UNIVERSITY OF UDINE
hic sunt futura

Department of
Mathematics, Computer Science and Physics

Department of
Artificial Intelligence and Cybersecurity

MASTER THESIS IN
ARTIFICIAL INTELLIGENCE & CYBERSECURITY

Automatic Discovery of Kernels for Quantum Anomaly Detection

CANDIDATE
Daniele Lizzio Bosco

SUPERVISOR
Prof. Giuseppe Serra

CO-SUPERVISORS
Prof. Martin Gebser
Prof.ssa Alessandra Di Pierro
Dott. Massimiliano Incudini

Academic Year 2022-2023

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor, Giuseppe Serra, for his unparalleled guidance, his insightful suggestions, and for his deep belief in this work. Without his encouragement, this thesis would not have been possible. I am thankful to Martin Gebser for his insights and for being a constant source of inspiration. Alessandra Di Pierro deserves my gratitude for her indispensable involvement during all phases of this work.

I must thank Massimiliano Incudini for his valuable expertise, suggestions, and advice, but above all his invaluable patience throughout the duration of this journey.

I am also grateful to all the people who accompanied me along these years: my somewhat far, but also very close friends, my colleagues, my fellow students, my flatmates... I would love to acknowledge them one by one, but this would easily become a long list which this margin is too narrow to contain (however, if you are reading this, you are probably one of them!). Suffice it to say, my life would have been very different without all of them.

My warmest thank goes to my family, for their relentless support, deep comprehension, and infinite affection. Without them, none of this would have been possible. A small thank you goes to my dogs, without whom everything would have been possible anyway, but which are still an important part of my life.

Lastly, I want to express my deepest appreciation to all the people that study, teach, and research. Many of the people I met throughout my education were a huge source of inspiration, that helped me renew and reinforce my passion for learning, and which in the end contributed to shaping my approach towards the world. If the university is the house of knowledge, then I can say that this house has welcomed me with open arms.

Abstract

Quantum Computing stands out as one of the most popular research fields poised to revolutionize our society and industry. Despite our current quantum devices being in the early stages of development, some applications in Machine Learning seem to be viable on current, noisy devices. One of the most promising techniques in this field is obtained by *quantum kernels*, which consists of embedding classical data into quantum state spaces that represent more complex, non-local correlation between data, to obtain the quantum analogous of a Support Vector Machine (SVM). Certain problems admit quantum models able to consistently outperform any classical counterpart. However, designing a quantum kernel suitable for a certain dataset is far from being a trivial task, as most kernels admit unfavorable statistical properties that drastically limit their applicability.

In this thesis we propose a technique to automatically discover and optimize quantum kernels by combining the selection of optimization algorithms with adequate evaluation metrics. We tested our algorithm on a dataset that is known to admit a quantum advantage over classical techniques, comparing the performance of models that use our discovered quantum kernels with the performance of classical models and state-of-the-art SVM with quantum kernels. Our tests were performed in a noiseless, simulated environment for both quantum models, to ensure a fair comparison. The results show that our quantum kernel discovery pipeline is able to produce models that outperform both classical and quantum state-of-the-art ones in a statistically significant way, offering insights for possible future applications to real-world problems.

Contents

1 Introduction **1**

2 Background on Quantum Computing and Quantum Machine Learning **3**
2.1 Qubits . . . 3
2.2 Postulates . . . 4
2.3 Representing states on the Bloch Sphere . . . 7
2.4 Circuit Model and Quantum Gates . . . 9
2.4.1 Single Qubit gates . . . 9
2.4.2 Multiple Qubits gates . . . 12
2.5 Universal set of gates . . . 14
2.6 Quantum computational complexity and some quantum algorithms . . . 16
2.6.1 Complexity classes and their definitions . . . 17
2.6.2 Deutsch–Jozsa's problem . . . 17
2.6.3 Simon's problem . . . 18
2.6.4 Shor's algorithm . . . 19
2.6.5 Structure required for an exponential advantage . . . 19
2.7 Quantum Machine Learning . . . 20
2.7.1 Solving linear systems of equations: HHL Algorithm . . . 21
2.7.2 Quantum Principal Component Analysis . . . 22
2.7.3 Quantum Approximate Optimization Algorithm . . . 22

3 Background on kernel methods **25**
3.1 Kernel Methods and Kernel Trick . . . 26
3.2 Properties and examples of kernels . . . 27
3.3 Support Vector Machines . . . 28
3.3.1 Hard and Soft margin SVM . . . 28
3.3.2 Dual SVM . . . 30
3.4 Other examples of Kernel Methods . . . 31
3.4.1 Kernel PCA . . . 31
3.4.2 One-Class SVM . . . 32

4 Quantum kernels and their challenges **35**
4.1 Characterization of quantum kernels . . . 36
4.1.1 Efficient classical simulability . . . 36
4.1.2 Exponential concentration . . . 37
4.1.3 Expressibility . . . 37
4.1.4 Eigenvalues distribution . . . 39
4.1.5 Quantifying expressibility . . . 39

5 Automatic Kernel Discovery **43**
5.1 Modeling and optimizing the kernel . . . 44
5.1.1 The algorithm . . . 45
5.2 Evaluating the kernel: metrics . . . 46

5.2.1 Kernel compatibility . . . 46
5.2.2 Task-Model alignment and Spectral bias . . . 47
5.2.3 Performance-based metrics . . . 47
5.3 Optimizers . . . 47
5.3.1 Greedy Algorithm . . . 48
5.3.2 Bayesian Optimization . . . 50
5.3.3 Genetic Algorithm . . . 52
5.3.4 Reinforcement Learning Approaches . . . 53
5.4 Details and optimizations . . . 54
5.4.1 Reducing the search space . . . 54
5.4.2 Qiskit and Transpiling . . . 56
5.4.3 Gates repetition . . . 56

6 Quantum Anomaly Detection **59**
6.1 Proton collision events at the LHC . . . 59
6.1.1 The dataset . . . 60
6.2 Approach and experimental setup . . . 60
6.3 Results . . . 61
6.3.1 Evaluating performance: ROC and AUC . . . 61

7 Conclusion and further improvements **67**

1

Introduction

Among the research fields promising to revolutionize our society and industry, Quantum Computing is perhaps one of the most popular. Such a paradigm of computation, rooted in the principles of Quantum Mechanics, requires a great change of perspective compared to classical computing, but allows, in principle, greater computational capabilities, solving certain classes of problems way faster than what classical computation can do.

Recently, Quantum Computing has become more than just a theory, as some working prototypes have been proposed having more than 433 qubits (quantum bits), a landmark achievement allowing for some possible benefits in using these devices. However, it is worth noting that such devices are sensitive to environmental noise, leading to errors. These noisy devices are denoted with NISQ, from *Noisy Intermediate Scale Quantum* [32].

While quantum algorithms have been applied to many possible applications and topics, the one discussed in this thesis is machine learning. Quantum Machine Learning (QML) [9, 5] applies quantum computation to obtain faster or more accurate machine learning models. It has been applied to many ML contexts, including supervised learning, generative models, and reinforcement learning.

Regarding supervised learning, different approaches have been proposed. While some of these require fault-tolerant (error-free) quantum devices, which are beyond the reach of our current hardware capabilities, many other approaches can work satisfactorily on the current NISQ devices. One widely known example is the quantum kernel, implemented in the landmark paper by Havlicek et al. [20]. Kernel methods in classical machine learning identify a technique to express a notion of similarity between data points that is richer, and more expressive than the usual Euclidean product between real vectors. Quantum kernels take advantage of the exponential dimensionality of the quantum state space to represent a more complex, non-local correlation between data points. They have found applications in some physics use cases, for which we conjecture classical kernels (and, in general, classical machine learning techniques) fail. To apply quantum kernel successfully, i.e. with a significant advantage in terms of accuracy compared to the classical techniques, we need to check some properties of the quantum kernel. First, it must be complex enough to not be classically simulable efficiently, as there would be no point in using a quantum device otherwise. Second, it must have favorable statistical properties, such as a non-flat, decaying distribution of eigenvalues. Third, it must effectively be tailored to the structure of the problem.

Current approaches to kernel design propose to select operators that are invariant or equivariant with respect to certain symmetries of the problem. However, in most real-world problems we lack a complete knowledge of the properties of the dataset, which forces us to design kernels based on heuristics.

In this thesis, we explore the automatic generation and optimization of quantum kernels that do not rely on prior knowledge of the system considered. To do so, we address the following questions: How should we evaluate the performance of a quantum kernel? What criteria are indicative of a good performance on a certain problem? Furthermore, which heuristic optimization algorithm is the most effective in this context?

We study some of these questions related to an anomaly detection problem. Specifically, our focus is on discerning between *standard model* (SM) and *beyond standard model* (BSM) events within the context of latent space encoding for proton collision events at the LHC, as introduced by Woźniak et al. in [46]. This dataset is known to have a statistically proven benefit in using quantum kernels. We compare the results from the authors, who have handcrafted the quantum kernel according to their experience, with the result of our agnostic, automatic procedure. The obtained results show that, even with a limited amount of qubits and operations, our optimization technique can find kernels that in certain scenarios are able to outperform classical kernels and the quantum kernels in [46]. This result shows that it is indeed possible to automatize the design of kernels in unknown scenarios, greatly increasing the cases in which we can use a quantum approach instead of classical machine learning techniques. Finally, this study may open the door to the examination of new questions. It remains unclear if the choice of a particular optimization technique has some benefit compared to the Bayesian optimization, or any other heuristic proposed in the literature. Furthermore, we do not know whether there is a single cost function or multiple ones that better leads the optimization technique toward a satisfactory solution that takes into account multiple properties of the quantum kernel. That implies that our problem might be better formulated as multi-objective optimization.

The main contributions of this thesis can be summarized as follows:

- we introduce a novel approach for the automated generation and optimization of quantum kernels for machine learning tasks, eliminating the need for prior knowledge of dataset properties;
- By considering different criteria for evaluating the performance of quantum kernels for specific problem domains, we observe that certain optimization algorithms perform better according to the considered cost function;
- A comparative analysis is conducted between automatically generated quantum kernels and handcrafted state-of-the-art quantum kernels, demonstrating the effectiveness of the proposed automated approach in certain scenarios.

2

Background on Quantum Computing and Quantum Machine Learning

In this Chapter we give a brief introduction to Quantum Computing, presenting some of its theoretical foundations, results, and algorithms. For a comprehensive and in-depth exploration of this topic, we refer to the book [26]. In the last section of the Chapter, we present some of the most well-known algorithms in Quantum Machine Learning.

2.1 Qubits

The fundamental concept of classical computations is the *bit*. It represents the basic building block of digital information and can take one of two possible values: 0 or 1. The analogous concept in the quantum computation setting is the *quantum bit*, or *qubit*. In our discussion, we will refer to qubits as mathematical objects, in a similar way as we are going to consider *quantum circuits*, *gates*, and so on. However, is important to note that in a similar way to how bits can be realized in a physical device, quantum bits can be realized as actual physical systems.

Qubits are quantum mechanical systems whose state is a 2-dimensional complex vector, such as $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, that can be intuitively considered as the quantum analogous to the states 0 and 1 for a classic bit. The symbol $|0\rangle$ ("ket 0") is in the so-called *Dirac notation*, the standard notation for states in quantum mechanics and, for finite dimensional systems such as the ones in this thesis, corresponds to column vectors. The main difference between bits and qubits is that the latter can be in a state other than $|0\rangle$ or $|1\rangle$. In particular, each *quantum state* can be considered as a linear combination of states in the following notation:

$$|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.1}$$

where α and β are complex numbers that satisfy the condition of *normalization*, i.e.

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2.2}$$

When α and β are both different from 0, we say that $|\varphi\rangle$ is a *superposition* of $|0\rangle$ and $|1\rangle$.

We can create systems with multiple qubits. In the classical setting, 2 bits would have 4 different states, that are $00, 01, 10$, and 11. A two qubits system has, therefore, 4 computational basis states, denoted as $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$, and each state vector describing the system is a (normalized) linear combination

$$|\varphi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle . \tag{2.3}$$

The idea of *how* we can make a composite system with more qubits will be given in the next session, after the fourth postulate.

2.2 Postulates

We are now ready to formally introduce the basic postulates of quantum mechanics. These postulates provide a connection between the physical world and the mathematical formalism of quantum mechanics.

Postulate 1 *Any isolated physical system is associated with a complex Hilbert space, i.e. complete vector space with an inner product. The system is completely described by its state vector, which is a unit vector in the system's state space.*

This is the formal definition of a Hilbert space:

Definition 2.2.1 *A* Hilbert space *is a complete inner product space. Specifically, it is a vector space H over the field of complex numbers $\mathbb{C}$ equipped with an inner product $\langle \cdot, \cdot \rangle$ that satisfies the following properties for all vectors $x, y, z \in H$ and scalars $\alpha, \beta \in \mathbb{C}$:*

1. *Linearity: $\langle \alpha x + \beta y, z \rangle = \alpha \langle x, z \rangle + \beta \langle y, z \rangle$*
2. *Conjugate symmetry: $\langle x, y \rangle = \overline{\langle y, x \rangle}$*
3. *Positive definiteness: $\langle x, x \rangle > 0$ for all $x \neq 0$*
4. *Completeness: Every Cauchy sequence in H converges to a limit that is also in H.*

In the proper language used in quantum mechanics, the inner product of the states $|\varphi\rangle$ and $|\psi\rangle$ is written as $\langle \varphi | \psi \rangle$. In particular, every state $|\varphi\rangle$ satisfies the *normalization condition*, that means $\langle \varphi | \varphi \rangle = 1$. The last terminology we need to introduce before going further is *orthonormality.*

Definition 2.2.2 *A set of states $\{|\psi_i\rangle\}$ in a Hilbert space is said to be* orthonormal *if the inner product of any two distinct states is zero, and the inner product of a state with itself is one. Formally, for all i and for all j*

$$\langle \psi_i | \psi_j \rangle = \delta_{ij}$$

where δ_{ij} is the Kronecker delta.

As introduced in the last section, the simplest quantum system we are concerned with is the qubit, in which every state can be described as a unitary vector in $\mathbb{C}^2$ or equivalently, as a superposition (linear combination of unitary length) of the states of some orthonormal basis. The set $\{|0\rangle, |1\rangle\}$ is an orthonormal basis of the Hilbert space representing a one-qubit system (in particular, it is called the *computational basis states*). However, is important to note that it is just one of the possible basis. In particular, any pair of orthonormal states forms a basis, such as, for example, the states

$$|+\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big), \quad |-\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big). \tag{2.4}$$

The second postulate gives a characterization of how a quantum mechanical system changes with time.

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

$$|\psi'\rangle = U|\psi\rangle. \tag{2.5}$$

An *unitary* can be defined in the following way:

Definition 2.2.3 *A square matrix U is said to be* unitary *if its conjugate transpose, denoted by $U^\dagger$, is equal to its inverse. Mathematically, a square matrix U is said unitary if the following condition holds:*

$$U^\dagger U = UU^\dagger = I, \tag{2.6}$$

where I is the identity matrix.

Is important to note that each U can be considered as a parameter-dependent operator $U : t \mapsto U(t)$. In practice, when we apply a *gate* to a quantum circuit, we consider "constant" gates, i.e. operators that do not depend on time. This apparent inconsistency will be clarified in the following section.

In short, Postulate 2 describes how the quantum states of a closed quantum system at two different times are related. A more refined version of this postulate can be given which describes the evolution of a quantum system in *continuous time*.

Postulate 2′ *The time evolution of the state of a closed quantum system is described by the Schrödinger equation:*

$$i\hbar\frac{d|\psi\rangle}{dt} = H|\psi\rangle \tag{2.7}$$

where $\hbar$ is a physical constant known as Planck's constant. In practice, it is common to absorb the factor $\hbar$ into H, effectively setting $\hbar = 1$. H is a fixed Hermitian operator known as the Hamiltonian of the closed system.

Despite not being strictly useful for our scopes, this alternative formulation contains one of the most famous equations in quantum mechanics, if not the most famous. Moreover, is important to note that

we can easily derive the first formulation from the second one. In particular, it is possible to show that the solution to the Schrödinger equation can be written as

$$|\psi(t_2)\rangle = U(t_1, t_2)|\psi(t_1)\rangle, \tag{2.8}$$

where we define

$$U(t_1, t_2) \equiv \exp\left(-i\frac{H(t_2 - t_1)}{\hbar}\right), \tag{2.9}$$

obtaining therefore the first formulation. Finally, it may be interesting to note that is possible to derive the second formulation from the first one, obtaining that these postulates are equivalent. However, in the following sections, we are going to use only the first formulation, since we are mostly interested in operators as gates for quantum circuits, and not in the practical construction of operators as real-world devices.

The assumptions of "closeness" for our quantum system, despite not being formally defined, are fundamental for the evolution of the system. A closed system, in the context of quantum mechanics and quantum computing, refers to a physical system that is isolated from its external environment. This means that the system does not exchange energy or information with the outside world. However, since in practical experiments we are interested in *observing* the results, we need to introduce some kind of interaction between the system and the external world: *measurement*.

Postulate 3 *Quantum measurements are described by a collection $\{M_m\}$ of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement, then the probability that result m occurs is given by*

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle, \tag{2.10}$$

and the state of the system after the measurement is

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}. \tag{2.11}$$

The measurement operators satisfy the completeness equation,

$$\sum_m M_m^\dagger M_m = I. \tag{2.12}$$

Informally, we can say that a measurement of a state "destroys superpositions": if we measure a state $|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ according to the so-called *measurement of a qubit in the computational basis* M_0 and M_1, where

$$M_i = |i\rangle\langle i|, \tag{2.13}$$

we are going to observe the outcome 0 with probability

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \langle\psi|M_0|\psi\rangle = |\alpha|^2. \tag{2.14}$$

Similarly, the probability of obtaining the outcome 1 is $|\beta|^2$. However, if we obtained the outcome 0 in the first place, each subsequent measurement is going to result in 0 with probability 1, therefore "destroying" the initial superposition. This strange property poses some challenges in programming quantum devices, since every time we measure a state we need to recreate it again in order to do another measurement. Finally, we can introduce the last postulate

Postulate 4 *The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is prepared in the state* $|\psi_i\rangle$*, then the joint state of the total system is* $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_n\rangle$.

This postulate gives us a way to compose quantum systems made up of two or more distinct physical systems. The most immediate consequence of it is that, as long as we are able to build a quantum system with a qubit (and we are able to do it), we can have a system with any amount of qubits. Of course, this does not mean that we are able to physically build a functioning quantum device with any amount of qubits, but it does not matter as long as we are interested in quantum systems as mathematical objects. Let's see an example. Given a one-qubit system in state $|0\rangle$ and another one in state $|1\rangle$, the resulting system obtained will be in state $|0\rangle \otimes |1\rangle$, which we can simply write as $|01\rangle$. More in general, given a system A with state $|A\rangle$ and a system B with state $|B\rangle$, the state of the resulting system will be $|AB\rangle$.

This postulate allows us to introduce one of the most interesting ideas associated with composite quantum systems: *entanglement*. A qubit state $|\varphi\rangle$ in the system AB, obtained by A and B, is said *entangled* is it cannot be written as $|a\rangle \otimes |b\rangle$ for any state $|a\rangle$, $|b\rangle$ for the system A and B respectively. An example of an entangled 2-qubit state is

$$|\varphi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.15}$$

The importance of these considerations is that, under certain conditions, entanglement is required to achieve an exponential speed-up compared to a classical algorithm. In particular, for each quantum algorithm operating on pure states, if it does not introduce any entanglement, then it can be classically efficiently simulated. This result can be strengthened by substituting "any entanglement" with "a small amount of global entanglement" [45]. The exact meaning of this last sentence will be clearer later, after the introduction of metrics to compute the "level of entanglement" of a system.

2.3 Representing states on the Bloch Sphere

As we said before, a quantum state of a single qubit system can be written as

$$|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.16}$$

where α and β are *normalized*. From this condition, we are able to rewrite a quantum state as

$$|\psi\rangle = e^{i\gamma}\Big(\cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\Big), \tag{2.17}$$

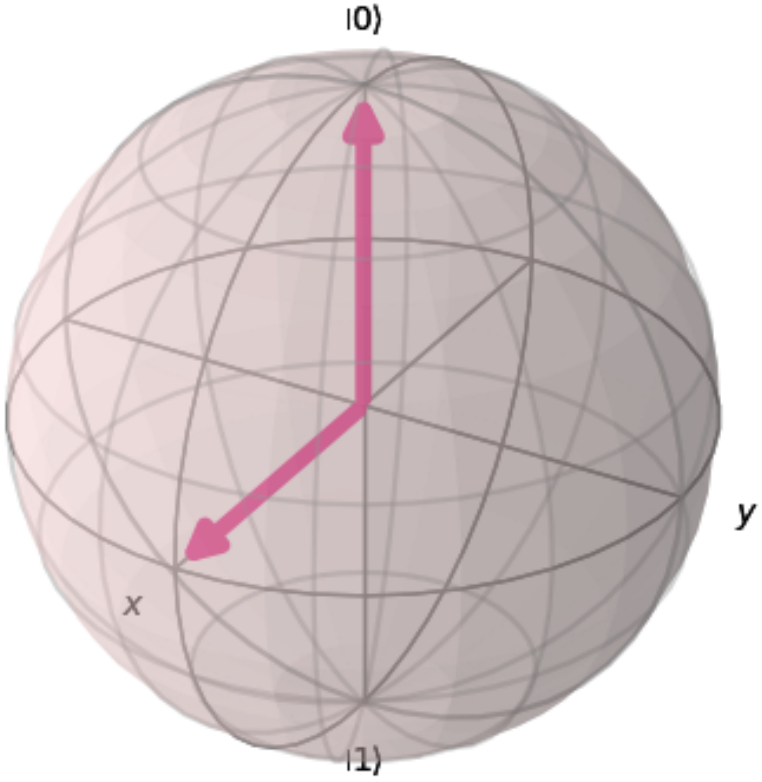


Figure 2.1: Bloch sphere with the vectors representing the states $|0\rangle$ and $|+\rangle$.

where θ, φ and γ are real numbers. The factor $e^{i\gamma}$ is called *phase*, and it does not have any observable effect, therefore can be omitted, allowing us to describe any state as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle\,. \tag{2.18}$$

A common way of depicting single qubit states is by visualizing them on the *Bloch sphere*, by associating to each state the point on the unit three-dimensional sphere corresponding to the angles θ and φ, as shown in this picture.

Visualizing a state on the Bloch sphere may give us the intuition of how much information a state contains. Of course, from a practical point of view describing a state in the former or the latter notation is irrelevant. Moreover, the Bloch sphere can even give a misleading intuition about qubits and states, as there is no simple generalization of the Bloch sphere on multiple qubits, raising some questions about how much we should imagine states as points on the unit sphere.

