainable parameters. With the addition of post-processing the output e.g. by using thresholds, it can also be used to perform classification tasks. Further, it is worth mentioning that linear refers to the model being linearly dependable on its unknown parameters.

Artificial Neural Networks

Neural networks are the working horses of nowadays machine learning applications. Their central building blocks are perceptrons, single neurons that have multiple weighted inputs and perform a linear classification resulting in a binary output produced by a non-linear activation function.¹ Stacked perceptrons build layers that then construct a neural network, also called multi-layer perceptrons. While perceptrons can build any function only over linearly separable data, neural networks act as universal function approximations. Figure 2.3 usual structure is one input layer, multiple hidden layers, and one output layer. Theoretically, only one hidden layer is enough to full fill universality, but it usually requires many perceptrons in that layer. The rise of deep neural networks in recent years shows only a moderate number of perceptrons

¹The whole idea of perceptrons was inspired by the back then state-of-the-art model of neurons in the brain.

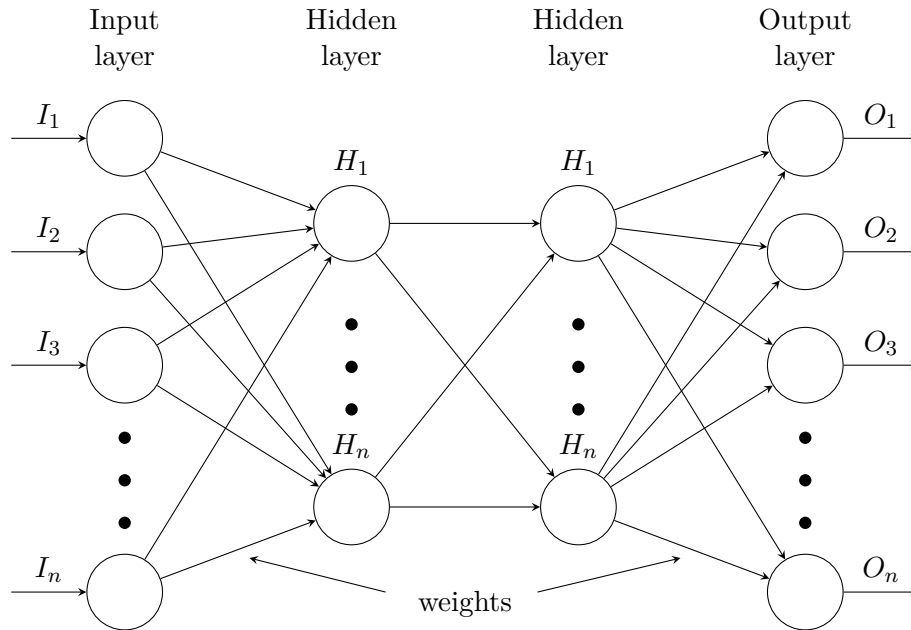


Figure 2.3. Classical feed-forward neural network structure consisting of connected perceptrons that form an input layer, several hidden layers and an output layer.

per layer suffice, but having many layers is often advantageous for many applications. As mentioned above, the activation function for each perceptron brings in the non-linearity. There are multiple options for activation function, ranging from Sigmoid, rectified linear units (ReLUs), or hyperbolic tangent, strongly depending on the problem one tries to solve. One of the biggest problems for this network structure with non-linear activation functions is that they impose a non-convex, non-linear optimization problem to find the fitting parameters. Such optimization problems are usually very hard to solve and some variation of a gradient descent algorithm is usually the method of choice when it comes to determining the optimal parameters. Especially after the introduction of the backpropagation algorithm [23], which essentially calculates the Jacobian of the parameters of the neural network by evaluating the model once and then passing its partial derivatives back through the network. It is a very efficient implementation to calculate the gradients of the network, so important that even with nowadays available computational power we are not able to train most of the models in use otherwise. We will see later that the general structure of a neural network is nearly identical to the one of a variational quantum circuit.

Graphical models

Next, we quickly introduce graphical models. Graphical models are probabilistic models that use graphs to express the structure and dependencies of random variables. One of its most

prominent applications is Bayesian networks. A chronically underdeveloped field in machine learning that has huge potential in the future. Here, quantum computers might be advantageous since due to their probabilistic nature, especially inference might be very easy.

Kernel methods

Kernel methods are ways of calculating distances in some feature space, which is given through a feature mapping from the classical to the feature space. Kernel methods are often used for classification algorithms that rely on distance measurements such as support vector machines or k-nearest neighbors. Contrary to neural networks, they are guaranteed to find the optimum due to the convex nature of their underlying optimization problem. Unfortunately their computation is often very hard, which made it unpracticable after the rise of deep neural networks. Nevertheless, its theory is still very important today and it will become clear that kernel methods are strikingly similar to quantum machine learning methods and at least partly efficiently implementable on a quantum computer.

2.3.2 Applications

In this thesis, we are in the realm of remote sensing. A field that would not exist in its current structure without the help of machine learning. Remotely sensed data is usually known for its complex nature and its sheer size. For most applications, without the help of aided classification and segmentation, the number of humans required to fulfill a similar task would just be unbearable. With the help of machine learning, we can monitor our earth in real-time, evaluate the risk of fire, risk of famines, human rights in conflict zones, or the impact of the climate crisis. Many challenges found a solution in recent years, but they either opened the door for new requirements or they often pointed in a direction of potential requirements.

Chapter 3

Background

Since quantum computation is an unknown field for many computer scientists and fundamentally differs from classical computational theory, we go into detail about the elements of its nature and the potential advantages of its use. We also go one step further and lay out the fundamentals of the core methodology we are using, quantum machine learning. It is the science of implementing machine learning techniques on quantum computers with a potential advantage over classical machine learning models. Lastly, we give a summary of the field of image processing for remotely sensed data. In particular, we define what hyperspectral images are and how they are currently processed and classified on classical computers. Thus the following section discusses the elements of quantum theory and provides necessary background knowledge in the area of quantum computation and how to embed classical data on a quantum circuit.

3.1 Elements of Quantum Theory

Richard Feynman famously said that everything about quantum mechanics can be explained with the double slit experiment [24]. The setup of the double slit experiment is fairly easy. We have a source that emits photons sequentially, a barrier that has two narrow slits, and a wall behind the barrier that detects incoming photons. As shown in Figure 3.1, after counting a certain number of photons, we receive a density pattern that shows that some areas are more likely than others to be hit by a photon. Contrary to common conception, the pattern alone is not the weird part. It is the difference between the single and the double slit pattern. Coming from basic probability theory, we would expect that the probabilities of the single-slit experiments would just add up to the double-slit pattern. But it does not. There are now photons in areas where no photons had been before and vice versa. It gets even weirder if we try to measure the photons before they hit the barrier. It resolves in Figure 3.1, which shows the collapse of the double-slit pattern into the two single-slit ones. This leakage of

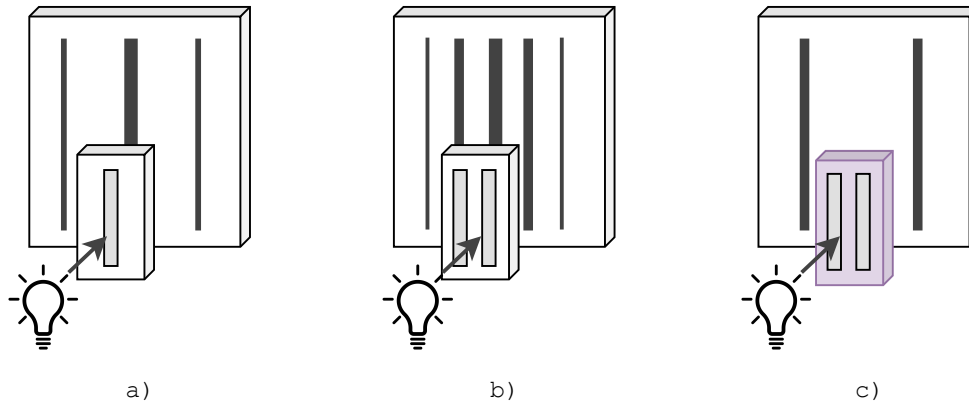


Figure 3.1. Double slit experiment: a) photon pattern with one slit b) photon pattern with double slit c) photon pattern with double slit and measurement

information on the quantum system to the outside is known as decoherence and is one of the reasons why building quantum computers is so difficult. In the above example, decoherence was purposely introduced to perform the measurement. But to build a quantum computer, we need to actively shield the quantum systems from the environment such that information can not be leaked. Physics is mostly about predicting future measurement outcomes when performing certain experiments and physicists in the early 20th century introduced complex numbers to describe the randomness of what they saw.¹ Each possible configuration of an isolated quantum system is given an amplitude α , which describes its probability of measuring it. The amplitudes are complex numbers with a real and an imaginary part, which can be either positive or negative.² More precisely, the probability of an outcome is the squared absolute value of the amplitude. This is also commonly known as the Born Rule:

$$P \in [0, 1] = |\alpha|^2 = \text{Re}(\alpha)^2 + \text{Im}(\alpha)^2 \quad (3.1)$$

The fact that the amplitudes are complex leads to the possibility of interference. Contrary to classical randomness, adding two probabilities can not only add to each other but can also cancel each other out. Subsequently, this produces the patterns as in Figure 3.1 and is one of the essences of quantum mechanics [27].

¹It was believed by many physicists that complex numbers were merely math sugar which made things easier, but not essential to the theory itself. Meaning that everything can also be described by real numbers. This was falsified by recent experiments in [25] meaning that complex numbers are inherently important for quantum mechanics.

²Interestingly, this leads to the existence of negative probabilities. Something that is intuitively hard to understand. If the reader is further interested in this topic, I recommend [26].

3.2 Elements of Quantum Computation

The Church-Turing thesis can be seen as the foundation of the theory of computer science. Developed by Alan Turing [28] and Alonzo Church [29] in the 1930's, it states, that *every physical process can be simulated by a Turing machine*. We can also reformulate it into:

A Turing machine (or lambda calculus) can simulate every mechanical computer.

In the following years, this thesis was strengthened by a theoretical computer scientist who thought that the Turing machine was at least as powerful as any other model of computation. This leads to the extended version of the Church-Turing thesis:

Every physical process can be simulated efficiently using a Turing machine.

This statement seems remarkable. One familiar with the building blocks of a Turing machine might think that it is a rather bold statement given the complexity of nature itself. To many, especially physicists, it seemed unintuitive to believe that such a mechanical structure would be able to simulate any physical process *efficiently*. So the question was risen, can we challenge this thesis? Or is it possible to derive an even stronger formulation of the Church-Turing thesis? In 1981, the aforementioned physicist Richard Feynman was the first person who openly suggested that we might need a quantum computer to simulate quantum systems [30]. The idea was further developed by David Deutsch in 1985 [31], who (tried) to define a computational device that could simulate any physical system efficiently.³ Deutsch's universal quantum computer is the prototype of the model of quantum computation that is most often used nowadays. To reformulate it in terms of the above discussed Church-Turing theses:

Is there a problem that can be solved efficiently by a universal quantum computer but not by a probabilistic Turing machine?

It was Peter Shor in 1994 who delivered one of the first extensions to David Deutsch's model. He proposed an algorithm in [32] that would solve the problem of finding prime factors of an integer efficiently using a universal quantum computer. A problem for which to this day, no efficient classical algorithm is known.⁴ Another step forward was made by Lov Grover in 1995 [34]. He proposed an algorithm that finds an element in an unsorted list in sub-linear time, more precisely $O(\sqrt{n})$. A quadratic speedup compared to the classical solution of going

³For completion, the quantum computational model was not the first computational model that challenged the extended Turing thesis. In the 70s the Solovay-Strassen test for primality was introduced. It uses randomness in its algorithm to determine whether an integer is prime or a composite with certainty. A problem for which no efficient classical algorithm is known. That leads to the discovery that an algorithm with access to a random number generator can solve certain problems efficiently that a classical deterministic Turing machine can not.

⁴Shor's algorithm is especially important and gained widespread recognition because many of our encryption techniques are based on the difficulty of solving this problem [33].

through the whole list and therefore has the complexity of $O(n)$ where n is the number of elements in the list. In comparison, Shor's algorithm delivers a potential exponential speedup in comparison to any classical known algorithm. Here, theoretical computer science or more specifically computational complexity theory gives us the overall motivation to find quantum algorithms.

3.2.1 Quantum states and observables

To understand quantum computing, we need to understand its core building blocks. The following is based on Chapter 1 in [35], which we recommend for further studies of quantum computation and information. In classical computation and digital communication, the basic unit of information is the bit. It is a binary logical state usually taking either the values 0 or 1. Its physical implementation is usually done by a two-state device such as a transistor. In quantum information theory, the basic unit of information is a quantum bit or qubit.⁵

Qubit

As the classical bit, a qubit can also be in the ground states $|0\rangle$ and $|1\rangle$, here notated by the Bra-Ket formulation. A quantum system is described by a unit vector in a complex vector space \mathbb{C}^N , the so-called Hilbert space \mathcal{H} . For now, we deal with discrete quantum systems where $N = 2$, so following a measurement we can only land in two possible basis states with a certain probability. So a qubit is a pure quantum state described by a normalized state vector $|\psi\rangle$. Given the two basis vectors $|0\rangle$ and $|1\rangle$, the qubit is in a general superposition,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad (3.2)$$

with $\{\alpha, \beta\} \in \mathbb{C}$ and normalized as

$$|\alpha|^2 + |\beta|^2 = 1. \quad (3.3)$$

The Bra-Ket or Dirac notation, named after its famous inventor Paul Dirac, simplifies common operations in quantum mechanics. While the Ket notations $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ refer to the two basis states or vectors, the scalar or inner product of two states is simply $\langle\psi|\psi\rangle$, where $\langle\psi|$ represents the entry-wise conjugate-transpose of a ket as a complex row vector. So for a pure state $\langle\psi|\psi\rangle = 1$, the norm of the ket, is true. The linear combination or superposition in Equation 3.2 reflects an arbitrary direction in which the state vector points within the Hilbert

⁵A formalism coined by Ben Schumacher, one of the pioneers of quantum information theory[36].

