# Computational speedups and learning separations in quantum machine learning

Gyurik, C.

# Chapter 2

# Background and definitions

In this chapter, we introduce the required background and definitions that will be used throughout this thesis. We begin with an explanation of the basics of quantum computing in Section 2.1. Following that, we delve into the basics of topological data analysis in Section 2.2, which will serve as the underpinning for Chapter 3. Next, Section 2.3 provides an in-depth look into structural risk minimization, a topic discussed in Chapter 4. Subsequently, in Section 2.4, we introduce the field of reinforcement learning, which will be central to our discussions in Chapter 5. Finally, in Section 2.5, we explore the basics of computational learning theory, which we further study in Chapter 6.

## 2.1 Quantum computing

In this section we will go over the basics of quantum computing. We will not cover all aspects of quantum computing, for more details we refer to [145, 69]. We will assume a basic understanding of linear algebra (for the required linear algebra see Appendix A of [69]). First, we will introduce the fundamental way quantum mechanics can be used to store, manipulate, and extract information. Next, in Section 2.1.1, we will discuss how these fundamental building blocks can be brought together to become quantum algorithms and we discuss two examples (i.e., quantum phase estimation and Hamiltonian simulation). Afterwards, in Section 2.1.2, we will give an introduction to complexity theory, and how quantum computing fits in that field. Finally, in Section 2.1.3, we will discuss how quantum computing can be used to evaluate certain families of linear classifiers (i.e., families of functions used in machine learning that separate classes of data by drawing hyperplanes between them).

#### From bits to qubits

In classical computing, the basic units of information are bits (i.e., $\{0, 1\}$). On the other hand, in quantum computing the basic units of information are *qubits*, which

are described by unit vectors $|\psi\rangle \in \mathbb{C}^2$, i.e.,

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle , \tag{2.1}$$

where $|\alpha_0|^2 + |\alpha_1|^2 = 1$, $|0\rangle = [1,0]^T$, and $|1\rangle = [0,1]^T$. The normalization constraint turns out to be important later on when we discuss how to extract classical information from qubits through measurements. Analogous to the classical case, we typically gather $n$ qubits into a single register. Following the postulates of quantum mechanics, an $n$-qubit register is described by a unit vector $|\psi\rangle \in \left(\mathbb{C}^2\right)^{\otimes n} \simeq \mathbb{C}^{2^n}$, i.e.,

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle , \tag{2.2}$$

where $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$, and $|i\rangle$ denotes the $i$th canonical basis vector (with zeroes everywhere except the $i$th entry). More generally, if an $n_1$-qubit register is in a state $|\psi\rangle \in \mathbb{C}^{2^{n_1}}$, and another $n_2$-qubit register is in a state $|\phi\rangle \in \mathbb{C}^{2^{n_2}}$, then their joint register is in the state $|\gamma\rangle$ given by

$$|\gamma\rangle = |\psi\rangle \otimes |\phi\rangle \in \mathbb{C}^{2^{n_1}} \otimes \mathbb{C}^{2^{n_2}}, \tag{2.3}$$

which is often referred to as the composition postulate.

**From classical to quantum circuits**

In classical computing, when performing computations the units of information (i.e., bits) are typically manipulated using Boolean circuits[1] that are build up from a basic set of logical gates. In fact, for any function manipulating bitstrings (i.e., so-called Boolean functions) one can build a Boolean circuit using just the logic gate set $\{\texttt{AND}, \texttt{OR}, \texttt{NOT}\}$ that implements the given Boolean function. In the quantum setting, Boolean functions are replaced by unitary transformations, Boolean circuits are replaced by quantum circuits, and logic gates are replaced by quantum gates. Specifically, instead of the logic gate set $\{\texttt{AND}, \texttt{OR}, \texttt{NOT}\}$ one typically consider the quantum gate set $\{X, S, T, H, \text{CNOT}\}$[2], which that are linear transformations described by the

[1] or Turing machines, but since they are in some sense equivalent to Boolean circuits, we will stick to the latter since it they are easier to translate to the quantum setting.

[2] There exists many different sets of quantum gates that are universal (two quantum gates can even be enough), though for our purposes we simply fix this to be our quantum gate set. Moreover, this set is "overcomplete" in the sense that the subset $\{\text{CNOT}, H, S, T\}$ is already universal, but we choose to introduce the $X$-gate anyways for future purposes.

matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{2.4}$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \tag{2.5}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \tag{2.6}$$

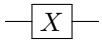
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \tag{2.7}$$

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.8}$$

Note that CNOT is a 2-qubit quantum gate, whereas the other quantum gates $X$, $Y$, $Z$ and $H$ act on a single qubit. Sometimes the quantum gate can have a dependence on some parameter $\theta \in \mathbb{R}$, in which case they are refered to as *paramaterized quantum gates*. For instance, one could consider the parameterized $X$-gate, which is described by the matrix

$$X(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \tag{2.9}$$

These quantum gates can be concatenated to build a quantum circuit. Here a quantum circuit is a collection of wires (each representing a qubit) and quantum gates (to be applied to the qubits) that one reads from left to right. If one of the quantum gates has a free parameter, then the resulting circuit is often called a *parameterized quantum circuit*. The single qubit gates are typically represented by boxes containing the letter corresponding to the gate respective gate, whereas the CNOT has a different notation. Specifically, the $X$-gate (and any other single qubit gate) is represented by




and the CNOT gate is represented by

Having fixed our notation for individual quantum gates, we can construct larger quantum circuit that specify how to manipulate an $n$-qubit register by concatenating individual quantum gates. For example, consider the 2-qubit quantum circuit in Figure 2.1 below.

We see that the circuit in Figure 2.1 first applies an $H$-gate to the first qubit, and afterwards we apply a CNOT to the output state. More precisely, the circuit in

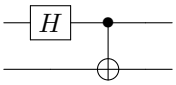



Figure 2.1: An example of a quantum circuit

Figure 2.1 corresponds to the unitary transformation

$$|\psi\rangle \mapsto U\,|\psi\rangle\,,$$

where $U$ is given by

$$U = \mathrm{CNOT} \cdot (H \otimes I).$$

Another important quantum circuit is that of the *quantum Fourier transform*. For the quantum Fourier transform the goal is to construct an $n$-qubit quantum circuit $U_n$ that implements the unitary transformation

$$|j\rangle \mapsto \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n}\,|k\rangle\,. \tag{2.10}$$

By exploiting a clever rewriting of Eq. (2.10), it turns out that for any given $n \in \mathbb{N}$ we can construct a quantum circuit $U_n$ consisting of $\mathcal{O}(\mathrm{poly}(n))$ quantum gates that implements the map in Eq. (2.10). The quantum Fourier transform is a pivotal building block for many important quantum algorithms (such as quantum phase estimation discussed in Section 2.1.1). For more details on the quantum Fourier transform see [69, 145].

### Measurements

Having discussed what the basic units of information are in quantum computing, and how to manipulate them, all that remains is how to extract classical information afterwards (i.e., extracting the output) through *measurements*. There are many different ways in which one can measure qubits to extract classical information, but we will focus on only two forms of measurements. First, we discuss what happens when we *measure in the computational basis*. Suppose we are given some $n$-qubit quantum state $|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle$, then if we measure this state in the computational basis, we obtain the outcome $i$ with probability $|\alpha_i|^2$. Next, we discuss what happens when one measures an *observable*, which is strictly more general than measuring in the computational basis. An $n$-qubit observable is an operator $O \in \mathbb{C}^{2^n \times 2^n}$ such that $O^\dagger = O$ (i.e., it is *Hermitian*). The possible outcomes when measuring $O$ are its eigenvalues $\{\lambda_j\}_{j=0}^{2^n-1}$. When we measure the observable $O$ on an $n$-qubit state $|\psi\rangle$ we obtain outcome $\lambda_j$ with probability

$$|\,\langle \phi_j \mid \psi\rangle\,|^2, \tag{2.11}$$

where $|\phi_j\rangle$ denotes an eigenvector of $O$ with eigenvalue $\lambda_j$, and $\langle v| := |v\rangle^\dagger$. Also, the expectation value of $O$ on an $n$-qubit state $|\psi\rangle$ is given by

$$\langle\psi| O |\psi\rangle . \tag{2.12}$$

**Mixed states**

Previously we have restricted ourselves to *pure states* (i.e., unit vectors), where we are certain about the state our qubit register is in. However, there can sometimes be uncertainty regarding the state of a qubit register in a classical sense. For example, one could have a register that is in some state $|\phi\rangle$ with probability $p$, and it is in some other state $|\psi\rangle$ with probabilty $1-p$. To model this, we consider *mixed states*, which are probability distributions (i.e., "mixtures") over pure states. Note that being in a superposition between basis states is sometimes misrepresented as a case of "uncertainty", even though we know for certain the precise superposition the quantum state is in. It is convenient to write down mixed states using *density matrices*, which are positive semidefinite operators (i..e, all eigenvalues are nonnegative) with trace 1. In particular, suppose we know that our qubit register is in one of the states $|\psi_1\rangle, \ldots, |\psi_r\rangle$ each with a respective probability $p_j$, then we write down this mixed states as a density matrix

$$\rho = \sum_{j=1}^{r} p_j |\psi_j\rangle \langle\psi_j| . \tag{2.13}$$

Following the conventions for pure states, we find that an $n$-qubit mixed state $\rho$ is manipulated by an $n$-qubit quantum circuit $U$ as follows

$$\rho \mapsto U\rho U^\dagger. \tag{2.14}$$

Moreover, when measuring an observable $O$ with orthonormal eigenvectors $\{|\psi_j\rangle\}_{j=0}^{2^n-1}$ and corresponding eigenvalues $\{\lambda_j\}_{j=0}^{2^n-1}$, the probability $p_j$ of outcome $\lambda_j$ is given by

$$p_j = \mathrm{Tr}\left[|\psi_j\rangle \langle\psi_j| \rho\right] \tag{2.15}$$

and the expectation value $\langle O\rangle$ (i.e., the probabilistic expected value of the measurement outcome) is given by

$$\langle O\rangle = \mathrm{Tr}\left[O\rho\right] . \tag{2.16}$$

### 2.1.1 Basics of quantum algorithms

Having discussed the basics of quantum computing, we are ready to define what a *quantum algorithm* is. In short, a quantum algorithm is a routine that gets a specification of an instance of a problem and efficiently (i.e., in time polynomial in the instance size) constructs a quantum circuit and measurement. This quantum circuit is then applied to the state $|0^n\rangle$, after which the measurement is preformed, and based on the outcomes of the measurement the quantum algorithm decides its

output. In the remainder of this section we will discuss two concrete examples of quantum algorithms: quantum phase estimation (QPE) and Hamiltonian simulation (HS).

**Quantum phase estimation**

Quantum phase estimation is an important building block for the algorithms discussed in Chapter 3. The input to the problem is an $n$-qubit unitary $U$ and an eigenvector $|\psi\rangle$ of $U$. The required output is the *eigenphase* of $|\psi\rangle$ with respect to $U$, i.e., $\phi \in [0, 1)$ such that

$$U |\psi\rangle = e^{2\pi i \phi} |\psi\rangle .$$

As originally proposed by Kitaev [120] and put in a broader context by Cleve et al. [64], given access to a black box capable of controlled-$U^{2^j}$ operations for $j = 1, \ldots, n$, we can construct a quantum circuit that uses $\mathcal{O}(\text{poly}(n))$ single-qubit gates together with $\mathcal{O}(\text{poly}(1/\delta, 1/\epsilon))$ calls to the black box such that with probability at least $1-\epsilon$ measuring the output state will produce an estimate $\tilde{\phi}$ such that $|\phi - \tilde{\phi}| < \delta$. Note that the use of black boxes suggests that quantum phase estimation by itself is not a complete algorithm in its own right. Rather, one should think of it as a kind of subroutine that, together with a realization of the unitary $U$ (capable of implementing the required black box) and a suitable quantum state $|\psi\rangle$, can perform interesting computational tasks.