All of the gates that act on a single gate that we use in this work can be easily represented as operations on the Bloch sphere. Each gate can (and should) be unequivocally represented with a matrix (in opposition to the Bloch sphere representation, this representation can be used for systems of any number of qubits), but sometimes is easier to visualize an operation as a rotation on the Bloch sphere. For this reason, when possible we will present a gate first with the corresponding matrix, and then with the "geometric" interpretation as a rotation of the Bloch sphere. When we say for example that "a X

gate is a rotation of π radians around the x-axis", we mean that when considering an initial state and the corresponding final state after the application of a X gate, their representations on the Bloch sphere are related by a rotation of π radians around the x-axis.

2.4 Circuit Model and Quantum Gates

In this section, we describe the *quantum circuit model* with some of the most used gates, and then we briefly introduce the idea of *universality* of a set of gates.

A device that performs *quantum computations* usually consists of a *quantum circuit*. A classical computer circuit is composed by wires and logic gates. Analogously, a quantum circuit is composed of wires and *quantum gates*, which are operators able to manipulate quantum information.

2.4.1 Single Qubit gates

The best way to introduce quantum gates is again with a direct comparison with the classical case. Consider the classical single bit logic gate `NOT`, which can be formally defined as the binary function $f : \{0, 1\} \to \{0, 1\}$ such that

$$0 \mapsto 1$$
$$1 \mapsto 0$$

If we want a quantum analogous of the `NOT` function, we expect that it would take the state $|0\rangle$ to $|1\rangle$ and vice versa. However, specifying this is not enough: we do not have enough information to know how the operator acts on superpositions of states $|0\rangle$ and $|1\rangle$. An important property of quantum gates is that they act *linearly*, and therefore we obtain that the quantum `NOT` gate must take the state

$$\alpha |0\rangle + \beta |1\rangle \tag{2.19}$$

to the corresponding state in which the role of $|0\rangle$ and $|1\rangle$ have been interchanges, that is

$$\alpha |1\rangle + \beta |0\rangle . \tag{2.20}$$

A convenient way to represent quantum gates is with matrices. For example, to represent the quantum `NOT` gate we can define the X matrix as follows:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

In that way, if we write the quantum state $\alpha |0\rangle + \beta |1\rangle$ in a vector notation as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

Table 2.1: Some of the most used single-qubit gates.

Quantum Gate	Matrix
Identity	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
X-gate (σ_x)	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Y-gate (σ_y)	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Z-gate (σ_z)	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

then the corresponding output from the quantum gate is

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}.$$

Notice that the action of the gate is to take the state $|0\rangle$ and replace it with the state corresponding to the first column of the matrix X. Similarly, the state $|1\rangle$ is replaced by the state corresponding to the second column of the matrix X. In particular, the X gate can be seen as a single-qubit rotation through π radians around the x-axis and is sometimes written as σ_x.

Thus, quantum gates acting on a single qubit can be described by 2×2 matrices. However, there are constraints on the matrices that may be used as quantum gates. Specifically, the normalization condition requires that $|\alpha|^2 + |\beta|^2 = 1$ for a quantum state $\alpha|0\rangle + \beta|1\rangle$. This condition must also hold for the quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ after the gate has acted.

The appropriate condition on the matrix representing the gate is that the matrix U describing the single-qubit gate be unitary, i.e., $U^\dagger U = I$, where $U^\dagger$ is the adjoint of U, obtained by transposing and then complex conjugating U, and I is the 2×2 identity matrix. For example, for the X gate, it is easy to verify that $X^\dagger X = I$.

Remarkably, this *unitarity* constraint is the only constraint on quantum gates. Any unitary matrix specifies a valid quantum gate. This interesting implication contrasts with the classical case, where only one non-trivial single-bit gate exists.

The most common single-qubit gates are presented in 2.1. The set of gates $P = \{I, X, Y, Z\} = \{I, \sigma_x, \sigma_y, \sigma_x\}$ is called the *Pauli* set, When the Hadamard gate operates on a quantum state, it trans-

forms the basis states as follows:

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

The Hadamard gate has the property to transform the computational basis into a "perfect superposition" of the states $|0\rangle$ and $|1\rangle$. In particular, when measuring the states $|+\rangle$ and $|-\rangle$ using the standard measure M_0, M_1, the outcomes 0 and 1 have the same probability of occurring. In particular, this gate is one of the main "ingredients" for several quantum algorithms.

Between the gates shown so far, the Pauli gates and H have the interesting property that applying them two times reverts their effects (e.g. $H^2 = I$). Any matrix with this property is unitary, and in particular, represents a quantum gate. However, this property does not hold for every unitary (e.g. the gates S and $\pi/8$).

Before moving on to multiple qubit gates, we can define three classes of unitary matrices that are "generated" from the Pauli gates. These matrices are called *rotation operators*.

The rotation operator about the $\hat{x}$ axis, $R_x(\theta)$, is defined as:

$$R_x(\theta) \equiv e^{-i\theta X/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

In the same way, the rotation operators about the $\hat{y}$ and $\hat{z}$ axis are defined as:

$$R_y(\theta) \equiv e^{-i\theta Y/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

$$R_z(\theta) \equiv e^{-i\theta Z/2} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}.$$

The notation e^A where A is a square matrix is used to represent the *matrix exponential*, which can be defined as

$$\exp A = \sum_{k=0}^{\infty} \frac{1}{k!}A^k. \tag{2.21}$$

The rotation operators are our first example of a *parametric gate*, which is a gate that depends on a certain parameter. This kind of gates are of fundamental importance for Quantum Machine Learning. Finally, we can use this class of operators to fill the gap between the definition of operators given in Postulate 2 and the one presented in the Quantum Circuit model. In particular, the former consisted of unitary operators that are time-dependent $U(t)$, while all of the gates defined so far are not. The explanation lies in the fact that a quantum device, in practice, does not "apply", for example, a gate X to the qubit, but applies a gate $R_x(t)$ for a certain time t such that its application is equivalent to (or at least, a good approximation of) the gate X.

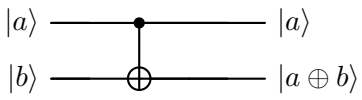


Figure 2.2: Representation of the CNOT gate. The dot indicates the wire on which the control is performed.

2.4.2 Multiple Qubits gates

Now, let us extend our discussion from single qubits to multiple qubits. An essential theoretical result in classical computing is that any function on bits can be computed using only `NAND` gates, making them universal gates. On the other hand, the XOR gate, alone or in combination with NOT, is not universal. The limitation of XOR gates becomes evident when considering parity-preserving operations, where circuits composed of only NOT and XOR gates will maintain the parity of inputs, thus restricting the class of computable functions and precluding universality.

In the realm of quantum computing, the prototypical multi-qubit quantum logic gate is the controlled-`NOT` or `CNOT` gate. This two-qubit gate has a control qubit and a target qubit. The circuit representation is shown in the figure 2.2. The top line represents the control qubit, and the bottom line represents the target qubit. The `CNOT` gate operates as follows: If the control qubit is set to 0, the target qubit remains unchanged. If the control qubit is set to 1, the target qubit is flipped. This behavior can be described by the following transformations for the computational basis states: $|00\rangle \to |00\rangle; |01\rangle \to |01\rangle; |10\rangle \to |11\rangle; |11\rangle \to |10\rangle$.

Another way to view the `CNOT` gate is as a generalization of its classical counterpart, where its action can be summarized as $|A, B\rangle \to |A, B \oplus A\rangle$, with $\oplus$ representing addition modulo two, which precisely reflects the gate's behavior. The control qubit's state is effectively copied and stored in the target qubit.

Furthermore, the `CNOT` gate's action can be represented using a matrix representation in the following way:

$$U_{\text{CNOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

It can be verified that the first column of the matrix U_{CNOT} describes the transformation applied to $|00\rangle$, and similarly for the other computational basis states $|01\rangle$, $|10\rangle$, and $|11\rangle$. Analogous to the single qubit case, the requirement for probability conservation is expressed by the fact that U_{CNOT} is a unitary matrix, i.e., $U_{\text{CNOT}}^{\dagger} U_{\text{CNOT}} = I$.

The fact that every gate must be a unitary matrix can apparently limit the expressivity of a quantum circuit. As we said before, the `CNOT` gate can be understood as a generalization of the `XOR` gate. However, doing the same kind of consideration on the other classical gates such as the `NAND` or the `XOR` gate is not possible, because the operations they describe are not *invertible*. In other words, we are not able to guess the inputs from the output. For example, if $A \oplus B = 1$, we know that either $A = 0$ or $B = 0$, but we are not able to determine which one of the two is correct. Therefore, we can say that there

is a *loss of information* associated with this gate. To simulate a 2 bit function we therefore need to have 2 outputs (of course, an analogous result holds for n-bit functions) and perform something called *reversible computation*, as we did before for the `CNOT` gate. However, is important to note that we can simulate any classical function with a quantum circuit and that therefore quantum circuits are at least as expressive as classical circuits.

Another interesting gate is the `SWAP` gate, also known as the exchange gate. It allows for the exchange of quantum states between two qubits. The `SWAP` gate is particularly useful when qubits need to be rearranged or when entanglement needs to be transferred between qubits. The `SWAP` gate can be defined as follows: it takes two qubits as input and swaps their quantum states. Mathematically, the `SWAP` gate can be represented using a matrix representation as:

$$U_{\text{SWAP}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The action of the `SWAP` gate on the basis states can be described as follows:

$$|00\rangle \to |00\rangle$$

$$|01\rangle \to |10\rangle$$

$$|10\rangle \to |01\rangle$$

$$|11\rangle \to |11\rangle$$

From these equations, it is evident that the `SWAP` gate swaps the amplitudes of the states $|01\rangle$ and $|10\rangle$, while leaving the states $|00\rangle$ and $|11\rangle$ unchanged.

Is interesting to note that the `SWAP` gate can be obtained from `CNOT` gates. In particular, is easy to verify that swapping the amplitudes of the first and the second qubits is equivalent to apply a `CNOT`$[0, 1]$, a `CNOT`$[1, 0]$ and then again a `CNOT`$[0, 1]$, where the first number in the bracket represents the control bit, and the second the bit on which we are applying the not.

The fact that a certain unitary can be represented in different ways is not surprising. A more unexpected fact is that we can actually represent *any* unitary on n-qubits only with single qubit gates and `CNOT` gates. We can say that, in a similar fashion to how the classical `NAND` is *universal* for classical circuits, the `CNOT` gate is universal for quantum circuits. More precisely, it can be used to form a *universal set of gates*, that can be used to provide a *finitary* representation of any unitary (equivalently, of any quantum gate or any quantum circuit). We are going to analyze this "decomposition" more in-depth in the following section. However, before that is useful to introduce another class of multiple qubits gate: rotations generated by a *Pauli string*.

Given a n-qubits system, n Pauli matrices $\sigma_k \in \{I, \sigma_x, \sigma_y, \sigma_z\}$, and $\theta \in \mathbb{R}$ we define the unitary

$$G(\theta) = \exp\left(-i\frac{\theta}{2}H \right), \tag{2.22}$$

where $H = \bigotimes_{k=1}^{n} \sigma_k$ and σ_k operates on the k-th qubit. The unitary $G(\theta)$ is said to be *generated* by the Pauli string $S_1 \dots S_n$ where $S_j \in \{\sigma_x, \sigma_y, \sigma_z, \sigma_0 = Id\}$. In practice, we are interested in Pauli strings of length two, as any rotation generated by a Pauli string can be decomposed in the product of rotations generated by Pauli strings of length two. In the following chapters, we are going to denote by $R_{S_1S_2}(\theta)[i,j]$ the operator induced by string S_1S_2 that can be obtained as the tensor product of the Pauli gate S_1 that operates on the qubit i, with the Pauli gate S_2 that operates on the qubit j. These unitaries are the fundamental building blocks of our model used to generate quantum kernels.

2.5 Universal set of gates

As we said before, any unitary operator corresponds to a quantum gate. We expect a quantum computer to be able to implement any quantum circuit. However, it is not possible to build a device able to exactly realize every possible gate. The idea that allows a quantum computer to implement a generic gate consists in the *universality* of certain sets of gates. In particular, we have that is possible to approximately implement any quantum gate with arbitrary precision by using a sufficient number of "elementary" gates from a finite set. If a set of gates has this property, it is called *universal*. The existence of a finite universal set of gates is given by the following theorem:

Theorem 2.5.1 *The set of gates consisting in Hadamard,* `CNOT` *and* $\pi/8$ *gates is universal.*

Proving in detail this theorem is beyond our scopes. However, it is interesting to present the intermediate steps required for this proof. The first two steps show that any unitary can be decomposed into simpler gates.

Property 1 *Any unitary matrix* U *which acts on a* n*-dimensional Hilbert space can be written as*

$$U = \prod_{i=1}^{4^n} T_i, \tag{2.23}$$

where T_i *acts non-trivially only on two or fewer vector components.*

A matrix with the latter property is called *two-level*. The number of the two-level matrices required for a general unitary is exponential, and is *optimal*: there exist some unitaries that cannot be decomposed in less than $O(4^n)$ matrices. For specific unitary matrices, it may be possible to find much more efficient decompositions.

For example, consider

$$U = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 & 0 & c \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 & d \end{bmatrix}$$

where $\widetilde{U} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ is a unitary matrix.

Is easy to verify that this matrix operates non-trivially only on the basis states $|000\rangle$ and $|111\rangle$, leaving unchanged the other 6 components. More in general, every two-level matrix has this form up to permutations of rows and columns.

This result implies that the set of every two-level matrices is universal. However, every two-matrix can be further decomposed into "simpler" gates:

Property 2 *Any two-level matrix can be decomposed into the product of single qubit gates and* `CNOT`*.*

Combining these two results, we obtain that the set composed from every single qubit gate with the `CNOT` gate is universal. This set is intuitively "smaller" than the latter, but is still "too big", as we cannot build a quantum device able to implement *any* single qubit gate. The next and last step is to find a discrete set of gates that approximates arbitrarily well any unitary.

How can we define the level of "approximation" to a given unitary? Suppose U and V are two unitary operators on the same state space. U is the target unitary operator that we wish to implement, and V is the unitary operator that is actually implemented in practice. We define the error when V is implemented instead of U by

$$E(U,V) \equiv \max_{|\psi\rangle} \left\| (U-V)|\psi\rangle \right\|^2 \equiv \|U-V\| \tag{2.24}$$

where the maximum is over all normalized quantum states $|\psi\rangle$ in the state space.

We can finally state the last property:

Property 3 *Hadamard and $\pi/8$ gates can approximate any single qubit operation to arbitrary accuracy.*

Using our notation, with "arbitrary accuracy" we actually mean "with an error arbitrary small". Even without dealing with the theory in detail, it should be clear that Theorem 2.5.1 is an easy corollary of these properties.

Before going further it is interesting to spend a few words about how this idea of arbitrary accuracy gets translated into the measurement part of the circuit. In other words, does the fact that $E(U,V)$ is small imply that the difference in the measurement of a state after performing U or V is small? The answer is yes. To show this important result, let M be a measurement, $|\psi\rangle$ the initial state of the system, and P_U and P_V the probability of obtaining the corresponding outcome after the operation U and V respectively. Then

$$|P_U - P_V| = \left| \langle\psi| U^\dagger M U |\psi\rangle - \langle\psi| V^\dagger M V |\psi\rangle \right| \tag{2.25}$$

Let $|\Delta\rangle \equiv (U-V)|\psi\rangle$. From the Cauchy-Schwarz inequality, we have that

$$\begin{aligned} |P_U - P_V| &= \left| \langle\psi| U^\dagger M |\Delta\rangle + \langle\Delta| M V |\psi\rangle \right| \\ &\leq \left| \langle\psi| U^\dagger M |\Delta\rangle \right| + \left| \langle\Delta| M V |\psi\rangle \right| \\ &\leq \left\| |\Delta\rangle \right\| + \left\| |\Delta\rangle \right\| \\ &\leq E(U,V). \end{aligned} \tag{2.26}$$

The inequality $|P_U - P_V| \leq 2E(U,V)$ gives a quantitative foundation of the idea that when the error $E(U,V)$ is small, the difference in probabilities between measurement outcomes is also small. This proves that a construction with a good approximation corresponds to outcomes that are a good approximation of the correct outcome.

To recap, an arbitrary unitary operation U can be implemented on a quantum computer using a circuit consisting of single qubit gates and controlled-`NOT` gates. This *universality* result is important because it ensures we can implement (up to arbitrary accuracy) any quantum gate from a finite set of elementary ones.

This result, however, does not give any information about how many gates we need to approximate a unitary with a given accuracy ϵ. A very important theorem shows that any single qubit gate U can be approximated with accuracy ϵ using only $O(log^c(1/\epsilon))$ gates from a fixed finite set of gates, where c is a constant that can be made to be less than 4 (the value of c can be made lower than this, but the optimal value is not relevant to our scopes). Before stating the theorem, we need some nomenclature.

First of all, $SU(2)$ is the *special unitary group* of degree 2. It is the group of the unitary matrices 2×2 having determinant 1. Formally,

$$\mathrm{SU}(2) = \left\{ \begin{pmatrix} \alpha & -\overline{\beta} \\ \beta & \overline{\alpha} \end{pmatrix} : \quad \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1 \right\} , \tag{2.27}$$

where the overline denotes the complex conjugate. A subset S of $SU(2)$ is said to be *dense* in $SU(2)$ if for any element U of $SU(2)$ and $\epsilon > 0$ there is an element $s \in S$ such that $\|s - U\| \leq \varepsilon$. We are now in the position to state the theorem.

Theorem 2.5.2 (Solovay–Kitaev) *Let $\mathcal{G}$ be a finite set of elements in $SU(2)$ containing its own inverses and such that the group $\langle \mathcal{G} \rangle$ they generate is dense in $SU(2)$. Consider some $\varepsilon > 0$. Then there is a constant c such that for any $U \in SU(2)$, there is a sequence S of gates from $\mathcal{G}$ of length $O(\log^c(1/\varepsilon))$ such that $\|S - U\| \leq \varepsilon$.*

A consequence of this important theorem is that a quantum circuit of m qubit gates can be approximated to ε error by a quantum circuit of $O(m \log^c(m/\varepsilon))$ gates from a desired finite universal gate set. By comparison, just knowing that a gate set is universal only implies that a general gate can be approximated by a finite circuit from the gate set, with no bound on its length. So, the Solovay–Kitaev theorem shows that this approximation can be made very efficiently provided that $\mathcal{G}$ is dense in $SU(2)$, thereby justifying that quantum computers need only implement a finite number of gates to gain the full power of quantum computation.

2.6 Quantum computational complexity and some quantum algorithms

In this section, we present some quantum complexity classes and we state some interesting *oracle separations* between classical complexity classes and quantum ones. In particular, we briefly present the Deutsch-Jozsa algorithm to separate `P` from `EQP`, and Simon's algorithm to separate `BPP` and `BQP`. After that, we present Shor's algorithm for the *factorization* problem, which is one of the most famous

quantum algorithms due to its possible consequences in cryptography. Finally, we present some considerations about the amount of *structure* is needed in a problem in order to achieve an exponential speed-up with a quantum device.

2.6.1 Complexity classes and their definitions

Computational complexity is one of the most known fields of Computer Science, and its analysis is fundamental to understanding how much an algorithm is efficient, and more in general, it allows us to understand if a problem is "easy" or "hard" to solve. Concepts such as the complexity classes `P`, `NP`, and `BPP` are widely known, therefore we are not going to present their formal definition. Informally, we can consider `P` as the class of problems that are efficiently solvable by a classical computer, and `NP` as the class of problems whose solution is verifiable in polynomial time. The class `BPP`, from *bounded-error probabilistic polynomial time*, can be considered as the class of problems that admits an efficient, probabilistic algorithm. For this reason, `BPP` is often called a *practical* class. Here, the word "efficiently" has been used as "that runs in polynomial time". Problems `NP`-*complete* (the "hardest" problems in the class) do not have a known polynomial solving algorithm, and so cannot be solved efficiently. However, some of them are commonly used and solved for many tasks (e.g. *Integer Linear Programming* and many more), as their intractability emerges only when the size of the input is "big" enough.

Quantum complexity classes can be defined in analogy with the classical ones. Specifically, we can use the formal definition of `P` and `BPP` by substituting the Turing machine with a quantum computational model, such as a *family of quantum circuits* or a *quantum Turing Machine*, to obtain the classes `EQP` (*exact quantum polynomial*) and `BQP` (*bounded-error quantum query*).

2.6.2 Deutsch–Jozsa's problem

The Deutsch-Jozsa algorithm, formulated by David Deutsch and Richard Jozsa in 1992, represents a deterministic quantum algorithm. While its practical utility remains limited at present, it stands as an early demonstration of a quantum algorithm that surpasses the computational speed of any deterministic classical algorithm exponentially. The Deutsch-Jozsa problem is intentionally tailored to be solvable by a quantum algorithm while posing a challenge to deterministic classical algorithms. This problem operates as a black box, efficiently solvable by a quantum computer without error, whereas a deterministic classical computer would require an exponential number of queries to the black box for problem resolution.

Problem 1 (Deutsch–Jozsa's problem) *Given an unknown function* $f : \{0, \dots, 2^n - 1\} \to \{0, 1\}$ *and a black box quantum gate* U_f *(the* oracle*) that implements* f*, given that* f *is either* constant *or* balanced*, determine which of the two types it belongs to.*

Before analyzing further the problem, is important to do a few considerations. First of all, this problem belongs to a class of *promise problems*, specific decision problems on which we already know some information about the input. Therefore, it would be more correct to talk about the *promise* version of quantum complexity classes. However, for simplicity's sake, we are going to omit this detail. Secondly, in the assumptions of the problem we already have an oracle that implements the function.

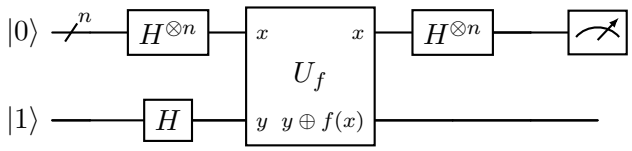


Figure 2.3: Scheme of the circuit for the Deutsch-Jozsa algorithm. The symbol $/^n$ indicates that we are considering n qubits and wires.

In practice, to implement the oracle we need way more information about f than just knowing if it is constant or balanced, therefore this problem does not have an important practical value.