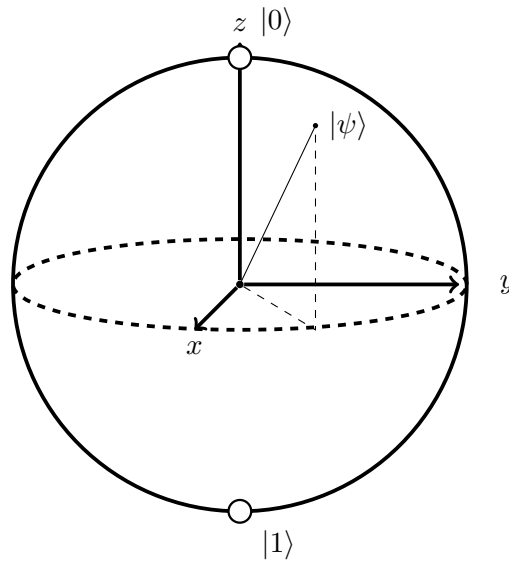


Figure 3.2. Bloch Sphere representation of a qubit state $|\psi\rangle$.

space. We can rewrite Equation 3.2 and create a three-dimensional geometric representation of ϕ by such that

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right). \quad (3.4)$$

where $e^{i\gamma}$ is the global phase which we ignore since it has no observable effects⁶ and simply write:

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle. \quad (3.5)$$

θ and φ define a point in the so-called Bloch sphere, which is illustrated in Figure 3.2. Given an n -dimensional quantum system, we can pick a basis $\mathcal{B} = \{|0\rangle, \dots, |n-1\rangle\}$. Every state vector can then be expressed in this basis so that,

$$|\psi\rangle = \sum_{j=0}^{n-1} c_j |j\rangle. \quad (3.6)$$

c_0, \dots, c_{n-1} are called coefficients and are complex numbers. Further, we pick our basis vector such that they are orthogonal to each other and normalized,

$$\langle \psi_m | \psi_n \rangle = \delta_{mn} = \begin{cases} 1 & m = n \\ 0 & m \neq n. \end{cases} \quad (3.7)$$

δ_{mn} is referred to as the *Kronecker delta symbol*. Besides the already mentioned computational

⁶Even though the global phase does have measurable effects, it does not affect our form of computation and is thus being ignored for simplicity.

basis states $|0\rangle$ and $|1\rangle$, there are four other basis states that occur frequently in the literature:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad |i\rangle = \frac{|0\rangle + i|1\rangle}{\sqrt{2}} \quad |-i\rangle = \frac{|0\rangle - i|1\rangle}{\sqrt{2}} \quad (3.8)$$

Density operator

So far, we described our quantum states with state vectors and simply used the Dirac notation. We were able to do this because we assumed pure states, hence perfect knowledge of the described state. In the absence of this knowledge, which is very common in quantum mechanics, we are required to use an alternative, stochastic description of the quantum states, the density matrix ρ . The density matrix of an arbitrary pure state $|\psi\rangle$ is defined as,

$$\rho = |\psi\rangle\langle\psi|, \quad (3.9)$$

which is, for non-physicists, essentially the covariance matrix of the state $|\psi\rangle$. The $|\cdot\rangle\langle\cdot|$, or "ket-bra", notation is equivalent to an outer product of the two-state. We are now also able to describe so-called mixed quantum states. A mixed quantum state is an ensemble of pure states, which is with probability p_i in the state $|\psi_i\rangle$. The density matrix for mixed states is given by,

$$\rho \equiv \sum_i p_i |\psi_i\rangle\langle\psi_i|. \quad (3.10)$$

This density matrix is now able to describe the uncertainty in which state the ensemble is. Further, the trace of the density matrix must satisfy either $\text{tr}(\rho^2) = 1$ for pure states or $\text{tr}(\rho^2) < 1$ for mixed states.⁷ Another useful property is the partial tracing of subsystems. Given a density matrix of two quantum systems A and B ,

$$\rho_{AB} = \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix}, \quad (3.11)$$

we can partially trace out the two subsystems as mixed states so that,

$$\text{tr}_A \{\rho_{AB}\} = \begin{pmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{pmatrix} \quad \text{and} \quad \text{tr}_B \{\rho_{AB}\} = \begin{pmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{pmatrix}. \quad (3.12)$$

⁷In a very nice way, one can show this through Schwartz inequality.

Multiple Qubits

We already got a glimpse of how a single qubit can potentially store and process a substantial amount of information, but to harness the full potential of quantum information and computation we need to consider the exponentially growing space of multiple qubits. Given a two qubit system, the four computational basis states are $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. We can describe the joint state of the system of qubits with the tensor product as $|0\rangle \otimes |0\rangle = |00\rangle$. Considering the definition in Equation 3.2, we write the state vector as a superposition of all four states so that,

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle. \quad (3.13)$$

The complex coefficients α_i are the state's amplitudes and squaring them gives the probability the state collapses in one of the corresponding classical states 00, 01, 10, or 11. As in Equation Equation 3.3, the normalization condition requires that the sum of all the probabilities of all possible states is 1.

3.2.2 Transforming Quantum States and Measurement

After introducing the basic units of information in quantum computing, we now describe how to manipulate them to perform computation. To perform quantum computation, we need to manipulate quantum information and therefore the state of a qubit. This is done by quantum logic gates, whose functionality is similar to classical gates. A quantum gate is a linear transformation of a quantum state. To model computation, we use quantum circuits which are composed of quantum wires, that represent qubits and carry the information. Quantum circuits are usually read from left to right and by applying quantum gates we can manipulate the qubits within.

Single Qubit Gates

Single qubit gates can always be described by 2x2 unitary matrices, so that for every quantum gate U , $U^\dagger U = I$, where U^\dagger is the adjoint of U . Potentially every unitary 2x2 matrix can be a quantum gate on a single qubit state and since the state of a qubit can be represented as a vector in the Bloch sphere, a quantum gate can be seen as a rotation of this vector. To give an example, we introduce one of the basic quantum gates, the X-gate. On a computational basis state, it acts similar to the classical NOT gate,

$$\begin{aligned} X |0\rangle &= |1\rangle \\ X |1\rangle &= |0\rangle \end{aligned}$$

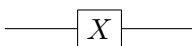
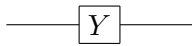
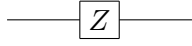
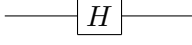
Gate	Circuit Representation	Matrix
X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Y		$\begin{bmatrix} 0 & -i \\ i & i \end{bmatrix}$
Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
H		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Figure 3.3. Basic single qubit gates, rotating around the X, Y and Z axis of the bloch sphere as well as the Hadamard gate (H). Shown are also the representation in a circuit and their matrix representation.

Further, it acts linearly on the superposition in Equation 3.2,

$$X(\alpha |0\rangle + \beta |1\rangle) = \alpha |1\rangle + \beta |0\rangle \quad (3.14)$$

For a more comprehensive overview of single qubit quantum gates and their circuit and matrix representations see Figure 3.3.

Observables

But how can we perform a measurement? To understand this, we need to learn about observables and their expectation values. Observables are Hermitian operations in Hilbert space. They are the quantities measured when we want to extract information after we do an experiment or our above stated computations. An operator A is Hermitian if,

$$A = A^\dagger. \quad (3.15)$$

Further, given a quantum state $|\psi\rangle$, the expectation value is given by,

$$\langle A \rangle = \langle \psi | A | \psi \rangle. \quad (3.16)$$

The expectation value of the observable A is defined as the probabilistic expected value of its measurement in a certain state. We can also define the expectation value of A through calculating the trace of the density operator. Here we define a density operator ρ as the projection $|\psi\rangle\langle\psi|$ and get the expectation value through,

$$\langle A \rangle = \text{Tr}(A\rho). \quad (3.17)$$

If we want to measure a given quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in our computational basis, we take the Pauli-Z operator and calculate its expectation value, so that

$$\langle\psi|Z|\psi\rangle = |\alpha|^2 - |\beta|^2. \quad (3.18)$$

The probability of receiving a specific measurement, say $|0\rangle$, is then, given the corresponding projected eigenspace $P_0 = |0\rangle\langle 0|$,

$$p(0) = \text{Tr}(P_0|\psi\rangle\langle\psi|) = \langle\psi|P_0|\psi\rangle = |\alpha|^2. \quad (3.19)$$

Multi Qubit Gates

To make quantum computation universal and harness its full potential, we need a way for qubits to interact with each other. Here we need quantum gates that act on two or more qubits. Given the multi qubit state in Equation 3.13, we introduce the most basic two qubit quantum gate, the CNOT gate. It is defined as,

$$CNOT = CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{cases} |00\rangle \rightarrow |00\rangle \\ |01\rangle \rightarrow |01\rangle \\ |10\rangle \rightarrow |11\rangle \\ |11\rangle \rightarrow |10\rangle \end{cases} \quad (3.20)$$

It sets the first qubit as control and if and only if it is the computational ground state $|1\rangle$, applies an X-gate on the second qubit. We can rewrite this as,

$$|x\rangle|y\rangle \rightarrow |x\rangle|x \otimes y\rangle. \quad (3.21)$$

Here \otimes acts as addition modulo 2. To get a comprehensive overview of multi-qubit gate representation see Figure 3.4.

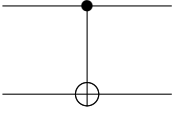
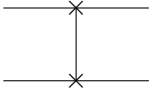
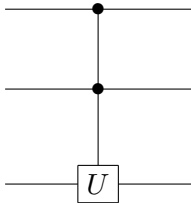
Gate	Matrix	Circuit Representation
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
CCU		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & 0 & 0 & 0 & 0 & U_{10} & U_{11} \end{bmatrix}$

Figure 3.4. Basic multi qubit gates: CNOT, the controlled NOT gate; SWAP, swaps the state of two qubits and CCU, controlled-controlled U gate that controls on two wires and then applies an arbitrary unitary U on the third wire.

3.2.3 Quantum Processing Unit

We execute quantum circuits on a quantum processing unit (QPU). A QPU describes the physical implementation which performs quantum computation. In contrast to classical computation, a QPU can come not only in different hardware types but also with different quantum computing paradigms. For this thesis, we choose the discrete quantum gate mode. A universal quantum computing paradigm that already showed some recent proof of advantage [10], has commercially available implementations [37] and has the overall advantage of being conceptual very similar to classical circuit models. A QPU can also run hybrid quantum-classical circuits that contain parameters that are optimized using a classical feedback loop. Such a scheme is pictured in Figure 3.5.

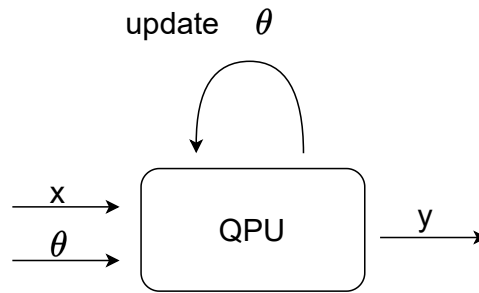


Figure 3.5. Quantum processing unit with input x , parameters θ and output y . The QPU runs quantum circuits, but its parameters can be optimized classically.

3.3 Quantum Embedding of Data

So far, we presented the basic model of quantum computation but did not think about what kind of information we wanted to process. The following section is solely dedicated to different ways of encoding information into a quantum state so that it is computable in a meaningful way.

3.3.1 Classical and Quantum Data

In quantum computing and especially quantum machine learning, the embedding of data is one of the most crucial steps within the overall algorithm. As we have seen before, we can have four different ways of quantum computation. We could either use quantum data on a quantum or classical device or use classical data on a quantum or classical device. While quantum data on a quantum device is the one setting that is expected to have the biggest impact [38], we will keep it an open research question in the end since its implementation would overshoot the current thesis. We will concentrate on the most common case where we use classical data on a quantum device. To be more precise, we want to find a way to encode classical data as quantum states in a Hilbert space through a so-called quantum feature map. Given a classical feature vector x , we want to find a set of gates that represent its information in a quantum state $|\psi_x\rangle$. Since embedding is usually a bottleneck in quantum computation, it will also be a central part of this thesis.

3.3.2 Universal Embedding Techniques

We define two categories of universal quantum feature maps for classical data, parametrized and non-parametrized. The difference is straightforward. Given some classical data x , we define the non-parametrized embedding $|x\rangle$ as an unitary $U(x)$ acting on the computational basis state

$|0\rangle$ so that,

$$|x\rangle = U(x) |0\rangle. \quad (3.22)$$

The parametrized embedding acts in the same way with the difference, that the unitary $U(x, \theta)$ also depends on a set of freely tuneable parameters θ ,

$$|x\rangle = U(x, \theta) |0\rangle. \quad (3.23)$$

In the following, we will present different schemes for non-parametrized embeddings. Parametrized embeddings will be shown in the quantum image presentation section and the kernel section.

Basis Embedding

Basis embedding is the most basic and straightforward way to represent classical (binary) data as a quantum state. Given a classical, 4-bit feature vector $x = 1100$, we simply represent it by 4-qubit state $|1100\rangle$. So we use the two computational basis states to embed the classical binary information, which allows us a direct mapping. It also means that, as a lower bound, we need at least as many qubits as we have bits in the feature vector x . But we can use the Hilbert space and therefore the superposition to embed an entire dataset. So considering a Dataset D which consists of M N -dimensional feature vectors, we have a specific feature vector $x^{(m)} = (b_1, \dots, b_N)$ with $b_i \in \{0, 1\}$ for $i = 1, \dots, N$. With binary precision, we can now represent the entire dataset as a superposition of N qubits so that,

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M |x^{(m)}\rangle. \quad (3.24)$$

The number of qubits grows linear with the number of bits.

Amplitude embedding

Another common embedding technique is amplitude embedding[39]. Here we take normalized classical data and encode it into the amplitudes of a quantum state, which is especially suited for floating point data. Broadly speaking, we prepare a quantum state in a way that the amplitude of every computational basis state represents a value in our feature vector. Given an N -dimensional data point x , we need a n -qubit quantum state $|\psi_x\rangle$ so that $N = 2^n$. $|\psi_x\rangle$ is then defined as the sum of all features in x , represented as the amplitude of the corresponding i -th computational basis state α_i so that,

$$|\psi_x\rangle = \sum_{i=1}^N \alpha_i |i\rangle. \quad (3.25)$$

For amplitude encoding to work, we need to make sure to normalize the feature vector $|\alpha|^2 = 1$ and potentially pad the input data if N is not a multiple of 2^n . As a lower bound, amplitude encoding requires at least $\log(N)$ qubits, which translates to exponentially fewer qubits than the basis embedding technique. But it comes with the cost of preparing the dense amplitude vectors, which can be viewed as a space-time trade-off.

Angle Encoding

Another remarkable simple, yet very effective way of encoding is angle encoding. Through the use of a single rotation gate $R \in \{R_x, R_y, R_z\}$, we encode each datapoint directly in an angle. The resulting state is given by,

$$|x\rangle = \bigotimes_i^n R(\mathbf{x}_i) |0^n\rangle. \quad (3.26)$$

The slightly different notation \bigotimes represents a tensor product of the n qubits present. The angle encoding is desirable for NISQ-era devices for two reasons. Firstly, it is hardware efficient. Meaning its implementation on real quantum hardware does not require much computational overhead and is, therefore, comparable efficient to implement. Further, rotation gates are usually within the set of native gates for many quantum computers. Secondly, due to its bijective mapping of qubits and features, we achieve a sub-polynomial depth of the circuit in the number of features.⁸

Within angle encoding, we can even go one step further. By exploiting the relative phase degree of freedom, we are able to encode two features per qubit. This is known as dense angle encoding.[40] Given a single, arbitrary angle encoding of one qubit $|\psi\rangle$, we can define this as,

$$|\psi\rangle = \cos(x) |0\rangle + \sin(x) |1\rangle. \quad (3.27)$$

Dense angle encoding now makes use of the relative phase, so that,

$$|\psi\rangle = \cos(\pi x_1) |0\rangle + e^{2\pi i x_2} \sin(\pi x_1) |1\rangle. \quad (3.28)$$

IQP Encoding

IQP or Instantaneous Quantum Polynomial time is a class of commuting quantum computations that are generally hard to simulate classically up to a constant additive error. IQP encoding schemes use data encoding circuits that fall into that class. One very general way of encoding

⁸Circuit depth corresponds to the coherence time of quantum hardware. This is essentially the maximum possible duration of computation until the qubit collapses. A look at the currently publicly available IBM quantum hardware reveals an estimated average of $100\mu s$.

a certain state $|x\rangle$ in the IQP style is,

$$|x\rangle = (U_Z(x)H^{\otimes n})^\Gamma |0^n\rangle, \quad (3.29)$$

where $U_Z(x)$ is defined as,

$$U_Z(x) = \prod_{[i,j] \in S} R_{Z_i Z_j}(x_i x_j) \bigotimes_{k=1}^n R_z(x_k). \quad (3.30)$$

The circuit is essentially a sequence of Hadamard gates, rotation gates, and a non-closing ring of controlled rotations. As mentioned above, the rotations are arbitrary as long as the overall circuit falls into the IQP class.