To get some intuition how quantum phase estimation works, consider the case where the eigenphase can be expressed as a $n$-bit string $j$ (i.e., $\phi = \sum_{j=1}^{n} j_i 2^{-i}$). Then, quantum phase estimation works as follows

1. Start with $|0^n\rangle |\psi\rangle$.
2. Apply the layer of gates $(H^{\otimes n} \otimes I)$.
3. Use access to $U$ to apply the map $|j\rangle |\psi\rangle \mapsto |j\rangle U^j |\psi\rangle = e^{2\pi i \phi j} |j\rangle |\psi\rangle$.
4. Apply the inverse quantum Fourier transform to the first $n$ qubits.
5. Measure the first $n$ qubits.

After Step 3., the first $n$ qubits are in the state $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} e^{2\pi i \phi j} |j\rangle$ which is the same state obtained by applying the quantum Fourier transform to the state $|\phi\rangle = |j_1, \ldots, j_n\rangle$. Since the quantum Fourier transform is self-inverse, we find that after Step 4. the first $n$ qubits must be in the state $|\phi\rangle$ which when measured in the computational basis will return $\phi$. For more details on quantum phase estimation (e.g., what happens in the case when the eigenphase cannot be expressed as a bitstring) we refer to [69, 145]. For our purposes, we highlight that quantum phase estimation can, with probability at least $1 - \epsilon$, obtain an estimate of the eigenphase up to additive error $\delta$ using $\mathcal{O}(\text{poly}(n))$ single-qubit gates and $\mathcal{O}(\text{poly}(1/\delta, 1/\epsilon))$ calls to the black box controlled-$U^{2^j}$ operations.

**Hamiltonian Simulation**

Hamiltonian simulation is another important building block for the algorithms discussed in Chapter 3. Here the input is a specification of a sparse Hermitian operator $H$ together with some time $t \in \mathbb{R}_{>0}$, and the goal is to construct a circuit $U$ that implements the unitary transformation $e^{iHt}$. We call a $2^n \times 2^n$ matrix *sparse* if at most $\mathcal{O}(\text{poly}(n))$ entries in each row are nonzero. A special class of sparse positive semidefinite matrices that we consider is the class of *log-local* Hamiltonians, i.e., $n$-qubit Hermitian operators that can be written as a sum

$$H = \sum_{j=1}^{m} H_j, \tag{2.17}$$

where each $H_j$ acts on at most $\mathcal{O}(\log n)$ qubits and $m \in \mathcal{O}(\text{poly}(n))$. We specify the the input matrix $H$ using either of the following two standard cases. First, the input matrix can be specified in terms of *sparse access*. That is, the input matrix $H \in \mathbb{C}^{2^n \times 2^n}$ is specified by quantum circuits that let us query the values of its entries, and the locations of the nonzero entries. More precisely, we assume that we are given classical descriptions of $\mathcal{O}(\text{poly}(n))$-sized quantum circuits that implement the oracles $O_H$ and $O_{H,\text{loc}}$, which map

$$O_H : |i,j\rangle\,|0\rangle \mapsto |i,j\rangle\,|H_{i,j}\rangle\,,$$
$$O_{H,\text{loc}} : |j,\ell\rangle\,|0\rangle \mapsto |j,\ell\rangle\,|\nu(j,\ell)\rangle\,,$$

where $0 \leq i,j,\ell \leq 2^n - 1$, and $\nu(j,\ell) \in \{0,\ldots,2^n-1\}$ denotes the location of the $\ell$-th nonzero entry of the $j$-th column of $H$. Secondly, for log-local Hamiltonians, we also consider specifying the input matrix $H$ by its *local-terms* $\{H_j\}$ as in Eq. (2.17). Note that any specification in terms of local-terms can be turned into a specification in terms of a sparse access oracle.

Depending on the type of access you have to the input matrix $H$, different quantum algorithms exist to turn this specification of $H$ into a quantum circuit $U$ that implements the unitary transformation $e^{iHt}$. If the access is in terms of the local-terms in Eq. (2.17) (with respect to a family of local terms $H_j$ that allow for Hamiltonian simulation in an efficient way, such as the family of Pauli-strings), then one can use Trotterization [129] to construct a quantum circuit $U$ consisting of $\mathcal{O}(\text{poly}(n, 1/\epsilon))$ many two- and single-qubit quantum gates that satisfies

$$||e^{iHt} - U||_2 < \epsilon.$$

On the other hand, if the access is in terms of sparse access oracles then more advanced methods such as [133] are required. These methods use sparse access to $H$ to produce a quantum circuit $U$ of size $\mathcal{O}(\text{poly}(n, \log(1/\epsilon)))$ such that[3]

$$||U - e^{iHt}||_2 < \epsilon.$$

[3]The runtime of certain more advanced methods also depend on the largest entry of $H$ and its sparsity, but we choose to omit these dependencies from the runtime since they are not relevant for the purposes of Chapter 3

### 2.1.2 Quantum complexity theory

In the previous section we discussed how classical and quantum computers perform computations in a fundamentally different way. In this section we will discuss some of the consequences that this fundamentally different way of computing has on the field of *complexity theory*. In short, complexity theory studies how much of a certain resource is required to solve a given problem as a function of the instance size. We will stick to an intuitive explanation of some classes of problems as categorized by complexity theory, and for the formal definitions we refer to [23]. The problems that one typically studies in complexity theory are so-called *decision problems*. In such problems, one is given some $x \in \{0,1\}^n$ and one is asked to decide whether the input encoded by $x$ satisfies a certain property. For instance, we could use $x \in \{0,1\}^n$ to encode a Boolean formula, and ask to decide whether $x$ has any satisfying assignment or not.

In complexity theory, the class $\mathsf{P}$ denotes all problems that are solvable by a classical circuit that is allowed a number of gates that is at most polynomial in the input length (with the caveat that there must exist an efficient procedure that generates these circuits for a given input size). Additionally, the class $\mathsf{BPP}$ denotes all problems that are solvable by a classical circuit that is allowed a number of gates that is at most polynomial in the input length and is additionally allowed to use some randomness (e.g., by providing it with a random number generator). One of the questions studied in complexity theory is what additional problems one could solve by allowing this additional internal randomness, if any such problems exist. This question can be reformulated as asking whether or not $\mathsf{BPP} = \mathsf{P}$, which sparked the study of *derandomization* techniques [57].

With the addition of quantum computing one can define a whole new family of complexity classes. For instance, the class $\mathsf{BQP}$ denotes all problems that are solvable by a quantum circuit that is allowed a number of quantum gates that is at most polynomial in the input length. Another important question in complexity theory is whether there exists problems that can be solved with polynomial-size quantum circuits, but which cannot be solved with polynomial-size (randomized) classical circuits. This question can be reformulated as asking whether or not $\mathsf{BPP} = \mathsf{BQP}$. Since quantum circuits can simulate any classical circuit (with a negligible increase in circuit size), it holds that $\mathsf{BPP} \subseteq \mathsf{BQP}$. Even though it is not formally proven that $\mathsf{BPP} \subsetneq \mathsf{BPP}$ (i.e., that $\mathsf{BQP}$ is strictly larger than $\mathsf{BPP}$), it is it is still widely-believed to be the case. For example, the problem of factoring lies in $\mathsf{BQP}$ due to the famous Shor's algorithm [176], but it is widely-believed to be outside of $\mathsf{BPP}$.

If we allow ourselves to operate under the assumption that $\mathsf{BQP} \neq \mathsf{BPP}$, then how could one use complexity theory to argue that a certain problem is likely to exhibit a (superpolynomial) quantum advantage? Intuitively, what we mean by a "quantum advantage" is that the classical resources required to solve this problem drastically outscale the quantum resources required to solve the problem. Under the assumption that $\mathsf{BQP} \neq \mathsf{BPP}$, one way to show that a problem exhibits a quantum advantage is to show that it is among the hardest problems in $\mathsf{BQP}$. More precisely, we would like to show that if we could solve this problem using certain resources, then we could solve any problem in $\mathsf{BQP}$ using comparable resources. In complexity theory, the above is

captured using the notion of *reductions*. In the broadest sense, we say that a problem $A$ is reducible to a problem $B$ if an algorithm for solving $B$ with a certain set of resources (if it were to exist) can be turned into an algorithm for solving $A$ that uses a comparable amount of resources (e.g., both algorithms use polynomial-size circuits). Now if any problem in a complexity class is reducible to a problem $A$, then we say that $A$ is *hard* for that class. If additionally $A$ is also contained in that complexity class, we say that $A$ is *complete* for that class.

If we were to show that a given problem $A$ is BQP-complete[4], then this is solid evidence that $A$ exhibits a quantum advantage under the assumption that $\mathsf{BPP} \neq \mathsf{BQP}$. In particular, suppose we could solve $A$ using a polynomial-size randomized classical circuit (i.e., it is in BPP). Since $A$ is among the hardest problems in BQP, through the reductions we find that any other problem is BQP can also be solved using the resources of BPP. This clearly contradicts the assumption that $\mathsf{BPP} \neq \mathsf{BQP}$, which establishes by contradiction that $A$ is not in BPP. Moreover, since $A$ was in fact complete for BQP, we know that is in BQP and that it is thus solvable using a polynomial-depth quantum circuit. Using a completeness-result to give evidence for quantum advantage is not limited to the class BQP. In fact, any class that contains problems in BQP that are widely-believed to be outside BPP would work for this purpose. In Section 3.2.1, we discuss another example of a complexity class whose completeness result provide evidence for quantum advantage called DQC1, and we use it to argue that certain problems in topological data analysis (or machine learning more general) are likely to exhibit a quantum advantage.

To end this section, we introduce two more complexity classes. In particular, we introduce two classes of problems whose solutions are *verifiable* using a given amount of resources. For instance, if our input $x \in \{0,1\}^n$ encodes a Boolean formula, then how much resources does it require to check whether a given assignment makes the formula evaluate to 1? In the classical setting, the class NP denotes all problems whose solutions are verifiable in time polynomial in the input length on a classical computer. Note that deciding whether a Boolean formula has a satisfying assignment is in NP, since the evaluation of the Boolean formula for a given assignment can be done using in polynomial-time on a classical computer. In the quantum setting, the class QMA denotes the set of problems whose solutions are verifiable in time polynomial in the input length on a quantum computer. It is a long-standing open question in complexity theory whether $\mathsf{NP} = \mathsf{P}$, i.e., whether problems whose solution can be *verified* in polynomial-time on a classical computer can also be *solved* using the same resources. However, it is widely-believed that $\mathsf{P} \subsetneq \mathsf{NP}$, though there is no proof (yet). Similarly, it is widely-believed that $\mathsf{BQP} \subsetneq \mathsf{QMA}$, though also here there is no proof (yet).

[4]Strictly speaking, when we say that problem is BQP-complete, we actually mean that it is complete for the class PromiseBQP (the class BQP is not known to have any complete problems). The difference between BQP and PromiseBQP is that in the latter one only needs to be correct on a subset of instances (i.e., those that satisfy a certain "promise").

### 2.1.3 Quantum linear classifiers

A fundamental family of classifiers (i.e., binary-valued functions) used throughout machine learning are those constructed from *linear functions*. Specifically, they are constructed from the family of real-valued functions on $\mathbb{R}^\ell$

$$\mathcal{F}_{\text{lin}} = \Big\{ f_w(x) = \langle w, x \rangle \ : \ w \in \mathbb{R}^\ell \Big\}, \tag{2.18}$$

where $\langle ., . \rangle$ denotes an inner product on the input space $\mathbb{R}^\ell$. These linear functions are turned into classifiers by adding an offset and taking the sign, i.e., the classifiers are given by

$$\mathcal{C}_{\text{lin}} = \Big\{ c_{w,d}(x) = \text{sign}\big(\langle w, x \rangle - d\big) \ : \ w \in \mathbb{R}^\ell, \ d \in \mathbb{R} \Big\}. \tag{2.19}$$

While this family of classifiers is relatively limited (e.g., it cannot solve the well-known `XOR` problem), it becomes powerful when introducing a *feature map*. Specifically, a feature map $\Phi : \mathbb{R}^\ell \to \mathbb{R}^N$ is used to (non-linearly) map the data to a (much) higher-dimensional space – called the *feature space* – in order to make the data more linearly-separable. We let $\mathcal{C}\left(\Phi\right) = \{c \circ \Phi \mid c \in \mathcal{C}\}$ denote the family of classifiers on $\mathbb{R}^\ell$ obtained by combining a family of linear classifiers $\mathcal{C} \subseteq \mathcal{C}_{\text{lin}}$ on $\mathbb{R}^N$ with a feature map $\Phi$. If the feature map is clear from the context we will omit it in the notation and just write $\mathcal{C}$. A well known example of a model based on linear classifiers is the *support vector machine* (SVM), which aims to finds the hyperplane that attains the maximal perpendicular distance to the two classes of points in the two distinct half-spaces (assuming the feature map makes the data linearly separable, though one could relax this using so-called soft-margin SVMs [66] in the non-separable case).