From a theoretical point of view, however, this problem is able to *separate* the classes `P` and `EQP`. From a classical point of view, to deterministically evaluate the behavior of f we need $2^{n-1}+1$ evaluations of the function. However, the quantum Deutsch–Jozsa algorithm requires only one evaluation of U_f. This proves that the classes `P` and `EQP` are different since the Deutsch–Jozsa problem belongs obviously to the latter, but not to the former.

2.6.3 Simon's problem

Another interesting limitation of the Deutsch–Jozsa problem is that, despite being "hard" to deterministically solve on a classical computer, it can be easily approached with a probabilistic algorithm, therefore this problem does not provide a separation between classically tractable problems and quantum tractable problems. A separation (again, with an oracle) of this kind is given instead by Simon's problem.

Problem 2 (Simon's problem) *Given an unknown function $f : \{0, \dots, 2^n - 1\} \to \{0,1\}^n$ and a black box quantum gate U_f that implements f, given that for some unknown $s \in \{0,1\}^n$ we have that for all $x, y \in \{0,1\}^n$,*

$$f(x) = f(y) \text{ if and only if } x \oplus y \in \{0^n, s\}, \tag{2.28}$$

determine s.

Intuitively, solving the Deutsch–Jozsa problem using classical methods, even with randomness and a small probability of error, is challenging. The underlying difficulty can be understood quite simply: in order to solve the problem classically, one needs to find two distinct inputs, denoted as x and y, such that $f(x) = f(y)$. However, the function f may lack any discernible pattern or structure that would aid in identifying such inputs. Specifically, our knowledge about f and its behavior is limited to instances where different inputs yield the same output. Consequently, in order to locate a pair of inputs on which f produces the same output, we would need to make a large number of guesses, on the order of $O(\sqrt{2^n})$, before having a reasonable likelihood of success. On the other hand, a quantum device is able to solve this problem in $O(n)$ queries of a certain quantum circuit (we are not interested in all the details).

In conclusion, Simon's problem proves a separation (again, an oracle separation of the promise variant) of the classes `EQP` and `BQP`, proving that at least from a theoretical point of view, and under

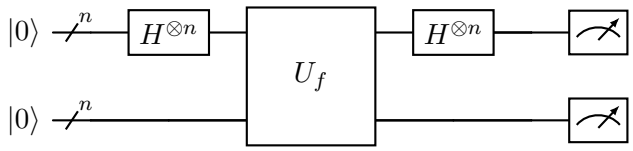


Figure 2.4: Scheme of the circuit for the Simon's algorithm.

certain conditions, quantum computing is able to solve problems that are *intractable* in the classical setting.

2.6.4 Shor's algorithm

Despite the importance of the last two results from a theoretical point of view, the problems presented are not very useful, as there is no known application to any real-world problem. Shor's algorithm is able to solve a very important problem that is believed to be hard: prime factorization.

Problem 3 (Prime Factorization) *Given a positive integer N, find its prime factors.*

On a quantum computer, Shor's algorithm provides a polylogarithmic-time solution for factoring an integer N. This implies that the time required by the algorithm is polynomial in $\log N$, where $\log N$ represents the size of the input integer. As a result, Shor's algorithm demonstrates the efficient solvability of the integer factorization problem on a quantum computer, placing it within the complexity class `BQP`. Notably, this algorithm outperforms the most efficient known classical factoring algorithm, the *general number field sieve*, which operates in sub-exponential time $O(e^{1.9(\log N)^{(1/3)}(\log\log N)^{(2/3)}})$.

In the scenario where a quantum computer possesses a sufficient number of qubits and can function without succumbing to quantum noise and other quantum-decoherence phenomena, Shor's algorithm could potentially be employed to break public-key cryptography schemes, including the widely-used RSA scheme. The RSA scheme relies on the assumption that factoring large integers is computationally intractable. Currently, there is no known classical algorithm capable of factoring integers in polynomial time, thereby affirming the validity of this assumption for classical (non-quantum) computers.

Is important to note that having no polynomial algorithm to classically solve the prime factorization problem does not mean that the problem is not in `P`. It is assumed that the factorization problem does not belong to `P` but is also not `NP`-complete. Clearly, proving the `NP`-completeness of the problem would establish the significant inclusion of `NP`$\subseteq$`BQP`, meaning that every `NP` problem can be solved efficiently by a quantum computer.

2.6.5 Structure required for an exponential advantage

The prime factorization problem is a very remarkable case of a useful, real-world problem on which is possible to achieve an exponential *quantum advantage*. Unfortunately, only a handful of similarly promising problems have been identified thus far. The "quantum ingredient" in Shor's algorithm consists of an application of the so-called *quantum phase estimation algorithm*, that is used to estimate the *phase* (or the eigenvalue) of an eigenvector of a unitary application, that uses in turn the *quantum Fourier*

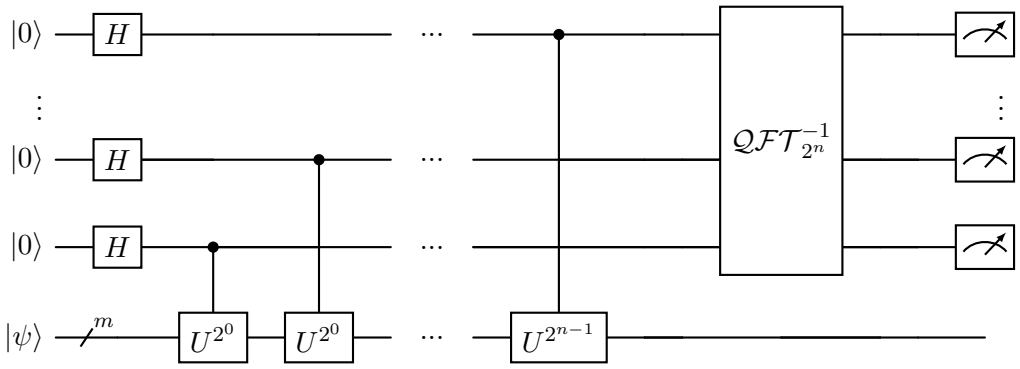


Figure 2.5: Circuit scheme for the phase estimation algorithm. The gate $\mathcal{QFT}_{2^n}^{-1}$ computes the inverse of the quantum Fourier transform.

transform (QFT). A substantial amount of quantum algorithms that present an exponential speed-up make use of the quantum phase estimation or QFT.

In [1], the author divides problems into four categories, according to the level of structure, and the achievable quantum speed-up. It turns out that, in order to achieve a quantum exponential speed-up, the problem must adhere to a specific kind of abelian group structure. Informally, if a problem has a certain structure involving periodicity, then a quantum algorithm may be able to exploit this property to achieve an exponential speed-up. This is the case for the prime factorization and the discrete logarithm problem. On the other hand, when there is minimal or no structure, we can achieve at most a polynomial speed-up.

Structure	**Speed-Up achievable**	**Example problem**
Abelian group structure	Exponential	Factorization
Nonabelian group structure	Exponential?	Graph Isomorphism
Minimal structure	Polynomial only	Collision Problem
No global structure	Polynomial only	PARITY

Table 2.2: Scheme of structure required and corresponding speed-up, from [1]. For nonabelian groups, is currently not known whether is possible a super polynomial speed-up.

In order to exploit the structure of the problem and achieve an exponential advantage, there are usually three steps required. First, we have to characterize the problem as the problem of estimating an eigenvalue of a unitary U. Second, we need to prepare efficiently the corresponding eigenvector $|\phi\rangle$. Third, we need to prepare efficiently the unitaries U^{2^j}, required for the application of the phase estimation subroutine, represented in Fig 2.5.

Understanding the level of structure of a given problem is therefore fundamental, as is conjectured that polynomial speed-ups, despite being interesting from a theoretical point of view, are not going to achieve a practical advantage over classical algorithms soon [3].

2.7 Quantum Machine Learning

Traditional machine learning methods, like deep neural networks, possess the intriguing ability to both recognize and generate data exhibiting consistent statistical patterns. Quantum Machine Learning

(QML) represents a paradigm shift in the field of artificial intelligence, harnessing the principles of quantum mechanics to enhance classical machine learning methods. QML algorithms leverage the power of quantum systems to not only recognize statistical patterns in data but also generate new data that should not be possible to simulate (in an efficient way) by a classical computer. The hope of QML is that, if a quantum device is able to produce patterns that are hard to simulate, then it is able to recognize similar patterns better than any possible classical device. The ability to find a quantum algorithm that over-performs any possible classical algorithm depends on a variety of considerations. One of the biggest is the current limitation of our quantum devices, which are currently prone to statistical noise and decoherence and are therefore very limited with respect to the number of qubits and operations. Our current stage of quantum computing technology is known as the *Noisy Intermediate-Scale Quantum* (NISQ) era. In this section, we are going to briefly present some of the most famous QML algorithms with some considerations about their possible NISQ usability. We will omit the discussion of Quantum Support Vector Machine, as it will be described in detail in Chapter 4.

2.7.1 Solving linear systems of equations: HHL Algorithm

Among all the proposed algorithms that promise an exponential speed-up over their classical counterparts, the one that seems to find most applications is the HHL, called after its inventors Aram Harrow, Avinatan Hassidim, and Seth Lloyd [19]. This algorithm addresses one of the most common problems in science: (approximately) solving a system of linear equations. Given a $n \times n$ real matrix A and a vector b, the HHL algorithm is able to find an approximated solution of

$$Ax = b \tag{2.29}$$

with logarithmic time complexity, therefore obtaining an exponential speed up. Despite not being originally formulated for Machine Learning applications, HHL laid the foundations of a wide class of QML algorithms, such as Quantum Linear Regression, Quantum recommendation systems, Quantum singular value thresholding, and many more (for a comprehensive list, see [17]). However, this algorithm provides a quantum speedup only under certain strict caveats that strongly limit its application. In particular, it needs

- the vector b to be loaded and stored in a QRAM (i.e. we need to prepare a quantum state $|b\rangle = \sum_{i=1}^{n} b_i |i\rangle$ of $\log_2 n$ qubits that encode b's entries)
- the quantum computer needs to apply the unitary e^{-iAt} for different values of t in an efficient way, which can be difficult when A is not *sparse*
- the matrix A needs to be *well-conditioned*, i.e. the ration $\lambda_{\max}/\lambda_{\min}$ must be "small"
- the output of HHL is not the vector x (as writing it down would require n steps as in the classical world), but a quantum state $|x\rangle$ of $\log_2 n$ qubits that approximately encode the entries of x as amplitudes.

These conditions, in addition to the fact that current NISQ devices do not implement error correction techniques and can utilize a small number of qubits, dramatically reduce the possibilities of effectively

implementing HHL [38].

2.7.2 Quantum Principal Component Analysis

The matrix's spectral decomposition characterizes a matrix by utilizing its eigenvectors and associated eigenvalues. Creating the optimal low-rank estimate of a matrix is achievable through the spectral decomposition technique, where eigenvalues and corresponding eigenvectors below a specified threshold are disregarded. This approach is commonly known as principal component analysis (PCA). It's applied, for instance, to form a low-rank approximation of the covariance matrix, which is positive semidefinite and symmetric, for sampled random vectors, and it finds extensive use in Machine Learning, mostly as a *dimensionality reduction* tool. However, the cost of PCA is prohibitive when one is presented with a large number of high-dimensional vectors. The Quantum PCA (or qPCA), presented in [24], allows the construction of the eigenvectors corresponding to large eigenvalues in time $O(\log d)$, where d is the dimension of the Hilbert space, providing therefore an exponential speed up.

As for HHL, qPCA requires some properties to be satisfied by the problem and significant resources, which are incompatible with our current NISQ devices.

2.7.3 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA), originally proposed in [18], is an algorithm that provides approximate solutions for certain combinatorial optimization problems. The most well-studied target problem for QAOA is Max-Cut, an NP-Hard graph problem. Max-Cut is equivalent to Unconstrained Quadratic Binary Optimization, which has a wide range of applications. Formally, given a graph (V, E), where V is the set of vertices, and E is the set of edges, the goal of the Max-Cut problem is to find a partition of V into two subsets V_1, V_2 (in other words, we want that $V_1 \cap V_2 =$ and $V_1 \cup V_2 = V$) such that the total number of edges connecting the two subsets is maximized, i.e.

$$\max \left| \{(u, v) \in E \quad \text{s.t.} \quad u \in V_1, v \in V_2\} \right|. \tag{2.30}$$

The first step to applying QAOA is to rewrite the problem as

$$\max_s \sum_{i,j \in V} w_{ij} s_i s_j$$
$$s_k \in \{-1, 1\}$$

where w_{ij} is 1 if $(i, j) \in E$ and 0 otherwise. In this formulation, the binary decision variables s_i tell if the vertex i belongs to V_1 or V_2. After that, it is possible to construct a *cost Hamiltonian* of the problem by mapping the binary variables s_k to eigenvalues of the Pauli Z operator σ_z, obtaining

$$H_C = \sum_{i,j \in V} w_{ij} \sigma_z^i \sigma_z^j. \tag{2.31}$$

QAOA is able to obtain a state of H_C that corresponds to a good approximation of the optimal solution of the original problem, by applying a series of p alternating operators, that in the limit $p \to \infty$ lead to

the true global optimum of the original problem.

In contrast to HHL and qPCA, QAOA requires a smaller amount of qubits and limited error corrections and is therefore suitable for NISQ applications. In particular, it is conjectured that a QAOA with hundreds of qubits would be able to solve problems that would be classically untractable [12]. However, it is expected that the class of problems on which a proper quantum advantage is obtainable is limited [25].

3

Background on kernel methods

In a supervised learning context, the *learning task* consists in finding a function $f : \mathcal{X} \to \mathcal{Y}$ that approximates the underlying mapping between the input space $\mathcal{X}$ and the output space $\mathcal{Y}$, given a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{M}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ for $i = 1, 2, \ldots, M$. If the problem is a *binary classification* one, we can assume that $\mathcal{Y} = \{-1, 1\}$. Moreover, we can generally assume that $\mathcal{X}$ is a subset of the Euclidean space $\mathbb{R}^d$.

One of the simplest models for this problem is the *Hard Margin Support Vector Machine*(SVM). In order to successfully use it, we need to assume that the training set $\mathcal{D}$ is *linearly separable*. In particular, it means that exists a $w \in \mathbb{R}$ with $\|w\|_2 = 1$ and a real number b such that

$$\begin{aligned} \langle w, x_i \rangle + b > 0 \quad &\text{for all } i \text{ such that } y_i = 1 \\ \langle w, x_i \rangle + b < 0 \quad &\text{for all } i \text{ such that } y_i = \text{-1}. \end{aligned} \tag{3.1}$$

In other words, we have that exists a hyperplane that perfectly separates the training set $\mathcal{D}$ into two sets. Under this hypothesis, a model that performs well on the training set is the function f defined by $f(x) = \text{sign}\langle w, x \rangle + b$. Of course, as long as the dataset is *strictly* separable, we have infinite pairs (w', b) that separate $\mathcal{D}$, and between all of them we are usually interested in the pair (w_D, b_D) that separates

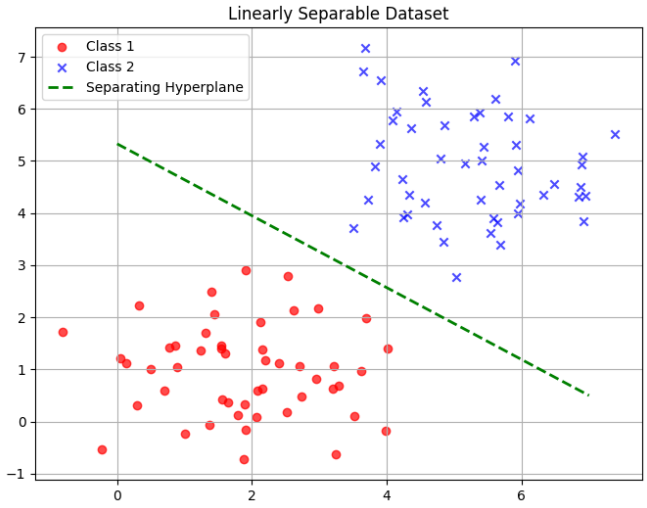
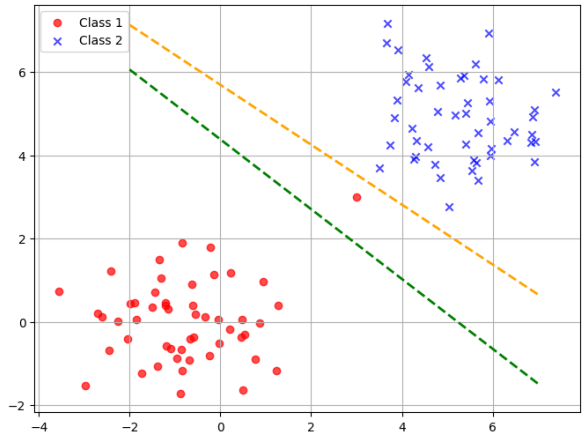
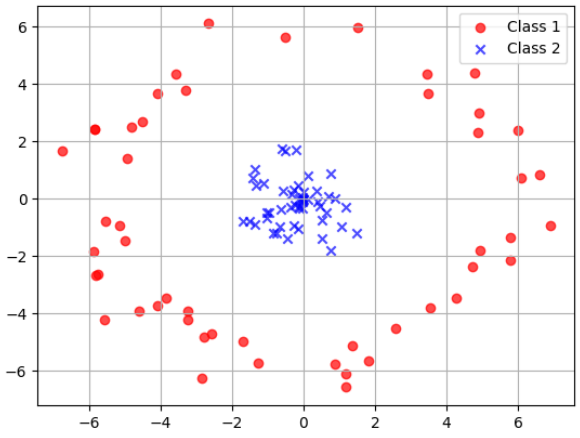


Figure 3.1: A linearly separable dataset with a corresponding separating hyperplane.



(a) Despite mislabelling a point in Class 1, the hyperplane in green gives a better margin and a better generalization than the one in orange.



(b) A classic example of a dataset on which a linear classifier cannot perform well. To correctly classify its data points, a non-linear model is required.

$\mathcal{D}$ with *maximal margin*, i.e. that maximize the minimum distance between the hyperplane and the training examples.

This model has two serious shortcomings:

- In the presence of noise, allowing some misclassifications may achieve a better generalization (Fig 3.2a);
- It can be used only when the relationship between data is linear. This means that this model should be used only for very "simple" tasks (Fig 3.2b).

To resolve the first issue, we can consider some models that take into account this possibility by relaxing the constraints (a more detailed explanation is given in the section with *soft-margin support vector machines*). The second issue is the central point of the application of *kernel-methods*. The idea consists of mapping the input data $(x_1, \dots, x_m)$ into a Hilbert space H_0 of higher dimension (possibly even infinite-dimensional), a non-linear function $\phi : \mathcal{X} \to H_0$. The Hilbert space H_0 and the function ϕ are usually called *feature space* and *feature map*. The obtained mapped dataset $\{(\phi(x_1), y_1), \dots, (\phi(x_m), y_m)\}$ may be linearly separable, allowing our linear classifier model to actually learn a non-linear decision function. Of course, this possibility depends on the choice of the feature map. However, certain feature maps exist such that any training set without contradicting examples (such that (x_i, y_i) and (x_j, y_j) satisfy $x_i = x_j$ and $y_i \neq y_j$) can be perfectly separated by a hyperplane in the feature space. However, this increased flexibility comes at the cost of the separating hyperplane now residing in a high-dimensional or even infinite-dimensional space.

3.1 Kernel Methods and Kernel Trick

Explicitly transforming the input data into the feature vector in the feature space, as we did before with the linear classifier, can have a serious computational cost, in particular, if the dimension of the feature space is high. However, certain models can operate in high-dimensional, *implicit* feature space without computing the coordinates of the data $\phi(x)$ in that space, by simply computing a certain inner product (called *kernel* function) between all the points in the starting space $\mathcal{X}$. This operation is often

computationally cheaper than the explicit computation of the coordinates. This approach is called the "*kernel trick*", and algorithms able to use linear classifiers to solve nonlinear problems with kernels are usually called *kernel machines*.

An example of a binary classifier that can be "kernelized" is a model that can be written as

$$f(x) = \operatorname{sign} \sum_{i=1}^{m} w_i y_i \langle \phi(x_i), \phi(x) \rangle, \tag{3.2}$$

where ϕ is a feature map $\mathcal{X} \to H_0$, and is possible to define a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for all $x, x' \in \mathcal{X}$, $k(x, x') = \langle \phi(x), \phi(x') \rangle$. In that case, is possible to write the same model as

$$f(x) = \operatorname{sign} \sum_{i=1}^{m} w_i y_i k(x_i, x), \tag{3.3}$$

without computing explicitly $\phi(x_i)$.

3.2 Properties and examples of kernels

In this section, we give a slightly more formal definition of kernel functions with some important properties.

Definition 3.2.1 *Let $\mathcal{X}$ be a non-empty set. Then, a function $k : \mathcal{X} \times \mathcal{X} \to \mathcal{K}$ is called a kernel on $\mathcal{X}$ if there exists a Hilbert space H_0 and a map $\phi : \mathcal{X} \to H$ such that for all $x, x' \in \mathcal{X}$, we have*

$$k(x, x') = \left\langle \phi(x), \phi(x') \right\rangle. \tag{3.4}$$

We call ϕ a feature map and H a feature space of k.

Kernels are closed under sum and product operations:

Property 4 (Sums of kernels) *Let $\mathcal{X}$ be a set, $\alpha \geq 0$, and k, k_1, and k_2 be kernels on $\mathcal{X}$. Then αk and $k_1 + k_2$ are also kernels on $\mathcal{X}$.*

Property 5 *Let k_1 be a kernel on $\mathcal{X}_1$ and k_2 be a kernel on $\mathcal{X}_2$. Then $k_1 \cdot k_2$ is a kernel on $\mathcal{X}_1 \times \mathcal{X}_2$. In particular, if $\mathcal{X}_1 = \mathcal{X}_2$, then $k(x, x') := k_1(x, x') \cdot k_2(x, x')$, $x, x' \in \mathcal{X}$, defines a kernel on $\mathcal{X}$.*

A trivial family of kernels is the one given by $k(x, x') = \langle x, x' \rangle + c$. The feature maps induced by kernels of this form are called *linear*, and they are not very useful on their own. However, given the last two properties, we are able to build from them the class of *polynomial kernels*:

Property 6 *Let $m \geq 0$ be an integer and $c \geq 0$ a real number. Then the function $k(x, x') = (\langle x, x' \rangle + c)^m$ is a kernel.*

Another famous family of kernels is the one that uses the exponential function.