3.4 Quantum Computational Complexity

After setting the stage with a basic overview of quantum computing, let us have a look at the true motivation for why quantum computing might be worth the effort. In classical computational science, we use complexity classes to group problems by their hardness.⁹ Our measurement is the scaling of the resources it takes to solve the problem as a function of the input size n . The accommodating notation is the so-called big O notation, which is part of the Bachmann-Landauer or asymptotic notations and shows lower bounds on time or space resources. So for linear runtime, we have $O(n)$, quadratic $O(n^2)$, and so on. We use complexity classes to define the groupings of the problems. The most well-known might be (i) P: problems that are solvable in polynomial time by a Turing machine and (ii) NP: problems that can be solved in polynomial time by a non-deterministic Turing machine and/or which are verifiable by a deterministic Turing-machine in polynomial time. We have (iii) NP-complete: the "hardest" problems in NP, which are NP-hard, so that every problem in NP can be reduced to it in polynomial time, and in NP. (iv) BPP: problems that are solvable by a probabilistic Turing machine in polynomial time with a bounded-error probability. (v) BQP: problems that are solvable by a quantum computer in polynomial time with a bounded error probability. It defines the quantum analog to (iv). (vi) QMA: quantum Merlin Arthur complexity class contains problems that are verifiable by a quantum computer in polynomial time. QMA is in a similar relation to BQP, as NP is to P. Further, (vii) PSPACE: problems that are solvable using polynomial space, and (viii) EXPTIME: problems that are solvable using exponential time on a deterministic Turing machine. One of the central questions in theoretical computer science is how the classes are in relation to each other, the most famous whether $P \stackrel{!}{=} NP$. We know for example that

⁹The following chapter is based on [41] and [42], references an interested reader might also enjoy getting a more comprehensive understanding.

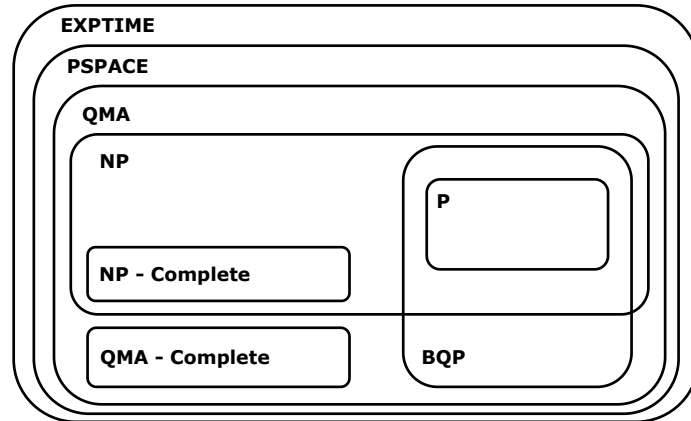


Figure 3.6. Potential relationships of complexity classes. Description of the classes are given in Chapter 3.4

$P \subseteq BQP$. It is easily verifiable by looking at Shor's factoring algorithm [43]. It factorizes integers in polynomial time using a quantum computer, a problem for which no polynomial time algorithm is known classically. Also, [44] showed that it is possible to simulate a quantum computer classically with exponential time and polynomial memory, thus $BQP \subseteq PSPACE$. Complexity classes are crucial when thinking about new algorithms and also perform well as a sanity check whether potential advantages are realizable. A diagram of potential and widely believed containment relations are shown in Figure 3.6. Nevertheless, so far, surprisingly little is known about quantum complexity classes.¹⁰

¹⁰A rich and up-to-date database of all sorts of complexity classes can be found at https://complexityzoo.net/Complexity_Zoo

Chapter 4

Quantum Machine Learning

In Chapter 1, we introduced two waves of quantum machine learning. The first wave concentrated on speeding up existing machine learning subroutines. Theoretically, this has a speed up over classical algorithms, but would require large, fault-tolerant quantum devices. Since such hardware is beyond the horizon, a second wave of quantum machine learning evolved around available NISQ era devices. There are several approaches on how such a NISQ algorithm might look like, but the most famous ones are based on Variational Quantum Circuits (VQC) [22], [45]. Instead of fixed quantum gates, VQCs have parametrized quantum gates that are trainable, which allows them to incorporate the existing noise. They became prominent with the introduction of the variational quantum eigensolver algorithm (VQE) and the quantum approximate optimization algorithm (QAOA) [46], [47]. VQE was especially interesting for quantum chemistry use cases, while QAOA was developed to solve combinatorial optimization algorithms. One should not underestimate the impact of those to algorithms, because even though they still lack a proof of superiority, they mainly drove the industry interest in quantum computing. Moreover, it was also the trigger for the second wave of quantum machine learning, the direct use of quantum circuits as machine learning models [48], [38]. The subsequent chapter will introduce variational quantum algorithms and kernel methods, and conclude the chapter with a section on quantum image representations.

4.1 Variational Quantum Algorithms

Throughout this thesis, we focus on the previously mentioned second wave of quantum machine learning. We accept the presence of noise in our quantum devices, the limitation of qubits and coherence time and therefor work with two NISQ era approaches.

Firstly, we describe how quantum computing and especially quantum machine learning is closely related to Kernel methods. We explain the idea behind kernel methods, how and why we can

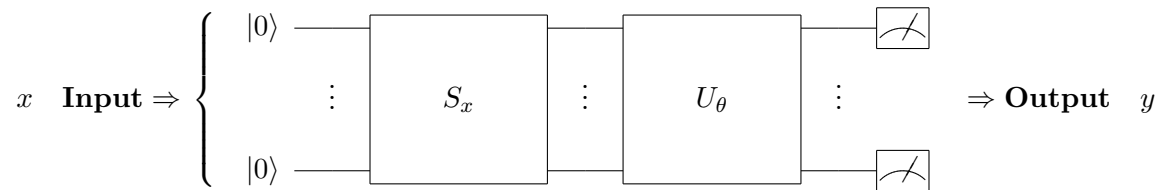


Figure 4.1. General structure of a variational quantum circuit consisting of the feature map S_x with input data x , a parametrized unitary U_θ with parameters θ and a subsequent measurement $f(\sigma) = y$ where f is the overall quantum function with measurement in basis σ producing output y .

implement them on quantum computers, discuss their advantage and give an outline why they might become more relevant in the future. Secondly, we extend the idea and show the remarkable similarity between parametrized quantum circuits and neural networks. The two ideas are not mutually exclusive, and we will see in the methodology section that both have their advantages and can be used together to improve results.

A variational quantum circuit is a quantum circuit that contains freely tuneable, parametrized quantum gates. The general structure of a variational quantum circuit is pictured in Figure 4.1. Given an n -dimensional input vector x , we encode the data using a unitary S_x which acts as a feature map using N qubits. The resulting quantum state ψ therefor only depends on the input feature, so that $|\psi_x\rangle = S_x |0\rangle^{\otimes N}$. The routine is also called state preparation and usually acts on an unprepared qubit input register in the computational basis state. In rare cases, the feature map also contains trainable parameters which we discuss in the following section. The input data can be quantum or classical and the choice of the feature map heavily depends on it and the classification task. Common feature maps are the previously discussed amplitude or angle encoding. Following the state preparation, we apply the variational model or Ansatz U_θ . The Ansatz computes the desired output of the model and consists of parametrized and non-parametrized single- and multi-qubit gates. It is often applied in multiple layers, similar to hidden layers of classical neural networks. The output is the expectation value of an observable. The parameters are then optimized given a certain objective function. In NISQ-era algorithm, the optimization is computed classically which coins the term hybrid quantum machine learning algorithms. The optimized parameters are then fed back into the Ansatz and the new output is again evaluated. Before we have a look at different forms of variational quantum circuits, let us take a look at how parametrized gates look like and how we can extract gradients.

Parametrized gates and the parameters-shift rule

One of the three basic single qubit parametrized rotation gate is the $R_x(\theta)$ gate,

$$R_x(\theta) = e^{-i\theta\sigma_x/2} = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}, \quad (4.1)$$

which performs a rotation around the x-axis. The rotation angle θ is freely choosable so that $\theta \in [0, 2\pi]$. To extract the gradient of $R_x(\theta)$, we simple apply the so-called parameter shift rule [49], [50]:

$$\frac{d}{d\phi} f(R_x(\phi)) = \frac{1}{2} [f(R_x(\phi + \pi/2)) - f(R_x(\phi - \pi/2))], \quad (4.2)$$

where $f(\theta)$ is the output of the variational circuit as a quantum function. It is important to notice, that the actual gradients can be computed with the same variational circuit only through slightly shifting the input parameters. Further, the parameter shift rule provides the exact gradient and not an approximation. It also does not depend on very small shifts, as we can see in Eq. 4.2. This is very crucial compared to other methods like the finite difference method since NISQ-era devices have a hard time outputting reliable small changes.

Optimization

Precision in measurement is also important when we search for a suitable optimizer. We only consider gradient-based approaches, but there are also gradient-free and hardware-aware approaches. In general, optimizing parameters in quantum circuits does not differ from the classical setting. But lack of precision increases the number of measurements that are required to estimate a suitable mean value, an additional overhead we should consider when choosing the optimizer.[22] We should also avoid methods that require high-depth analytical gradient circuits and are prone to error since NISQ-era devices are of course noisy and coherence times might not be long enough to construct such circuits. One of the biggest obstacles parametrized quantum circuits suffer from are so-called Barren plateaus. Barren plateaus describe a very flat optimization landscape which makes optimizing through gradient descent algorithms extremely difficult [51]. The resulting local minima are believed to be partly introduced by noise [52], or in our cases, since we work with noiseless simulators, by the circuit itself. [53]. It could be mitigated through better parameter initialization.

Ansätze

The parametrized quantum circuit or ansatz is not only essential when it comes to optimization, but also dictates the performance of the model. Primarily, it affects the convergence speed and

the final distance to the model’s aim. We can distinguish two objectives when constructing an Ansatz. They can either be problem-inspired or hardware-inspired. In general, an Ansatz acts as a parameter θ depending unitary $V(\theta)$ on the prepared state $|\psi\rangle$ so that,

$$|\psi(\theta)\rangle = V(\theta) |\psi\rangle. \quad (4.3)$$

Usually, a problem-inspired Ansatz is rooted in either the underlying physics of the solvable problem or uses some properties of quantum mechanics to optimize the circuit. Famous and often used Ansätze are the Quantum Approximate Optimization Algorithm (QAOA) [47] or the Variational Hamiltonian Ansatz (VHA) [54]. Even though they solve problems theoretically very efficiently, they are sometimes hard to implement. On the other hand, hardware-efficient Ansätze try to optimize the circuits without violating hardware constraints. See [55], [56] or [22] for more comprehensive reviews.

4.2 Quantum Kernel methods

Besides the variational quantum algorithms, kernel methods are on the verge to become very important in quantum machine learning. The following chapter is based on a similarity of quantum computing and kernel methods, closely following the paper of [57] with the title ‘Quantum machine learning models are kernel methods’.

4.2.1 Theoretical Foundation of Kernel Methods

First and foremost, kernel methods are similarity measures that solve machine learning tasks by calculating the distance between data points. Those distances are then for example used to compare new data points to training samples in a classification task.¹ Common classical methods to achieve this are k-nearest neighbor, which compares the distance to a k number of the closest data points and classifies according to the most common neighbor, and support vector machines, which find a linear separation of the data.

Linear Classification

To understand the purpose of kernel methods, it is worth reminding us of one of the simplest binary classification techniques, linear classification. As the name suggests, it draws a line in our data space to separate the data points into their respective class. Given the binary label $y = \pm 1$, the input data x , the intercept term b and an orthogonal, direction determining vector

¹Kernel methods are not only classification methods, but they also can replace every dot product in a machine learning model which can be reformulated with dot products.

w , we can formalize this classification task as,

$$y(x) = \text{sign}(\langle w, x \rangle + b). \quad (4.4)$$

In the case of high dimensional space, w defines a hyperplane. In a more visual way, b defines the position of the separating line or hyperplane, while w describes its direction. The dot product then indicates on which 'side' (± 1) the corresponding datapoint is.

This linear classification states a simple and powerful classification tool but quickly loses its accuracy and relevance when the data is not linearly separable. But through the application of simple and genius linear algebra, we can increase the accuracy of this method also for the non-linear case.

Kernel definition

Following [58], we first define a mapping of the data from its original set X to a space which allows us to perform similarity measurement, the dot-product space H ,

$$\phi : X \rightarrow H. \quad (4.5)$$

As the name suggests, we subsequently also define a dot product k as our similarity measurement in H , so that,

$$k : X \times X \rightarrow \mathbb{R} \quad \text{and} \quad (x, x') \mapsto k(x, x'), \quad (4.6)$$

with,

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \quad \forall \quad x, x' \in X. \quad (4.7)$$

We call k the kernel and $\phi(x)$ the feature map, while H is our feature space. A kernel that satisfies Equation 4.7 is also a positive definite kernel. This has far-reaching effects, as we do not have to know the explicit form of the feature map and can operate on data through the dot product by replacing $\langle \phi(x), \phi(x') \rangle$ with the kernel evaluation $k(x, x')$, which is called the kernel trick.[59] So in other words, through Equation 4.7 we do not have to compute the explicit embedding, which is mostly very costly, but we can implicitly use the embedding through the kernel. In terms of our linear classification task defined in Equation 4.4, we replace w with a separating hyperplane in the feature space w' and x with the feature map ϕ applied on it so that,

$$y(x) = \text{sign}(\langle w', \phi(x) \rangle + b). \quad (4.8)$$

We are now able to do a non-linear classification through a linear classification in the feature

space. Combining it with the kernel trick, we can do it by simply computing the inner product in the initial space while implicitly computing the inner product in the feature space. [60] gave a good example in showing that for an embedding,

$$\phi : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}, \quad (4.9)$$

we can explicitly calculate the corresponding kernel:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \left\langle \phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \phi \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} \right\rangle \\ &= x_1^2x_1'^2 + 2x_1x_2x'_1x'_2 + x_2^2x_2'^2 \\ &= (x_1x'_1 + x_2x'_2)^2 \\ &= \langle \mathbf{x}, \mathbf{x}' \rangle^2. \end{aligned} \quad (4.10)$$

We can see that through calculating the dot product (and squaring) in our original space, we get the dot product in the associated feature space. As mentioned above, we usually do not know the explicit feature mapping function and calculating it is also expensive, so making use of the kernel trick is very useful. To explain the effect of the feature map, we can take a look at another, more visual example.