The linear-algebraic nature of linear classifiers makes them particularly well-suited for quantum treatment. In the influential works of Havlíček et al. [103], and Schuld & Killoran [168], the authors propose a model where the space of $n$-qubit Hermitian operators – denoted $\text{Herm}\left(\mathbb{C}^{2^n}\right)$ – takes the role of the feature space. Note that $\text{Herm}\left(\mathbb{C}^{2^n}\right)$ is a $4^n$-dimensional real vector space equipped with the Frobenius inner product $\langle A, B \rangle = \text{Tr}\big[A^\dagger B\big]$. Their feature map maps classical inputs $x$ to $n$-qubit density matrices $\Phi(x) := \rho_\Phi(x)$ (i.e., positive semi-definite matrices of trace one). Finally, the hyperplanes that separates the states $\rho_{\Phi(x)}$ corresponding to the different classes are given by $n$-qubit observables. In short, the family of functions their model uses is given by

$$\mathcal{F}_{\text{qlin}} = \Big\{ f_{\mathcal{O}}(x) = \text{Tr}\left[\mathcal{O}\rho_\Phi(x)\right] \ : \ \mathcal{O} \in \text{Herm}\big(\mathbb{C}^{2^n}\big) \Big\}, \tag{2.20}$$

and the family of classifiers – which we refer to as *quantum linear classifiers* – is given by

$$\mathcal{C}_{\text{qlin}} = \Big\{ c_{\mathcal{O},d}(x) = \text{sign}\big(\text{Tr}\left[\mathcal{O}\rho_\Phi(x)\right] - d\big) \ : \ \mathcal{O} \in \text{Herm}\big(\mathbb{C}^{2^n}\big), \ d \in \mathbb{R} \Big\}. \tag{2.21}$$

We can estimate $f_{\mathcal{O}}(x)$ defined in Equation (2.20) by preparing the state $\rho_{\Phi(x)}$ and

measuring the observable $\mathcal{O}$. In particular, approximating $f_{\mathcal{O}}(x)$ up to additive error $\epsilon$ requires only $\mathcal{O}\left(1/\epsilon^2\right)$ samples. While the error creates a fuzzy region around the decision boundary, this turns out to not cause major problems in practical settings [31].

Using parameterized quantum circuits both the preparation of a quantum state that encodes the classical input and the measurement of observables can be done efficiently for certain feature maps and families of observables. We now briefly recap two ways in which parameterized quantum circuits can be used to efficiently implement a family of quantum linear classifiers, as originally proposed by Havlíček et al. [103], and Schuld & Killoran [168]. Both ways use a parameterized quantum circuit to implement the feature map. Specifically, let $U_\Phi$ be a parameterized quantum circuit, then we can use it to implement the feature map given by

$$\Phi : x \mapsto \rho_\Phi(x) := |\Phi(x)\rangle \langle \Phi(x)| , \tag{2.22}$$

where $|\Phi(x)\rangle := U_\Phi(x) |0\rangle^{\otimes n}$. The key difference between the two approaches is which observables they are able to implement (i.e., which separating hyperplanes they can represent) and how the observables are actually measured (i.e., how the functions $f_{\mathcal{O}}$ are evaluated). An overview of how the two approaches implement quantum linear classifiers can be found in Figure 2.2, and we discuss the main ideas behind the two approaches below.

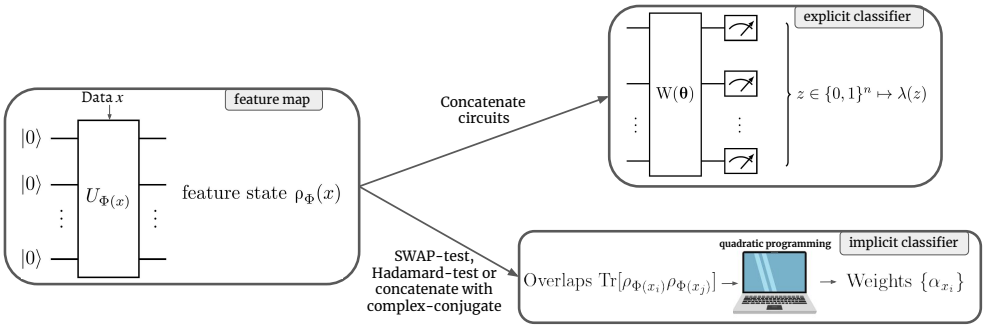



Figure 2.2: An overview of the explicit and implicit quantum linear classifiers defined in Equations (2.24) and (2.25), respectively (adapted from [95]). Note that in the case of the explicit classifier, a universal circuit $W(\theta)$ (specifying the eigenbasis) followed by a computational basis measurement and universal postprocessing $\lambda$ (specifying the eigenvalues) allows one to measure any observable (albeit not efficiently with respect to the number of qubits).

**Explicit quantum linear classifier**[5] The observables measured in this approach are implemented by first applying a parameterized quantum circuit $W(\theta)$, followed by a computational basis measurement and postprocessing of the measurement outcome

[5] Also called the *quantum variational classifier* [103].

$\lambda : [2^n] \to \mathbb{R}$. Upon closer investigation, one can derive that the corresponding observable is given by

$$\mathcal{O}_\theta^\lambda = W^\dagger(\theta) \cdot \mathrm{diag}\Big(\lambda(0), \lambda(1), \ldots, \lambda(2^n - 1)\Big) \cdot W(\theta). \tag{2.23}$$

Examples of efficiently computable postprocessing functions $\lambda$ include functions with a polynomially small support (implemented using a lookup table), functions that are efficiently computable from the input bitstring (e.g., the parity of the bitstring, which is equivalent to measuring $Z^{\otimes n}$), or parameterized functions such as neural networks. Note that the postprocessing function $\lambda$ plays an important role in how the measurement of the observable in Eq. (2.23) is physically realized. Altogether, this efficiently implements the family of linear classifiers – which we refer to as *explicit quantum linear classifiers* – given by

$$\begin{aligned}\mathcal{C}_{\mathrm{qlin}}^{\mathrm{explicit}} = \Big\{ & c_{\mathcal{O}_\theta^\lambda, d}(x) = \mathrm{sign}\big(\mathrm{Tr}\left[\rho_\Phi(x)\mathcal{O}_\theta^\lambda\right] - d\big) \\ & \qquad : \ \mathcal{O}_\theta^\lambda \text{ as in Equation (2.23)}, \ d \in \mathbb{R} \Big\}.\end{aligned} \tag{2.24}$$

The power of this model lies in the efficient parameterization of the manifold (inside the $4^n$-dimensional vector space of Hermitian operators on $\mathbb{C}^{2^n}$) realized by the quantum feature map together with the parameterized separating hyperplanes that can be attained by $W(\theta)$ and $\lambda$. Here also lies a restriction of the explicit quantum linear classifier compared to standard linear classifiers, as in the latter all hyperplanes are possible and in the former only the hyperplanes that lie in the manifold parameterized by $W(\theta)$ and $\lambda$ are possible. Furthermore, explicit quantum linear classifiers can likely not be efficiently evaluated classically, as computing expectation values $\mathrm{Tr}\left[\rho_\Phi(x)\mathcal{O}_\theta^\lambda\right]$ is classically intractable for sufficiently complex feature maps and observables [188, 41].

**Implicit quantum linear classifier**[6] Another way to implement a linear classifier is by using the so-called *kernel trick* [164]. In short, this trick involves expressing the normal vector of the separating hyperplane, – i.e., the observable $\mathcal{O}$ in the case of quantum linear classifiers – on a set of training examples $\mathcal{D}$ as a linear combination of feature vectors, resulting in the expression

$$\mathcal{O}_\alpha = \sum_{x' \in \mathcal{D}} \alpha_{x'} \rho_{\Phi(x')} = \sum_{x' \in \mathcal{D}} \alpha_{x'} \left|\Phi(x')\right\rangle \left\langle\Phi(x')\right|.$$

Using this expression we can rewrite the corresponding quantum linear classifier as

$$c_{\mathcal{O}_\alpha, d}(x) = \mathrm{sign}\big(\mathrm{Tr}\left[\rho_\Phi(x)\mathcal{O}_\alpha\right] - d\big) = \mathrm{sign}\big(\sum_{x' \in \mathcal{D}} \alpha_{x'} \mathrm{Tr}\left[\rho_\Phi(x)\rho_\Phi(x')\right] - d\big).$$

These type of linear classifiers can also be efficiently realized using parameterized quantum circuits. Using quantum protocols such as the SWAP-test or the Hadamard-

[6] Also called the *quantum kernel estimator* [103].

test, or by using the inverse of the circuit that implements the feature map $U_\Phi(x)^{-1}$, it is possible to efficiently evaluate the overlaps $\mathrm{Tr}[\rho_\Phi(x)\rho_\Phi(x')]$ for the feature map defined in Equation (2.22). Afterwards, the optimal parameters $\{\alpha_{x'}\}_{x'\in\mathcal{D}}$ are obtained on a classical computer, e.g., by solving a quadratic program. Altogether, this allows us to efficiently implement the family of linear classifiers – which we refer to as *implicit quantum linear classifiers* – given by

$$\mathcal{C}_{\mathrm{qlin}}^{\mathrm{implicit}} = \Big\{ c_{\mathcal{O}_\alpha,d}(x) = \mathrm{sign}\big(\mathrm{Tr}\left[\rho_\Phi(x)\mathcal{O}_\alpha\right] - d\big) \\ : \ \mathcal{O}_\alpha = \sum_{x'\in\mathcal{D}} \alpha_{x'}\rho_\Phi(x'),\ \alpha\in\mathbb{R}^{|\mathcal{D}|},\ d\in\mathbb{R}\Big\}. \tag{2.25}$$

The power of this model comes from the fact that evaluating the overlaps $\mathrm{Tr}\left[\rho_\Phi(x)\rho_\Phi(x')\right]$ is likely classically intractable for sufficiently complex feature maps [103], demonstrating that classical computers can likely neither train nor evaluate this quantum linear classifier efficiently. Moreover, from the well-known *Representer theorem* we know any quantum linear classifier that is the minimizer of a loss functions that includes *regularization* of the Frobenius norm of the observable can be expressed as an implicit quantum linear classifier [165]. However, as we indicate later in Section 4.3, this does not mean that we can forego explicit quantum linear classifiers entirely, as in the explicit approach there are unique types of meaningful regularization for which there is no straightforwards correspondence to the implicit approach.

## 2.2 Topological data analysis

Topological data analysis is a recent approach to data analysis that extracts robust features from a dataset by inferring properties of the *shape* of the data. This is perhaps best explained in analogy to a better-known method: much like how principal component analysis extracts features (i.e., the singular values characterizing the spread of the data in the directions of highest variance) that are invariant under translation and rotation of the data, topological data analysis goes a step further and extract features that are also invariant under bending and stretching of the data (i.e., by inferring properties of its general shape). Because of this invariance of the extracted features, topological data analysis techniques can be inherently more robust to noise in the data.

The theory behind topological data analysis is fairly extensive, but most of it we will not need for our purpose. Namely, we can set most of the topology aside and tackle the issue in linear-algebraic terms, which are well-suited for quantum approaches. In this section we introduce the relevant linear-algebraic concepts, and we briefly review the quantum algorithm for topological data analysis of Lloyd, Garnerone and Zanardi (LGZ) [130].

### 2.2.1 Betti numbers as features

In topological data analysis the dataset is typically a point cloud (i.e., a collection of points in some ambient space) and the aim is to extract the shape of the underlying

data (i.e., the 'source' of these points). This is done by constructing a connected object – called a *simplicial complex* – composed of points, lines, triangles and their higher-dimensional counterparts, whose shape one can study. After constructing the simplicial complex, features of the shape of the data – in particular, the number of connected components, holes, voids and higher-dimensional counterparts – can be extracted using linear-algebraic computations based on *homology*. An overview of this procedure can be found in Figure 2.4.