Property 7 *Given $\gamma \geq 0$, the function $k(x, x') = \exp(-\gamma \left\| x - x' \right\|^2)$ is a kernel, called Gaussian radial basis function (RBF) kernel. Moreover, the corresponding feature space has an infinite number of dimensions.*

More in general, the theorem stated at the end of this section gives us a necessary and sufficient condition on a function k to be a kernel. To present it, we first need the following definition.

Definition 3.2.2 *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called positive definite if, for all $n \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ and all $x_1, \ldots, x_n \in \mathcal{X}$, we have*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_j, x_i) \geq 0. \quad (4.5)$$

Furthermore, k is said to be strictly positive definite if, for mutually distinct $x_1, \ldots, x_n \in \mathcal{X}$, equality in (4.5) only holds for $\alpha_1 = \ldots = \alpha_n = 0$.

Theorem 3.2.1 (Characterization of kernel functions) *A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if it is symmetric and positive definite.*

This fundamental theorem allows us to characterize kernel functions. Before moving further, we give another characterization of kernel function by a property of matrices.

Definition 3.2.3 *An $n \times n$ symmetric real matrix M is said to be positive-definite if $\mathbf{x}^T M \mathbf{x} > 0$ for all non-zero $\mathbf{x} \in \mathbb{R}^n$. Formally,*

$$M \text{ positive definite} \iff \mathbf{x}^T M \mathbf{x} > 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

Given $x_1, \ldots, x_n \in \mathbb{R}^n$, the $n \times n$ matrix $K := (k(x_j, x_i))_{i,j}$ is usually casayingd the *Gram matrix*. The condition of k being positive definite is equivalent to say that the correspondent Gram matrix is positive definite.

3.3 Support Vector Machines

In this section we see in detail an example of one of the most used Kernel Methods, the Support Vector Machine (SVM), to justify and apply some of the considerations done in this chapter.

3.3.1 Hard and Soft margin SVM

Let's recall the formulation of the linear classifier. Supposing that our dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{M}$ is separable, we want to find a pair (w, b) such that

$$y_n(\langle w, x_n \rangle + b) > 0 \quad \text{for } i \text{ in } 1, \ldots, M \quad (3.5)$$

$$\|w\| = 1, \quad (3.6)$$

where we can choose w *normalized* without loss of generality. Intuitively, we are interested in finding not a generic pair (w, b) that satisfies the constraint, but the one that does it with the maximal *margin*.

The margin is defined as the minimum of the distances between points x_i and the hyperplane. To take into account the margin r, we can reformulate our problem in this way:

$$\max_{w,b,r} \quad r \tag{3.7}$$

$$\text{subject to} \quad y_n(\langle w, x_n \rangle + b) > r \tag{3.8}$$

$$\|w\| = 1 \tag{3.9}$$

This formulation, despite being intuitively clear, is not the only possible, as we can just *scale* our dataset in a way such that the example x_a with minimum distance r from the hyperplane in the first formulation has distance 1. By doing so, simple calculations and geometric considerations allow us to rewrite the problem as

$$\max_{w,b} \quad \frac{1}{\|w\|} \tag{3.10}$$

$$\text{subject to} \quad y_n(\langle w, x_n \rangle + b) > 1. \tag{3.11}$$

Instead of maximizing the reciprocal of the norm of w, we can just minimize the squared norm, by obtaining the following quadratic optimization problem:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \tag{3.12}$$

$$\text{subject to} \quad y_n(\langle w, x_n \rangle + b) > 1, \tag{3.13}$$

where the factor $\frac{1}{2}$ is used to write in a more compact way the gradient of the function to minimize. This problem is known as the *hard margin SVM*, as in its formulation we are supposing that the dataset is separable and that therefore we can find a hyperplane that perfectly separates the data points. As considered in the first section, usually this is not the case, as we can have noise in the dataset, or the dataset is just not linearly separable. However, is possible to relax the constraints to obtain the so-called *soft margin SVM*, which allows some examples to fall within the margin region, or even on the wrong side of the hyperplane. The key idea is to introduce some *slack variables* ζ_n that corresponds to how much the correspondent x_n example is not well classified by the model.

$$\min_{w,b,\zeta} \quad \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{M}\zeta_n \tag{3.14}$$

$$\text{subject to} \quad y_n(\langle w, x_n \rangle + b) > 1 - \zeta_n, \tag{3.15}$$

$$\zeta_n \geq 0. \tag{3.16}$$

The parameter $C > 0$, usually called *regularization parameter*, trades off the size of the margin and the total amount of slack that we have. A large value of C implies low regularization, as it assigns higher weights to the slack variables. Consequently, more emphasis is placed on examples that do not lie on the correct side of the margin. In other words, the optimization process prioritizes minimizing

the classification errors on the training data, possibly leading to a narrower margin that may result in overfitting.

3.3.2 Dual SVM

The representation of SVM in the preceding sections, in terms of variables w and b, is referred to as the primal SVM. Recall that we deal with input vectors $x \in \mathbb{R}^D$ with D features. As w shares the same dimension as x, this implies that the number of parameters (the dimension of w) in the optimization problem increases linearly with the number of features. In this formulation, using a feature map of high dimensions increases the complexity of the problem.

In the following, we explore an equivalent optimization problem (known as the dual view) that is independent of the number of features. Instead, the number of parameters grows with the number of examples in the training set. Moreover, the dual formulation allows us to apply kernels, increasing the expressivity without increasing the complexity of the model.

The dual formulation is obtained by the application of the Lagrange multiplier to the primal formulation. Since we are not interested in the mathematical derivation, we are not going to present all the calculations or the explicit form of the partial derivatives in the Lagrangian formulation of the problem. Between these formulas is however important the following one that allows us to write the vector w as a linear combination of the examples x_n:

$$w = \sum_{n=1}^{M} \alpha_n y_n x_n, \tag{3.17}$$

where $\alpha_n \geq 0$ are the Lagrangian multipliers corresponding to the classification constraints in the primal formulation. This formula is a particular instance of the *representer theorem*, which states that the solution of the optimization problem lies in the span of training data (more specifically, it holds that the optimal weight vector is an affine combination of the examples). This theorem also provides an explanation for the term "support vector machine." The examples x_n, where the corresponding parameters $\alpha_n = 0$, do not influence the solution w in any way. However, the remaining examples with $\alpha_n > 0$ are referred to as *support vectors* since they actively "support" the positioning of the hyperplane. Is important to note that the representer theorem can be formulated in a way more general setting, by using concepts that we did not introduce such as *empirical risk minimization*.

We can finally present the dual SVM problem:

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \sum_{i=1}^{M} \alpha_i \tag{3.18}$$

$$\text{subject to} \quad \sum_{i=1}^{M} y_i \alpha_i = 0 \tag{3.19}$$

$$0 \leq \alpha_i \leq C. \tag{3.20}$$

As before, we are not interested in the mathematical derivation of this formulation or in the meaning of the constraints on the Lagrange multipliers α_i. There are two points that are relevant to us in the dual formulation. The first one is the size of the problem does not depend on the number of features d,

nor on the dimension of the feature map we use. This means that we can use a feature map of arbitrary complexity, without increasing the computational cost of solving this problem. The second point is that the actual values of x_i and x_j are never used in the function to minimize or in the constraints since we only need the inner product $\langle x_i, x_j \rangle$. This allows us to use a feature map ϕ without explicitly computing $\phi(x_i), \phi(x_j)$, which can be computationally expensive. The SVM formulation only requires the product $\langle \phi(x_i), \phi(x_j) \rangle$, therefore allowing us to use an appropriate kernel k instead. With these considerations, the dual formulation of a non-linear SVM that operates with the kernel k is

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} y_i y_j \alpha_i \alpha_j k(x_i, x_j) - \sum_{i=1}^{M} \alpha_i \tag{3.21}$$

$$\text{subject to} \quad \sum_{i=1}^{M} y_i \alpha_i = 0 \tag{3.22}$$

$$0 \leq \alpha_i \leq C. \tag{3.23}$$

From the solution α of the dual problem is then possible to easily compute the vector w solution of the primal problem.

3.4 Other examples of Kernel Methods

3.4.1 Kernel PCA

Principal component analysis (PCA) is a technique for extracting structure from possibly high-dimensional data sets. It is performed by solving an eigenvalue problem or using iterative algorithms that estimate *principal components*, coordinate values by which we can represent data in a space with a lower dimension. By using kernel functions we can find not principal components in input space, but in features that are non-linearly related to the initial input variables. In the classic case, given a set of *centered* observation $x_1, \ldots, x_M \in \mathbb{R}^D$, $\sum_{i=1}^{M} x_i = 0$, PCA diagonalizes the covariance matrix

$$C = \frac{1}{M} \sum_{i=1}^{M} x_i x_i^T. \tag{3.24}$$

Diagonalizing C results in a set of D orthonormal eigenvectors $v_1, \ldots, v_D$, and their corresponding eigenvalues $\lambda_1, \ldots, \lambda_D$. These eigenvectors are sorted in descending order of their associated eigenvalues, indicating the amount of variance they capture. The first principal component, v_1, corresponds to the eigenvector with the largest eigenvalue λ_1 and represents the direction along which the data exhibits the highest variance. PCA allows for dimensionality reduction by projecting the data onto a lower-dimensional subspace spanned by the selected principal components. By retaining the top K principal components, where K is usually much smaller than D, we obtain a reduced representation of the data in a K-dimensional space. PCA finds numerous applications across various domains, such as image processing, finance, genetics, and data compression. One common use case is in data visualization, where high-dimensional data is projected onto a $2D$ or $3D$ space to facilitate visualization and gain insights into the underlying patterns. Although PCA is widely used in many linearly separable datasets,

it may not be effective when dealing with complex, nonlinear data. To address this limitation, kernel PCA extends the technique by using a kernel function ϕ to map the data into a higher-dimensional space H_0. By finding principal components in this higher-dimensional feature space, it becomes feasible to capture nonlinear relationships among data points.

The kernel PCA consists of applying the standard PCA not to the space $\mathbb{R}^D$ containing the dataset x_i, but to the space H_0 containing the datapoints $\phi(x_i)$. The kernel trick in that setting is performed by directly computing the covariance matrix

$$\tilde{C} = \frac{1}{M} \sum_{i}^{M} \phi(x_i)\phi(x_i)^T \tag{3.25}$$

by exploiting the kernel representation of the feature map ϕ , i.e. the fact that we can compute $\phi(x_i) \cdot \phi(x_j)$ as $k(x_i, x_j)$ without explicitly computing $\phi(x_i)$ and $\phi(x_j)$.

3.4.2 One-Class SVM

Support Vector Machines are very good at classifying patterns in the domain of supervised learning. SVMs can be "adapted" to work in an unsupervised setting, such as in the field of *novelty detection*. In this section, we present the One-Class SVM model, which uses the same ideas of SVMs to detect a "region" in the input space where is located "most of the data". More formally, the strategy is to map the data into the feature space corresponding to the kernel and to separate them from the origin with the maximum margin possible. For each new point x, the model computes the value $f(x)$ by evaluating if x lies on the same hyperplane as the other data points. The corresponding optimization problem is the following:

$$\min_{w,b,\zeta} \quad \frac{1}{2}\|w\|^2 + \frac{C}{\nu}\sum_{n=1}^{d}\zeta_n - b \tag{3.26}$$

$$\text{subject to} \quad (w \cdot \phi(x_i)) > b - \zeta_n, \tag{3.27}$$

$$\zeta_n \geq 0. \tag{3.28}$$

In this formulation, w is a vector in the feature space H_0 and ν is a parameter in $(0, 1)$. The decision function f can be expressed as

$$f(x) = \text{sign}\Big(\sum_{n=1}^{M} k(x_i, x) - b\Big). \tag{3.29}$$

The dual problem can be expressed as

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i}^{M}\sum_{j}^{M}\alpha_i\alpha_j k(x_i, x_j) \tag{3.30}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{C}{\nu}, \tag{3.31}$$

$$\sum_{i}\alpha_i = 1. \tag{3.32}$$

Note that if ν approaches 0, the upper boundaries on the Lagrange multipliers tend to infinity, i.e. the slack variables ζ_i are strongly penalized. The problem then resembles the corresponding hard margin algorithm, since the penalization of errors becomes infinite. Therefore, the parameter ν characterizes the fraction of outliers. More formally, it holds:

Property 8 *If the solution of the primal problem satisfies $b \neq 0$, then:*

- *ν is an upper bound on the fraction of outliers*
- *ν is a lower bound on the fraction of Support Vectors (i.e. vectors x_i such that $\alpha_i \neq 0$).*

One Class SVM were first introduced as novelty detection algorithms, due to their versatility and generalization capability. In the following part of the thesis, we used the quantum equivalent (Quantum One-Class SVM) to address an anomaly detection problem.

4

Quantum kernels and their challenges

In this chapter, we describe how to build a "hybrid" model based on the classical SVM with the implementation of quantum kernels, and we present some issues and challenges in its applications. Before going further, is important to note that there exist different quantum variants of the classical SVM model, such as the Quantum Support Vector Machine (QSVM), a fault-tolerant quantum algorithm based on the HHL procedure, originally presented in [35]. In the following, we will refer to the hybrid classical-quantum learning model that takes classical input and evaluates a kernel function on a quantum device. The hybrid aspect of the model consists of the fact that the classification is performed in the standard classical manner, as described in Chapter 3. In particular, we will refer to the so-called *implicit formulation* [37, 27]. This approach relies on the observation that both quantum computing and kernel methods consist of performing computations in a high dimensional (possibly infinite) Hilbert space via an efficient manipulation of inputs. More in detail, in this model a quantum device takes classical input x and maps it to a quantum state by the use of a quantum circuit $U_\phi(x)$. Formally, the mapping

$$\phi : x \mapsto \left|\phi(x)\right\rangle = U_\phi(x) \left|000\ldots0\right\rangle \tag{4.1}$$

defines the kernel

$$K(x_i, x_j) = \left\langle 000\ldots0\right| U_\phi^\dagger(x_i) U_\phi(x_j) \left|000\ldots0\right\rangle . \tag{4.2}$$

Any state $|\psi_0\rangle$ can be used as the initial state instead of $|000\ldots0\rangle$. The computation described in the (4.1) is known as *fidelity test*, and it can be considered as a similarity measure between $\left|\phi(x_i)\right\rangle$ and $\left|\phi(x_j)\right\rangle$. An alternative to the fidelity test is the so-called *swap test*. The swap test is a procedure that, given two states $|\phi\rangle$ and $|\psi\rangle$, returns the "overlap" between them. More formally, the circuit (showed in 4.1) to implement the swap test requires the states $|\phi\rangle$, $|\psi\rangle$, and an ancilla qubit initialized to $|0\rangle$. At the beginning of the circuit, a Hadamard transformation maps the ancilla qubit into an equal superposition of $|0\rangle$ and $|1\rangle$. Then a controlled SWAP gate is performed on $\ker\phi$ and $\ker\psi$, where the ancilla is the control qubit. Finally, another Hadamard gate maps the ancilla qubit to the state $|\theta_{SW}\rangle$. Measuring the ancilla qubit results in state $|0\rangle$ with probability

$$P\left(|0_{ancilla}\rangle\right) = \frac{1}{2} + \frac{1}{2}\left|\langle\phi|\psi\rangle\right|^2 . \tag{4.3}$$

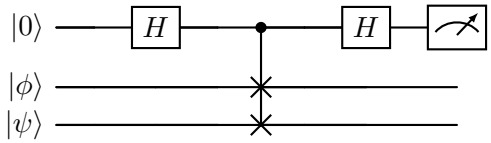


Figure 4.1: Circuit for the swap test. The probability of measuring 0 on the ancilla is proportional to the inner product.

By sampling this value more times we can estimate the squared inner product between the two states. The kernel $K(x_i, x_j)$ is then obtained by performing the swap test on the states $U_\phi(x_i), U_\phi(x_j)$.

4.1 Characterization of quantum kernels

The construction of a "good" quantum kernel is far from being an easy task. To date, the development of such constructions has predominantly relied on trial-and-error experimental methodologies or well-educated conjectures. This approach highlights the inherent complexity in quantum kernels' design, which lacks a unique evaluation criterion to measure their effectiveness and suitability for specific tasks.

The most common problems that should be addressed while constructing a kernel are related to *classical simulability*, *exponential concentration*, and *eigenvalue distribution*. The last two issues are both related to the *curse of dimensionality*, i.e. unfavorable properties of the model that arise when working with very high-dimensional spaces. Since exploiting the exponential (in the number of qubits n) dimension of the Hilbert space is one of the main advantages of quantum models compared to the classical ones, addressing these challenges is of crucial importance.

4.1.1 Efficient classical simulability

When talking about simulability it is important to remember that, theoretically, every quantum circuit can be simulated by a classic device. The fundamental consideration is that for "most" quantum circuits, we do not know how to simulate them *efficiently*, i.e. without an exponential blow-up in complexity. However, if a quantum circuit on n qubits can be simulated on a classical computer in $O(\text{poly}(n))$ time, then it does not provide any kind of quantum advantage, therefore making the quantum approach ineffective. To achieve a quantum advantage and justify the use of QML we need to identify and exclude these circuits from consideration. Unfortunately, determining whether a quantum circuit is efficiently simulable or not is a hard task. An interesting result that gives us a necessary condition for ensuring non-efficient simulability is given by the level of entanglement. A quantum circuit with no entanglement can be easily simulated by a classical computer (to get an intuitive understanding of the reasons, just consider that at each step of the computation, the quantum state can still be expressed as a single tensor product between basis states). However, this result holds even for circuits that present a low level of entanglement, as shown in [45]. This allows us to exclude from consideration circuits according to certain entanglement metrics, for example the Schmidt number [42] or the Meyer–Wallach Measure, an index able to quantify the ability of a circuit to produce entangled states [39]. On the other hand, having a high entanglement level does not mean that the circuit is not efficiently simulable, as shown in

[2]. Some more advanced techniques can give us a better approximation of the classical simulability of a quantum system (for example, in [41]), although at a high computational cost.

4.1.2 Exponential concentration

The *exponential concentration* is an issue that has an impact on the *trainability* of a model. Even though kernel methods do not have trainability problems (as the optimization problem is convex) when provided with exact kernel values, in a realistic quantum scenario the values of $k(x, x')$ can be only estimated via probabilistic measurements. Informally, when the difference between kernel values becomes too small, we need to do more measurements. If the number of measurements required is exponential in the number of qubits n, the model becomes infeasible. This phenomenon is known as exponential concentration. A more formal definition of exponential concentration in quantum kernel methods, adapted from the one given in [43], is the following:

Definition 4.1.1 *A quantum kernel $k(x, x')$ is said to be* deterministically exponentially concentrated *in the number of qubits n towards a fixed value μ if*

$$|k(x, x') - \mu| \leq \beta \in O(1/b^n) \tag{4.4}$$

for some $b > 1$ and all x, x'.

Analogously, $k(x, x')$ is said to be probabilistically exponentially concentrated *if*

$$Pr_{x,x'}\left[|k(x, x') - \mu| \geq \delta\right] \leq \frac{\beta^2}{\delta^2}, \beta \in O(1/b^n) \tag{4.5}$$

for some $b > 1$ and all x, x', δ.

In other words, the probability that $k(x, x')$ deviates from μ by a small amount δ is exponentially small for every x and x'. An equivalent definition of probabilistic exponential concentration is given by the variance of k, in particular if

$$\mathrm{Var}_{x,x'}\left[k(x, x')\right] \in O(1/b^n). \tag{4.6}$$

When we are able to obtain exactly the value $k(x, x')$ (e.g. for the classic SVM model), the exponential concentration is not a problem. In the quantum setting, in practice, we can only do a statistical estimation of the value of $k(x, x')$. If we perform N measurements, we have that the sampling error is of the order of $O(1/\sqrt{N})$. If a certain kernel suffers from exponential concentration, then distinguishing $k(x, x')$ from μ requires a number of measurements in the order of $O(1/b^n)$. However, for large-scale quantum devices, only a polynomial number of measurement shots is practically feasible. That leads us to an accuracy of $O(1/\mathrm{poly}(n))$. Under these conditions, the estimated value of $k(x, x')$ is dominated by statistical noise, and the differences in kernel values between different outputs cannot be discerned.

4.1.3 Expressibility

In [43], the authors identify four different possible sources of exponential concentration in a quantum kernel: *expressibility, entanglement level, global measurements* and *noise*. In the following, we will address only the first source.

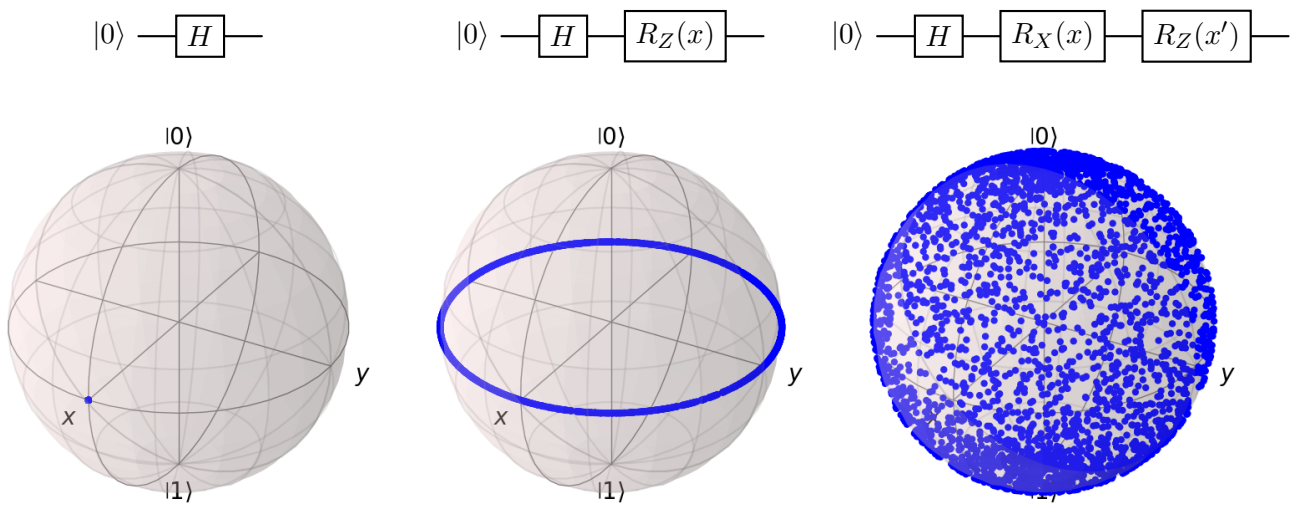


Figure 4.2: Representation of states on the Bloch sphere for different circuits, where each x, x' is sampled uniformly from $[0, 2\pi]$. Circuits with higher expressibility are able to "cover" higher portions of the Bloch sphere.