XOR Example The XOR problem is straightforward: we have 4 datapoints situated in a 2D plane like given in Figure 4.2. The set of inputs is $X = \{(-1, -1), (-1, 1), (1, -1), (1, 1)\}$ with the respective outcomes of $Y = \{1, -1, -1, 1\}$. It is the exclusive OR operation known in computer science, where we only receive a positive outcome if the input is different. In Figure 4.2, squares and circles are representing the binary outcomes ± 1 respectively. Given a), we are not able to find a linear separation. But applying the feature map,

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \quad (4.11)$$

that maps,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1x_2 \end{bmatrix} \quad (4.12)$$

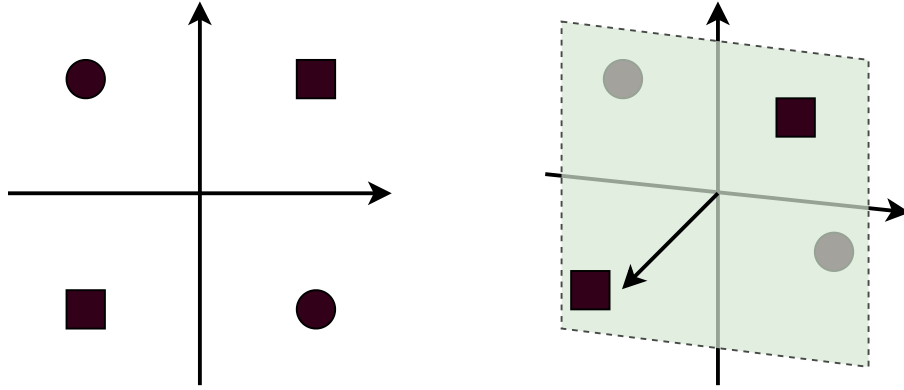


Figure 4.2. (a) Linearly not separable XOR problem with 4 datapoints on a 2D plane (b) The feature map projects the data into a 3D space, where we can now draw a separating hyperplane.

with the associated kernel,

$$k(x, y) = \begin{bmatrix} x_1 \\ x_2 \\ x_1x_2 \end{bmatrix}^T \begin{bmatrix} y_1 \\ y_2 \\ y_1y_2 \end{bmatrix}. \quad (4.13)$$

As mentioned before, we do not need to know the explicit feature map, but we can simply define a valid kernel associated with it. The result is the projected data in \mathbb{R}^3 as seen in Figure 4.2 b, which is now linearly separable.

Valid Kernels

To check whether a kernel is actually valid for a certain feature map, we can check the 'Mercer condition' with respect to the 'Gram matrix'. The 'Gram matrix' K , or the kernel matrix given x_1, \dots, x_n , is defined as,

$$K_{ij} = k(x_i, x_j), \quad (4.14)$$

satisfies,

$$\sum_{i,j=1}^n c_i c_j k_{ij} \geq 0 \quad \forall \quad c_1, \dots, c_n \in \mathbb{R}, \quad (4.15)$$

namely if K_{ij} is positive definite. Subsequently, a positive definite kernel is a kernel that outputs a positive definite Gram matrix for elements in its original input set [58] [19].

Quantum Feature Map

Generally, we can say that every encoding of classical information on a quantum computer is in fact a mapping from the classical input space to a quantum state space. If this state space is

a Hilbert space, so that there exists an inner product for every pair of points within the space, we can call this mapping a feature map. Luckily, our quantum state space is per nature always a Hilbert space, and therefore we can generally say that encoding classical data on a quantum computer is a data-encoding feature map or "quantum feature map". Following the literature [38], we can either use Dirac vectors as feature vectors $x \rightarrow |\phi(x)\rangle$ or density matrices as feature encoding states $x \rightarrow \rho(x)$. Pure quantum states can be described by both representations. Mixed states on the other hand can only be described completely by density matrices, which makes the second encoding method more universal. For Dirac vectors the inner product of two vectors is defined by $\langle\phi(x)|\phi(x')\rangle$ and for density matrices it is simply the trace so that $\text{tr}\{\rho(x)\rho(x')\}$. In our quantum computational model, the mapping itself happens through the use of quantum gates that depend on the specific classical input x which can be described through a Unitary U , so that,

$$|\phi(x)\rangle = U(x)|0\rangle. \quad (4.16)$$

In summary, we can say that like classical feature maps, our quantum feature map projects from the classical input space to a higher dimensional quantum Hilbert space. We access the quantum Hilbert space through measurement and ideally, this measurement is trainable.

4.2.2 Quantum Kernels

We are doing this, because quantum models are always linear in some feature space, regardless if they are fault-tolerant or variational algorithms. Since the space we are mapping in is a Hilbert space, we can calculate inner products within the quantum feature space. And as we learned from classical kernel theory, we can use this to operate and train our quantum models in low-dimensional space, while the model itself explores the high-dimensional feature space. First, we need to define two quantum kernels that calculate the distance of two quantum states, one for pure and one for mixed states. For two pure states $\{x, x'\}$, the distance is simply the squared absolute of their inner product,

$$\kappa(x, x') = |\langle\phi(x')|\phi(x)\rangle|^2. \quad (4.17)$$

For two mixed states, we need to take the trace of the product of their density matrices $\rho(x)$ and $\rho(x')$ so that,

$$\kappa(x, x') = \text{tr}\{\rho(x')\rho(x)\}. \quad (4.18)$$

Further, with this kernel, we can define an alternative version of a feature space, the reproducing Kernel Hilbert Space (RKHS). It is a feature space of functions that are identical to the one observed in the respective quantum machine learning model and only depends on the kernel

presented in Equation 4.18. If we, for example, want to minimize the cost function over the space of our quantum models, we just need to minimize the cost over the RKHS of the quantum kernel. We minimize by finding the optimal measurement and through the representer theorem from above, we can show that those measurements can be written as kernel expansion in the data. We skip this for now, but under further consideration of the regularisation, we can then show that we can find the minima of a cost function that is used to train a quantum model solely through the optimal encoding of the data.[38] In other words and in comparison to variational models, we are guaranteed to find the global optima. In terms of optimization, its importance very much depends on the structure of the underlying optimization problem. If the optimization problem is convex, only one global optimum exists, which we then are guaranteed to find in a finite number of steps. But if the problem has many local minima, finding the global minimum might actually be not that important. Recent research might tend to the latter, which would partially explain the success of deep learning models over kernel methods for similar tasks. One should also consider the scalability here.

4.3 Quantum Image Representation

The following section deals with the essential methods to encode a classical image into a quantum mechanical system. It is the image processing extension to chapter ???. A classical image is usually defined as matrices of numbers that describe the intensity levels for each matrix entry, a pixel. Images are usually rendered in three different ways: colored, grayscale, or binary. In a color image, each pixel has three 'channels' corresponding to the intensity levels of the three colors of the additive color model red, green, and blue. Every color can be described as a mix of those three channels. In grayscale images, there is only one channel that describes the intensity value of each pixel. As the name suggests, the bounds for this intensity value are either white or black, resulting in some shade of grey for every value in between. Lastly, the most simple image form is the binary image that for example captures only black and white pixels. A comprehensive review of Quantum Image Processing can be found in [61].

Qubit Lattice

The Qubit Lattice model was one of the first methods for representing images in a quantum state. [62] In comparison to the other models, it does not convert digitally detected RGB or HSI values to phases or amplitudes but imagines a certain machine that is capable of translating the electromagnetic wave of input 'pixel' to a quantum state of a qubit. Given the nature of a

quantum system of the form,

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\gamma}\sin\frac{\theta}{2}|1\rangle \quad (4.19)$$

we can always find a real parameter θ that initializes qubits in a different state for different frequency values of some monochromatic electromagnetic wave. Notably, this works also for frequencies in other non-visible spectra. The image is then stored in a set of qubit lattices. Each qubit lattice represents a copy of the image in which every pixel is substituted by a qubit and in addition, we have k number of lattice 'copies' to retrieve the information later. We can see the qubit lattice method as a quantum-analog representation of a classical image. Since it uses no immediate quantum effect, it requires quite a lot of qubits and is therefore widely unused in current literature. But with the potential rise of photonic quantum computers and quantum sensors to enable the aforementioned translating 'machine', the algorithm and the underlying idea might experience a revival.

Flexible Representation of Quantum Images

The Flexible Representation of Quantum Images method (FRQI) uses angle encoding to store the respective image. The image as a quantum state can be written as,

$$|I(\theta)\rangle = \frac{1}{2^n} \sum_{i=0}^{2^n-1} (\sin(\theta_i)|0\rangle + \cos(\theta_i)|1\rangle) |i\rangle. \quad (4.20)$$

where,

$$\sin(\theta_i)|0\rangle + \cos(\theta_i)|1\rangle = |c_i\rangle. \quad (4.21)$$

$|i\rangle$ and $|c_i\rangle$ represent the position of the respective pixel and its intensity respectively. Here, the method shows the FRQI for a grayscale pixel value. It is shown in [63] that the encoding can be done in polynomial many steps with simple quantum operations. Next to its exponential space reduction advantage, FRQI grants also a setup that allows fast image operations such as geometrical transformations. Due to the use of real and not complex-valued coefficients, FRQI is also able to accurately retrieve the original image. FRQI can be extended to represent color or, more generally, multi-band images with the multi-channel representation for quantum images (MCQI). It was also further optimized as FRQCI in [64] and IFRQI in [65].

Novel Enhanced Quantum Representation

With the classical trade-off of the number of qubits and the lengths of the circuit, the Novel Enhanced Quantum Representation NEQR can achieve a quadratic speedup in preparing the

quantum image, by using more qubits. Through entangling the color information, a $2^n \times 2^n$ image is stored with location information so that,

$$|I\rangle = \frac{1}{2^n} \sum_{y=0}^{2^{2n}-1} \sum_{x=0}^{2^{2n}-1} |f(y, x)\rangle |yx\rangle. \quad (4.22)$$

Here, $f(y, x)$ is the pixel intensity at position (y, x) . Especially in the multi-band image cases, NEQR comes in handier and can perform more complex operations. Like FRQI, NEQR got improved since its introduction by [66] as INEQR and generalized by [67] and [68] as GNEQR and CQIR, respectively.

Quantum Probability Image Encoding

o encode an image into a quantum circuit, QPIE uses basic amplitude encoding techniques to store the pixel values and position. The number of qubits required for this method is given by,

$$n = \lceil \log_2 N \rceil, \quad (4.23)$$

where N is the number of pixels. Through generating the superposition of the n qubits, the number of represented pixel values is 2^n . If the number of pixels is not equal to any 2^n , we need to pad the input image to ceil up to the next viable number of qubits. Let us take a simple gray-scale 2x2 classical image first. We can define the image through its pixel intensities as,

$$I = (I_{yz})_{N_1 \times N_2}. \quad (4.24)$$

I_{xy} defines the pixel intensity at position (x, y) in an image made of $N_1 \times N_2$ pixels. Like amplitude encoding, QHED encodes the intensity values I as the respective amplitudes of its pixel position state within the overall superposition spanned over n qubits. We achieve this through normalizing the intensity value I so that we receive a normalized c_{xy} for every I_{xy} ,

$$c_i = \frac{I_{xy}}{\sqrt{\sum I_{xy}^2}}. \quad (4.25)$$

The resulting quantum state is,

$$|(Img)\rangle = c_0 |00\rangle + c_1 |01\rangle + c_2 |10\rangle + c_3 |11\rangle, \quad (4.26)$$

with integer sub-script indices for c and $\sum_{i=0}^{2^n-1} c_i = 1$. Similar to amplitude encoding, QPIE 'fits' n pixels into $\log(n)$ qubits, which could falsely be seen as some kind of compression. While it is true that we compressed the input information from its classical to a quantum representation,

we are not able to efficiently decompress back from quantum to classical. This would require 2^n steps. But fortunately, our classical output can also be the result of an operation on our quantum data and not the explicit processed input image itself.

Chapter 5

Methodology

This chapter describes the core method we use to perform a pixel-wise classification of hyperspectral images with a quantum computer. We use a quantum neural network and a quantum support vector machine to perform this task. For quantum neural networks, we define an amplitude efficient feature map and a strongly entangling Ansatz as our model. Both are hard to simulate classically and optimized by using a classical feedback loop, whose classical methods are also presented. Further, we present the classical support vector machine and lay out how we integrate the quantum kernel. An overview of different quantum kernel methods is given and a method to further improve performance through a projected quantum kernel is introduced. We close the chapter by presenting an approach to further train a form of parametrized quantum kernels and selecting a feature map.

5.1 Quantum Neural Networks

Generally, quantum neural networks combine the concepts of artificial neural networks and quantum computing. There are several usages of the name, ranging from describing the first efforts to model a quantum version of the feed-forward neural networks [69], quantum versions of Boltzmann machines [70] or in general for variational quantum circuits. Note that all the above-mentioned ways to implement a "quantum neural network" fundamentally differ from each other. In the concept described in [69], the authors imitate the same linear/non-linear structure on a quantum computer that is present in a classical neural network through the inclusion of linear layers and non-linear activation functions. Boltzmann machines are rather graphical models that work as stochastic recurrent neural networks. Quantum computers work here as a sampler, mainly due to their probabilistic nature.

Then there is the notion of quantum neural networks as variational quantum circuits. Here, the term quantum neural networks refers more to the strikingly similar structure to classical

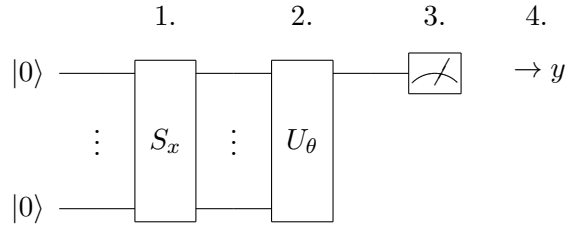


Figure 5.1. Typical four steps of a quantum neural network: 1. State preparation, 2. Model Circuit, 3. Measurement and 4. Postprocessing

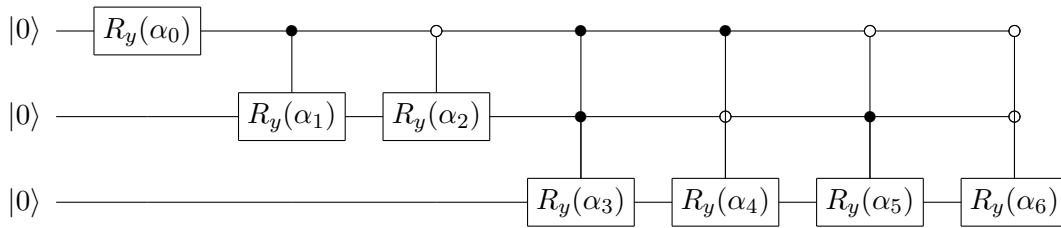


Figure 5.2. Amplitude encoding scheme for $n = 3$ qubits. Given $N = 8$ datapoints, we require $N - 1$ angles $(\alpha_0, \dots, \alpha_{N-1})$ to encode the information into the amplitudes of 2^n states.

neural networks, than the mathematical proximity. We have encoding and hidden layers, as well as parameters for which we need to find gradients. NISQ era algorithms are hybrid quantum-classical algorithms, meaning that while the parametrized circuit is executed on a quantum device or quantum processing unit, the parameters are optimized through a classical feedback loop. The following describes the two parts in more detail and the corresponding methods we are using to perform the desired classification.