Consider a dataset of points $\{x_i\}_{i=1}^n$ embedded in some space equipped with a distance function $d$ (typically $\mathbb{R}^m$ equipped with the Euclidean distance). The construction of the simplicial complex from this point cloud proceeds as follows. First, one constructs a graph by connecting datapoints that are "close" to each other. This is done by choosing a *grouping scale* $\epsilon$ (defining which points are considered "close") and connecting all datapoints that are within $\epsilon$ distance from each other. This yields the graph $G = ([n], E_\epsilon)$, with vertices $[n] := \{1, \ldots, n\}$ and edges

$$E_\epsilon = \{(i, j) \mid d(x_i, x_j) \leq \epsilon\}.$$

After having constructed this graph, one relates to it a particular kind of simplicial complex called a *clique complex*, by associating its cliques (i.e., complete subgraphs) with the building blocks of a simplicial complex[1]. That is, a 2-clique is considered a line, a 3-clique a triangle, a 4-clique a tetrahedron, and $(k+1)$-cliques the $k$-dimensional counterparts[2].

To fix the notation, let $\mathrm{Cl}_k(G) \subset \{0,1\}^n$ denote the set of $(k+1)$-cliques in $G$ – where we encode a subset $\{i_1, \ldots, i_k\} \subset [n]$ as an $n$-bit string where the indices $i_k$ specify the positions of the ones in the bitstring – and let $\chi_k := |\mathrm{Cl}_k(G)|$ denote the number of these cliques. Throughout this thesis, we will discuss everything in terms of clique complexes, as this is sufficient for our purposes and allows us to use the more familiar terminology of graph theory.

The constructed clique complex exhibits the features that we want to extract from our dataset – i.e., the number of $k$-dimensional holes. For example, in Figure 2.3 we see a clique complex where we can count three 1-dimensional holes. Interestingly, counting these holes can be done more elegantly using linear algebra by employing constructions from homology.

To extract these features using linear algebra, embed the clique complex into a Hilbert space $\mathcal{H}_k^G$, by raising the set of bitstrings that specify $(k+1)$-cliques to labels of orthonormal basis vectors. Let $\mathcal{H}_k$ denote the Hilbert space spanned by computational basis states with Hamming weight[3] $k+1$. Due to the way we encode cliques as bitstrings, we have that $\mathcal{H}_k^G$ is a subspace of $\mathcal{H}_k$. Moreover, each $\mathcal{H}_k$ is an $\binom{n}{k+1}$-dimensional subspace of the entire $n$-qubit Hilbert space $\mathbb{C}^{2^n}$, and $\mathbb{C}^{2^n} \simeq \bigoplus_{k=-1}^{n-1} \mathcal{H}_k$.

The next step towards extracting features using linear algebra involves studying properties of the *boundary maps* $\partial_k : \mathcal{H}_k \to \mathcal{H}_{k-1}$, which are defined by linearly

[1] The resulting simplicial complex coincides with the Vietoris-Rips complex common in topological data analysis literature [83].

[2] The shift in the indexing is due to different terminologies in graph theory and topology (e.g., in graph theory a triangle is called a 3-clique, whereas in topology it is called a 2-simplex).

[3] The Hamming weight of a bitstring is the number of 1s in it.

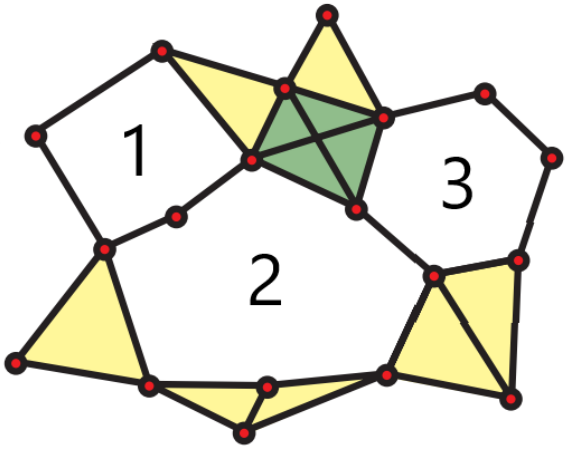



Figure 2.3: Example of a clique complex with three 1-dimensional holes (adapted from [83]). The number of these holes is equal to the first *Betti number*.

extending the action on the basis states given by

$$\partial_k |j\rangle := \sum_{i=0}^{k} (-1)^i \, |\widehat{j(i)}\rangle \,, \tag{2.26}$$

where $\widehat{j(i)}$ denotes the $n$-bit string of Hamming weight $k$ that encodes the subset obtained by removing the $i$-th element from the subset encoded by $j$ (i.e., we set the $i$-th one in the bitstring $j$ to zero). By considering the restriction of $\partial_k$ to $\mathcal{H}_k^G$ – which we denote by $\partial_k^G$ – these boundary maps can encode the connectivities of the graph $G$, in which case their image and kernel encode various properties of the corresponding clique complex. Intuitively, these boundary maps map a $(k+1)$-clique to a superposition (i.e., a linear combination) of all $k$-cliques that it contains, as seen in Eq. (2.26).

These boundary maps allow one to extract features of the shape of a clique complex by studying their images and kernels, and in particular their quotients. Specifically, the quotient space

$$H_k(G) := \ker \partial_k^G / \mathrm{Im} \partial_{k+1}^G, \tag{2.27}$$

which is called the $k$-th *homology group*, captures features of the shape of the underlying clique complex. The main feature is the $k$-th *Betti number* $\beta_k^G$, which is defined as the dimension of the $k$-th homology group, i.e.,

$$\beta_k^G := \dim H_k(G).$$

By construction, the $k$-th Betti number is equal to the number of $k$-dimensional holes in the clique complex.

The main problem in topological data analysis that we study in this thesis is the computation of Betti numbers. To do so, we study the *combinatorial Laplacians* [76], which are defined as

$$\Delta_k^G = \left(\partial_k^G\right)^\dagger \partial_k^G + \partial_{k+1}^G \left(\partial_{k+1}^G\right)^\dagger . \tag{2.28}$$

These combinatorial Laplacians can be viewed as generalized (or rather, higher-order) graph Laplacians in that they encode the connectivity between cliques in the graph as opposed to encoding the connectivity between individual vertices. We study the combinatorial Laplacians because the discrete version of the Hodge theorem [76] tells us that

$$\dim \ker \left(\Delta_k^G\right) = \beta_k^G, \tag{2.29}$$

which is often used as a more convenient way to compute Betti numbers [81], particularly in the case of the quantum algorithm that we discuss in the next section.

In conclusion, if the clique complex is constructed from a point cloud according to the construction discussed above, then computing these Betti numbers can be viewed as a method to extract features of the shape of the data (specifically, the number of holes are present at scale $\epsilon$). By recording Betti numbers across varying scales $\epsilon$ in a so-called *barcode* [83], one can discern which holes are "real" and which are "noise", resulting in feature extraction that is robust to noise in the data. Even though barcodes are very important in topological data analysis, we will mostly focus on the problem of estimating the number of holes at a given scale.

### 2.2.2 Quantum algorithm for Betti number estimation

The algorithm for Betti number estimation of Lloyd, Garnerone and Zanardi (LGZ) [130] utilizes Hamiltonian simulation and phase estimation to estimate the dimension of the kernel (i.e., the *nullity*) of the combinatorial Laplacian (which by Eq. (2.29) is equal to the corresponding Betti number). After the initial algorithm of Lloyd et al. several different improvements were made, focusing either on reducing the number of $T$-gates [33], making it more amenable to NISQ requirements [189], or on exploiting the quantum singular value transform and achieving an exponential saving in the number of qubits [135]. To make our presentation self-contained, we review the original quantum algorithm for Betti number estimation of Lloyd et al.

Estimating the nullity of a sparse Hermitian matrix can be achieved using some of the most fundamental quantum-algorithmic primitives. Namely, using Hamiltonian simulation and quantum phase estimation one can estimate the eigenvalues of the Hermitian matrix, given that the eigenvector register starts out in an eigenstate. Moreover, if instead the eigenvector register starts out in the maximally mixed state (which can be thought of as a random choice of an eigenstate), then measurements of the eigenvalue register produce approximations of eigenvalues, sampled uniformly at random from the set of all eigenvalues. This routine is then repeated to estimate the nullity by simply computing the frequency of zero eigenvalues (recall that the

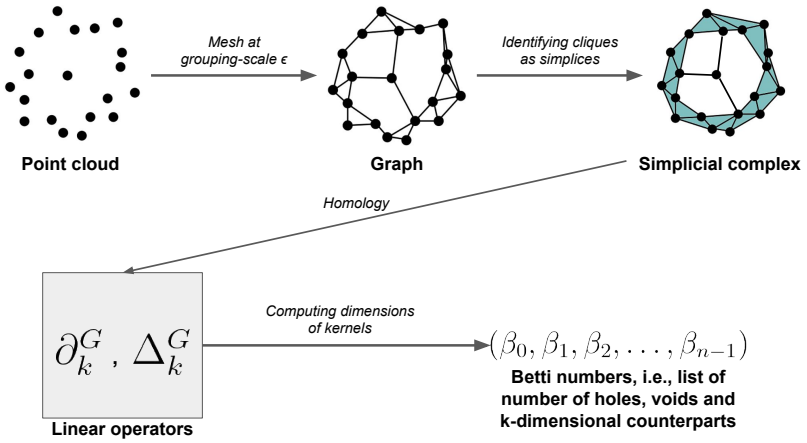



Figure 2.4: The pipeline of topological data analysis (adapted from [90]). First, points within $\epsilon$ distance are connected to create a graph. Afterwards, cliques in this graph are identified with simplices to create a simplicial complex. Next, homology is used to construct linear operators that encode the topology. Finally, the dimensions of the kernels of these operators are computed to obtain the Betti numbers (which correspond to the number of holes).

dimension of the kernel is equal to the multiplicity of the zero eigenvalue). Note that this procedure does not strictly speaking estimate the nullity, but rather the number of small eigenvalues, where the threshold is determined by the precision of the quantum phase estimation (see Section 2.2.2 for more details). The steps of the quantum algorithm for Betti number estimation of LGZ are summarized in Figure 2.5.

In Step 1(a), Grover's algorithm is used to prepare the uniform superposition over $\mathcal{H}_k^G$, from which one can prepare the state $\rho_k^G$ by applying a CNOT gate to each qubit of the uniform superposition into some ancilla qubits and tracing those out. When given access to the adjacency matrix of $G$, one can check in $\mathcal{O}\left(k^2\right)$ operations whether a bitstring $j \in \{0,1\}^n$ encodes a valid $k$-clique and mark them accordingly in the application of Grover's algorithm. By cleverly encoding Hamming weight $k$ strings we can avoid searching over all $n$-bit strings, which requires $\mathcal{O}\left(nk\right)$ additional gates per round of Grover's algorithm plus an additional one-time cost of $\mathcal{O}\left(n^2k\right)$ [93]. Hence, the runtime of this first step is

$$\mathcal{O}\left(n^2k + nk^3\sqrt{\binom{n}{k+1}/\chi_k}\right)$$

where $\chi_k$ denotes the number of $(k+1)$-cliques. This runtime is polynomial in the

Quantum algorithm for Betti number estimation

1. For $i = 1, \ldots, M$ repeat:
   (a) Prepare the state:
   $$\rho_k^G = \frac{1}{|\dim \mathcal{H}_k^G|} \sum_{j \in \mathrm{Cl}_k(G)} |j\rangle \langle j| . \tag{2.30}$$
   (b) Apply quantum phase estimation to the unitary $e^{i\Delta_k^G}$, with the eigenvector register starting out in the state $\rho_k^G$.
   (c) Measure the eigenvalue register to obtain an approximation $\widetilde{\lambda}_i$.
2. Output the frequency of zero eigenvalues:
   $$\left|\{i \mid \widetilde{\lambda}_i = 0\}\right| / M.$$

Figure 2.5: Overview of the quantum algorithm of Lloyd, Garnerone and Zanardi (LGZ) [130]

number of vertices $n$ when

$$\binom{n}{k+1} / \chi_k \in \mathcal{O}\left(\mathrm{poly}(n)\right). \tag{2.31}$$

Throughout this thesis we say that a graph is *clique-dense* if it satisfies Eq. (2.31). Note that $\rho_k^G$ can of course also be directly prepared without the use of Grover's algorithm by using rejection sampling: choose a subset uniformly at random and accept it if it encodes a valid clique. This is quadratically less efficient, however it has advantages if one has near-term implementations in mind, as it is a completely classical subroutine. As we will discuss in more detail in Section 2.2.2, this state preparation procedure via Grover's algorithm or uniform clique-sampling is a crucial bottleneck in the quantum algorithm.