The expressibility of a quantum circuit U can be defined as its capability of simulating a wide range of unitary operations by the adjustment of its rotational parameters. More in detail, a circuit is expressive if it can be used to uniformly explore the unitary group $\mathcal{U}(d)$ [21], where $d = 2^n$ and n is the number of qubits. For example, when the system consists of a single qubit, this corresponds to a circuit's ability to explore the Bloch sphere, as shown in Figure 4.2.

One approach for computing this notion of expressibility is to compare the distribution of states obtained from the embedding by sampling its parameters to the uniform distribution of states in H, i.e. the so-called ensemble of *Haar random states*. The former object can be defined in the following way: given a unitary U, an initial state $|\phi_0\rangle$, and an input space $\mathcal{X}$, we can define the distribution of states

$$\mathcal{Y} = \{U(x)\,|\phi_0\rangle : x \in \mathcal{X}\}, \tag{4.7}$$

where the data x are sampled from $\mathcal{X}$. A way to quantify the non-uniformity is by using the definition of an ϵ-approximate state t-design [30]. A t-design of an ensemble is defined as an ensemble that simulates up to the t-th order statistical moment of the original ensemble on average, while an ϵ-approximate state t-design requires to simulate the t-th order statistical moment with an error of at most ϵ. Formally, $\mathcal{Y}$ is an ϵ-approximate state t-design of $\mathcal{Y}'$ if

$$\left\| \mathrm{E}_{|\phi\rangle\in\mathcal{Y}} \,|\phi\rangle\langle\phi|^{\otimes t} - \mathrm{E}_{|\phi\rangle\in\mathcal{Y}'} \,|\phi\rangle\langle\phi|^{\otimes t} \right\| \leq \epsilon, \tag{4.8}$$

where $\|\cdot\|$ is the unitary norm. A way of measuring the expressibility of an embedding U is by the distance between the set $\mathcal{Y}$ induced by U and an ensemble that forms a 2-design. Formally, this corresponds to the superoperator

$$A_{U,\mathcal{X}} = \int_{\text{Haar}} \left(|\phi\rangle\langle\phi|\right)^{\otimes 2} d\phi - \int_{\mathcal{X}} \left(U(x)\,|\phi_0\rangle\langle\phi_0|\,(U(x))^\dagger \right)^{\otimes 2} dx. \tag{4.9}$$

Analytically, the expressibility of U can be measured as the norm of the operator $A_{U,\mathcal{X}}$. From the formula 4.9 is clear that the expressibility of a circuit U depends not only on the embedding but also on the feature space $\mathcal{X}$ and on the initial state vector $|\phi_0\rangle$ (Figure 4.3).

It is possible to prove (*Theorem* 1 in [43]) that a higher expressibility leads to a greater quantum kernel concentration, requiring exponentially many measurements to evaluate the kernel on a real device. The intuition behind the mathematical proof is that for a highly expressive embedding, kernel estimation is equivalent to evaluating the inner product between two approximately random vectors, which, due to the high dimension of the space, are almost orthogonal, leading to values that are exponentially small.

4.1.4 Eigenvalues distribution

Another problem related to the curse of dimensionality is the *flat eigenvalues distribution* of quantum kernels, first noticed and proven in [22]. According to [7], a kernel function can be decomposed via Mercer's theorem, obtaining an eigendecomposition in an orthonormal basis of functions (and their corresponding non-negative real eigenvalues). When a component has a large eigenvalue, it can be learnt "easily" (in terms of sample complexity, i.e. the number of samples required in the training dataset), while components with small eigenvalues require more samples. In [22], the authors proved that expressible unitaries on n qubits lead to a Mercer decomposition with most of the eigenvalues having a magnitude of $O(e^{-n})$, that is therefore "unlearnable" (unless is provided an exponential amount of samples, which is usually not the case). In particular, according to their model, the task-model alignment, each orthogonal eigenfunction in the Mercer decomposition can be learnt with an amount of data inversely proportional to the corresponding eigenvector. Thus, the eigenfunctions corresponding to such small eigenvalues will need $O(2^n)$ data sample to be learnt.

4.1.5 Quantifying expressibility

The expressibility of a circuit can be computed as the operator norm of A given in 4.9. In practice, computing it analytically is very challenging. Since limiting the expressibility of a circuit is of fundamental importance to obtain a useful kernel, some less precise, but tractable estimations of expressibility have been proposed.

In [39], the authors proposed an estimation based on the *frame potential*, a quantity firstly introduced in [36] that measures whether an ensemble is a k-design. However, estimating the frame potential with good accuracy requires a large number of samples. A way to estimate the expressibility of an embedding with a lower computational cost (but also with lower precision) is presented in [16]. In this paper, the authors used some tools from the field of statistical learning theory to estimate the size of the hypothesis space generated by the circuit via its covering number. By doing so, they have shown that the covering number of the circuit, which can be expressed by the number of gates, is an upper bound on the expressibility of the circuit.

To limit the expressibility of a quantum circuit, different approaches have been used. One proposed solution, first presented in [23], imports some ideas from the field of Geometric Deep Learning (GDL) to take advantage of symmetries in the data. The key point is the same as the one used in the "classical" *GDL blueprint* [6]: given a problem that admits a set of symmetries $\mathfrak{G}$, models for it can be built from

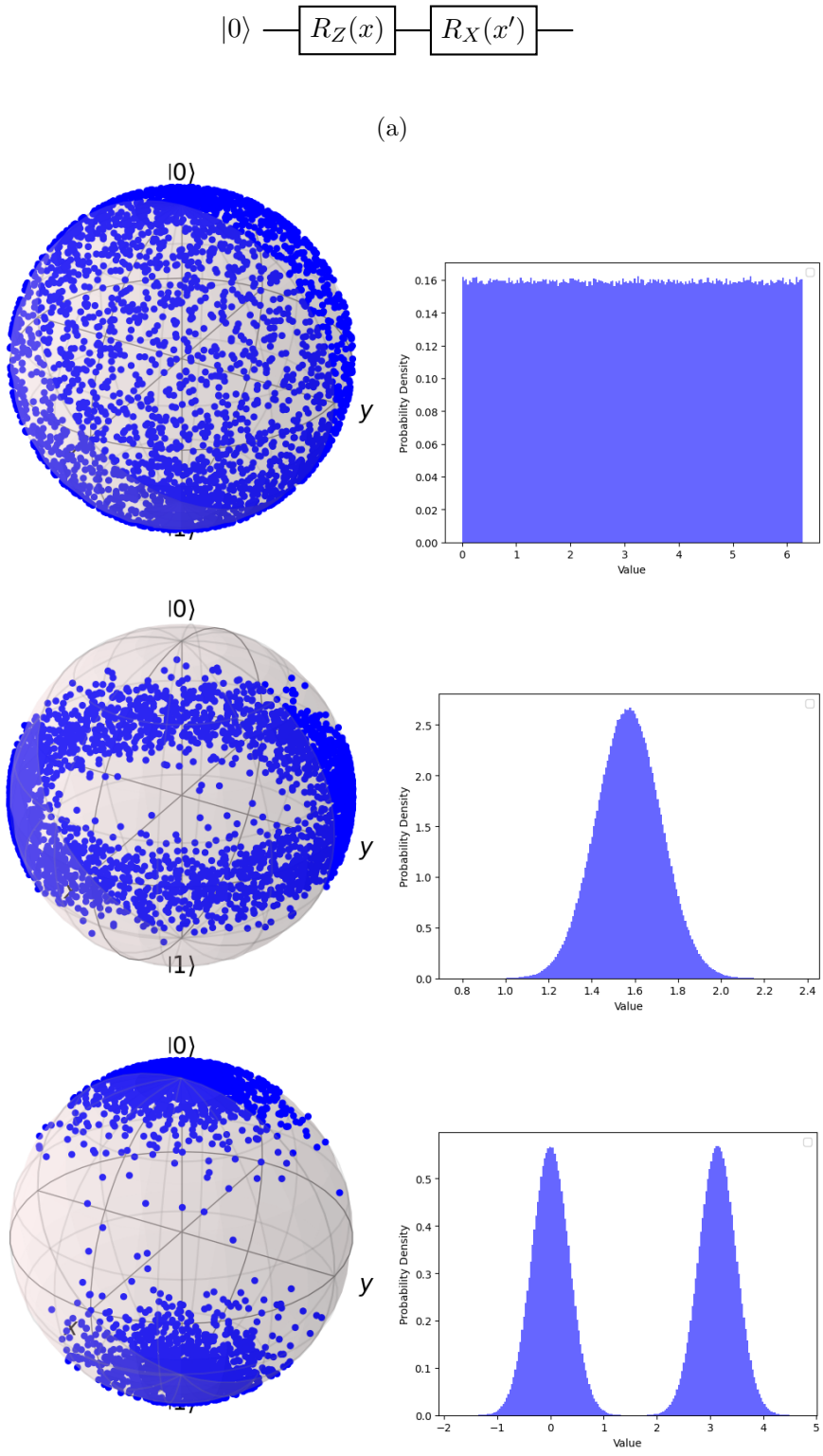
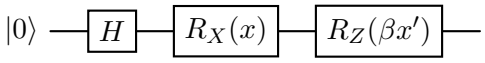
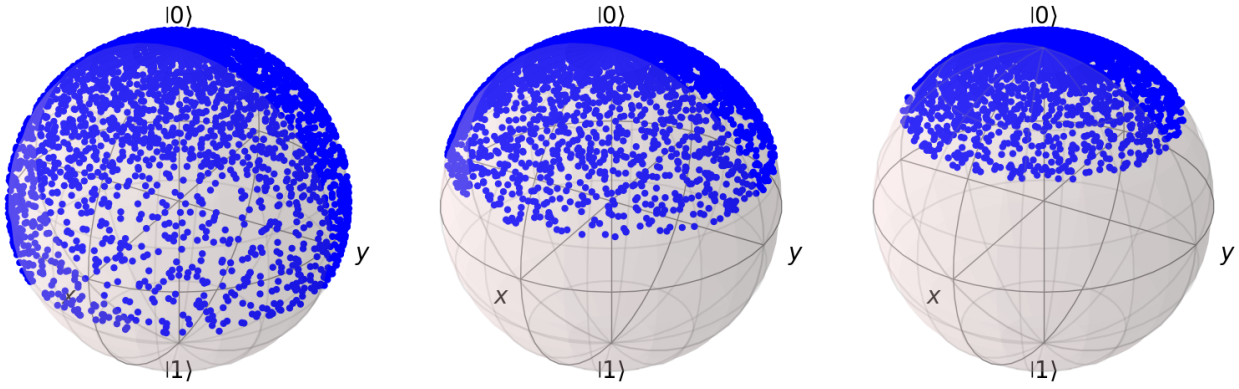


Figure 4.3: Representation of states on the Bloch sphere for the circuit (a), where x' is uniformly sampled from $[0, 2\pi]$ and x is sampled according to the distribution on the right.



(a)



(b) $\beta = 0.3$ (c) $\beta = 0.2$ (d) $\beta = 0.15$

Figure 4.4: Representation of states on the Bloch sphere for the circuit (a) for different values of the bandwidth parameter β. The values of x and x' are randomly sampled in $[0, 2\pi]$. Smaller values of β restrict the exploration of the Bloch sphere to smaller regions.

functions that are $\mathfrak{G}$-*invariant* and $\mathfrak{G}$-*equivariant*. By restricting circuits to ones with this property, we can obtain kernel with a reduced expressibility. A different, more general approach is the *bandwidth limitation*, proposed in [8]. The idea consists of restricting the exploration of the quantum space by introducing a parameter (called *bandwidth*) $\beta \in]0, 1[$, such that each rotation $R_i(x_j)$ in the embedding U can be written as $R_i(\beta x_j)$. The value of β can be determined heuristically. It can be proven that a sufficiently small bandwidth can reduce the circuit's expressibility, leading to better performance. More in general, we can consider a set of values $\beta_1, \ldots, \beta_k$ and write each rotation as $R_i(\beta_i x_j)$. This is the approach we decided to use for our solution. A visual representation of the effect of varying the bandwidth on a simple circuit is given in Figure 4.4.

5

Automatic Kernel Discovery

In the last chapter, we have presented some of the challenges that arise when using a quantum kernel. In general, finding a kernel that outperforms classical ML algorithms on a real-world dataset is a hard task. One of the most performing techniques for the design of suitable quantum kernels relies on exploiting the symmetries of the problem [23, 4, 28]. The ease of quantum kernels to recognize non-trivial symmetries justifies the conjecture that QML is well-suited to address problems from the fields of quantum chemistry, particle physics, and, more in general, any physical system that admits conserved quantities; in fact, we know from Noether's theorem that conserved quantities correspond to symmetries in the underlying system. The idea of exploiting the symmetries of the problem to generate a kernel has a shortcoming: to build such a kernel, we need some *knowledge* of the problem properties. In certain well-known scenarios, this is not a problem (e.g. a symmetry that is used in quantum chemistry application is that the energy of a state does not change if all spins are flipped, which leads to the invariance of the gate $X^{\otimes n}$). However, the lack of prior knowledge is a common situation for most real-world datasets. This happens in particular when dealing with anomaly detection, where knowing the properties of the phenomenon would often make the task trivial.

Another less knowledge-dependent idea is to choose kernels heuristically, for example by adapting them from kernels that perform well on similar problems. Despite being easier to implement, this idea is far from optimal, as it requires knowing that the addressed problem is similar to another one that can be solved with a certain kernel. The concept of "similarity" between two problems is fundamental, as due to the no free lunch theorem, a kernel that performs well on a certain problem may perform very badly on another one. The lack of a formal, knowledge-independent technique to construct a quantum kernel for a general problem makes it necessary to develop new ideas. In this work, we implement an algorithm able to automatically obtain and optimize a quantum kernel (by modifying the corresponding embedding) for a given problem, by using different metrics and optimization techniques. In the remaining part of the chapter, we give a description of our algorithm, with a focus on the circuit modeling, the metrics used to evaluate it, and the optimizers.

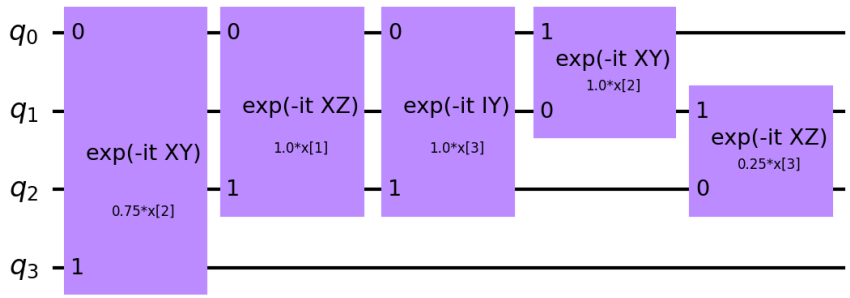
5.1 Modeling and optimizing the kernel

In our approach, the structure of an embedding U can be modeled as a sequence of L gates that operates on n qubits. Each gate acts on one or two qubits and is associated with a generator in $P = \{\sigma_0 = \mathrm{Id}, \sigma_x, \sigma_y, \sigma_z\}$ for single qubit gates or in $P^{\otimes 2}$ for two qubits gates, and an angle corresponding to one of the features of the dataset. Moreover, each gate can use a bandwidth parameter $0 < \beta < 1$ to limit expressibility. In this way, an embedding can be described by a $5 \times L$ matrix, where each column represents a gate, while the rows correspond to:

1. the generator $g \in P \cup P^{\otimes 2}$
2. the qubit on which the gate acts when g is a single qubit gate or the first qubit of the two when it is a two qubits one
3. the second qubit on which g acts, if g is a two qubits gate
4. the feature used as angle rotation
5. the bandwidth parameter.

An example of an embedding with the corresponding matrix is given in Figure 5.1.

Defined a parameterized quantum circuit, we can create a "traditional" quantum kernel, for which two data points are encoded in two quantum states and then their inner product is estimated via the SWAP test. Otherwise, we can calculate a projected quantum kernel by specifying a vector of n symbols in P, each referring to a qubit, where the Identity symbol means the qubit is not measured, Z means the usual measurement and X, Y denote the measurement in another basis. The final model of the kernel is given by the circuit matrix and the "measurement vector".



Generator	XY	XZ	IY	XY	XZ
First qubit	0	0	0	1	2
Second qubit	3	2	2	0	1
Feature	2	1	3	2	3
Bandwidth	0.75	1	1	1	0.25

Figure 5.1: An embedding with 5 gates, 4 qubits, and 4 features, represented as a circuit and as a matrix. The underlying transformation is $U(x) =$
$\exp(-i0.25 \cdot x_3 \sigma_x^2 \otimes \sigma_z^1) \cdot \exp(-ix_2 \sigma_x^0 \otimes \sigma_y^2) \cdot \exp(-ix_3 \sigma_z^2) \cdot \exp(-ix_1 \sigma_x^2 \otimes \sigma_z^0) \cdot \exp(-i0.75 \cdot x_2 \sigma_x^0 \otimes \sigma_y^3)$.

5.1.1 The algorithm

The Kernel Optimization algorithm aims to identify an optimal kernel for a given problem. The algorithm employs a combination of random kernel generation, optimization, and evaluation processes. Given parameters such as the number of gates L, the number of qubits n, and the set of generators G, the algorithm iteratively refines the kernel to minimize a specified cost function that depends on the evaluator.

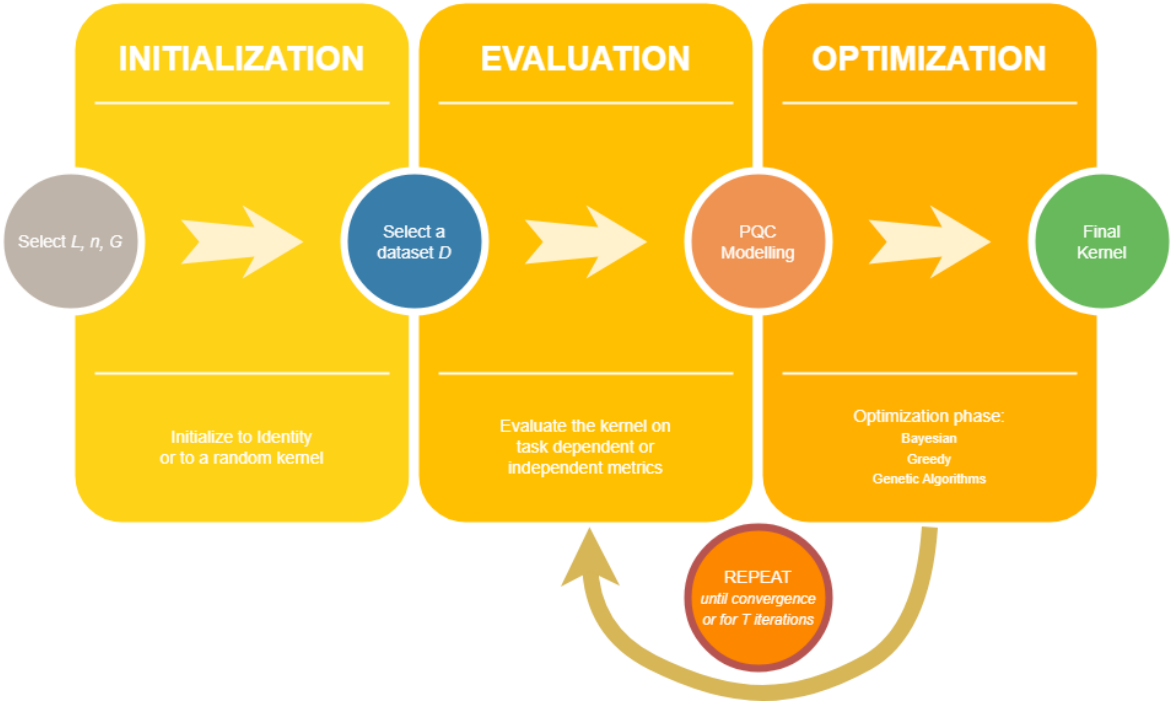


Figure 5.2: Scheme of our proposed algorithm for Automatic Kernel Discovery.

Algorithm 1 Kernel Optimization Algorithm

Require: Parameters: L, n, G
Require: Evaluator: *evaluator*
Require: Optimizer: *optimizer*
Require: (optional) Task: $\mathcal{D}$
  Initialize kernel: $kernel \leftarrow RandomKernel(L, n, G)$
  Compute starting performance: $performance \leftarrow evaluator(kernel, \mathcal{D})$
  while termination criterion not met **do**
    Perform optimization step: $kernel \leftarrow optimizer(kernel, performance)$
    Evaluate updated kernel: $performance \leftarrow evaluator(kernel, \mathcal{D})$
  end while
  return *kernel*

The key points of the algorithm are the evaluator and the optimizer. In our modeling, any function computed on the kernel can be used as a cost function. Different evaluators can compute functions with a wide range of desired properties, that can greatly differ in precision and computational cost. On the other hand, any optimization algorithm that is able to explore the search space of possible embeddings can be used to minimize the chosen cost. This allows us to have considerable flexibility in the selection of both evaluator and optimizer. In the following sections, we are going to present the implemented

evaluators and optimizers, with more emphasis on the ones that showed a better performance.

5.2 Evaluating the kernel: metrics

In order to optimize a kernel, we need an *evaluator*, i.e. a function e such that, given a kernel k, computes a value $e(k)$ that is correlated to the performance of the kernel and that we want to maximize. Our modeling allows us to use a very general class of functions without much requirements. For example, it would be possible to choose as evaluator a value that indicates the classical simulability of k, or the distance between the desired amount of expressibility. Of course, selecting the right evaluator is of fundamental importance, as some metrics can be less or more correlated to the actual performance of the model. Is important to distinguish two classes of evaluators: the ones *task-independent*, and the ones *task-dependent*. Evaluators that are related to the level of entanglement or expressibility belong to the former class. However, task-dependent metrics are of fundamental importance, as for the *no-free lunch theorem* we know that we cannot find a single kernel that performs well on all tasks. Another important consideration is regarding the complexity of the evaluator. Some evaluators can be easy to compute (e.g. kernel compatibility) or hard, so when choosing an evaluator is important to be sure that it is compatible with the number of calls required by the optimizer. Finally, is important to note that given two evaluators e and e' is possible to combine them to obtain a new evaluator (e.g. $e''(k) = e(k) + e'(k)$). This may be particularly useful when wanting to combine task-dependent and independent metrics.