Quantum Processing Unit

In our implementation of a QNN, the QPU will load the classical data onto quantum hardware and compute a prediction value by the use a parametrized circuit and subsequent measurement. To perform the classification of a hyperspectral image, the QNN is divided into four steps. The following will describe the four steps and our methodological choice as well as its implementation. Figure 5.1 can be seen as accompanying graphical support.

Step 1: State Preparation

We decide to use a variation of the previously presented amplitude encoding method, the Möttönen state preparation [71]. Amplitude encoding methods have high popularity since we can

develop algorithms that perform computations on 2^n amplitudes with a polynomial number of qubits n . Here we trade circuit-width against circuit-depth.¹ For arbitrary state preparation, the theoretical lower bound of depth is given by $\frac{1}{n}2^n$ [38]. This is given by the fact that even though we used the superposition over all qubits to represent our state space, we still need to encode the classical data into every single amplitude. It is achieved through a series of expensive controlled rotation gates, for which we can see an example implementation in Figure 5.2. The Moettoenen state preparation essentially initializes the same state using a sequence of uniformly controlled rotations, but in linear time and is therefore considered an amplitude efficient state preparation. Depending on the structure of the data, we might also achieve polylogarithmic runtimes with the number of features [72], [73]. It takes as input any real or complex-valued vector $x \in \mathbb{C}^N$. We require the input to be normalized so that $\sum_i |x_i|^2 = 1$ and the number of elements in x is a power of two. For most datasets, this is not the case. The length of x can simply be solved by padding the input, that is, by adding arbitrary values like zero until the next power of two is reached. Normalization is more difficult, since the distance between data points can be distorted. We can circumvent this by embedding the data in a higher dimension, similar to distance preserving feature maps, which is achieved through padding the data first. Given an input vector x_1, \dots, x_N and the padding terms p_1, \dots, p_M , we can define the preprocessing as,

$$\mathbf{Norm} = \frac{1}{\sqrt{\sum_i x_i^2 + \sum_j |p_j|^2}}, \quad (5.1)$$

so that,

$$(x_1, \dots, x_N)^T \rightarrow \mathbf{Norm}(x_1, \dots, x_N, p_1, \dots, p_M)^T. \quad (5.2)$$

Further, padding and normalization prevent the input vector from being homogenous to the encoded state vector and add additional dimension [48]. In summary, since our input data is in the order of magnitude 2, we choose an encoding that minimizes required qubits. In that way, we can fit all data of a hyperspectral image on a NISQ-era circuit, without the need for dimensionality reduction.

Step 2: Model Circuit

Next, we select a model or Ansatz that resembles the expressibility of a neural network and can classify the input. In addition, we want it to be low in-depth and wide in reach within the Hilbert space. Such shallow quantum circuits show promising results as universal function approximators and seem to be at least as easy to handle as classical neural networks [74], [75] Here, we select the quantum equivalent to a fully connected layer, the strongly entangled

¹Width and depth of a circuit describe the number of qubits and the number of gates respectively.

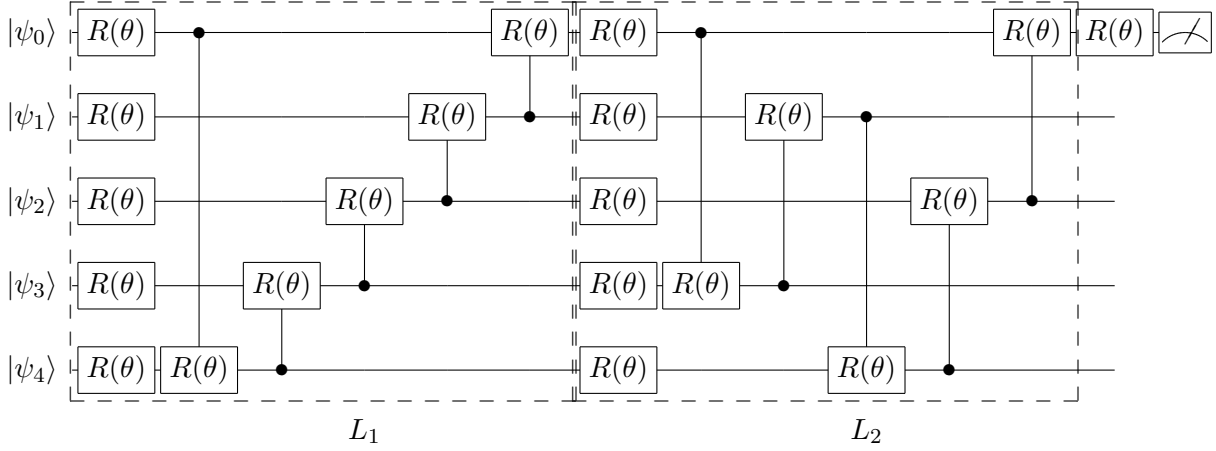


Figure 5.3. Strongly entangled layers Ansatz consisting of two blocks L_1 and $L_2R(\theta)$ are rotation gates with an individual set of parameters $\theta = (\phi, \gamma, \omega)$ for every gate. For this configuration only the first qubit gets measured.

layer. A strongly entangled layer is generic for shallow circuits that have the goal to maximally entangle the input states through parametrized gates. We choose an architecture proposed in [48]. As shown in Figure 5.3, the layer consists of two blocks. Each block is structured in the same fashion: we have an all qubit spanning single qubit rotation gates $R(\phi, \gamma, \omega)$ layer and a layer of controlled rotation gates $CR(\phi, \gamma, \omega)$. (ϕ, γ, ω) define the set of rotation angles around the respective axis of the Bloch sphere. Those are ring-connecting in the first block and oscillating as given in Figure 5.3 in the second block. The single qubit rotation gate is defined as,

$$R(\phi, \gamma, \omega) = RZ(\omega)RY(\gamma)RZ(\phi) = \begin{bmatrix} e^{-i(\phi+\omega)/2} \cos(\gamma/2) & -e^{i(\phi-\omega)/2} \sin(\gamma/2) \\ e^{-i(\phi-\omega)/2} \sin(\gamma/2) & e^{i(\phi+\omega)/2} \cos(\gamma/2) \end{bmatrix}, \quad (5.3)$$

while the controlled qubit rotation is given by,

$$CR(\phi, \gamma, \omega) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{-i(\phi+\omega)/2} \cos(\gamma/2) & -e^{i(\phi-\omega)/2} \sin(\gamma/2) \\ 0 & 0 & e^{-i(\phi-\omega)/2} \sin(\gamma/2) & e^{i(\phi+\omega)/2} \cos(\gamma/2) \end{bmatrix}. \quad (5.4)$$

The rotation angles are thus our trainable parameters.

To verify the trainability, we can provide the gradient recipe for each gate. For $R(\phi, \gamma, \omega)$, the

recipe holds for every angle parameter so that,

$$\frac{d}{d\phi}f(R(\phi, \gamma, \omega)) = \frac{1}{2}[f(R(\phi + \pi/2, \gamma, \omega)) - f(R(\phi - \pi/2, \gamma, \omega))], \quad (5.5)$$

and for controlled rotation gate $CR(\phi, \gamma, \omega)$ we get,

$$\frac{d}{d\mathbf{x}_i}f(CR(\mathbf{x}_i)) = c_+[f(CR(\mathbf{x}_i + a)) - f(CR(\mathbf{x}_i - a))] \quad (5.6)$$

$$- c_-[f(CR(\mathbf{x}_i + b)) - f(CR(\mathbf{x}_i - b))]. \quad (5.7)$$

In both equations, f is the expectation value depending on the respective gate. Equation 5.6 additionally satisfies a specific parameter-shift rule as proven in [76]. Further, the variables in Equation 5.6 are defined as $a = \pi/2$, $b = 3\pi/2$ and $c_{\pm} = (\sqrt{2} \pm 1)/4\sqrt{2}$.² While the number of parameters is fixed for this layer structure, the number of layers is a changeable hyperparameter. As a rule of thumb, we should choose the number of layers so that the number of tuneable parameters matches the length of the feature vector [77].

Step 3: Measurement

In this case, we perform binary classification and Figure 5.3 shows the measurement of one qubit, typically in the Z -basis with an expectation value that ranges from $f(x, \theta) \in [-1, 1]$. Here one major difference between simulators and real quantum devices gets obvious. While simulators can calculate an expectation value, quantum circuits can only be measured once and produce either -1 or 1, a binary output. To get an equivalent estimation value, we need to run the circuit a sufficient number of times and calculate the estimation value from the collected samples. This overhead increases with the precision of the estimation value.

Step 4: Postprocessing

After sampling the estimation value, we simply plug the output in a threshold function so that,

$$f(x; \theta) = \begin{cases} 1 & \text{if } \mathbb{E}(x; \theta) \geq 0 \\ 0 & \text{else} \end{cases} \quad (5.8)$$

So given a measurement in the Z -basis, we classify a given input as 1 if the expectation value is above or equal zero and one if the expectation value is below zero.

²For further information on possible gates, state preparations and differentiable quantum methods, we recommend the documentation of the PennyLane framework.

Summary

To summarize, in Step 1 we loaded the classical data onto the quantum computer via a state preparation routine. This routine can be seen as a non-linear map, or feature map. In Step 2, we applied a unitary transformation in form of a strongly entangled Ansatz on the prepared state. The unitary transformation acts as a linear layer, which is one of the core mathematical differences from classical machine learning. Here, activation functions follow every layer to bring in non-linearity. In discrete quantum machine learning, measurements are one of the few actions that can introduce non-linearity. This turns Step 3 into a nonlinear layer with one neuron. We now leave the quantum processing unit and discuss the classical feedback loop.

Classical feedback loop

The classical feedback loop has the task to optimize the models parameters. To achieve this, we need to define a loss function, calculate the gradient and define an optimization method.

Loss function

As a loss function, we choose the least-square method. Essentially, given a dataset of tuples consisting features and labels, the least square methods calculates the sum of the squared subtraction of predictions and labels so that,

$$LSM = \frac{1}{2} \sum_{m=1}^M |f(x^m, \theta) - y^m|^2. \quad (5.9)$$

Since the loss function does not need to be further computed on a quantum device, we also add L1 or L2 regularization terms [38].

Gradient extraction

On a QPU, we can not directly extract the gradient, since we do not have access to the circuit itself. But as presented in ??, we can make use of the parameter-shift rule. It calculates the analytical gradient with little overhead. For cases where this is not possible, we can also use finite difference or similar methods. There are also promising results using a quantum natural gradient [78].

Optimization Method

As an optimization method, we choose stochastic gradient descent. Classical gradient descent updates parameters following the rule,

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla \mathcal{L}(\theta^{(t)}). \quad (5.10)$$

Stochastic gradient descent, given the step-size η and g^t being a sequence of random variables, is a small modification so that,

$$\theta^{(t+1)} = \theta^{(t)} - \eta g^{(t)}(\theta^{(t)}). \quad (5.11)$$

It is mostly preferred over classical gradient descent because it helps escape local minima, it has better convergence properties and g^t can be computed way more efficiently than \mathcal{L} [48].

5.2 Quantum Support Vector Machine

After presenting methods concerning quantum neural networks, we come back to kernel methods. More precise, we show how we can use quantum kernel methods within support vector machines (SVM) to perform classification tasks.

Step 1: Classification Method

To use kernel methods for classification, we integrate them into support vector machines. Support vector machines as kernel-based classification methods were introduced in the 90s and are one of the best-known classification methods in machine learning [79], [80].³ They linearly separate two classes by fitting a margin-maximizing hyperplane, which also leads to the equivalent term of the maximum-margin classifier. The margin describes the distance from the hyperplane to the nearest data points of each class, the support vectors. Through maximization of the margins, we find the best possible position for the separating hyperplane that acts as a decision boundary of the classifier. Figure 5.4 shows a dataset of two classes, respectively labeled by filled or empty circles. The classification follows a simple scheme. Let us consider the linear model,

$$f_w(x) = w^T x + b, \quad (5.12)$$

where w is a set of parameters, x is a feature vector and b is some bias. w is the normal vector to the hyperplane, which is defined by all the points that satisfy $w^T x + b = 0$. The resulting

³In recent years, especially with the rise of deep and convolutional networks they lost quite some traction. This is not due to their performance, but more due to their computational cost. A potential angle of attack for quantum algorithms.

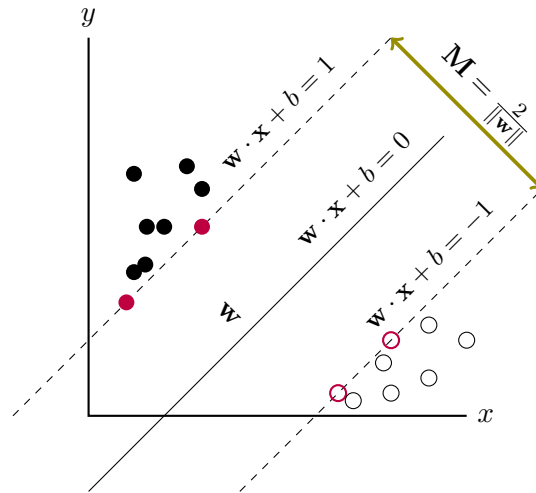


Figure 5.4. Support Vector Machine that maximizes the margin M . The separating hyperplane are all points satisfying $w \cdot x + b = 0$, where w is a set of parameters, x is a feature vector and b is some bias. The closest points of each dataset to the hyperplane are the support vectors, here purple colored datapoints.

support vector machine classifies a given input by assigning it the output -1 if $f_w(x_i) < 0$ and 1 if $f_w(x_i) > 0$ for the i -th data point. To make the support vector machine also classify non-linearly separable datasets, we let it act on the data in the feature space. So similarly to Equation 4.8, we use the previously defined kernel and define the classifier as $f_w(x) = w^T \phi(x) + b$. To complete the description of Fig. Figure 5.4, $\frac{b}{\|w\|}$ describes the offset of the hyperplane to the origin and the margin is $M = \frac{2}{\|w\|}$. Thus, maximization of the margin requires minimization of $\|w\|$. To formulate the actual optimization problem, we further need to define a loss function. Due to its strong performance and applicability to soft-margins⁴, we choose the hinge-loss,

$$l(y_i) = \max(0, 1 - y_i(w^T x_i + b)). \quad (5.13)$$

Here, $w^T x_i + b$ is the output of $f_w(x_i)$ and y_i is the i -th target label. Support vector machines are designed for binary classification tasks. To perform multi-class classification, we can use the One-vs-Rest or One-vs-One strategy. One-vs-Rest creates a binary classification for every class and compares it with the rest. One-vs-One considers each pair of classes and creates a binary classification for each. The strategy of choice depends on the number of labels and the dataset.