In Step 1(b), standard methods for Hamiltonian simulation of sparse Hermitian matrices are used together with quantum phase estimation to produce approximations of the eigenvalues of the simulated matrix. In the original algorithm, the matrix that LGZ simulates (i.e., the matrix it applies Hamiltonian simulation on) in this step is

the *Dirac operator*, which is defined as

$$B_G = \begin{pmatrix} 0 & \partial_1^G & 0 & \dots & \dots & 0 \\ \left(\partial_1^G\right)^\dagger & 0 & \partial_2^G & \dots & \dots & 0 \\ 0 & \left(\partial_2^G\right)^\dagger & 0 & \ddots & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 & \partial_{n-1}^G \\ 0 & 0 & 0 & \dots & \left(\partial_{n-1}^G\right)^\dagger & 0 \end{pmatrix}$$

and satisfies

$$B_G^2 = \begin{pmatrix} \Delta_0^G & 0 & \dots & 0 \\ 0 & \Delta_2^G & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \Delta_{n-1}^G \end{pmatrix}. \tag{2.32}$$

From Eq. (2.32) we gather that the probability of obtaining an approximation of an eigenvalue that is equal to zero is proportional to the nullity of the combinatorial Laplacian. Because $B_G$ is an $n$-sparse Hermitian matrix with entries $0, -1$ and $1$, to which we can implement sparse access using $\mathcal{O}(n)$ gates, we can implement $e^{iB}$ using $\widetilde{\mathcal{O}}\left(n^2\right)$ gates [133] (here $\widetilde{\mathcal{O}}$ suppresses logarithmically growing factors).

We remark that it is also possible to simulate $\Delta_k^G$ directly (as depicted in Figure 2.5). Namely, as $\Delta_k^G$ is an $n^2$-sparse Hermitian matrix whose entries are bounded above by $n$, to which we can implement sparse access using $\mathcal{O}\left(n^4\right)$ gates (e.g., see Theorem 3.3.4 [86]), we can implement $e^{i\Delta_k^G}$ using $\widetilde{\mathcal{O}}\left(n^6\right)$ gates [133].

The disadvantage to directly simulating $\Delta_k^G$ is that it requires more gates. However, the advantage is that the Hamiltonian simulation of $\Delta_k^G$ requires fewer qubits compared to the Hamiltonian simulation of $B_G$, namely, $\log\binom{n}{k+1}$ qubits instead of $n$. Moreover, when the graph is clique-dense one can bypass Step 1(a) by padding $\Delta_k^G$ with all-zero rows and columns and letting the eigenvector start out in the maximally mixed state $I/2^n$ (see Section 3.1.1 for more details).

Let $\lambda_{\max}$ denote the largest eigenvalue and let $\lambda_{\min}$ denote the smallest nonzero eigenvalue of $\Delta_k^G$. By scaling down the matrix one chooses to simulate (i.e., either $B$ or $\Delta_k^G$) by $1/\lambda_{\max}$ to avoid multiples of $2\pi$, we can tell whether an eigenvalue is equal to zero or not if the precision of the quantum phase estimation is at least $\lambda_{\max}/\lambda_{\min}$. By the Gershgorin circle theorem (which states that $\lambda_{\max}$ is bounded above by the maximum sum of absolute values of the entries of a column or row) we know that $\lambda_{\max} \in \mathcal{O}(n)$. For the general case not much is known in terms of lower bounds on $\lambda_{\min}$. Nonetheless, even if we do not have such a lower bound, the number of small eigenvalues (as opposed to zero eigenvalues) still conveys topological information about the underlying graph (see Section 2.2.2 for more details). By taking into account the cost of the quantum phase estimation [145], the total runtime of Step 1(b) becomes $\widetilde{\mathcal{O}}\left(n^3/\lambda_{\min}\right)$.

Finally, estimating $\beta_k^G / \dim \mathcal{H}_k^G$ up to additive precision $\epsilon$ can be done using $M \in$

$\mathcal{O}\left(\epsilon^{-2}\right)$ repetitions of Step 1(a) through 1(c). This brings the total cost of estimating $\beta_k^G / \dim \mathcal{H}_k^G$ up to additive precision $\epsilon$ to

$$\widetilde{\mathcal{O}}\left(\left(nk^3\sqrt{\binom{n}{k+1}/\chi_k} + n^3/\lambda_{\min}\right)/\epsilon^2\right).$$

In conclusion, the quantum algorithm for Betti number estimation runs in time polynomial in $n$ under two conditions. Firstly, the graph has to be clique-dense, i.e., it has to satisfy Eq. (2.31) (see Section 2.2.2 for more details). Secondly, the smallest nonzero eigenvalue $\lambda_{\min}$ has to scale at least inverse polynomial in $n$ (see Section 2.2.2 for more details). If both these conditions are satisfied, then the quantum algorithm for Betti number estimation achieves an exponential speedup over the best known classical algorithms if the size of the combinatorial Laplacian – i.e., the number of $(k+1)$-cliques – scales exponentially in $n$ (see Section 2.2.3 for more details).

**Approximate Betti numbers**

As mentioned in the previous section, the quantum algorithm for Betti number estimation does not strictly speaking estimate the Betti number (i.e., the nullity of the combinatorial Laplacian), but rather the number of small eigenvalues of the combinatorial Laplacian. This is because little is known in terms of lower bounds for the smallest nonzero eigenvalue of combinatorial Laplacians, and hence it is unclear to what precision one has to estimate the eigenvalues in the quantum phase estimation. In any case, it is conjectured that for high-dimensional simplicial complexes the smallest nonzero eigenvalue will generally be at least inverse polynomial in $n$ [81], which would imply that quantum phase estimation can in time $\mathcal{O}\left(\text{poly}(n)\right)$ determine whether an eigenvalue is exactly equal to zero.

Even without knowing a lower bound on the smallest nonzero eigenvalue of the combinatorial Laplacian, we can still perform quantum phase estimation up to some fixed inverse polynomial precision. The quantum algorithm for Betti number estimation then outputs an estimate of the number of eigenvalues of the combinatorial Laplacian that lie below this precision threshold. Throughout this thesis we will refer to this as *approximate Betti numbers*, which turn out to still convey information about the underlying graph. For instance, Cheeger's inequality – which relates the sparsest cut of a graph to the smallest nonzero eigenvalues of its standard graph Laplacian – turns out to have a higher-order generalization that utilizes the combinatorial Laplacian [92]. Moreover, there are several other spectral properties of the combinatorial Laplacian beyond the number of small eigenvalues that also convey topological information about the underlying graph. Some of these spectral properties can also be efficiently extracted using quantum algorithms (see Section 3.3.2 for more details).

**Efficient state preparation**

In Section 2.2.2 we saw that the quantum algorithm for Betti number estimation can efficiently estimate approximate Betti numbers if the input graph satisfies certain criteria. In particular, the graph has to be such that one can efficiently prepare the

maximally mixed state over all its cliques of a given size (i.e., the state in Eq. (2.30) in Figure 2.5). In this section we highlight that this state preparation constitutes one of the main bottlenecks in the quantum algorithm for Betti number estimation.

One way to prepare the maximally mixed state over all $k$-cliques of an $n$-vertex graph is to sample $k$-cliques uniformly at random and feed them into the quantum algorithm. For the quantum algorithm for Betti number estimation to run in time sub-exponential in $n$, we have to be able to sample a $k$-clique uniformly at random in time $n^{o(k)}$. However, for general graphs finding a $k$-clique cannot be done in time $n^{o(k)}$ unless the exponential time hypothesis fails [56]. Nonetheless, for certain families of graphs, uniform clique sampling can be done much more efficiently, e.g., in time polynomial in $n$ (in which case the quantum algorithm also runs in time polynomial in $n$). In particular, the graph's clique-density (i.e., probability that a uniformly random subset of vertices is a clique), or the graph's arboricity (which up to a factor $1/2$ is equivalent to the maximum average degree of a subgraph) are important factors that dictate the efficiency of uniform clique sampling algorithms. In Section 3.2.4 we outline concrete families of graphs (based on their clique-density or arboricity) for which the quantum algorithm achieves a (superpolynomial) speedup over classical algorithms.

### 2.2.3 Classical algorithms for Betti number estimation

In this section we will closely investigate the state-of-the-art classical algorithms, to analyze whether it is possible to strengthen the argument for quantum advantage (or, to actually find an efficient classical algorithm) for the topological data analysis problem. In particular, we will cover classical algorithms based on numerical linear algebra or random walks and analyze the theoretical hurdles that, at least currently, stymie them from performing equally as well as the quantum algorithm.

To the best of our knowledge, the best known classical algorithms for approximate Betti number estimation is based on a numerical linear algebra algorithm for low-lying spectral density estimation [190, 59, 70, 124]. These algorithms typically run in time linear in the number of nonzero entries. Since combinatorial Laplacians are $n$-sparse, the number of nonzero entries of the combinatorial Laplacian – and hence also the runtime of the best known classical algorithm for approximate Betti number estimation – scales as

$$\mathcal{O}\left(n \cdot \chi_k\right) \in \mathcal{O}\left(n^{k+1}\right).$$

Recall that the quantum algorithm for Betti number estimation can estimate approximate Betti numbers in time polynomial in $n$ if we can efficiently prepare the maximally mixed state over the cliques of a given size (e.g., if it satisfies Eq. (2.31)). For graphs that satisfy this condition, we conclude that the quantum algorithm for Betti number estimation achieves an exponential speedup over the best known classical algorithms if the size of the combinatorial Laplacian – i.e., the number of $(k+1)$-cliques – scales exponential in $n$ (which requires $k$ to scale with $n$). For exponential speedups for Betti number estimation, we also require that the smallest nonzero eigenvalue of the combinatorial Laplacian scales at least inverse polynomially in $n$.

To investigate the actual hardness of approximate Betti number estimation, we go one step further and discuss new possibilities for efficient classical algorithms. In particular, we investigate potential classical algorithms that take into account the specifics of the combinatorial Laplacian by using carefully designed random walks. Firstly, there exists a classical random walk based algorithm that can approximate the spectrum of the 0th combinatorial Laplacian (i.e., the ordinary graph Laplacian) up to $\epsilon$ distance in the Wasserstein-1 metric in time $\mathcal{O}\left(\exp(1/\epsilon)\right)$ (i.e., independent of the size of the graph) [65]. To generalize this to higher-order combinatorial Laplacians, one would have to construct an efficiently implementable walk operator whose spectral properties coincide with the higher-order combinatorial Laplacian. While potential candidates for such higher-order walk operators have previously been studied [143, 154], relatively little is known about such higher-order walk operators. Note that such a construction must take into account the specifics of the combinatorial Laplacian, since if the construction would work for arbitrary sparse Hermitian matrices, then this would lead to an efficient classical algorithm for LLSD (which by Theorem 5 is widely-believed to be impossible). Recently, Berry et al. [33] and Apers et al. [22] proposed new classical algorithms for Betti number estimation based on random walks that works best precisely in the regime where the quantum algorithm works best. However, the scaling of these algorithm with respect to the spectral gap is exponentially worse compared to the existing quantum algorithms. Thus, to obtain a quantum speedup, we must ensure that the spectral gap of the combinatorial Laplacian is not too large such that these classical algorithms become efficient.