5.2.1 Kernel compatibility

Kernel compatibility is a term used to describe if a certain kernel function is suitable (or compatible) for a certain task and is, therefore, a *task* dependent evaluator. A basic measure of compatibility is the *kernel-target alignment* [11] (KTA), which, as suggested by the name, measures the level of agreement between a kernel and a given learning task. It is defined by the normalized inner product between the Gram matrix of the kernel K and the label matrix $Y = yy^T$. The value is given by

$$\mathrm{KTA}(K, Y) = \frac{\langle K, Y \rangle_F}{\sqrt{\langle K, K \rangle_F \langle Y, Y \rangle_F}}, \tag{5.1}$$

where $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product. KTA between two matrices can be viewed as the cosine of the angle between two bi-dimensional vectorized matrices K and Y, and is, in particular, a similarity index that takes values between -1 and 1. Values close to 1 indicate a good level of compatibility. A variant of this metric, called Centered Kernel Alignment (CKA) [10], improves KTA by centering kernel matrices. In particular, CKA is computed as

$$\mathrm{CKA}(K, Y) = \mathrm{KTA}(HKH, HYH), \tag{5.2}$$

where H is the so-called *centering matrix*, defined as

$$H = \mathbb{I}_m - \frac{1}{m} \mathbb{1}\mathbb{1}^T. \tag{5.3}$$

The centered index is a better measurement for the generalization error. Despite not being very precise, kernel compatibility metrics have in general a very important advantage, which is being very computationally cheap. For this reason, they can be easily used with optimizers that require a considerable number of calls of the evaluator, such as for example a greedy optimizer.

5.2.2 Task-Model alignment and Spectral bias

An important model to evaluate a kernel, called the *task-model alignment*, has been introduced in [7]. The target function f and the learnt kernel machine $\hat{f}$ can be both expressed in the form $\sum \lambda_i \phi_i$ and $\sum \hat{\lambda}_i \phi_i$ where ϕ_i are the orthonormal eigenfunction of the kernel found via the Mercer's theorem and λ, $\hat{\lambda}$ are nonnegative eigenvalues. In this case, if the sequence of λ_i matches $\hat{\lambda}_i$ then the kernel machine works well for the function at hand. Analytically, the level of alignment can be measured by the *cumulative power distribution* $C(\rho)$, a function that measures the compatibility between the kernel and the target function f and that is correlated to the generalization error of the model. Furthermore, the amount of data needed to learn a certain component ϕ_i is proportional to $\lambda_i/\hat{\lambda}_i$. This phenomenon is called *spectral bias*. As described in Section 4.2.4, expressive quantum kernels lead to kernel machines having all the components with amplitude $O(2^{-n})$, that therefore need an exponential amount of data to be learnt. For these reasons, both task-model alignment and spectral bias provide parameters that can computed by the eigenvalues decomposition of the kernel, and that are good evaluators (respectively, task-dependent and task-independent) of the final performance of a kernel.

5.2.3 Performance-based metrics

The generality of the selection of the evaluation function allows us to select any kind of metric that depends on the kernel, which can be problem-dependent or not. Of course, since our aim is to find a good kernel for a certain problem, the most intuitive and simple idea is that a kernel can be considered good if it leads to a model that is good for the current task. In Machine Learning, the metric to evaluate a model can be usually selected from a wide range of functions that depend on the type of problem (e.g. *accuracy* for a classification problem, MSE for a regression one, etc.). It turns out that these functions can be used to evaluate a kernel. After selecting the type of model, the learning algorithm, and the relevant hyperparameters, given a kernel k we can build a model $\mathcal{M}$ that uses k as the kernel function, train it, and evaluate its performance according to a chosen metric m. Despite being one of the most accurate types of metric to evaluate performance, is important to note that according to the model, it can have a very consistent computational cost, and its usage is dependent on the chosen optimizer. However, performance-based metrics are well suited when training and testing a model is cheap, or when the selected optimizer requires a small number of calls of the evaluation function, for example when using optimizers suited to compute expensive, black-box functions such as in the Bayesian optimization.

5.3 Optimizers

An *optimizer* is an algorithm that takes in input the matrix of an embedding as described in Section 5.1 and eventually the corresponding measurement vector and gives in output a new kernel, optimized according to a certain evaluator. A strong point of the model proposed is that any optimization algorithm

can be used as the optimizer, as long as the algorithm is able to work on the data structure representing the circuit. An important consideration is that most of the values to optimize are discrete variables, therefore some optimizers that require continuous values can have a lower performance. On the other hand, the *bandwidth* parameter can be easily discretized. When selecting an optimizer, is crucial that it has a complexity cost suitable with the number of evaluations required by the optimizer (e.g. if the evaluation function is expensive to compute, then an optimizer that requires multiple evaluations per step may not be suitable). A scheme of the algorithms implemented with the corresponding costs is presented in 5.1. In our implementations, the only optimizer able to modify not only the circuit but also the measurement vector of the kernel is the *greedy algorithm* with measurement selection (Algorithm 3). However, the possibility of modifying the measurements does not generally imply a better performance.

Table 5.1: Optimization Approaches

Optimizer	**Optimization Cost**	**Number of evaluations**
Bayesian	High	Low
Greedy	Low	High
Genetic Algorithm	Medium	Medium
RL Approaches	Medium	Medium

5.3.1 Greedy Algorithm

A *greedy algorithm* is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. In our implementation, this translates to the selection of the optimal gate for each gate. In other words, at the beginning, a kernel with L gates is randomly generated and evaluated. Then, for each gate l in $\{1, \ldots, L\}$, the algorithm tries every possible combination of qubits, features, and bandwidths to produce a gate l', and then proceeds to test the kernel with l instead of l'. The algorithm scheme is presented in Algorithm 2. In the variant Algorithm 3, called greedy algorithm with *measure selection*, at each step (i.e. for every gate) is performed a subroutine of greedy measure selection. In other words, for each component i of the measurement vector M, the algorithm tries to change the i-th component with a different measure in $\{I, X, Y, Z\}$.

From a computational point of view, the optimization step is very cheap, as it consists of just building the kernel from the circuit. Every step requires an evaluation, therefore this optimization technique can be applied only for evaluations that are very cheap to compute. However, as shown in Table 5.2, the number of evaluations required grows quadratically in the number of qubits n and may not be suitable even for circuits with a limited number of qubits.

Variant	Evaluations required
without measure selection	$L \times \Big(\lvert G \rvert \times n \times (n-1) \times f \times b \Big)$
with measure selection	$L \times \Big(\lvert G \rvert \times n \times (n-1) \times f \times b + 4n \Big)$

Table 5.2: Comparison of the number of evaluations required for the standard greedy algorithm proposed and the variant with measure selection, where L is the number of gates, G the set of generators, n the number of qubits, f the number of features, and b the amount of possible bandwidth values.

Algorithm 2 Greedy Algorithm

Require: Parameters: L, n, G
Require: Evaluator: *evaluator*
Initialize kernel: *kernel* $\leftarrow$ RandomKernel(L, n, G)
Compute starting performance: *performance* $\leftarrow$ *evaluator*(*kernel*)
for gate l in $1, 2, \ldots, L$ **do**
  for generator g, qubits $q1$ and $q2$, feature f and bandwidth b **do**
    new_kernel $\leftarrow$ swap_gate(*kernel*, $l, g, q1, q2, f, b$)
    if *evaluator*(*new_kernel*) $>$ *performance* **then**
      Update *kernel* $\leftarrow$ *new_kernel*
      Update *performance* $\leftarrow$ *evaluator*(*new_kernel*)
    end if
  end for
end for

Algorithm 3 Greedy Algorithm with Measurement Selection

Require: Parameters: L, n, G
Require: Evaluator: *evaluator*
Initialize kernel: *kernel* $\leftarrow$ RandomKernel(L, n, G)
Compute starting performance: *performance* $\leftarrow$ *evaluator*(*kernel*)
for gate l in $1, 2, \ldots, L$ **do**
  for measure M in I, X, Y, Z **do**
    for qubit q in $1, 2, \ldots, n$ **do**
      new_kernel $\leftarrow$ swap_measurement(*kernel*, M, q)
      if *evaluator*(*new_kernel*) $>$ *performance* **then**
        Update *kernel* $\leftarrow$ *new_kernel*
        Update *performance* $\leftarrow$ *evaluator*(*new_kernel*)
      end if
    end for
  end for
  for generator g, qubits $q1$ and $q2$, feature f and bandwidth b **do**
    new_kernel $\leftarrow$ swap_gate(*kernel*, $l, g, q1, q2, f, b$)
    if *evaluator*(*new_kernel*) $>$ *performance* **then**
      Update *kernel* $\leftarrow$ *new_kernel*
      Update *performance* $\leftarrow$ *evaluator*(*new_kernel*)
    end if
  end for
end for

5.3.2 Bayesian Optimization

As in other kinds of optimization, in Bayesian optimization we are interested in finding the minimum of a function $f(x)$ on some bounded set X, which we can suppose to be a subset of $\mathbb{R}^D$. The idea behind Bayesian optimization is to construct a probabilistic model for $f(x)$ and then exploit this model to make decisions about where in X we need to evaluate the function next while integrating out uncertainty. Moreover, Bayesian optimization is able to use all of the information available from previous evaluations of $f(x)$ and not simply rely on local gradient and Hessian approximations. This results in a procedure that can find the minimum of difficult non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. Bayesian optimization shines when the evaluation of the function f is particularly expensive, as we can perform more computation about the next points to test, instead of computing more evaluations of the function. For this reason, this optimizer is often used in ML models to select hyperparameters, as searching for the optimal hyperparameters in a model is indeed an optimization problem, where each evaluation requires a full training of the model, often very time-consuming. This kind of advantage can be translated in our problem setting, when the Bayesian optimization can be paired with more expensive evaluations, such as the ones based on the model performance.

The core idea of the Bayesian optimization relies on the Bayes Theorem, which states that the *posterior* probability of a model M, according to the evidence E, depends on the *likelihood* of E given M, multiplied by the *prior* probability of M. In formulas,

$$P(M|E) \propto P(E|M)P(M). \tag{5.4}$$

In this setting, the prior is represented by our belief about the space of possible objective functions. In general, the space of functions is unknown, however is often possible to suppose that we have some information about the property of the target function (e.g. smoothness, etc.), therefore making some functions more plausible than others. The posteriors are given by our samples. Let's define x_i as the i-th sample and $f(x_i)$ as the corresponding evaluation. Then our dataset of observations can be written as

$$\mathcal{D}_{1,\dots,t} = \big(x_i, f(x_i)\big)_{i\in\ \{1,\dots,t\}}. \tag{5.5}$$

The posterior distribution can be therefore written as

$$P(f|\mathcal{D}_{1,\dots,t}) \propto P(\mathcal{D}_{1,\dots,t}|f)P(f). \tag{5.6}$$

The posterior represents our updated beliefs about the unknown objective function. After each iteration t, the optimization process updates the set $\mathcal{D}_{1,\dots,t}$ with the new sample $(x_{t+1}, f(x_{t+1}))$. To do so, we have to decide the next point x_{t+1} to evaluate. This process is made by the so-called *acquisition function*, a function that performs a decision based on a trade-off between *exploration* (where the objective function is very uncertain and there is a higher margin of improvement) and *exploitation* (trying values of x where the objective function is expected to be high). An example of a Bayesian optimization is represented in Figure 5.3. A difference between standard Bayesian optimization and the process in our approach consists of the values of the function, which in the latter case are (mostly) discrete. The typical approach

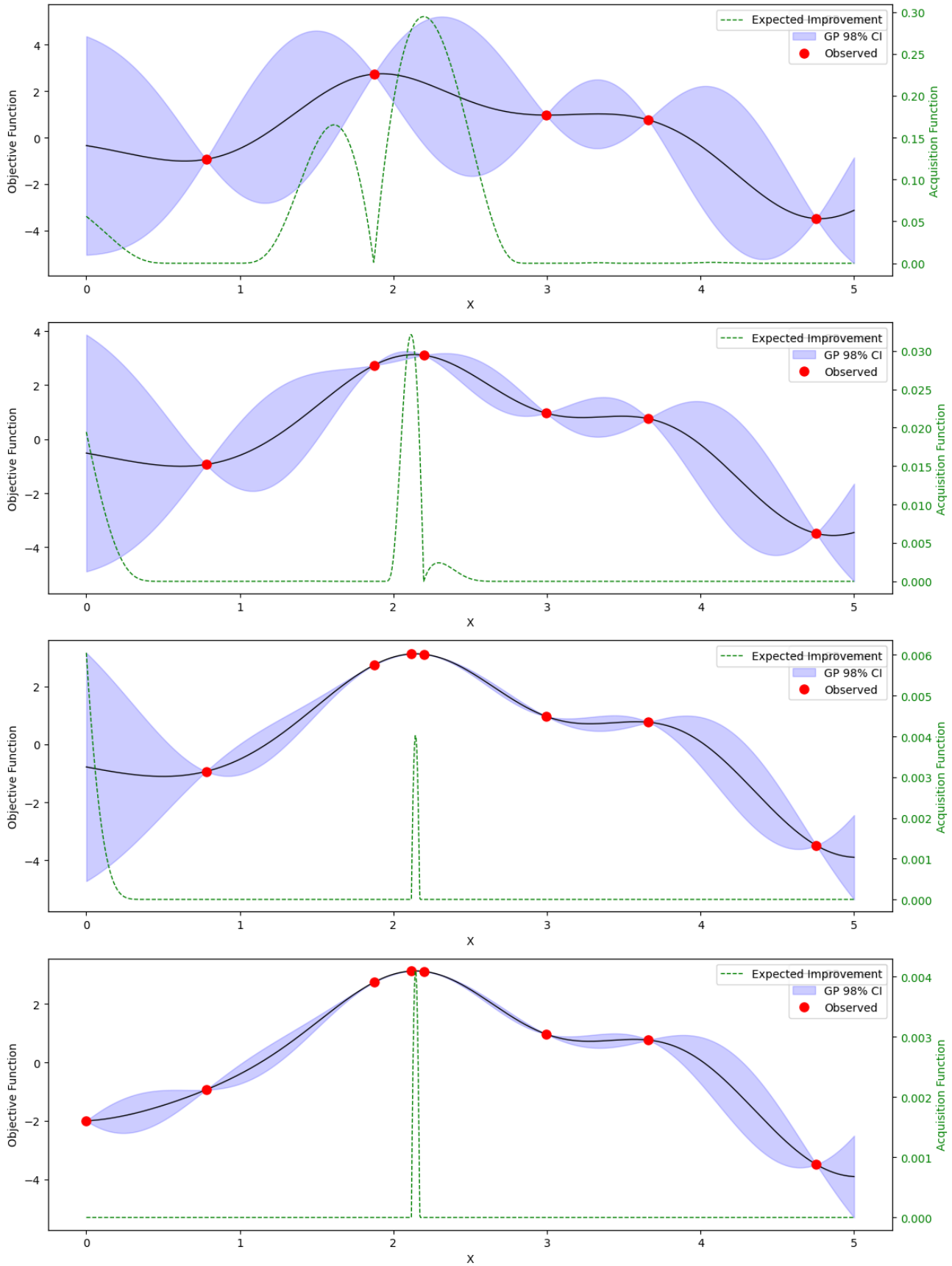


Figure 5.3: An example of Bayesian optimization process on a $1D$ problem. The figures show the approximation of the objective function over four iterations. The line in green represents the expected improvement in the maximum of the objective function, in function of the new x sampled.

to using Bayesian optimization for such functions is to apply the standard method after rounding the suggested continuous points to the nearest discrete points at the end.

Algorithm 4 Bayesian Optimization

Require: maxIterations
1: Initialize two starting points: x_1, x_2
2: Compute initial $\mathcal{D}_{1,2} = \{(x_1, f(x_1)), (x_2, f(x_2))\}$
3: **while** $t \leq$ maxIterations **do**
4: Find x_t by optimizing the acquisition function over the function space: $x_t = \arg\max_x u(x|D_{1,\dots,t-1})$.
5: Evaluate the function at x_t: $y_t = f(x_t)$.
6: Update the samples set $D_{1,\dots,t} = D_{1,\dots,t-1} \cup (x_t, y_t)$.
7: **end while**

5.3.3 Genetic Algorithm

Metaheuristic algorithms are a very active area of research aimed at finding solutions to intractable optimization problems. Notable examples of metaheuristic algorithms include genetic algorithms, simulated annealing, large neighborhood search, and many more [15]. Between these, population-based metaheuristics find good solutions by iteratively selecting and then combining existing solutions from a set, usually called the *population*. The most important members of this class are *evolutionary algorithms*, called after their similarity to the principles of natural evolution. We decided to implement for our problem a *Genetic Algorithm* (GA) [29]. In this algorithm, described in Algorithm 5, we start with an initial population of kernels P with size m, where each kernel represents a potential solution. The algorithm proceeds over a predefined number of generations N_{gen}. In each generation, the fitness of each kernel in the population is evaluated using the provided evaluator function.

The core of the GA lies in the creation of the offspring population $P_{\text{offspring}}$. During this step, for each new kernel in $P_{\text{offspring}}$, we either perform a crossover operation by selecting two parent kernels from P based on their fitness and combining them to create a new kernel or apply a mutation operation by selecting one parent kernel and introducing small changes to create a new kernel. After generating the offspring population, we select the top-performing kernels to form the next generation's population. This selection process is based on fitness, ensuring that the best-performing kernels have a higher chance of being included. The algorithm continues for a specified number of generations, and at the end of this process, the top kernels in the final population represent the optimized solutions for the given problem.

The success of the GA depends on the design of the *crossover* and *mutation* operations, which determine how new kernels are generated from parent kernels. Crossover can be defined as combining two parent kernels to create a new one. Depending on the problem and kernel representation, different crossover strategies can be employed. In our implementation, we decided first to perform a vectorization of the two considered kernels, and then a *one-point* crossover by choosing a certain element e in the vector, in order to create one offspring by copying the components before e from the first kernel and the components after e from the second one. The mutation operation is then performed after the crossover to introduce small random changes. Mutations can involve modifying gate types, qubit assignments, feature selections, or bandwidth parameters. The mutation rate p_{mutation} determines the probability of applying a mutation to each vector component.

Algorithm 5 Genetic Algorithm

Require: Parameters: m, n, G, *evaluator*
Require: Population: P of size m
Require: Maximum number of generations: N_{gen}
Require: Crossover rate: $p_{\text{crossover}}$
Require: Mutation rate: p_{mutation}
Initialize population: $P \leftarrow \text{RandomKernels}(m, n, G)$
for i in 1 to N_{gen} **do**
  Evaluate the fitness of each kernel in P using *evaluator*
  Create an empty offspring population: $P_{\text{offspring}} \leftarrow \text{EmptyPopulation}()$
  while Size of $P_{\text{offspring}}$ is less than m **do**
    if $\text{Random}() < p_{\text{crossover}}$ **then**
      Select two parent kernels p_1 and p_2 from P based on fitness
      Apply crossover operation to create a new kernel: $c \leftarrow \text{Crossover}(p_1, p_2)$
    else
      Select one parent kernel p from P based on fitness
      Apply mutation operation to create a new kernel: $c \leftarrow \text{Mutation}(p)$
    end if
    Add c to $P_{\text{offspring}}$
  end while
  Select the top m kernels from $P_{\text{offspring}}$ based on fitness
  Replace P with the selected kernels: $P \leftarrow \text{SelectTopKernels}(P_{\text{offspring}}, m)$
end for

In our setting, this optimizer did not perform well, mostly due to the implementation of the vectorization of the circuit that made the search space inefficient to explore. With a loss of generalization in the model, it would be possible to change the modalities of generation and crossover in the population by considering a restricted set of gates (e.g. $R_X(\beta x[i],)$, H, and $CNOT$). The crossover would be performed as a one-point selection on the list of operations, while the parameter β would be changed as a mutation.

Regarding the computational costs, is interesting to note that operations such as crossover and mutations are not very expensive to perform, therefore the optimization step has a computational cost dependent mostly on the size of the population m. Finally, we can observe that every step requires exactly m evaluations. Values such as m, number of iterations N_{gen}, and the mutation probability can be tuned according to the computational cost of the evaluations and the size of the model.

5.3.4 Reinforcement Learning Approaches

Reinforcement Learning (RL) offers a distinct approach to optimizing quantum kernels in the context of Automatic Kernel Discovery. Unlike traditional optimization methods, RL leverages a learning agent that interacts with its environment, making decisions (in this case, selecting gates, qubits, and parameters) to maximize a cumulative reward signal. In this context, RL can be used to develop a policy for selecting the parameters and components of quantum circuits that result in kernels with better performance. The usual components of a RL method are the following:

- *States (S):* States represent the current configuration of the quantum kernel under optimization. These states encompass various aspects of the quantum circuit, such as the gate sequence, qubit

assignments, feature mappings, and bandwidth parameters.

- *Actions (A):* Actions represent the choices that the RL agent can make at each state. These choices correspond to modifying components of the quantum circuit, such as selecting different gates, and qubits, or adjusting parameters like bandwidth.

- *Rewards (R):* The reward signal is a measure of the performance of the quantum kernel, given by a given evaluation function.

- *Policy (π):* The policy is a mapping of states to actions. It defines the strategy that the agent follows to maximize its cumulative reward. The policy is improved iteratively based on the experiences of the agent.

In our approach, we tested *SARSA* (State-Action-Reward-State-Action), an on-policy temporal difference learning algorithm that allows the agent to learn the value of state-action pairs and make decisions accordingly. The SARSA algorithm operates iteratively, alternating between exploration and exploitation. Within each episode, the agent selects an action based on its current policy, interacts with the environment (modifying the quantum kernel), receives a reward, and updates its policy accordingly. The objective is to learn the optimal policy that maximizes the cumulative expected reward over time. The application of SARSA to Automatic Kernel Discovery involves defining a state representation that captures relevant quantum circuit characteristics and designing an action space encompassing circuit modifications. This was performed in the same way as for the Genetic Algorithm, i.e. by considering a vectorization of the circuit.

Our implementation of SARSA did not perform very well, mostly due to the unfeasibility of selecting long-window horizons for the selection of actions. As for the GA, a way to improve the performance of this optimizer may consist of changing the representation of the problem, for example by reducing the action space. However, despite the low performance reached, we observed that an RL approach for automatic kernel discovery is applicable. Considering the fact that there is a wide range of RL algorithms, further explorations may show that the design of states and action spaces that align better with the specific requirements and characteristics of the problem can achieve a good performance.

5.4 Details and optimizations

Finally, in this section, we present some technical details and some improvements that we considered.