⁴Depending on whether the data is linearly separable or not, we use either hard- or soft-margins respectively. Further information can be found in [79]

Step 2: Quantum Kernel

To maximize the margins, we need to know the distances between each pair of data points. To obtain this information, we calculate the Gram matrix as defined in Equation 4.14. While we use inner products classically, we define a quantum circuit that performs the equivalent distance measurement.

We first show the realization of the kernel/overlap calculation, then we introduce performance-enhancing projected kernels and how we can also train the kernels. Finally, we show how we can integrate the quantum kernel implementations into a classical support vector machine to classify hyperspectral images.

Implementing the Overlap Calculation

In the following, we will present two ways of calculating the overlap of two quantum states, depending on whether they are pure or mixed.⁵

Kernel implementation for pure states For the first method, we make use of the fact that a unitary embedding results solely in a pure state. A known and also previously mentioned property of unitaries,

$$U * U = UU^* = UU^{-1} = I, \quad (5.14)$$

or, more specifically, as we work with the Hermitian adjoint of the matrix,

$$U^\dagger U = UU^\dagger = I. \quad (5.15)$$

This boils down to the following: Given our data-embedding unitary $U(x)$, if we are able to construct its Hermitian adjoint U^\dagger , we can calculate the overlap of two states $U(x), U^\dagger(x)$ by simply applying the unitaries one after the other while one of them is its Hermitian adjoint. If we have identical states and we started in a certain computational basis state like $|0\rangle$, we should measure the same basis state afterward. Every other measurement and also its difference from the basis states indicates the distance of the two encoded data points x, x' . Mathematically, we can write this overlap as,

$$\kappa(x, x') = |\langle \phi(x') | \phi(x) \rangle|^2 = |\langle 0 | U^\dagger(x') U(x) | 0 \rangle|^2. \quad (5.16)$$

Kernel implementation of mixed states To estimate the overlap of two mixed states, we can make use of a well-known quantum algorithm, the SWAP test. The SWAP test as

⁵We use the term mixed more loosely here. The presented method will not only work for mixed states, but also for pure ones.

well calculates the absolute squared value of two arbitrary quantum states, but through the measurement of an ancilla qubits' probability. To achieve this, we put an ancilla qubit into a superposition and perform a controlled swap operation of the two quantum states following another Hadamard gate on the ancilla qubit. The resulting interference under superposition and the subsequent measurement reveals the actual overlap the two states. The probability of measuring the ancilla qubit in the original computational basis state $P(|anc\rangle == |0\rangle)$ is,

$$p_0 = \frac{1}{2} - \frac{1}{2} |\langle x|x'\rangle|^2, \quad (5.17)$$

for pure states and,

$$p_0 = \frac{1}{2} - \frac{1}{2} tr\{ab\}, \quad (5.18)$$

for mixed states. The overlap is then given by,

$$|\langle x|x'\rangle|^2 = 1 - 2p_0 \quad \text{and} \quad tr\{ab\} = 1 - 2p_0. \quad (5.19)$$

The SWAP test method can also be used for pure states and must be used for pure states where the embedding unitary can not be inverted.

The methods presented thus far can be described as fidelity measurements. Using these methods, we are able to estimate the fidelity up to an accuracy of $\pm\sqrt{F(1-F)/k}$ where F is the overlap/fidelity measurement and k are the number of repetitions. For the implementing circuit, we can again see a width vs. depth trade-of. While the SWAP test method requires $2n + 1$ qubits, where n is the number of qubits required to represent the state in the feature space, the method presented in Equation 5.16 needs only n qubits but twice the circuit-depth. For example, an embedding using 20 qubits to operate in a 2^{20} dimensional Hilbert space would subsequently require 41 qubits to implement the SWAP method. Apart from the presented fidelity measurements, there are also other measurements known in quantum information theory. A prominent one is the Helstrom measurement which is known to be optimal to discriminate two quantum states. Even though its performance is very efficient, we do not consider it in this work since the required circuit size is much larger compared to the presented fidelity measurements.

Projected Quantum Kernels

While the power of quantum kernels lies in their potential high dimensionality, this is where one of its major flaws lies, too. While the corresponding Hilbert space's dimensionality grows exponentially with the number of qubits, at some point, all quantum states will be perpendicular to each other. This leads to them being indistinguishable from each other and the kernel/gram matrix being an identity matrix. This problem occurs, when we calculate the kernels in the

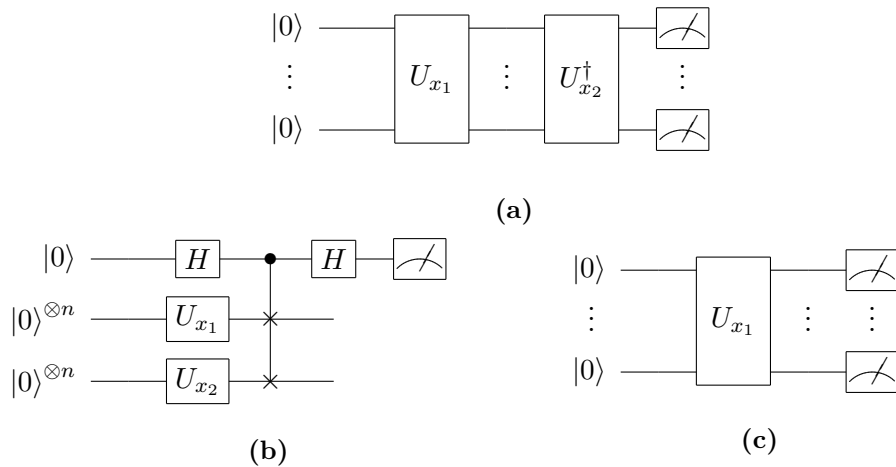


Figure 5.5. Three possible implementations of a quantum kernel: (a) Adjoint method (b) SWAP-Test (c) Projected Kernel. U is an arbitrary unitary encoding the datapoints (x_1, x_2) .

quantum space, but as proposed in [77], we can use projected quantum kernels to solve this problem. For this, we identify the relevant features in the quantum Hilbert space and project them back into the (approximate) classical space to circumvent the dimensionality problem. This is achieved by using reduced physical observables like partial traces or classical shadows. Subsequently, we combine the advantage of low dimensionality of the classical space and the valuable features from the quantum feature/Hilbert space, which reduces the complexity of training. A visual comparison of the respective circuits of the presented kernels can be seen in Figure 5.5. To project the data back from the quantum space to the classical space we can select from a variety of linear and non-linear kernel methods. We choose a simple and yet powerful projected quantum kernel $\kappa^{Proj}(x, x')$ as presented in [77] so that,

$$\kappa^{Proj}(x, x') = \exp \left(-\gamma \sum_{\kappa} \sum_{P \in X, Y, Z} (\mathbf{Tr}(P\rho(x)_\kappa) - \mathbf{Tr}(P\rho(x')_\kappa))^2 \right). \quad (5.20)$$

Like the classical and quantum kernel, the projected quantum kernel calculates the entries of the respective kernel or Gram matrix. In Equation 5.20, ρ is the density matrix of the state after applying the initial quantum feature map. Following that, we sum up the measurements in all of the Pauli operators P , while γ acts as a positive, tuneable hyperparameter to optimize prediction accuracy. Our implementation calculates the inner product of the feature vectors classically while the feature vectors themselves being the result of a quantum feature map. Figure 5.5 provides an overview of the implementation of the kernel methods presented so far.

Training Quantum Kernels

So far we only talked about fixed kernels, meaning we have a single, non-parametrized unitary which is not subject to any optimization. Theoretically, we can learn every function that a parametrized quantum circuit can learn, but this might require an exponential number of samples [77].⁶ It strongly depends on the kernel or feature-map we choose, a problem that is also present in classical kernel methods. One attempt to solve this problem is building a bridge to the theory of parametrized quantum circuits and by making use of their similar structure. We simply define a data fitting quantum feature map and add some freely tuneable parameters, converting a part of the feature map, the ansatz, into a variational quantum circuit. Through optimizing the parameters, we then try to find a better version of our kernel e.g. a kernel that leads to a higher accuracy if we use it to train a support vector machine to classify given data. One way of finding the optimal parameter is simply using an exhaustive search in a pre defined set of parameters. Unfortunately, this grows exponentially in the number of parameters since we need to calculate the accuracy of each parameter in the set of parameters for every possible combination of the remaining parameters. Thus it is only suitable for parametrized feature maps with few parameters. For more parameters, [60] proposed using the kernel-target alignment method from [81]. Essentially, it uses the labels of the training set to form an ideal kernel which kernel matrix we get by calculating the outer product of the label vectors. The alignment comes from the idea to compare this perfect kernel with the kernel we have geometrically since we can see the kernel matrices themselves as a multidimensional vector in some space. To calculate the distance of two kernel matrices in terms of their angle, the alignment, we use the Frobenius inner product. Given two kernel matrices A and B , the Frobenius inner product is defined as,

$$\langle A, B \rangle = \sum_{i,j} A_{ij}^* B_{ij} = \text{Tr}(A^\dagger B), \quad (5.21)$$

where the asterisk represents the complex conjugate of matrix A and the dagger represents the Hermitian conjugate of B . Like calculating the alignment of two vectors, we can now calculate the kernel-target alignment $\Xi(K)$ by using the Frobenius inner product $\langle \cdot \rangle$ on the optimal (label)

⁶Parametrized quantum circuits can be seen as an encoding with a subsequent optimization of a measurement. This is strikingly similar to our definition of quantum kernel methods. See Appendix A for more information.

kernel \bar{K} and the current kernel K and normalizing it so that,

$$\Xi(K) = \frac{\langle K, \bar{K} \rangle}{\sqrt{\langle K, K \rangle, \langle \bar{K}, \bar{K} \rangle}} \quad (5.22)$$

$$= \frac{\sum_{ij} y_i y_j k(x_i, x_j)}{\sqrt{\left(\sum_{ij} k(x_i, x_j)^2\right) \left(\sum_{ij} y_i^2 y_j^2\right)}} \quad (5.23)$$

$$= \frac{\sum_{ij} y_i y_j k(x_i, x_j)}{n \sqrt{\left(\sum_{ij} k(x_i, x_j)^2\right)}}. \quad (5.24)$$

Equation 5.22 and Equation 5.23 represent the same, but Equation 5.23 is using the kernel function and training data. Since the alignment value for $\Xi(K)$ will be between $[-1, 1]$, where -1 is the kernel being the negative vector of the target and 1 is perfect alignment, $y^2 = 1$ and n is the number of samples. Here the kernel-target alignment $\Xi(K)$ assumes a balanced dataset, but [60] extended it also to unbalanced datasets by rescaling the labels. ⁷

Step 3: Feature Map

Since we do want to harness the full power of quantum computing, we need to define quantum feature maps. If the goal is to show quantum advantage, the feature map must at least be classically hard to simulate. We will use Instantaneous Quantum Polynomial time encoding (IQP) as presented in Equation 3.29 and test it against angle encoding. We choose IQP, because it is hard to simulate classically, hardware efficient and promises a larger feature space through highly entangled input states. Further, we gain an additional hyperparameter that specifies how often the IQP circuit will be repeated. Repetition increases the computational cost for both simulators and quantum computers, but also makes the embedding extra complex through interference.

⁷A dataset is balanced if each class has the same number of samples.

Chapter 6

Implementation

In this section, we use the previously introduced methods and implement them using Xanadu’s PennyLane framework. We optimize the baseline simulator method using JAX so that we achieve a two order of magnitude runtime decrease for classifying the Pavia University hyperspectral image dataset. The results are comparable accuracy scores to the classical support vector machine benchmark implementation. This shows that the implementation of all three methods was successful.

6.1 Pavia University Dataset

Hyperspectral imaging is a new technology that leads to a little pool of publicly available datasets. This will likely change in the foreseeable future through missions like the recently launched EnMap hyperspectral spacecraft, sent into space by the German Aerospace Centre. We apply our methods to the Pavia University dataset, a publicly available dataset that consists of a 610x340 pixel hyperspectral image of Pavia University in Italy. Figure 6.1 shows its unprocessed three RGB channels and the respective label ground-truth. It was taken from an aircraft using the ROSIS sensor with a high spatial resolution of 1.3 meters. Each pixel has a spectral resolution of 103 bands, covering a spectral range from 0.43 - 0.84 μm . The dataset contains nine classes comprising common urban surface materials like metal, asphalt, vegetation, or water. For classification, we drop the undefined pixel class as well as some pixels that have no spectral information. Other than that no preprocessing is required.

6.1.1 Dimensionality Reduction

Depending on our embedding strategy and the used hardware, we must reduce the dimensionality of the Pavia University dataset. Current real-world quantum hardware is limited to a few noisy qubits, we use simulators that can handle up to 30 qubits. We distinguish between

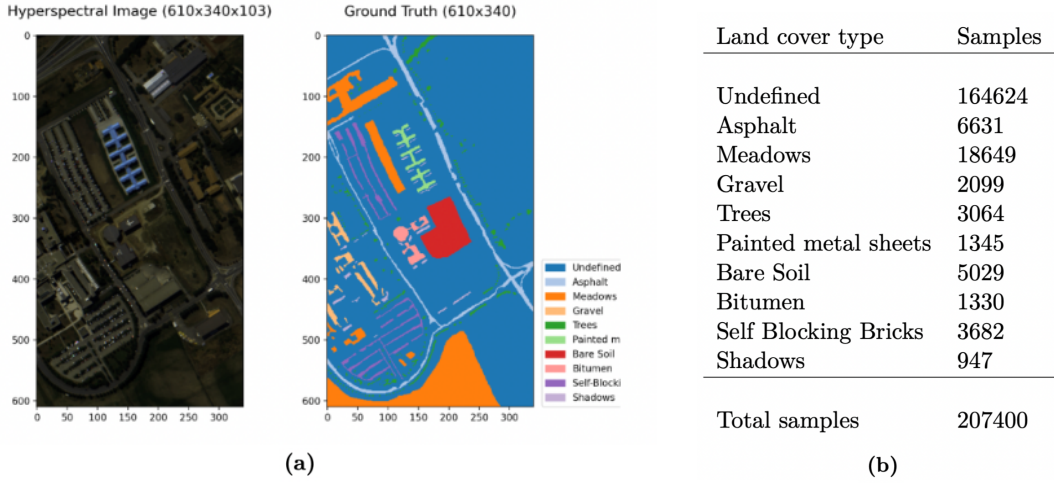


Figure 6.1. (a) Pavia University dataset, only the red-blue-green channels on the left side and the ground truth with corresponding labels on the right side, (b) Number of samples per class.

two main encoding strategies, linear or sub-linear mappings. Amplitude encoding, a sub-linear mapping, requires \sqrt{x} qubits with x datapoints. IQC embedding on the other hand is a linear mapping and requires x qubits for x datapoints. Since our dataset has 103 bands for each pixel, thus is a 103-dimensional input vector, we need to reduce the number of dimensions so that we can embed it with the current hard- and software. We choose the Principal Component Analysis (PCA) for this task. It is to date one of the most common approaches to reduce dimensionality, and effectively defines an orthogonal projection of the data onto a lower dimensional subspace [82].¹ The resulting principal components replace the feature in our input feature vector. The goal is to pin down how many components we need to explain a certain threshold of variance in the original data. We see the cumulative explained variance with a growing number of principal components in Figure 6.2. After setting the cut-off threshold to 95% explained variance, we see that 3 principal components are necessary. We reduced the number of dimensions from 103 to 3 and can now embed the data with any embedding technique since we require a maximum of three qubits.² However, we need to be careful when applying unsupervised reduction methods. We might explain most of the variance in the data through the extracted principal components but we can still lose what might be important for classification.