## 2.3 Structural risk minimization

When looking for the optimal family of classifiers for a given learning problem, it is important to carefully select the family's *complexity* (also known as expressivity or capacity). For instance, in the case of linear classifiers, it is important to select what kind of hyperplanes one allows the classifier to use. Generally, the more complex the family is, the lower the training errors will be. However, if the family becomes overly complex, then it becomes more prone to worse generalization performance (i.e., due to overfitting). Structural risk minimization is a concrete method that balances this trade-off in order to obtain the best possible performance on unseen examples. Specifically, structural risk minimization aims to saturate well-established upper bounds on the expected error of the classifier that consist of the sum of two inversely related terms: a *training error* term, and a *complexity term* penalizing more complex models.

In statistical learning theory it is generally assumed that the data is sampled according to some underlying probability distribution $P$ on $\mathcal{X} \times \{-1, +1\}$. The goal is to find a classifier that minimizes the probability that a random pair sampled according to $P$ is misclassified. That is, the goal is to find a classifier $c_{f,d}(x) = \operatorname{sign}(f(x) - d)$ that minimize the *expected error* given by

$$\operatorname{er}_P(c_{f,d}) = \Pr_{(x,y)\sim P}\big(c_{f,d}(x) \neq y\big). \tag{2.33}$$

As one generally only has access to training examples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^m$ that are sampled according to the distribution $P$, it is not possible to compute $\mathrm{er}_P$ directly. Nonetheless, one can try to approximate Equation (2.33) using *training errors* such as

$$\widehat{\mathrm{er}}_{\mathcal{D}}(c_{f,d}) = \frac{1}{m}\Big|\big\{i \;:\; c_{f,d}(x_i) \neq y_i\big\}\Big|, \tag{2.34}$$

$$\widehat{\mathrm{er}}^{\gamma}_{\mathcal{D}}(c_{f,d}) = \frac{1}{m}\Big|\big\{i \;:\; y_i \cdot \big(f(x_i) - d\big) < \gamma\big\}\Big|, \;\; \gamma \in \mathbb{R}_{\geq 0}. \tag{2.35}$$

Intuitively, $\widehat{\mathrm{er}}_{\mathcal{D}}$ in Equation (2.34) represents the frequency of misclassified training examples, and $\widehat{\mathrm{er}}^{\gamma}_{\mathcal{D}}$ in Equation (2.35) represents the frequency of training examples that are either misclassified or are "within margin $\gamma$ from being misclassified". In particular, for $\gamma = 0$ both training error estimates are identical (i.e., $\widehat{\mathrm{er}}_{\mathcal{D}} = \widehat{\mathrm{er}}^{0}_{\mathcal{D}}$). When selecting the optimal classifiers from a given model one typically searches for the classifier that minimizes the training error (in practice more elaborate and smooth loss functions are used), which is referred to as *empirical risk minimization*. The problem that structural risk minimization aims to tackle is how to optimally select a model such that one will have some guarantee that the training error will be close to the expected error.

Structural risk minimization uses expected error bounds – two of which we will discuss shortly – that involve a training error term, and a complexity term that penalizes more complex models. This complexity term usually scales with a certain measure of the complexity of the family of classifiers. A well known example of such a complexity measure is the Vapnik-Chervonenkis dimension.

**Definition 1** (VC dimension [194])**.** *Let $\mathcal{C}$ be a family of functions on $\mathcal{X}$ taking values in $\{-1, +1\}$. We say that a set of points $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$ is shattered by $\mathcal{C}$ if for all $y \in \{-1, +1\}^m$, there exists a classifier $c_y \in \mathcal{C}$ that satisfies $c_y(x_i) = y_i$. The VC dimension of $\mathcal{C}$ defined as*

$$\mathrm{VC}\big(\mathcal{C}\big) = \max\big\{m \mid \exists\{x_1, \ldots, x_m\} \subset \mathcal{X} \text{ that is shattered by } \mathcal{C}\big\}.$$

Besides the VC dimension we also consider a complexity measure called the *fat-shattering dimension*, which can be viewed as a generalization of the VC dimension to real-valued functions. An important difference between the VC dimension and the fat-shattering dimension is that the latter also takes into account the so-called *margins* that the family of classifiers can achieve. Here the margin of a classifier $c_{f,d}(x) = \mathrm{sign}\big(f(x) - d\big)$ on a set of examples $\{x_i\}_{i=1}^m$ is given by $\min_i |f(x_i) - d|$. Throughout the literature, this is often referred to as the functional margin.

**Definition 2** (Fat-shattering dimension [117])**.** *Let $\mathcal{F}$ be a family of real-valued functions on $\mathcal{X}$. We say that a set of points $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$ is $\gamma$-shattered by $\mathcal{F}$ if there exists an $s \in \mathbb{R}^m$ such that for all $y \in \{-1, +1\}^m$, there exists a function*

$f_y \in \mathcal{F}$ *satisfying*

$$f_y(x_i) \begin{cases} \leq s_i - \gamma & \text{if } y_i = -1, \\ \geq s_i + \gamma & \text{if } y_i = +1. \end{cases}$$

*The fat-shattering dimension of* $\mathcal{F}$ *is a function* $\mathrm{fat}_{\mathcal{F}} : \mathbb{R} \to \mathbb{Z}_{\geq 0}$ *that maps*

$$\mathrm{fat}_{\mathcal{F}}(\gamma) = \max\big\{m \;\big|\; \exists\{x_1, \ldots, x_m\} \subset \mathcal{X} \text{ that is } \gamma\text{-shattered by } \mathcal{F}\big\}.$$

We will now state two expected error bounds that can be used to perform structural risk minimization. These error bounds theoretically quantify how an increase in model complexity (i.e., VC dimension or fat-shattering dimension) results in a worse expected error (i.e., due to overfitting). First, we state the expected error bound that involves the VC dimension.

**Theorem 1** (Expected error bound using VC dimension [201])**.** *Consider a set of functions* $\mathcal{C}$ *on* $\mathcal{X}$ *taking values in* $\{-1, +1\}$*. Suppose* $\mathcal{D} = \big\{(x_i, y_i)\big\}_{i=1}^{m}$ *is sampled using* $m$ *independent draws from* $P$*. Then, with probability at least* $1-\delta$*, the following holds for all* $c \in \mathcal{C}$*:*

$$\mathrm{er}_P(c) \leq \widehat{\mathrm{er}}_{\mathcal{D}}(c) + 62\sqrt{\frac{k}{m}} + 3\sqrt{\frac{\log(2/\delta)}{2m}} \tag{2.36}$$

*where* $k = \mathrm{VC}\big(\mathcal{C}\big)$*.*

Next, we state the expected error bound that involves the fat-shattering dimension. One possible advantage of using the fat-shattering dimension instead of the VC dimension is that it can take into account the margin that the classifier achieves on the training examples. This turns out to be useful since this margin can be used to more precisely fine-tune the expected error bound.

**Theorem 2** (Expected error bound using fat-shattering dimension [28])**.** *Consider a set of real-valued functions* $\mathcal{F}$ *on* $\mathcal{X}$*. Suppose* $\mathcal{D} = \big\{(x_i, y_i)\big\}_{i=1}^{m}$ *is sampled using* $m$ *independent draws from* $P$*. Then, with probability at least* $1 - \delta$*, the following holds for all* $c(x) = \mathrm{sign}\big(f(x) - d\big)$ *with* $f \in \mathcal{F}$ *and* $d \in \mathbb{R}$*:*

$$\mathrm{er}_P(c) \leq \widehat{\mathrm{er}}^{\gamma}_{\mathcal{D}}(c) + \sqrt{\frac{2}{m}\Big(k \log(34em/k) \log_2(578m) + \log(4/\delta)\Big)}. \tag{2.37}$$

*where* $k = \mathrm{fat}_{\mathcal{F}}(\gamma/16)$*.*

**Remark(s).** *If the classifier can correctly classify all examples in* $\mathcal{D}$*, then the optimal choice of* $\gamma$ *in the above theorem is the margin achieved on the examples in* $\mathcal{D}$*, i.e.,* $\gamma = \min_{x_i \in \mathcal{D}} \big|f(x_i) - d\big|$*.*

Generally, the more complex a family of classifiers is, the larger its generalization errors are. This correlation between a family's complexity and its generalization errors is theoretically quantified in Theorems 1 and 2. Specifically, the more complex the family is the larger its VC dimension will be, which strictly increases the second

term in Equation 2.36 that corresponds to the generalization error. Note that for the fat-shattering dimension in Theorem 2 this is not as obvious. In particular, a more complex model could achieve a larger margin $\gamma$, which actually decreases the second term in Equation 2.37 that corresponds to the generalization error.

Theorems 1 and 2 establish that in order to minimize the expected error, we should aim to minimize either of the sums on the right-hand side of Equations (2.36) or (2.37) (depending on which complexity measure one wishes to focus on). Note that in both cases the first term corresponds to a training error and the second term corresponds to a complexity term that penalizes more complex models. Crucially, the effect that the complexity measure of the family of classifiers has on these terms is inversely related. Namely, a large complexity measure generally gives rise to smaller training errors, but at the cost of a larger complexity term. Balancing this trade-off is precisely the idea behind structural risk minimization. More precisely, structural risk minimization selects a classifier that minimizes either of the expected error bounds stated in Theorem 1 or 2, by selecting the classifier from a family whose complexity measure is fine-tuned in order to balance both terms on the right-hand side of Equations (2.36) or (2.37). Note that limiting the VC dimension and fat-shattering dimension does not achieve the same theoretical guarantees on the generalization error, and it will generally give rise to different performances in practice (as also discussed Section 4.2). An overview of the trade-off in the error bounds stated in Theorems 1 and 2 is depicted in Figure 2.6.

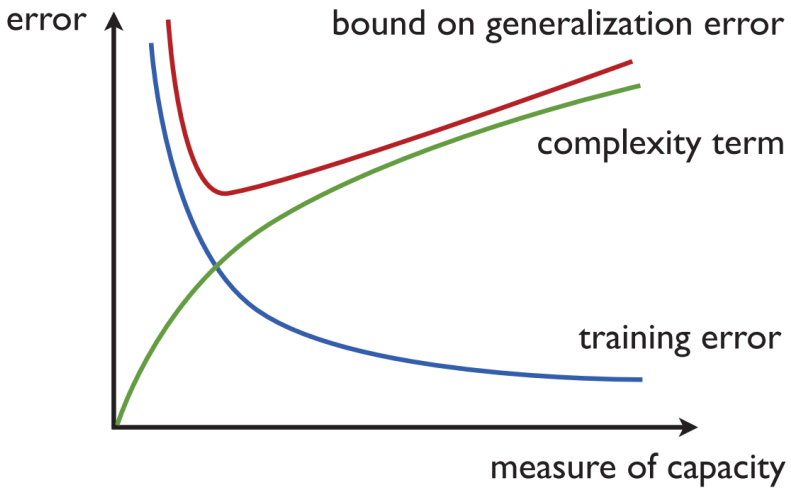



Figure 2.6: Illustration of structural risk minimization taken from [139]. Increasing the complexity causes the training error (blue) to decrease, while it increases the complexity term (green). Structural risk minimization selects the classifier minimizing the expected error bound in Eqs. (2.36) and (2.37) given by the sum of the training error and the complexity term (red).

## 2.4 Reinforcement learning

In this section, we introduce the basic ideas of reinforcement learning (for a more in-depth treatment we refer to [182]). Intuitively, in reinforcement learning the problem is to learn from interactions to achieve a goal. The way this is generally modelled is through the *agent-environment* interaction setup depicted in Figure 2.4. In this setup, the learner takes the role of the agent which continually interacts with the environment by performing *actions*. After every action from the agent, the environment responds by presenting the agent with a new *state*, and some numerical value called the *reward* that the agent tries to maximize over time.

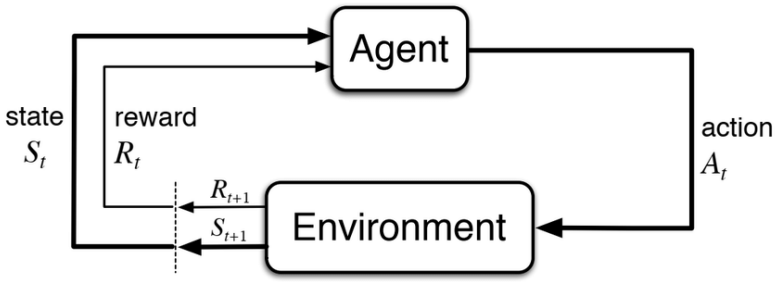



Figure 2.7: The agent-environment interaction setup (taken from [182]).