5.4.1 Reducing the search space

From the formulation presented in Section 5.1, is clear that the search space can have be very big even for circuits with a limited number of gates. More specifically, given n qubits, m features and supposing bandwidth can be selected from b different values, we obtain

$$|G| \times n \times (n-1) \times f \times b \tag{5.7}$$

possible gates, where G is the generators group and the factor $n \times (n-1)$ takes into account that 2-qubits generators act on 2 different qubits. The number of possible circuits with L gates is therefore

$$\big(|G| \times n \times (n-1) \times f \times b\big)^L . \tag{5.8}$$

If we consider that for each qubit we can choose a measure between I, X, Y, and Z, we obtain that the number of possible kernels is

$$\big(|G| \times n \times (n-1) \times f \times b\big)^L \times 4^n. \tag{5.9}$$

From this number is clear that reducing the search dimension is of fundamental importance, as every optimizer has a computational cost that depends on the size of the exploration space. To reduce the exploration space without reducing the expressive power of our model, we would need a way to recognize if some different circuit representations lead to the same circuit. If this is the case, we would be able to remove some states, reducing therefore the search space. As we saw in Chapter 2, different combinations of elementary gates can result in the same unitary matrix, therefore we expect that a consistent number of gates combination is redundant. Unfortunately, checking whether two circuits are equivalent is in general a very hard task. However, sometimes, we can easily identify that some combinations of gates are equivalent. For example, every time we have two adjacent gates g_1, g_2 that operate on different qubits, we know that they commute, so a circuit "starting" with g_1 and g_2 and the same circuit where the order of g_1 and g_2 is swapped lead to the same kernel, and therefore evaluating both of them is useless. In certain scenarios, we can instead prune the exploration space way more easily by removing some redundant generators. In particular, we noticed that applying a gate based on the generator PP' on the qubits n_1, n_2 is equivalent to applying the gate generated by $P'P$ on the qubits n_2, n_1. This allowed us to easily reduce the set of 2 qubits generators from

$$[\ \text{II}\ ,\ \text{IX}\ ,\ \text{IY}\ ,\ \text{IZ}\ ,\ \text{XI}\ ,\ \text{XX}\ ,\ \text{XY}\ ,\ \text{XZ}\ ,\ \text{YI}\ ,\ \text{YX}\ ,\ \text{YY}\ ,\ \text{YZ}\ ,\ \text{ZI}\ ,\ \text{ZX}\ ,\ \text{ZY}\ ,\ \text{ZZ}\ ] \tag{5.10}$$

to

$$[\ \text{II}\ ,\ \text{IX}\ ,\ \text{IY}\ ,\ \text{IZ}\ ,\ \text{XX}\ ,\ \text{XY}\ ,\ \text{XZ}\ ,\ \text{YY}\ ,\ \text{YZ}\ ,\ \text{ZZ}\ ], \tag{5.11}$$

effectively reducing the factor $|G|$ in (5.5) from 16 to 10. However, we decided to test a stricter set of generators

$$G_r = [\text{IX, IY, IZ, XY, XZ, YZ}]. \tag{5.12}$$

To test if it was possible to reduce the generators set from G to G_r we decided to perform an empirical evaluation of our algorithm on some artificially generated tasks, with different optimizers and with Centered Kernel Alignment as the evaluator and a small number of gates ($L = 4$). The result we found was that the resulting circuits with generators from G and G_r, despite having in some cases different representations, corresponded to the same kernel, allowing us to use G_r in the main part of the experiment.

Optimizer	2X	2XZ	3X	3XZ	4X	4XZ	5X	5XZ
Bayesian	0.9916	0.7840	0.9997	0.9960	0.9924	0.5797	0.4274	0.4316
Greedy_1	0.1327	0.7017	0.1957	0.9956	0.4585	0.3064	0.3950	0.3981
Greedy_2	0.1327	0.7841	0.9534	0.9956	0.4585	0.2997	0.4092	0.8057
Greedy_3	0.9968	0.7703	0.9998	0.8892	0.9936	0.9222	0.8526	0.9914
SARSA	0.1218	0.5746	0.0101	0.3615	0.1237	0.0646	0.0273	0.0993

Table 5.3: Values of TKA of different evaluators for artificial tasks. Each column corresponds to a dataset generated by a simulated quantum process with N qubits and different measurements, e.g. $3XZ$ represents a process obtained from a 3 qubits circuit with a measurement vector composed of X and Z. The Bayesian optimization is performed for 10 iterations. Greedy_1 uses the algorithm 2 without bandwidth selection, Greedy_2 uses the same algorithm with 3 possible bandwidths $(0.33, 0.66$, and $1)$, while Greedy_3 considers 3 bandwidth and the possibility to change measurements (algorithm 3). Finally, SARSA has the same 3 possible bandwidths used by Greedy_2 and Greedy_3.

5.4.2 Qiskit and Transpiling

As described in Chapter 2, a quantum device is not able to implement any possible gate, but only a finite (but universal) set of them. The number of elementary gates and their performance varies significantly between device and device. Before "submitting" a program on a quantum device, the circuit is usually optimized for the target device, in order to minimize errors or computation times, or the ensure that the connectivity between qubits is adequate for the circuit. This operation is called *transpilation*. Instead of using real quantum devices, in our project, we worked with quantum devices simulated by the Qiskit library, by using the *statevector simulator*. A classic computer is able to fully simulate a quantum device with an exponential slowdown, which even for a limited amount of qubits can require significant time. The Qiskit transpiler method is able to convert a generic quantum circuit into an equivalent one that is optimized for the classical statevector simulator in a way that is similar to the optimization performed for actual quantum devices. By transpiling every circuit before evaluating the kernel we were able to dramatically reduce the required time for each optimization cycle. Finally, considering that a quantum device is usually able to perform a set of elementary gates different from the ones in G, the transpiled equivalent of a circuit gives us a better understanding of its performance on a real quantum device, for example by considering the number or equivalent elementary operations, its depth, etc.

5.4.3 Gates repetition

In the execution of the automatic kernel discovery algorithm, the most important parameter in the space search space is the number of gates L. A low number of operations leads to circuits with a small entanglement, that therefore cannot overperform classical techniques. On the other hand, since the search space is exponential in L, some optimizers may be unfeasible even for $L = 20$ or 30. An easy way to guarantee circuits with a higher number of gates without impacting the performance of the optimization process is by obtaining a kernel by repeating its gates many times. Formally, given an embedding U with L gates $o_1, \ldots, o_L$ and a repetition parameter R, we can construct a new circuit RU with $L \times R$ operations by concatenating the set of gates R times. The main advantage of this implementation is that the exploration space is the same, and therefore we can perform our optimization process on U while evaluating RU. We tested this idea on an artificial dataset for a number of gates of

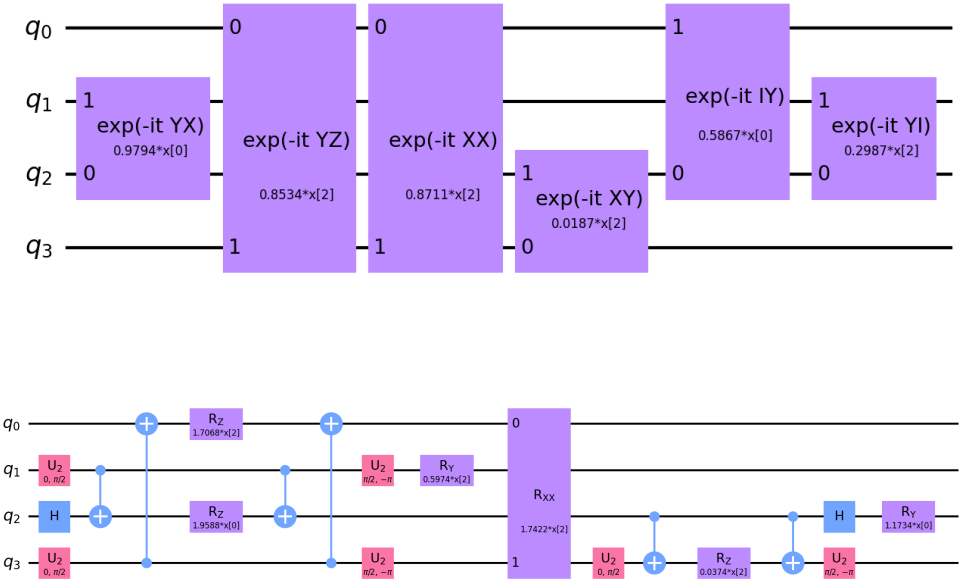


Figure 5.4: A circuit with 6 gates and the corresponding transpiled version. The transpiled circuit is composed of only $\exp(-iXX)$ gates, CNOT and single qubit gates.

4 and 12, and for $R = 1, 2$ and 4, with the Bayesian optimization and TKA as evaluator. The results obtained are shown in Table 5.5. In general, we observed that increasing R led sometimes to better performance, in particular for a higher number of qubits. However, the results were inconsistent, and we decided to not use the parameter R in the remaining part of the project.

	3	4	5	8	10
No repetition	0.8635	0.5611	0.4193	0.0091	0.1049
$R = 2$	0.6742	0.5599	0.2637	0.0058	0.0333
$R = 4$	0.6282	0.4578	0.6569	0.4121	0.2783

(a) Results for $L = 4$.

	3	4	5	8	10
No repetitions	0.8516	0.4934	0.6251	0.4718	0.4001
$R = 2$	0.7331	0.4322	0.7821	0.4003	0.6273
$R = 4$	0.6992	0.5393	0.6331	0.5054	0.7382

(b) Results for $L = 12$.

Figure 5.5: TKA obtained with the Bayesian optimization after 10 iterations. The experiments were performed on an artificially generated dataset with a different number of qubits and 4 features.

6

Quantum Anomaly Detection

Most of the datasets that show a quantum advantage in ML tasks are artificially generated, or with unknown practical applications, such as the one based on the Discrete Logarithm Problem (DLP) given in [47]. In this work, the authors constructed a family of datasets and showed that no classical learner can classify them better than a quantum approach. The key point is that the DLP, assumed to be hard from a classical point of view, is strictly related to the factorization problem, that can be efficiently solved by a quantum device. Another example of a dataset that is guaranteed to exhibit a quantum advantage is presented in [22]. The idea is that, given a scalar function f that is efficiently computable on a quantum device, but that requires exponential time on a classic computer, by generating the data as $Y = f(X) + \epsilon$, then the kernel defined by $k(x, x') = f(x)f(x')$ has a quantum exponential advantage. These examples, despite being interesting from a theoretical point of view, provide close to no insights into applications to real-world problems.

Datasets generated by real-world phenomena that present a quantum advantage are rare, but crucially important to study and understand the practical applicability of QML techniques. Between them, one remarkable example is given in [46]. In this work, the authors proposed several unsupervised QML algorithms for the anomaly detection of High Energy Physics (HEP) events. Between them, the Quantum SVM obtained a consistent performance advantage compared to the classical version. In our work, we decided to test our kernel discovery algorithm on the same dataset, to compare our results to theirs. In this way, we can evaluate the performance of our "discovered" kernels with the performance of a hand-crafted kernel that is known to outperform the classical counterpart. We will refer to this kernel as the "fixed" kernel. This chapter is structured as follows. In the next section, we describe in detail the problem and the dataset considered. In Section 2 we describe our experiments and methodology. Finally, we discuss our results in Section 3.

6.1 Proton collision events at the LHC

Being able to determine if an HEP event deviates significantly from a "standard" or "normal" behavior is of fundamental importance. One of the main goals of particle physics research concerns the discovery of new particles, new forces, new symmetries, and more in general, new phenomena. Our most complete theory about particle physics is called *Standard Model* (SM), and it describes our current understanding

of the laws that characterize particles' behavior. Anomaly detection applied to HEP events may help us to understand if a certain phenomenon belongs to the SM or if it is *Beyond Standard Model* (BSM). Doing so would allow us to have a better understanding of the fundamental constituents of our universe, leading to an enrichment of the theory.

The *Large Hadron Collider* (LHC) is the world's largest and highest-energy particle collider, known, among other results, for the discovery of the Higgs boson [44]. In LHC, proton beams are made to collide, after being accelerated to energy levels that are comparable to the ones of collisions that were typical in the early universe. In most proton collisions, quarks and gluons inside the protons interact to form a wide array of usually low-energy particles that decay to *jet pairs*. A *jet* is a spray of close-by particles that is produced after a *Quantum Chromodynamics* (QCD) multijet event, one of the most frequent processes occurring in LHC collisions.

6.1.1 The dataset

The dataset considered, available at [31], contains the simulations of three different BSM processes, in particular the narrow Randall-Sundrum gravitons decaying to W-bosons, broad Randall-Sundrum gravitons decaying to W-bosons, and scalar boson A decaying to a Higgs and a Z bosons [34]. These three different sets are the *signals* of the Anomaly Detection, i.e. the events that deviate from the majority of the data. On the other hand, the SM events (the so-called *background*) consist of simulated QCD multijets production. Both signals and background were simulated with the software `PYTHIA` [40], a program widely used for the generation of HEP events, and processed with the library `DELPHES` [13]. The details of the preprocessing are exhaustively discussed in [46]. After this first preprocessing, for each event are selected the 2 jets with the highest "energy", measured as a parameter called *transverse momentum*. Finally, each event is mapped to a latent, lower-dimension space by a convolutional autoencoder. The model maps every particle to a one-dimensional vector of length l, where $l = 4, 8, 16$ is the dimension of the latent space. Since each event consists of the jets of two particles, the final dimension of the dataset is $2l$. Is important to note that encoding the pre-processed data with a classical model may introduce biases that decrease the performance of a quantum model. However, such a kind of post-processing is fundamental, as current NISQ devices are not able to load and process datasets with a high dimension.

6.2 Approach and experimental setup

Our approach consists of implementing the kernel discovery algorithm described in the last chapter to the task of anomaly detection applied to the three BSM processes, with a latent dimension encoding of sizes $l = 4$ and $l = 8$. We decided to use the Bayesian Optimizer (BO) with a performance-based evaluation function. This decision was made according to our experiments on artificial data, where the BO performed better than the other optimizers implemented. Moreover, to evaluate a kernel the algorithm needed to train and test a new model, which has a serious computational cost. The choice of BO, considering that it requires a very low number of calls of the evaluation function, was therefore optimal.

The experiments were performed on a classical device with Qiskit [33]. Qiskit is an open-source development kit produced by IBM that can be used to create and test quantum circuits. In particular,

a quantum circuit with Qiskit can be tested on any real quantum device that follows the circuit model for quantum computing or can be tested on a classical device by simulating a quantum computer. For our experiments, we decided to use the so-called *statevector simulator*, an algorithm that calculates and returns the qubits statevector in an ideal, noiseless environment. The code was written in Python and is highly based on a QuASK (*Quantum Advantage Seeker with Kernels*) variant, a software presented in [14] that can be used to design and evaluate quantum kernels.

We performed different experiments for each task with a latent encoding $l = 4$ and $l = 8$. In our modeling, we selected a number of qubits n equal to l. We repeated the experiments with a different number of operations $L = 8, L = 12$ and with a different number of optimization steps $T = 5$, $T = 10$, and $T = 15$. The number of operations was decided according to the observation that a circuit with few gates corresponds to a kernel that is "too simple", and that therefore does not provide any kind of quantum advantage over a classical model. On the other hand, a circuit with too many gates is unfeasible for our current NISQ devices and would have required too much time for its simulation on our hardware.

In order to evaluate a quantum kernel K on the task t, we:

- build a One Class SVM based on K
- train the model on $n_{\text{train}} = 75$ samples from t
- return the accuracy of the model, computed on $n_{\text{test}} = 75$ samples.

The values of $n_{\text{train}} = 75$ and $n_{\text{test}} = 75$ were selected to achieve a balance between the performance and the time required by the evaluation.

After the optimization process, we trained a new model based on the final kernel on 200 samples, and we tested it on 1500 data points. To evaluate the final performance of the model we repeated the same process on a model built on the fixed kernel used in [46] and with the highest performance classical SVM that uses Radial Basis Function kernel. We chose the same number of training and testing samples to ensure a fair comparison between the models.

6.3 Results

In order to evaluate our technique, we consider the following points:

- final performance versus classical SVM and SVM with quantum kernels;
- depth of the discovered circuit.

6.3.1 Evaluating performance: ROC and AUC

To evaluate the models' performance, we computed the *Receiver Operating Characteristic* (ROC) curve and the corresponding *Area Under the Curve* (AUC). The ROC curve is a graphical representation of a classification model's performance across various threshold values. It illustrates the trade-off between true positive rate and false positive rate as the classification threshold is varied. In the context of

anomaly detection, the ROC curve helps us assess the model's ability to distinguish between normal and anomalous data points.

The AUC, on the other hand, quantifies the overall performance of the model by summarizing the ROC curve into a single value. Specifically, it represents the probability that the model will rank a randomly chosen anomalous data point higher than a randomly chosen normal data point. A higher AUC indicates better discrimination between normal and anomalous instances, with a perfect classifier having an AUC of 1.0.

The values for AUC to compare our results with the classical SVM and the fixed quantum kernel SVM are given in Tables 6.1a and 6.1b. The testing was performed with 1500 samples. A 5-fold testing is performed to assess the statistical significance of the results. A scheme with the ROA and the corresponding band of uncertainty for each experiment, compared to the classical values, is given in Figure 6.2.

Number of gates	Iterations	Narrow G	A → HZ	Broad G
8	5	97.57 ± 1.21	94.22 ± 1.44	48.81 ± 4.58
8	10	97.09 ± 1.21	94.22 ± 1.44	43.77 ± 3.82
8	15	97.57 ± 1.21	94.22 ± 1.44	47.52 ± 3.41
12	5	98.10 ± 1.45	92.39 ± 1.64	48.15 ± 4.78
12	10	98.10 ± 1.45	93.79 ± 1.50	53.56 ± 2.91
12	15	98.10 ± 1.45	93.79 ± 1.50	53.56 ± 2.91
Classic		$\mathbf{98.64} \pm 0.74$	$\mathbf{94.86 \pm 1.40}$	49.06 ± 4.29
Fixed		96.37 ± 0.63	89.72 ± 3.16	$\mathbf{58.72 \pm 4.28}$

(a) Results for latent dimension $l = 4$.

Number of gates	Iterations	Narrow G	A → HZ	Broad G
8	5	99.28 ± 0.58	97.66 ± 0.61	47.62 ± 4.17
8	10	99.28 ± 0.58	97.66 ± 0.61	$\mathbf{61.61 \pm 4.67}$
8	15	99.28 ± 0.58	91.24 ± 1.97	61.61 ± 4.67
12	5	99.37 ± 0.46	94.57 ± 1.60	58.70 ± 4.38
12	10	$\mathbf{99.73 \pm 0.20}$	$\mathbf{98.34 \pm 0.37}$	58.70 ± 4.38
12	15	99.73 ± 0.20	98.34 ± 0.37	40.04 ± 5.57
Classic		99.54 ± 0.42	97.94 ± 0.69	50.90 ± 3.97
Fixed		99.65 ± 0.23	98.05 ± 0.58	55.20 ± 3.96

(b) Results for latent dimension $l = 8$.

Figure 6.1: Summary of the values of AUC obtained in each experiment, compared with the values obtained by the fixed kernel and the classical SVM. Each test was performed on 1500 samples with a 5-fold testing. Each value is considered with one standard deviation of uncertainty.

A first, general observation is that by increasing the latent space dimension the performance increases. This is not surprising, as by reducing the dimensionality of a dataset there is a loss of information.

Let discuss now the results for for $l = 4$. Regarding the first two problems, *Narrow G* and $A \rightarrow HZ$, there is no quantum advantage observed. Our method was able to find quantum kernels that overperformed the fixed kernel, but not the classical model. For the third problem, *Broad G*, our method found a quantum kernel better than the classical one, but not better than the fixed one.

On the other hand, for $l = 8$, is possible to observe that the quantum models obtained a better performance than the classical one on every problem. In particular, the fixed kernel SVM obtained

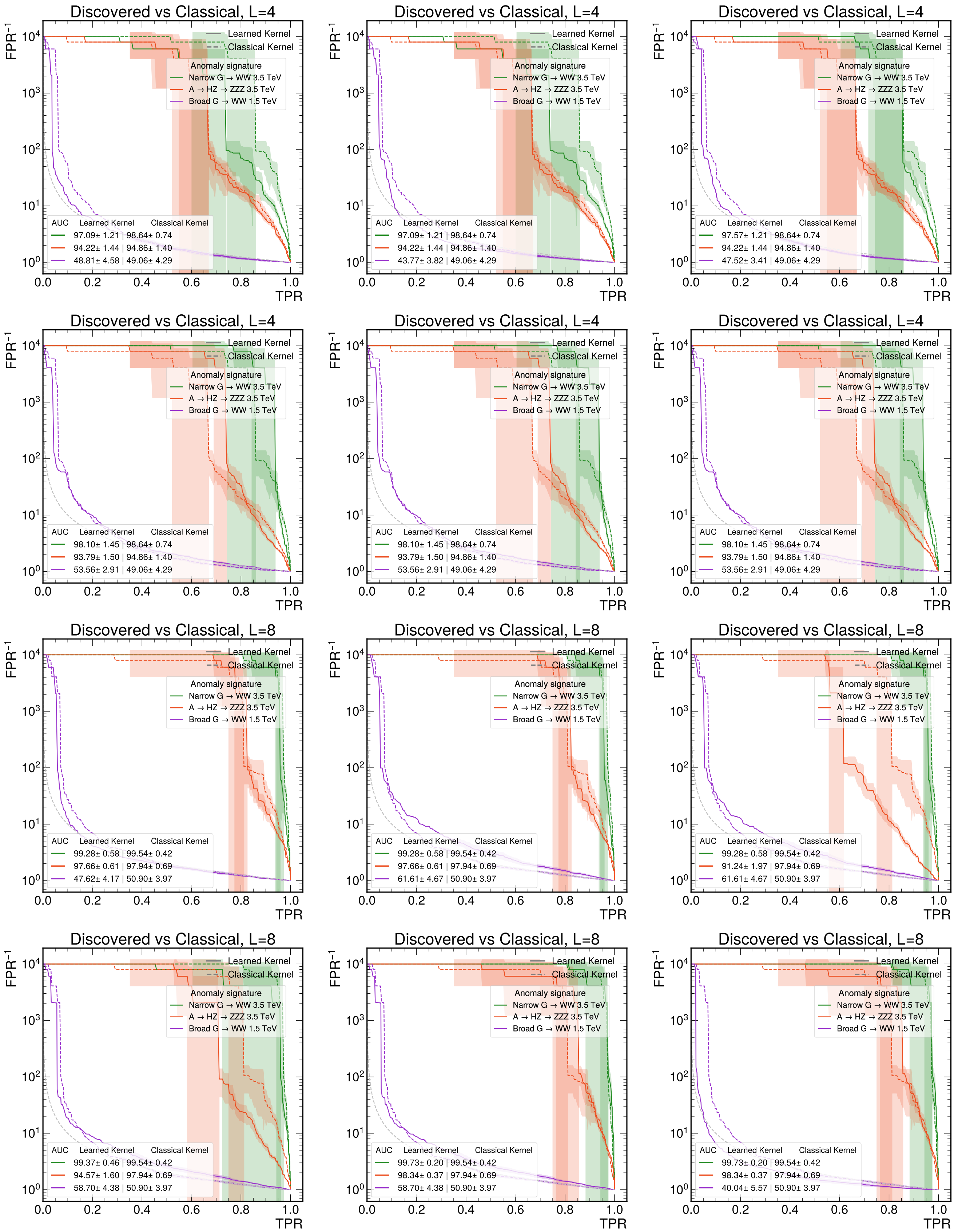
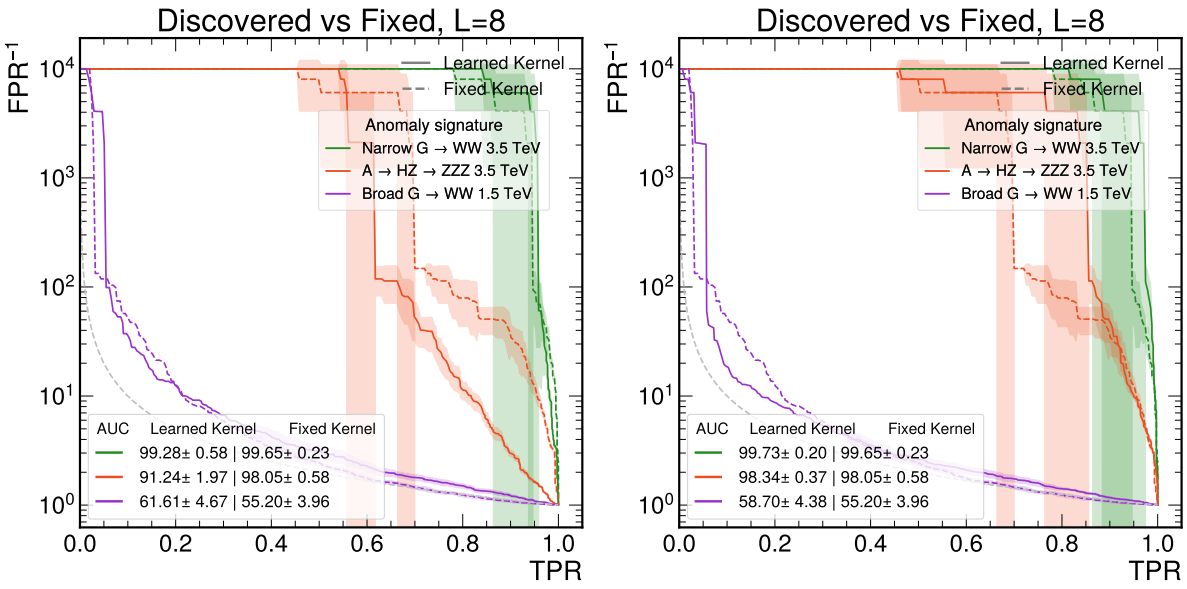


Figure 6.2: Plots of the ROA for latent dimensions $l = 4$ and $l = 8$, compared to the classical performance. Each test was performed on 1500 samples. The colored band represents the uncertainty, considered with one standard deviation. Images in odd rows represent the results of the discovered kernel with 8 gates, while figures in the even ones are for 12 gates.