¹Interestingly, there also exists a quantum version of the PCA due to efficient execution of linear algebra methods on quantum computers [83].

²There are also embedding techniques that require additional ancilla qubits, but they are not subject to this thesis.

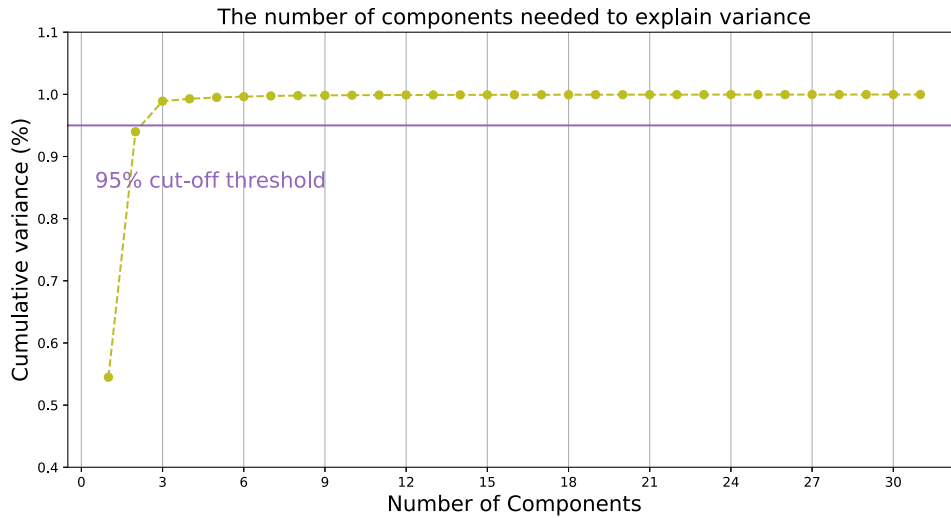


Figure 6.2. Cumulative explained variance with growing number of principal components. We break the threshold of 95% explained variance with 3 principal components.

6.1.2 Framework

Most quantum software is linked with a specific hardware producer. This comes in very naturally, since at this stage of development most producers have the system’s close knowledge and are also interested in finding users. The most used programming frameworks for quantum computers are qiskit [37] (IBM), PennyLane [84] (XANADU) and cirq [85] (Google). Extending on that are TensorFlow quantum [86], the quantum version of the machine learning package TensorFlow, and Pytorch as well as JAX integration for PennyLane. We execute all our algorithms using PennyLane. It is designed as a cross-platform library for differentiable programming of quantum computers. Cross-platform means that we can use multiple backends to execute the circuits we are designing. This ranges from PennyLane’s quantum simulator, over IBM’s super-conducting qubit hardware to Strawberry Fields photonic simulators. It focuses on differentiable programming through built-in automatic differentiation methods of the quantum circuits. Hence it is specially designed for NISQ era hybrid quantum computing algorithms and is compatible with existing machine learning frameworks like JAX or PyTorch. In PennyLane, circuits are defined as so-called QNodes. QNodes are differentiable and can handle classical and quantum input. They also output either classical or quantum data, which makes them seamlessly integrate into classical machine learning pipelines. For our implementation, we used the JAX interface for PennyLane and further optimized it. JAX uses just-in-time compilation for speeding up linear algebra operations while supporting parallelization and GPUs/TPUs. This results in some order of magnitude performance increase, which made most of the experiments

feasible to execute on a MacBook Pro M1 2021. Through PennyLane’s device interface, we are also able to compile and execute our circuits on a diverse set of real quantum backends such as IBM’s superconducting or Xanadu’s photonic quantum hardware.

6.2 Spectral classification methods

All methods we implement are supervised spectral classification methods. They perform pixel-wise classification, considering only the respective spectrum. It is worth noting, that today’s best-performing machine learning methods in remote sensing are mostly spatial-spectral classification methods[16]. But due to their large input size, they are not considered for our implementation.

Quantum Support Vector Machine

For the implementation of the QSVM, we use a classical support vector machine and a quantum kernel to classify our dataset. Since we use noise-less simulators to execute our quantum circuits, we choose the adjoint overlap method as our kernel. Further, we define IQP embedding as our feature map. The quantum circuit itself runs on PennyLanes ’default.qubit’ backend and is optimized using JAX and just-in-time compilation to speed up the qubit simulation. We use scikit-learn for the support vector machine implementation where our quantum kernel is given as a custom kernel to fit the data. We implement the quantum kernel using three qubits and six repetitions for the IQP embedding.

Projected Quantum Support Vector Machine

The projected quantum support vector machine runs identical to the presented quantum support vector machine with the difference of using a projected quantum kernel. We implement the projected quantum kernel by first defining the feature map as IQP embedding with subsequent measurement of the expectation value of the three Pauli observables ($\sigma_x, \sigma_y, \sigma_z$) over all qubits. Then we define a gaussian kernel κ_g that takes the previously defined feature map ϕ and calculates the distance of two datapoints x_1 and x_2 such that,

$$\kappa_g(x_1, x_2) = \exp -\gamma \sum (\phi(x_1) - \phi(x_2))^2, \quad (6.1)$$

where γ is a hyperparameter set to 20 in our implementation. We use κ_g as the custom kernel in scikit-learn’s support vector machine while also utilizing three qubits and six repetitions for the IQP embedding.

Method	Pavia University Accuracy
SVM linear	0.78
SVM rbf	0.88
QSVM	0.73
PQSVM	0.79
QNN	0.78

Table 6.1. Experimental results of the three quantum models compared to two classical support vector machines with linear and radial basis function kernel. The abbreviations translate to Support Vector Machine (SVM), Quantum Support Vector Machine (QSVM), Projected Quantum Support Vector Machine (PQSVM) and Quantum Neural Network (QNN).

Quantum Neural Network

For the QNN we choose amplitude encoding as embedding and the strongly entangled layer as presented in chapter section 5.1 as Ansatz. We define the resulting QNode using a PyTorch interface to make it a PyTorch layer and run it on the PennyLane 'default.qubit' backend. The QNode is then together with a subsequent softmax layer integrated into a sequential PyTorch model. The model parameters are optimized using stochastic gradient descent at a learning rate of 0.5 and the hinge loss as loss function. Since we measure only one qubit, we perform a binary classification task and have to apply the one-vs-rest approach to classify all classes.

6.3 Experiments

Before we applied the presented methods, we split the dataset into train and test sets. Sampling from a hyperspectral image is not a trivial task, since neighboring pixels tend to be highly correlated. We divide our dataset into test and train sets by randomly sampling over the whole image, excluding the dropped labels. The train set is chosen to be 20% of the original dataset, a particularly small share to ensure reasonable training times. Since our focus lies on demonstrating the implementation and not scoring the highest possible performance, we avoid further optimization to guarantee comparability. The resulting accuracies on the Pavia University test set are presented in Table 6.1. To justify the hyperparameter selection of our best performing model, namely the number of repetitions for the IQP embedding, we show the change of performance for increasing repetitions in Figure 6.4. Further, we show PQSVM's prediction as overlay next to the ground truth in Figure 6.3. The results show comparable results for all three quantum machine learning implementations to the two baseline support vector machine methods. This leads us to the conclusion that the implementation of the methods were successful and a pixel-wise classification was performed with an accuracy of up to 79%.

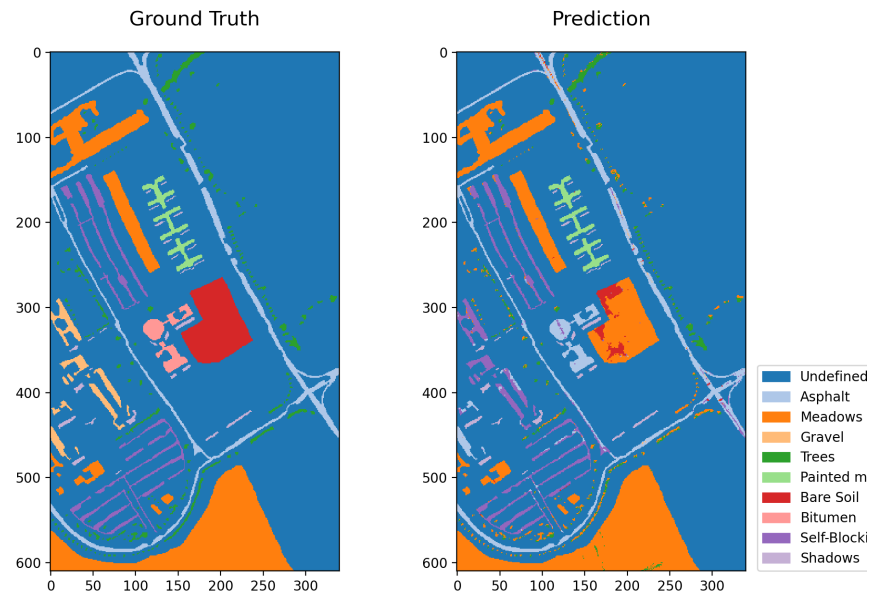


Figure 6.3. Ground truth and predicted classes of the Projected Quantum Support Vector Machine. To generate the predicted classes image, we overlaid the predicted classes for test and training set as well as the previously dropped undefined classes.

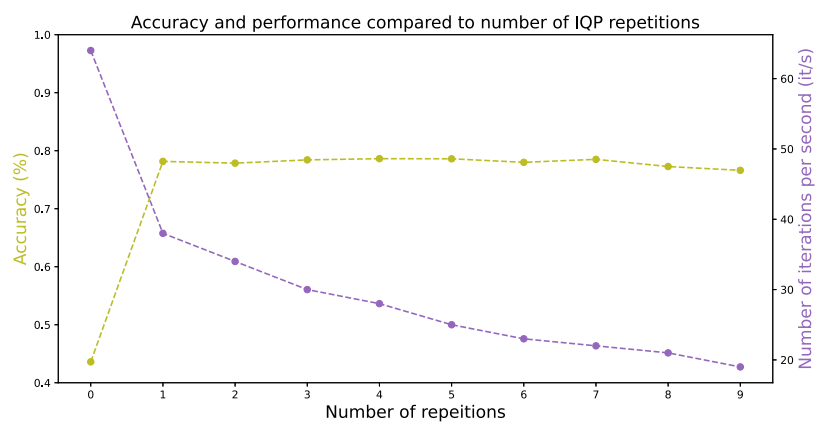


Figure 6.4. Comparison of accuracy and performance of the Projected Quantum Support Vector Machine with an increasing number of IQP repetitions.

Chapter 7

Conclusion

This thesis implemented three supervised learning methods to classify hyperspectral images on a quantum computer. We applied our models to the Pavia University dataset which consists of a 610x320 pixel scene of the University District in Pavia, Italy. It was taken by an aircraft using the ROSIS sensor and has a spectral resolution of 103 bands per pixel. As NISQ-era quantum devices are scarce in available qubits, we extracted the principal components to reduce the dimensionality. We first implemented a quantum support vector machine, that uses a quantum kernel to project the classical input data into a quantum feature space. Then, a classical support vector machine is applied to find the separating hyperplane. Our focus was on designing a quantum kernel that is optimized for the classical hyperspectral image data and the classification results showed that the implementation was successful. We improved the quantum support vector machine by introducing a method that projects the data back into classical space before fitting the model. With all other hyperparameters being equal, it showed an increase in classification accuracy over the standard quantum support vector machine. Further, we designed a trainable quantum neural network that acts similarly to a classical feed-forward neural network. The challenge here was to find a state preparation that encodes the classical hyperspectral data and a variational quantum circuit that acts as a quantum machine learning model to process the encoded data. We successfully implemented a state preparation that encodes the classical hyperspectral data with a sub-linear number of qubits in linear time with the number of data points. As the subsequent model, we chose a strongly entangled variational circuit that concludes with a single qubit measurement. The model gets evaluated using the hinge loss function and the parameters are updated using a stochastic gradient descent method. Similar to the quantum support vector machine, the quantum neural network reached accuracies comparable to the classical benchmark method and therefore indicates a successful implementation. Since both models are binary classification models, we classified the dataset using the one-vs-rest principle. In our implementation, we used the PennyLane framework for

all quantum methods and scikit-learn as well as PyTorch for the classical machine learning routines. Additionally, we optimized the underlying quantum simulation tool using JAX and achieved two orders of magnitude increase in runtime performance. All presented models are also scalable so that with the growing availability of larger circuits, we can fit larger datasets. Although, we are unsure of its effects on the growing parameter optimization landscape.

Future Work Expanding on this thesis, there are several subjects to work on in the future. On a theoretical note, we still lack a full understanding of the importance of classical data structures for quantum machine learning algorithms. We should investigate if hyperspectral data has an underlying structure that might be exploitable and whether we can find specific and efficient encoding methods for it. For quantum kernel methods and quantum neural networks, one could explore the expressibility of different feature maps and model circuits, the effect of trainable encoding routines, the use of quantum aware cost functions, and the potential of real multi-class classification. In terms of quantum image representation, the question of how much classical data can we store in a quantum state and whether the underlying structure of hyperspectral images has any advantage. While we are limited to small-scale simulations, it would also be interesting to investigate the effects of large input sets on classical parameter optimization. It would also be interesting to study the potential use cases for quantum sensors in remote sensing and subsequent processing of quantum data with the presented methods. Further, we can integrate the presented quantum neural network into existing classical machine learning architecture. Another promising method is a quantum convolutional neural network which could, like its classical counterpart, consider both spectral and spatial components. Looking in the future, we should also check whether replacing the classical machine learning subroutine with quantum methods poses any performance improvement.