More precisely, the agent and the environment interact with each other for a number of discrete time steps $t = 0, 1, 2, \ldots$. At every time step $t$, the agent is presented with a state $S_t$ from the environment, at which point it has to select an action $A_t$, after which the environment updates its state to a new state $S_{t+1}$ that it sends to the agent together with some reward $R_{t+1}$ Note that the actions the agent can choose from can depend on the current state of the environment. The way an action $A_t$ causes the environment to change $S_t \to S_{t+1}$ (together with the associated reward $R_{t+1}$) is often modelled using a Markov Decision Process (MDP). We choose not to discuss the MDP-setup in detail here, and instead provide a more high-level overview of reinforcement learning (see [182] for more details).

Generally, the way the agent chooses its actions based on the state of the environment is modelled by what is called its *policy* $\pi(. \mid .)$. The agent's policy maps the current state to a probability distribution over all possible actions, and the agent simply samples from it to selects its next step. More precisely, we let $\pi(a \mid s)$ denote the probability that the agent selects action $A_t = a$ given that the current state of the environment is $S_t = s$.

A standard way of training a reinforcement learning agent is through *policy iteration*, which consists of two consecutive steps: *policy evaluation* and *policy improvement*. First, in the policy evaluation step, the agent uses a policy to interact with

the environment and collect rewards for a given number of steps. Afterwards, in the policy improvement step, the agent updates its policy based on the interactions it had gained in the policy evaluation step. There are many different ways to perform the policy improvement step, e.g., in state of the art deep-learning methods one often uses Q-learning. In short, in Q-learning the agent trains a model (e.g., a deep neural network) that learns what the expected rewards are for a given action in a given state, and uses this to selects its next action. For more details on Q-learning or other policy improvement methods see [182]. Next, we focus on a different method called the *policy gradient method*, which we will use throughout Chapter 5.

In the policy gradient method, the agent is equipped with a differential policy $\pi_\theta$, where $\theta \in \mathbb{R}^\ell$ denote the differential parameters. For instance, $\pi_\theta$ could be implemented using a neural network, in which case the $\theta$ correspond to the weights of the neural network. In Chapter 5 we show how one can use parameterized quantum circuits to encode a differential policy, and how these policies are able to outperform classical policies. The main idea behind policy-gradient methods is to use a scalar performance measure $J(\theta)$, and to update the parameters using a gradient-ascent step

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta)}, \tag{2.38}$$

where $\widehat{\nabla J(\theta)}$ denotes a (stochastic) estimate of the gradient of $J$ with respect to $\theta$. In our case, our performance measure $J(\theta)$ will be the *value function* of the initial state of the environment with respect to the current policy $v_{\pi_\theta}(s_0)$. The value function $v_\pi(s)$ denotes the expected discounted rewards obtained by an agent following policy $\pi$ when starting in state $s$. More precisely, we set $J(\theta) = v_{\pi_\theta}(s_0)$ where $s_0$ denotes the initial state of the environment and

$$v_{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \tag{2.39}$$

where $\gamma \in [0, 1)$ is some discount factor chosen beforehand. What remains is to find a way to obtain a (stochastic) estimate of the gradient of $v_{\pi_\theta}(s_0)$ in order to implement the update step in Eq. (2.38). A priori, when computing the gradient of $v_{\pi_\theta}(s_0)$, we see that it depends on the effect of the policy on the state distribution of the environment. But, since we do not know what the effect of the policy is on the state distribution of the environment, how are we able to compute the gradient of $v_{\pi_\theta}(s_0)$? Fortunately, there is a nice theoretical answer to this question in the form of the *policy gradient theorem*, which provides an analytic expression for the gradient of $v_{\pi_\theta}(s_0)$ that does *not* involve the derivative of the state distribution of the environment.

For our purposes, we will stick to the vanilla version of the policy-gradient algorithm called REINFORCE [182], as used throughout Chapter 5. In this setting, the policy-gradient theorem tells us that

$$\nabla v_{\pi_\theta}(s_0) = \mathbb{E}_\pi\left[G_t \frac{\nabla \pi_\theta(A_t \mid S_t)}{\pi_\theta(A_t \mid S_t)}\right], \tag{2.40}$$

where $G_t = \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$ is called the *return*. The expression on the right

hand side of Eq. (2.40) is exactly what we need, a quantity that we can sample (i.e., it does not require us to know the state distribution of the environment) whose expectation value is equal to the gradient. In other words, we can use the right hand side of Eq. (2.40) to obtain a (stochastic) estimate of $\nabla v_{\pi_\theta}(s_0)$. In particular, we can let the agent interact with the environment for a given number of steps $T$ and collect the interactions

$$\{(s_0, a_0, r_1), (s_1, a_1, r_2), \ldots, (s_T, a_T, r_{T+1})\}.$$

Next, we loop over each step in the interactions $t = 0, \ldots, T$, and at each step $t$ we compute $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_t$, and we update the parameters

$$\theta = \theta + \alpha \gamma^t G_t \nabla \frac{\nabla \pi_\theta(a_t \mid a_t)}{\pi_\theta(a_t \mid a_t)}. \tag{2.41}$$

This is all summarized in Algorithm 1 in Figure 5.1.2.

## 2.5 Computational learning theory

Quantum machine learning (QML) [35, 26] is a bustling field with the potential to deliver quantum enhancements for practically relevant problems. An important goal of the community is to find practically relevant learning problems for which one can prove that quantum learners have an exponential advantage over classical learners. In Chapter 6 of this thesis, we study how to achieve such exponential separations between classical and quantum learners for problems with classical data in the efficient probably approximately correct (PAC) learning framework. But before we do so, we will first introduce the required background and definitions from computational learning theory together with some more computational complexity theory.

### 2.5.1 Learning separations in the PAC learning framework

As already mentioned, we use the standard terminology of the efficient *probably approximately correct* (PAC) learning framework, and we focus on the supervised learning setting (for an overview of the generative modelling setting see [185]). In this framework a learning problem is defined by a *concept class* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where each $\mathcal{C}_n$ is a set of *concepts*, which are functions from some *input space* $\mathcal{X}_n$ (in this thesis we assume $\mathcal{X}_n$ is either $\{0,1\}^n$ or $\mathbb{R}^n$) to some *label set* $\mathcal{Y}_n$ (in this thesis we assume $\mathcal{Y}_n$ is $\{0,1\}$, with the exception of Section 6.1.3 where it is $\{0,1\}^n$)[7]. As input the learning algorithm has access to a procedure $EX(c, \mathcal{D}_n)$ (sometimes called an *example oracle*) that runs in unit time, and on each call returns a labeled *example* $(x, c(x))$, where $x \in \mathcal{X}_n$ is drawn according to *target distributions* $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$. Finally, the learning algorithm has associated to it a hypothesis class $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$, and its goal is to output a *hypothesis* $h \in \mathcal{H}_n$ – which is another function from $\mathcal{X}_n$ to $\mathcal{Y}_n$– that is in some sense "close" to the concept $c \in \mathcal{C}_n$ generating the examples (we will make this more precise shortly).

[7]Since our focus is on the computational complexity of the learner, we choose to explicitly highlight the relevance of the instance size $n$ in our notation.

In the statistical version of the PAC learning framework the learning algorithm has to identify (and/or evaluate) a good hypothesis using $\mathcal{O}(\text{poly}(n))$ many queries to $EX(c, \mathcal{D}_n)$, and the computational complexity (i.e., "runtime") of the learning algorithm is not considered. In this thesis however, we focus on the *efficient* PAC learning framework, where the learning algorithm must output such a good hypothesis in *time* $\mathcal{O}(\text{poly}(n))$ (note that this also implies that the learning algorithm can only use $\mathcal{O}(\text{poly}(n))$ many queries to $EX(c, \mathcal{D}_n)$). Moreover, in this thesis, we study exponential separations specifically with respect to the time complexity of the learning algorithms.

The PAC learning framework formalizes supervised learning. For instance, in the learning scenario where one wants to detect a specific object in an image, the concepts are defined to attain the value 1 when the object is present and 0 otherwise. Moreover, the oracle represents the set of training examples that is available in supervised learning. We formally define efficient PAC learnability as follows.

**Definition 3** (Efficient PAC learnability)**.** *A concept class $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ is efficiently PAC learnable under target distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ if there exists a hypothesis class $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ and a (randomized) learning algorithm $\mathcal{A}$ with the following property: for every $c \in \mathcal{C}_n$, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if $\mathcal{A}$ is given access to $EX(c, \mathcal{D}_n)$ and $\epsilon$ and $\delta$, then with probability at least $1 - \delta$, $\mathcal{A}$ outputs a specification[8] of some $h \in \mathcal{H}_n$ that satisfies*

$$\mathsf{Pr}_{x\sim\mathcal{D}_n}\big[h(x) \neq c(x)\big] \leq \epsilon.$$

*Moreover, the learning algorithm $\mathcal{A}$ must run in time $\mathcal{O}(\text{poly}(n, 1/\epsilon, 1/\delta))$.*

In the above definition, the probability $1 - \delta$ is over the random examples drawn from $EX(c, \mathcal{D}_n)$ and over the internal randomness of $\mathcal{A}$. If the learning algorithm is a polynomial-time classical algorithm (or, a quantum algorithm), we say that the concept class is *classically learnable* (or, *quantumly learnable*, respectively). An important thing to note in the above definition is that the learner itself consists of two parts: a hypothesis class, and a learning algorithm. More precisely, the learner consists of a family of functions that it will use to approximate the concepts (i.e., the hypothesis class), and of a way to select which function from this family is the best approximation for a given concept (i.e., the learning algorithm). This is generally not very different from how supervised learning is done in practice. For example, in deep learning the hypothesis class consists of all functions realizable by a deep neural network with some given architecture, and the learning algorithm uses gradient descent to find the best hypothesis (i.e., the best assignment of weights).

To solve a given learning problem according to Definition 3, one thus needs to construct a learner, which consists of both a learning algorithms as well as a hypothesis class. Specifically, one is thus able to specifically tailor the hypothesis class to the learning problem that one is trying to solve. Another way to define a learning problem and how to solve it, would be to constrain the learner to only output hypotheses from

[8]The hypotheses (and concepts) are specified according to some enumeration $R : \cup_{n\in\mathbb{N}}\{0,1\}^n \to \cup_n \mathcal{H}_n$ (or, $\cup_n \mathcal{C}_n$) and by a "specification of $h \in \mathcal{H}_n$" we mean a string $\sigma \in \{0,1\}^*$ such that $R(\sigma) = h$ (see [116] for more details).

a given hypothesis class (which would then be part of the specification of the learning problem). In this thesis, our main focus is on investigating the limitations of *all* possible classical learners for a given task. To do so, we primarily focus on setting where constructing the right hypothesis class is also part of the learning problem. Our goal is to demonstrate separations that establish the inability of classical learners, regardless of the hypotheses they use, to efficiently solve a learning problem that can be solved by a quantum learner. Nonetheless, in certain cases, it is more natural to consider the setting where the learner is constrained to output hypotheses from a given hypothesis class. We explore this setting, along with an instance of it which is called *proper PAC learning*, in Sections 2.5.1 and 6.3.3.

When one is able to specifically tailor the hypothesis class to a given learning problem (as is the case in Definition 3), it turns out to be necessary to limit the computational power of the hypotheses. Constraining the learning algorithm to run in polynomial-time turns out to be pointless if one allows arbitrary superpolynomial-time hypotheses. More precisely, if we allow superpolynomial-time hypotheses, then any concept class that can be learned by a superpolynomial-time learning algorithm, can also be learned by a polynomial-time learning algorithm (see Appendix D.1.1 for more details). Intuitively, this is because by tailoring the hypothesis class one can "offload" the learning algorithm onto the evaluation of the hypotheses, which makes any constraints on the learning algorithm pointless. This is different when we constrain the learner to only be able to output hypotheses from a given hypothesis class, in which case it can be meaningful and natural to consider hypotheses with superpolynomial runtimes (see also Sections 2.5.1 and 6.3.3). Additionally, because we are studying separations between classical and quantum learners, we make the distinction whether the hypotheses are efficiently evaluatable classically or quantumly.