(a) Setting with $L = 8$ and 15 optimization steps. (b) Setting with $L = 12$ and 10 optimization steps.

Figure 6.3: Comparison between two of the best-performing models and the corresponding fixed kernel ones, for latent dimension equal to 8. For this latent dimension, the fixed kernel one showed an AUC higher than the classical counterpart, but lower than the ones obtained with our method. In particular, (a) obtained the highest performance on the *Broad G* dataset, showing a high quantum advantage, while (b) obtained the highest AUC for the other two datasets.

better results on *Narrow G*, $A \to HZ$, and *Broad G*. Our method, according to the setting of the parameters L and the number of iterations, obtained a performance higher than the SVM with the fixed quantum kernel on all of them. A comparison between the results obtained with $L = 8$, $T = 15$ and $L = 12$, $T = 10$ and the fixed kernel results is shown in Figure 6.3.

Is important to note that the first two problems are "easy", as the final AUC is close to 1, and the small differences between the classical and quantum performances are not enough to talk about a real quantum advantage. However, for the *Broad G* dataset there is a consistent margin of improvement. For this reason, is interesting to see in detail the differences between the performance of this problem, presented in Figure 6.4.

Regarding latent dimension 4, our method was able to overperform the classical kernel only with 12 operations. Is interesting to note that the AUC obtained considering the kernels with 10 and 15 steps is the same. This is due to the fact that the algorithm gives in output the same kernel. A similar behavior can be observed for latent dimension 8, with 8 gates and 10 and 15 optimization steps. We suppose that the kernel is "optimal" for our evaluation method and according to the search space considered, or that the Bayesian Optimizer is not able to explore the portion of the search spaces where better kernels are located. For the higher latent dimension, we observe that our algorithm finds a kernel able to achieve a higher performance than the classical and fixed kernel ones for 8 gates with 10 and 15 steps, and for 12 gates with 5 and 10 steps. However, our algorithm with 12 gates and 15 steps has a significant drop in performance. This may happen due stochasticity of our evaluation process, as the evaluation step is performed with only 75 samples, both in training and testing. Another possible reason may be that the kernel discovery process "overfitted" on the samples set used for the evaluation. Despite this "anomaly", our quantum kernel discovery pipeline achieved a discrete success on the *Broad*

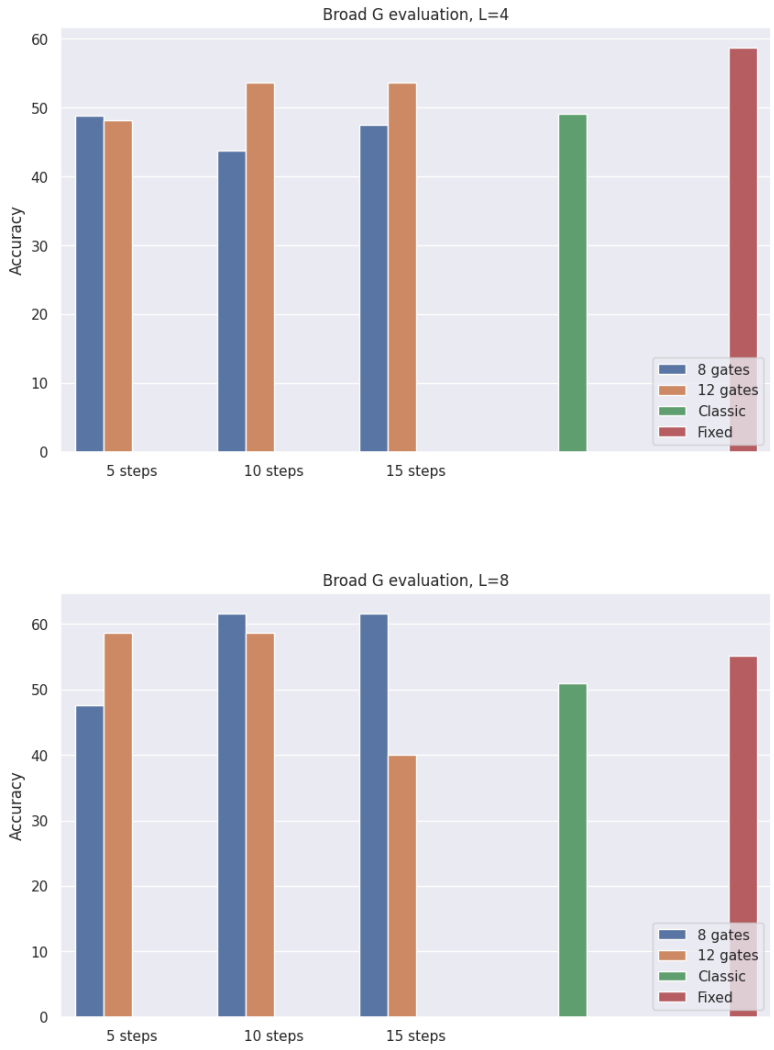


Figure 6.4: AUC plots on the *Broad G* dataset, for latent dimension $l = 4$ and $l = 8$. The values presented consider the discovered kernel with 8 and 12 gates, for $5, 10$, and 15 optimization steps, compared with the performed reached by the classical SVM and the SVM with the fixed quantum kernel.

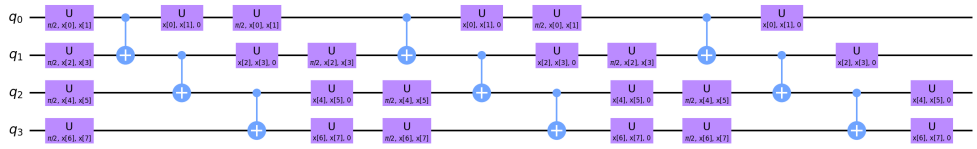
G dataset, outperforming both classical SVMs and the SVM with a "handcrafted" quantum kernel with a consistent margin. This consideration, in addition to the small advantages obtained in the other two datasets, shows that quantum models can achieve a better performance than classical SVMs (at least on simulated devices), and that is completely possible to find an appropriate quantum kernel on certain datasets with our agnostic, automated kernel discovery algorithm proposed in this work.

NISQ applicability as circuit *depth*

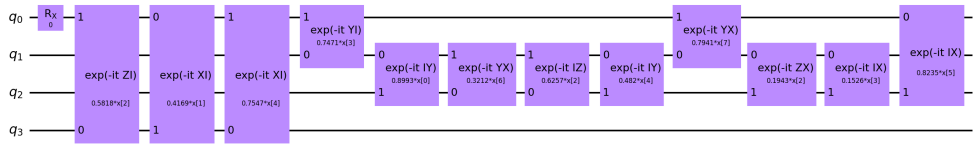
Among the main parameters affecting the usability of a circuit on a noisy device, there is the *depth* of the circuit, which can be defined as a measure of how many "layers" of quantum gates, executed in parallel, it takes to complete the computation defined by the circuit. Circuits with high depth require more execution time, which can be unfeasible for devices with a small decoherence time [32]. Intuitively, when the number of gates L is low, the depth of the circuit is low. In particular, if we consider the set of generators $G = P \cup P^{\otimes 2}$ as the set of elementary gates that can be applied by a quantum device, then the depth of the circuit is less or equal to the number of gates. However, different quantum devices are

able to implement different sets of elementary gates, usually "simpler" gates than the ones in G, and the depth obtained by the generators in G is not a valid metric to ensure that a circuit is suitable on a current device.

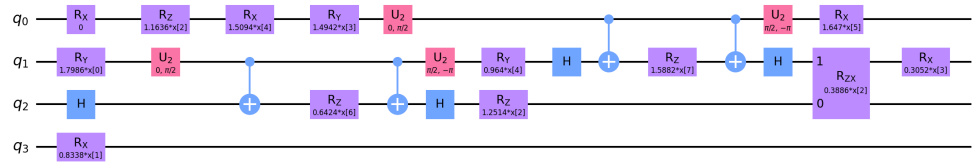
In order to have a better discussion on the real NISQ applicability of the kernels discovered by our algorithm, we decided to consider the depth and the number of operations of the transpiled equivalent circuit, described in Section 5.4.2. Given a circuit, the transpiled version is obtained with the Qiskit library by the command `transpile`. The result we observed is that the discovered transpiled kernels have usually a depth between 10 and 25, which is considered suitable for current devices. A comparison between the circuit used in [46], our best-performing discovered kernel for the *Broad* G dataset, and its transpiled equivalent is given in 6.5.



(a) Fixed kernel for a latent dimension $l = 4$



(b) Discovered kernel for $l = 4$, $L = 12$ and 15 iterations for the *Narrow* G dataset.



(c) Same kernel as in (b), transpiled.

Figure 6.5: A comparison in the structure of the fixed kernel for $l = 4$ with a kernel discovered and its transpiled circuit. After the transpile, the number of elementary gates used is 24, versus 33 used for the fixed kernel, while the depths are 14 and 13 respectively. From these parameters, both embeddings are suitable for NISQ devices.

7

Conclusion and further improvements

In this work, we have explored the state-of-the-art of quantum kernel methods, introducing an algorithm capable of autonomously discovering quantum kernels tailored to specific problem domains, without relying on prior domain knowledge. Our research followed a logical sequence of key steps.

We initiated our exploration by discussing the complexities and nuances inherent in quantum kernels. This foundational step was crucial to developing a deep understanding of the challenges associated with quantum machine learning methods. Next, we introduced our model, which makes use a diverse array of optimization techniques and evaluation criteria, to tackle domain-specific machine learning problems. Finally, we tested our algorithm on a specific quantum anomaly detection problem that exhibited clear potential for a quantum advantage. To do so, we conducted a rigorous comparative analysis. Our model's performance was evaluated against both classical machine learning methods and state-of-the-art quantum approaches.

Our research has yielded promising results, although our evaluations were conducted in a simulated quantum environment. Notably, we observed a significant quantum advantage in tackling the challenging task of detecting broad Randall-Sundrum gravitons decaying into W-bosons, demonstrating the potential of our automated kernel discovery pipeline.

While our accomplishments are significant, there remains ample room for improvement within our kernel discovery pipeline. First and foremost, we decided to explore on this dataset the Bayesian optimization with a performance-based evaluation. However, it is worth considering the exploration of alternative formulations as multi-objective optimization problems, coupled with different optimization techniques, which may yield superior results. Specifically, we did not explore the use of optimizers conducive to less resource-intensive evaluations, such as genetic algorithms and other metaheuristic approaches, which have the potential to minimize a composite cost function composed of various metrics.

Furthermore, our experiments were executed within the confines of a simulated quantum environment. To gain a more comprehensive understanding of the practical performance of our algorithm, it is fundamental to transition to real NISQ quantum devices, including those offered by prominent providers like IBM. This transition holds the promise of providing invaluable insights into the real-world feasibility and efficacy of our approach.

In conclusion, our research has shed light on the potential of quantum kernel methods and autonomous kernel discovery. As quantum computing technology continues to progress, the practical

applications of these techniques become increasingly promising. Our work is a small step in this direction, offering insights into the possibilities of quantum machine learning applications to real-world problems. It contributes to the ongoing exploration in these fields, leaving the door open for further discoveries and advancements.

Bibliography

[1] Scott Aaronson. How Much Structure Is Needed for Huge Quantum Speedups?, September 2022. arXiv:2209.06930 [quant-ph].

[2] Scott Aaronson and Daniel Gottesman. Improved Simulation of Stabilizer Circuits. *Physical Review A*, 70(5):052328, November 2004. arXiv:quant-ph/0406196.

[3] Ryan Babbush, Jarrod McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2(1):010103, March 2021. arXiv:2011.04149 [quant-ph].

[4] George S. Barron, Bryan T. Gard, Orien J. Altman, Nicholas J. Mayhall, Edwin Barnes, and Sophia E. Economou. Preserving Symmetries for Variational Quantum Eigensolvers in the Presence of Noise. *Physical Review Applied*, 16(3):034003, September 2021. arXiv:2003.00171 [quant-ph].

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

[6] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. arXiv:2104.13478 [cs, stat].

[7] Abdulkadir Canatar, Blake Bordelon, and Cengiz Pehlevan. Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nature Communications*, 12(1):2914, May 2021.

[8] Abdulkadir Canatar, Evan Peters, Cengiz Pehlevan, Stefan M. Wild, and Ruslan Shaydulin. Bandwidth Enables Generalization in Quantum Kernel Models, June 2023. arXiv:2206.06686 [quant-ph].

[9] Carlo Ciliberto, Mark Herbster, Alessandro Davide Ialongo, Massimiliano Pontil, Andrea Rocchetto, Simone Severini, and Leonard Wossnig. Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170551, January 2018.

[10] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for Learning Kernels Based on Centered Alignment, April 2014. arXiv:1203.0550 [cs].

[11] Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz Kandola. On Kernel-Target Alignment. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[12] Alexander M. Dalzell, Aram W. Harrow, Dax Enshan Koh, and Rolando L. La Placa. How many qubits are needed for quantum computational supremacy? *Quantum*, 4:264, May 2020. arXiv:1805.05224 [quant-ph].

[13] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. *Journal of High Energy Physics*, 2014(2):57, February 2014. arXiv:1307.6346 [hep-ex, physics:hep-ph].

[14] Francesco Di Marcantonio, Massimiliano Incudini, Davide Tezza, and Michele Grossi. QuASK – Quantum Advantage Seeker with Kernels, June 2022. arXiv:2206.15284 [quant-ph].

[15] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040, November 2019.

[16] Yuxuan Du, Zhuozhuo Tu, Xiao Yuan, and Dacheng Tao. Efficient measure for the expressivity of variational quantum algorithms. *Physical Review Letters*, 128(8):080506, February 2022. arXiv:2104.09961 [quant-ph].

[17] Bojia Duan, Jiabin Yuan, Chao-Hua Yu, Jianbang Huang, and Chang-Yu Hsieh. A survey on HHL algorithm: From theory to application in quantum machine learning. *Physics Letters A*, 384(24):126595, August 2020.

[18] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm, November 2014. arXiv:1411.4028 [quant-ph].

[19] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for solving linear systems of equations. *Physical Review Letters*, 103(15):150502, October 2009. arXiv:0811.3171 [quant-ph].

[20] Vojtech Havlicek, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019. arXiv:1804.11326 [quant-ph, stat].

[21] Zoë Holmes, Kunal Sharma, M. Cerezo, and Patrick J. Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1):010313, January 2022. arXiv:2101.02138 [quant-ph, stat].

[22] Jonas M. Kübler, Simon Buchholz, and Bernhard Schölkopf. The Inductive Bias of Quantum Kernels, November 2021. arXiv:2106.03747 [quant-ph, stat].

[23] Martin Larocca, Frederic Sauvage, Faris M. Sbahi, Guillaume Verdon, Patrick J. Coles, and M. Cerezo. Group-Invariant Quantum Machine Learning. *PRX Quantum*, 3(3):030341, September 2022. arXiv:2205.02261 [quant-ph, stat].

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

[25] Danylo Lykov, Jonathan Wurtz, Cody Poole, Mark Saffman, Tom Noel, and Yuri Alexeev. Sampling Frequency Thresholds for Quantum Advantage of Quantum Approximate Optimization Algorithm, July 2023. arXiv:2206.03579 [quant-ph].

[26] Riccardo Manenti and Mario Motta. *Quantum information science*. Oxford University Press, New York, 2023.

[27] Riccardo Mengoni and Alessandra Di Pierro. Kernel methods in Quantum Machine Learning. *Quantum Machine Intelligence*, 1(3-4):65–71, December 2019.

[28] Johannes Jakob Meyer, Marian Mularski, Elies Gil-Fuster, Antonio Anna Mele, Francesco Arzani, Alissa Wilms, and Jens Eisert. Exploiting symmetry in variational quantum machine learning. *PRX Quantum*, 4(1):010328, March 2023. arXiv:2205.06217 [quant-ph].

[29] Melanie Mitchell. *An introduction to genetic algorithms*. Complex adaptive systems. Cambridge, Mass., 7. print edition, 2001.

[30] Yoshifumi Nakata, Masato Koashi, and Mio Murao. Generating a state t -design by diagonal quantum circuits. *New Journal of Physics*, 16(5):053043, May 2014.

[31] M. Pierini and K. A. Wozniak. Dataset for Quantum anomaly detection in the latent space of proton collision events at the LHC, 2023.

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

[33] IBM Quantum. Qiskit. https://qiskit.org/.

[34] Lisa Randall and Raman Sundrum. Large Mass Hierarchy from a Small Extra Dimension. *Physical Review Letters*, 83(17):3370–3373, October 1999.

[35] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum Support Vector Machine for Big Data Classification. *Physical Review Letters*, 113(13):130503, September 2014.

[36] Daniel A. Roberts and Beni Yoshida. Chaos and complexity by design. *Journal of High Energy Physics*, 2017(4):121, April 2017. arXiv:1610.04903 [hep-th, physics:quant-ph].

[37] Maria Schuld and Nathan Killoran. Quantum machine learning in feature Hilbert spaces. *Physical Review Letters*, 122(4):040504, February 2019. arXiv:1803.07128 [quant-ph].

[38] Changpeng Shao. Reconsider HHL algorithm and its related quantum machine learning algorithms, March 2018. arXiv:1803.01486 [quant-ph].

[39] Sukin Sim, Peter D. Johnson, and Alan Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, December 2019. arXiv:1905.10876 [quant-ph].

[40] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An Introduction to PYTHIA 8.2. *Computer Physics Communications*, 191:159–177, June 2015. arXiv:1410.3012 [hep-ph].

[41] Rolando Somma, Howard Barnum, Gerardo Ortiz, and Emanuel Knill. Efficient Solvability of Hamiltonians and Limits on the Power of Some Quantum Computational Models. *Physical Review Letters*, 97(19):190501, November 2006.

[42] J. Sperling and W. Vogel. The Schmidt number as a universal entanglement measure, March 2011. arXiv:0908.3974 [quant-ph].

[43] Supanut Thanasilp, Samson Wang, M. Cerezo, and Zoë Holmes. Exponential concentration and untrainability in quantum kernel methods, August 2022. arXiv:2208.11060 [quant-ph, stat].

[44] The ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716(1):1–29, September 2012. arXiv:1207.7214 [hep-ex].

[45] Guifre Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):147902, October 2003. arXiv:quant-ph/0301063.

[46] Kinga Anna Woźniak, Vasilis Belis, Ema Puljak, Panagiotis Barkoutsos, Günther Dissertori, Michele Grossi, Maurizio Pierini, Florentin Reiter, Ivano Tavernelli, and Sofia Vallecorsa. Quantum anomaly detection in the latent space of proton collision events at the LHC, March 2023. arXiv:2301.10780 [hep-ex, physics:quant-ph].

[47] Sau Lan Wu, Shaojun Sun, Wen Guan, Chen Zhou, Jay Chan, Chi Lung Cheng, Tuan Pham, Yan Qian, Alex Zeng Wang, Rui Zhang, Miron Livny, Jennifer Glick, Panagiotis Kl Barkoutsos, Stefan Woerner, Ivano Tavernelli, Federico Carminati, Alberto Di Meglio, Andy C. Y. Li, Joseph Lykken, Panagiotis Spentzouris, Samuel Yen-Chi Chen, Shinjae Yoo, and Tzu-Chieh Wei. Application of Quantum Machine Learning using the Quantum Kernel Algorithm on High Energy Physics Analysis at the LHC. *Physical Review Research*, 3(3):033221, September 2021. arXiv:2104.05059 [hep-ex, physics:quant-ph].