Bibliography

- [1] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [2] A. Turing, “Computing machinery and intelligence in “mind”, vol,” 1950.
- [3] J. McCarthy and E. A. Feigenbaum, “In memoriam: Arthur samuel: Pioneer in machine learning,” *AI Magazine*, vol. 11, no. 3, pp. 10–10, 1990.
- [4] W. McCulloch and W. Pitts, “A logical calculus of the idea immanent in neural nets,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- [5] T. Mitchell and M. L. McGraw-Hill, “Edition,” 1997.
- [6] S. Lloyd, “Universal quantum simulators,” *Science*, vol. 273, no. 5278, pp. 1073–1078, 1996.
- [7] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.
- [8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
- [9] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.
- [10] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
- [11] D. Herman, C. Googin, X. Liu, A. Galda, I. Safro, Y. Sun, M. Pistoia, and Y. Alexeev, “A Survey of Quantum Computing for Finance,” *arXiv*, Jan. 2022.
- [12] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, S. Sim, L. Veis, and A. Aspuru-Guzik, “Quantum

- Chemistry in the Age of Quantum Computing,” *Chem. Rev.*, vol. 119, no. 19, pp. 10 856–10 915, Oct. 2019.
- [13] A. Dendukuri and K. Luu, “Image Processing in Quantum Computers,” *arXiv*, Dec. 2018.
- [14] D. A. Zaidenberg, A. Sebastianelli, D. Spiller, B. L. Saux, and S. L. Ullo, “Advantages and Bottlenecks of Quantum Machine Learning for Remote Sensing,” *arXiv*, Jan. 2021.
- [15] P. Gawron and S. Lewiński, “Multi-spectral image classification with quantum neural network,” in *IGARSS 2020-2020 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2020, pp. 3513–3516.
- [16] N. Audebert, B. Le Saux, and S. Lefèvre, “Deep learning for classification of hyperspectral data: A comparative review,” *IEEE geoscience and remote sensing magazine*, vol. 7, no. 2, pp. 159–173, 2019.
- [17] R. Gogineni and A. Chaturvedi, “Hyperspectral image classification,” in *Processing and Analysis of Hyperspectral Data*. IntechOpen London, UK, 2019.
- [18] G. Hughes, “On the mean accuracy of statistical pattern recognizers,” *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.
- [19] G. Camps-Valls and L. Bruzzone, *Kernel methods for remote sensing data analysis*. John Wiley & Sons, 2009.
- [20] F. Melgani and L. Bruzzone, “Classification of hyperspectral remote sensing images with support vector machines,” *IEEE Transactions on geoscience and remote sensing*, vol. 42, no. 8, pp. 1778–1790, 2004.
- [21] R. M. Mohamed and A. A. Farag, “Advanced algorithms for bayesian classification in high dimensional spaces with applications in hyperspectral image segmentation,” in *IEEE International Conference on Image Processing 2005*, vol. 2. IEEE, 2005, pp. II–646.
- [22] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W.-K. Mok, S. Sim, L.-C. Kwek, and A. Aspuru-Guzik, “Noisy intermediate-scale quantum algorithms,” *Rev. Mod. Phys.*, vol. 94, no. 1, p. 015004, Feb. 2022.
- [23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [24] S. Aaronson, *Quantum computing since Democritus*. Cambridge University Press, 2013.

- [25] M.-O. Renou, D. Trillo, M. Weilenmann, T. P. Le, A. Tavakoli, N. Gisin, A. Acín, and M. Navascués, “Quantum theory based on real numbers can be experimentally falsified,” *Nature*, vol. 600, pp. 625–629, Dec 2021.
- [26] R. P. Feynman, “Negative probability,” *Quantum implications: essays in honour of David Bohm*, pp. 235–248, 1987.
- [27] R. P. Feynman, R. B. Leighton, and M. Sands, *The Feynman lectures on physics, Vol. III: The new millennium edition: Quantum Mechanics*. Basic books, 2011, vol. 3.
- [28] A. M. Turing *et al.*, “On computable numbers, with an application to the entscheidungsproblem,” *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.
- [29] P. Bernays, “Alonzo church. an unsolvable problem of elementary number theory. american journal of mathematics, vol. 58 (1936), pp. 345–363.” *The Journal of Symbolic Logic*, vol. 1, no. 2, pp. 73–74, 1936.
- [30] R. P. Feynman, “Simulating physics with computers,” in *Feynman and computation*. CRC Press, 1981, pp. 133–153.
- [31] D. Deutsch, “Quantum theory, the church–turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.
- [32] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.
- [33] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [34] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.
- [35] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.
- [36] B. Schumacher, “Quantum coding,” *Physical Review A*, vol. 51, no. 4, p. 2738, 1995.
- [37] A. et. al., “Qiskit: An open-source framework for quantum computing,” 2021.
- [38] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*. Cham, Switzerland: Springer International Publishing, 2018. [Online]. Available: <https://link.springer.com/book/10.1007/978-3-319-96424-9>

- [39] M. Schuld, I. Sinayskiy, and F. Petruccione, “Prediction by linear regression on a quantum computer,” *Physical Review A*, vol. 94, no. 2, p. 022342, 2016.
- [40] R. LaRose and B. Coyle, “Robust data encodings for quantum classifiers,” *Physical Review A*, vol. 102, no. 3, p. 032420, 2020.
- [41] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [42] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, “Noisy intermediate-scale quantum algorithms,” *Reviews of Modern Physics*, vol. 94, no. 1, p. 015004, 2022.
- [43] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” *arXiv*, Aug. 1995.
- [44] E. Bernstein and U. Vazirani, “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.
- [45] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, “Variational quantum algorithms,” *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.
- [46] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, “A variational eigenvalue solver on a quantum processor,” *arXiv*, Apr. 2013.
- [47] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.
- [48] M. Schuld, A. Bocharov, K. Svore, and N. Wiebe, “Circuit-centric quantum classifiers,” *arXiv*, Apr. 2018.
- [49] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, “Quantum circuit learning,” *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.
- [50] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, “Evaluating analytic gradients on quantum hardware,” *Physical Review A*, vol. 99, no. 3, p. 032331, 2019.
- [51] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” *Nature communications*, vol. 9, no. 1, pp. 1–6, 2018.

- [52] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, “Noise-induced barren plateaus in variational quantum algorithms,” *Nature communications*, vol. 12, no. 1, pp. 1–11, 2021.
- [53] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. Coles, “Cost-function-dependent barren plateaus in shallow quantum neural networks (2020),” *arXiv preprint arXiv:2001.00550*, 2001.
- [54] D. Wecker, M. B. Hastings, and M. Troyer, “Progress towards practical quantum variational algorithms,” *Physical Review A*, vol. 92, no. 4, p. 042303, 2015.
- [55] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets,” *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.
- [56] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, “Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms,” *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.
- [57] M. Schuld, “Supervised quantum machine learning models are kernel methods,” *arXiv*, Jan. 2021.
- [58] T. Hofmann, B. Schölkopf, and A. J. Smola, “Kernel methods in machine learning,” *arXiv*, Jan. 2007.
- [59] B. Schölkopf, A. Smola, and K.-R. Müller, “Nonlinear component analysis as a kernel eigenvalue problem,” *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
- [60] T. Hubregtsen, D. Wierichs, E. Gil-Fuster, P.-J. H. S. Derks, P. K. Faehrmann, and J. J. Meyer, “Training Quantum Embedding Kernels on Near-Term Quantum Computers,” *arXiv*, May 2021.
- [61] A. Anand, M. Lyu, P. S. Baweja, and V. Patil, “Quantum image processing,” *arXiv preprint arXiv:2203.01831*, 2022.
- [62] S. E. Venegas-Andraca and S. Bose, “Storing, processing, and retrieving an image using quantum mechanics,” in *Quantum Information and Computation*, vol. 5105. SPIE, 2003, pp. 137–147.
- [63] F. Yan and S. E. Venegas-Andraca, *Quantum Image Processing*. Singapore: Springer Nature, 2020. [Online]. Available: <https://link.springer.com/book/10.1007/978-981-32-9331-1>

- [64] P. Li, H. Xiao, and B. Li, “Quantum representation and watermark strategy for color images based on the controlled rotation of qubits,” *Quantum Inf. Process.*, vol. 15, no. 11, Nov. 2016.
- [65] R. A. Khan, “An improved flexible representation of quantum images,” *Quantum Inf. Process.*, vol. 18, no. 7, May 2019.
- [66] R. Zhou, W. Hu, G. Luo, X. Liu, and P. Fan, “Quantum realization of the nearest neighbor value interpolation method for INEQR,” *Quantum Inf. Process.*, vol. 17, no. 7, p. 166, Jul. 2018.
- [67] H.-S. Li, X. Chen, S. Song, Z. Liao, and J. Fang, “A Block-Based Quantum Image Scrambling for GNEQR,” *IEEE Access*, vol. 7, pp. 138 233–138 243, Sep. 2019.
- [68] S. Caraiman and V. Manta, “Image Representation and Processing Using Ternary Quantum Computing,” *undefined*, 2013. [Online]. Available: <https://www.semanticscholar.org/paper/Image-Representation-and-Processing-Using-Ternary-Caraiman-Manta/19039b89e91348655b2678d8d2db3f1e3a01c0e8>
- [69] M. Schuld, I. Sinayskiy, and F. Petruccione, “The quest for a Quantum Neural Network,” *arXiv*, Aug. 2014.
- [70] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, “Quantum boltzmann machine,” *Physical Review X*, vol. 8, no. 2, p. 021050, 2018.
- [71] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, “Quantum circuits for general multiqubit gates,” *Physical review letters*, vol. 93, no. 13, p. 130502, 2004.
- [72] L. Grover and T. Rudolph, “Creating superpositions that correspond to efficiently integrable probability distributions,” *arXiv preprint quant-ph/0208112*, 2002.
- [73] A. N. Soklakov and R. Schack, “Efficient state preparation for a register of quantum bits,” *Physical review A*, vol. 73, no. 1, p. 012307, 2006.
- [74] M. Arjovsky, A. Shah, and Y. Bengio, “Unitary evolution recurrent neural networks,” in *International conference on machine learning*. PMLR, 2016, pp. 1120–1128.
- [75] Y. Levine, D. Yakira, N. Cohen, and A. Shashua, “Deep learning and quantum entanglement: Fundamental connections with implications to network design,” *arXiv preprint arXiv:1704.01552*, 2017.

- [76] G.-L. R. Anselmetti, D. Wierichs, C. Gogolin, and R. M. Parrish, “Local, expressive, quantum-number-preserving VQE ansätze for fermionic systems,” *New J. Phys.*, vol. 23, no. 11, p. 113010, Nov. 2021.
- [77] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, “Power of data in quantum machine learning,” *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.
- [78] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, “Quantum Natural Gradient,” *arXiv*, Sep. 2019.
- [79] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [80] B. Schölkopf, A. J. Smola, F. Bach *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [81] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola, “On kernel-target alignment,” *Advances in neural information processing systems*, vol. 14, 2001.
- [82] C. M. Bishop, “Pattern recognition and machine learning (information science and statistics),” 2007.
- [83] S. Lloyd, M. Mohseni, and P. Rebentrost, “Quantum principal component analysis,” *arXiv*, Jul. 2013.
- [84] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri *et al.*, “PennyLane: Automatic differentiation of hybrid quantum-classical computations,” *arXiv preprint arXiv:1811.04968*, 2018.
- [85] A. Hancock, A. Garcia, J. Shedenhelm, J. Cowen, and C. Carey, “Cirq: A python framework for creating, editing, and invoking quantum circuits.”
- [86] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa *et al.*, “Tensorflow quantum: A software framework for quantum machine learning,” *arXiv preprint arXiv:2003.02989*, 2020.

List of Figures

2.1	(a) Passive sensors detect the reflection of the sunlight from the object. (b) Active sensors have their own source of energy to illuminate the object.	4
2.2	(a) 3-dimensional hypercube representation of the Pavia University dataset showing the RGB image and its spectra (b) The spectral signature shows the normed reflectance intensities of the respective bands. This signature was randomly sampled from the class meadows within the Pavia University dataset.	4
2.3	Classical feed-forward neural network structure consisting of connected perceptrons that form an input layer, several hidden layers and an output layer.	7
3.1	Double slit experiment: a) photon pattern with one slit b) photon pattern with double slit c) photon pattern with double slit and measurement	10
3.2	The Bloch Sphere	13
3.3	Basic single qubit gates, rotating around the X, Y and Z axis of the bloch sphere as well as the Hadamard gate (H). Shown are also the representation in a circuit and their matrix representation.	16
3.4	Basic multi qubit gates: CNOT, the controlled NOT gate; SWAP, swaps the state of two qubits and CCU, controlled-controlled U gate that controls on two wires and then applies an arbitrary unitary U on the third wire.	18
3.5	Quantum processing unit with input x , parameters θ and output y . The QPU runs quantum circuits, but its parameters can be optimized classically.	19
3.6	Potential relationships of complexity classes. Description of the classes are given in Chapter 3.4	23
4.1	General structure of a variational quantum circuit consisting of the feature map S_x with input data x , a parametrized unitary U_θ with parameters θ and a subsequent measurement $f(\sigma) = y$ where f is the overall quantum function with measurement in basis σ producing output y	26

4.2	(a) Linearly not separable XOR problem with 4 datapoints on a 2D plane (b) The feature map projects the data into a 3D space, where we can now draw a separating hyperplane.	31
5.1	Typical four steps of a quantum neural network: 1. State preparation, 2. Model Circuit, 3. Measurement and 4. Postprocessing	38
5.2	Amplitude encoding scheme for $n = 3$ qubits. Given $N = 8$ datapoints, we require $N - 1$ angles $(\alpha_0, \dots, \alpha_{N-1})$ to encode the information into the amplitudes of 2^n states.	38
5.3	Strongly entangled layers Ansatz consisting of two blocks L_1 and $L_2R(\theta)$ are rotation gates with an individual set of parameters $\theta = (\phi, \gamma, \omega)$ for every gate. Fir this configuration only the first qubit gets measured.	40
5.4	Support Vector Machine that maximizes the margin M . The separating hyperplane are all points satisfying $w\hat{x} + b = 0$, where w is a set of parameters, x is a feature vector and b is some bias. The closest points of each dataset to the hyperplane are the support vectors, here purple colored datapoints.	44
5.5	Three possible implemenations of a quantum kernel: (a) Adjoint method (b) SWAP-Test (c) Projected Kernel. U is an arbitrary unitary encoding the datapoints (x_1, x_2)	47
6.1	(a) Pavia University dataset, only the red-blue-green channels on the left side and the ground truth with corresponding labels on the right side, (b) Number of samples per class.	52
6.2	Cumulative explained variance with growing number of principal components. We break the threshold of 95% explained variance with 3 principal components.	53
6.3	Ground truth and predicted classes of the Projected Quantum Support Vector Machine. To generate the predicted classes image, we overlaid the predicted classes for test and training set as well as the previously dropped undefined classes.	56
6.4	Comparison of accuracy and performance of the Projected Quantum Support Vector Machine with an increasing number of IQP repetitions.	56

List of Tables

6.1	Experimental results of the three quantum models compared to two classical support vector machines with linear and radial basis function kernel. The abbreviations translate to Support Vector Machine (SVM), Quantum Support Vector Machine (QSVM), Projected Quantum Support Vector Machine (PQSVM) and Quantum Neural Network (QNN).	55
-----	---	----