**Definition 4** (Efficiently evaluatable hypothesis class)**.** *A hypothesis class* $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ *is* classically (quantumly) efficiently evaluatable *if there exists a classical (respectively quantum) polynomial-time evaluation algorithm* $\mathcal{A}_{\text{eval}}$ *that on input* $x \in \mathcal{X}_n$ *and a specification of a hypothesis* $h \in \mathcal{H}_n$*, outputs* $\mathcal{A}_{\text{eval}}(x, h) = h(x)$*.*

For example, the hypotheses could be specified by a polynomial-sized Boolean circuit, in which case they are *classically* efficiently evaluatable. On the other hand, the hypotheses could also be specified by polynomial-depth quantum circuits, in which case they are *quantumly* efficiently evaluatable. If the family of quantum circuits that make up the hypothesis class is $\mathsf{BQP}$-complete, then the hypothesis class will be quantumly efficiently evaluatable, but not classically efficiently evaluatable (assuming $\mathsf{BPP} \neq \mathsf{BQP}$). In this thesis we will drop the "efficiently" and simply call a hypothesis class classically- or quantumly evaluatable.

Given the definitions above one may assume that there is only one way to define a learning separation in the PAC learning framework. However, it is in fact more subtle, and there are various definitions that each have operationally different meanings. In particular, one needs to differentiate whether the learning algorithm is classical or quantum, and whether the hypothesis class is classically- or quantumly- evaluatable. As a result, we define four categories of learning problems: concept classes that are either *classically-* or *quantumly- learnable* (i.e., whether the learning algorithm is classical or quantum), using a *classically-* or *quantumly-* evaluatable hypothesis class.

We denote these categories by $\mathsf{CC}, \mathsf{CQ}, \mathsf{QC}$, and $\mathsf{QQ}$, where the first letter signifies whether the concept class is classically- or quantumly- learnable , and the second letter signifies whether the learner uses a classically- or quantumly- evaluatable hypothesis class. These distinctions are *not* about the nature of the data (i.e., we only consider the setting where the examples are classical) as it often occurs in literature, and even on the Wikipedia-page on quantum machine learning.

**Definition 5** (Categories of learning problem)**.**

- *Let* $\mathsf{CC}$ *denote the set of tuples* $(\mathcal{C}, \mathcal{D})$ *such that* $\mathcal{C}$ *is* ***classically*** *learnable under target distributions* $\mathcal{D}$ *with a* ***classically*** *evaluatable hypothesis class.*
- *Let* $\mathsf{CQ}$ *denote the set of tuples* $(\mathcal{C}, \mathcal{D})$ *such that* $\mathcal{C}$ *is* ***classically*** *learnable under target distributions* $\mathcal{D}$ *with a* ***quantumly*** *evaluatable hypothesis class.*
- *Let* $\mathsf{QC}$ *denote the set of tuples* $(\mathcal{C}, \mathcal{D})$ *such that* $\mathcal{C}$ *is* ***quantumly*** *learnable under target distributions* $\mathcal{D}$ *with a* ***classically*** *evaluatable hypothesis class.*
- *Let* $\mathsf{QQ}$ *denote the set of tuples* $(\mathcal{C}, \mathcal{D})$ *such that* $\mathcal{C}$ *is* ***quantumly*** *efficiently learnable under target distributions* $\mathcal{D}$ *with a* ***quantumly*** *evaluatable hypothesis class.*

We remark that our definitions do not (yet) talk about the computational tractability of the concepts, the importance of which we will discuss in Section 2.5.2 and throughout Sections 6.1 and 6.2. We now proceed with a few observations. Firstly, since any classical algorithm can be simulated by a quantum algorithm it is clear that $\mathsf{CC} \subseteq \mathsf{CQ}$, $\mathsf{CC} \subseteq \mathsf{QC}$, $\mathsf{CC} \subseteq \mathsf{QQ}$, $\mathsf{CQ} \subseteq \mathsf{QQ}$, and $\mathsf{QC} \subseteq \mathsf{QQ}$. Secondly, we make the non-trivial observation that if the hypothesis class can use a quantum evaluation algorithm (i.e., it is quantumly evaluatable), then it does not matter whether we constrain the learning algorithm to be a classical- or a quantum- algorithm. More precisely, as a first result we show that any learning problem that is quantumly learnable using a quantumly evaluatable hypothesis class is also classically learnable using *another* quantumly evaluatable hypothesis class. This observation is summarized in the lemma below, and we defer the proof to Appendix D.1.2.

**Lemma 3.** $\mathsf{CQ} = \mathsf{QQ}$.

The above lemma is analogous to why we constrain the hypotheses to be efficiently evaluatable, in the sense that by changing the hypothesis class one can "offload" the *quantum* learning algorithm onto the evaluation of the *quantum* hypotheses. We reiterate that it is critical that one can change the hypothesis class when mapping a learning problem in $\mathsf{QQ}$ to $\mathsf{CQ}$. If the learner is constrained to output hypotheses from a fixed hypothesis class, then such a collapse does not happen.

Having studied the relations between the categories learning problems, we can now specify what it means for a learning problem to exhibit a separation between classical and quantum learners.

**Definition 6** (Learning separation)**.** *A learning problem* $L = (\mathcal{C}, \mathcal{D})$ *is said to exhibit a*

- $\mathsf{CC}/\mathsf{QC}$ *separation if* $L \in \mathsf{QC}$ *and* $L \notin \mathsf{CC}$.
- $\mathsf{CC}/\mathsf{QQ}$ *separation if* $L \in \mathsf{QQ}$ *and* $L \notin \mathsf{CC}$.

Firstly, note that due to the previously listed inclusions any $\mathsf{CC}/\mathsf{QC}$ separation is also a $\mathsf{CC}/\mathsf{QQ}$ separation. Secondly, note that by fully relying on the classical intractability of concepts one can construct trivial learning separations that are less about "learning" in an intuitive sense. More precisely, consider the separation exhibited by the concept class $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where each $\mathcal{C}_n$ consists of a *single* concept that is classically hard to evaluate on a fraction of inputs even in the presence of data, yet it can be efficiently evaluated by a quantum algorithm. This singleton concept class is clearly quantumly learnable using a quantumly evaluatable hypothesis class. Also, it is not classically learnable using any classically evaluatable hypothesis class, since this would violate the classical intractability of the concepts. However, note that the quantum learner *requires no data* to learn the concept class, so it is hard to argue that this is a genuine learning problem. We will discuss how to construct examples of such concept classes in Sections 6.1.1 and 6.1.2.

**Observation 1** (Trivial learning separation without data)**.** *Consider a family of concept classes* $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$*, where each* $\mathcal{C}_n = \{c_n\}$ *consists of a single concept that is classically hard to evaluate on a fraction of inputs when given access to examples, yet it can be efficiently evaluated by a quantum algorithm. Then,* $\mathcal{C}$ *exhibits a* $\mathsf{CC}/\mathsf{QQ}$ *separation which is quantum learnable without requiring data.*

We want to emphasize that some concept classes are *efficiently evaluatable on a classical computer*, yet they are *not classically learnable*. One such example is the class of polynomially-sized logarithmic-depth Boolean circuits [116]. Moreover, in Section 6.1.3, we provide an example of concept class which (assuming a plausible but relatively unexplored hardness assumption) exhibits a $\mathsf{CC}/\mathsf{QC}$ separation where the concepts are efficiently evaluatable on a classical computer.

### Learning separations with a fixed hypothesis class and proper PAC learning

In some practical settings, it can be natural to constrain the learner to only output hypotheses from a fixed hypothesis class. To give a physics-motivated example, when studying phases of matter one might want to identify what observable properties characterize a phase. One can formulate this problem as finding a specification of the correct hypothesis selected from a hypothesis class consisting of possible *order parameters*. More precisely, we fix the hypotheses to be of a particular form, e.g., those that compute certain expectation values of ground states given a specification of a Hamiltonian[9]. We further discuss this setting of characterizing phases of matter in Section 6.3.3, where we also discuss Hamiltonian learning as a natural setting in which the learner is constrained to output hypotheses from a fixed hypothesis class.

[9]Note the computation of these hypotheses can be $\mathsf{QMA}$-hard, as it involves preparing ground states. Nonetheless, we can still study whether a learner is able to *identify* which of these hypotheses matches the data.

Recall that in the standard PAC learning framework discussed in the previous section, the learner is free to output arbitrary hypotheses (barring tractability constraints discussed in Appendix D.1.1). It therefore fails to capture the setting where one aims to characterize phases of matter, as the learner might output hypotheses that are not order parameters, which will not allow one to identify physical properties that characterize a phase. To remedy this, one could consider the setting where the learner is constrained to output hypotheses from a fixed hypothesis class.

**Definition 7** (Efficient PAC learnability with fixed hypothesis class)**.** *A concept class $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ is efficiently PAC learnable with a fixed hypothesis class $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ under target distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ if there exists a (randomized) learning algorithms $\mathcal{A}$ with the following property: for every $c \in \mathcal{C}_n$, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if $\mathcal{A}$ is given access to $EX(c, \mathcal{D}_n)$ and $\epsilon$ and $\delta$, then with probability at least $1 - \delta$, $\mathcal{A}$ outputs a specification of some $h \in \mathcal{H}_n$ that satisfies*

$$\mathsf{Pr}_{x\sim\mathcal{D}_n}\big[h(x) \neq c(x)\big] \leq \epsilon.$$

*Moreover, the learning algorithm $\mathcal{A}$ must run in time $\mathcal{O}(\mathrm{poly}(n, 1/\epsilon, 1/\delta))$.*

In the above definition, the probability $1 - \delta$ is over the random examples from $EX(c, \mathcal{D}_n)$ and over the internal randomization of $\mathcal{A}_n$. If the learning algorithm is a polynomial-time classical algorithm (or, a quantum algorithm), we say that the concept class is *classically learnable with fixed hypothesis class* (or, *quantumly learnable with fixed hypothesis class*, respectively). An example of learning with a fixed hypothesis class is that of *proper PAC learning*. In proper PAC learning the learner is constrained to only output hypothesis from the concept class it is trying to learn.

We emphasize again that if the learner is constrained to output hypotheses from a fixed hypothesis class, then it is allowed and reasonable for the hypothesis class to be (classically- or quantumly-) intractable. In particular, doing so will not trivialize the definitions as it did in the standard PAC learning framework (see Appendix D.1) as this requires one to be able to change the hypotheses.

In the setting where the learner is constrained to output hypotheses from a fixed hypothesis class, it is relatively clear how to define a learning separation. In particular, one only has to distinguish whether the learning algorithm is an efficient classical- or quantum- algorithm, which we capture by defining the following categories of learning problems.

**Definition 8** (Categories of learning problem – fixed hypothesis class $\mathcal{H}$)**.**

- *Let $\mathsf{C}_{\mathcal{H}}$ denote the set of tuples $\big(\mathcal{C}, \mathcal{D}\big)$ such that $\mathcal{C}$ is **classically** learnable with fixed hypothesis class $\mathcal{H}$ under target distributions $\mathcal{D}$.*
- *Let $\mathsf{Q}_{\mathcal{H}}$ denote the set of tuples $\big(\mathcal{C}, \mathcal{D}\big)$ such that $\mathcal{C}$ is **quantumly** learnable with fixed hypothesis class $\mathcal{H}$ under target distributions $\mathcal{D}$.*

We can now specify what it means for a learning problem to exhibit a separation between classical and quantum learners in the setting where the learner is constrained to output hypotheses from a fixed hypothesis class.

**Definition 9** (Learning separation – fixed hypothesis class $\mathcal{H}$)**.**
*A learning problem $L = (\mathcal{C}, \mathcal{D}) \in \mathsf{Q}_{\mathcal{H}}$ is said to exhibit a $\mathsf{C}_{\mathcal{H}}/\mathsf{Q}_{\mathcal{H}}$ separation if $L \notin \mathsf{C}_{\mathcal{H}}$.*

In Sections 6.1.3 and 6.1.4, we provide examples of learning separations in the setting where the learner is constrained to output hypotheses from a fixed hypothesis class. Moreover, in Section 6.3.3, we further discuss the practical relevance of this setting by discussing how it captures certain physics-motivated examples of learning settings.

#### Identification versus Evaluation

An important difference in what exactly entails a learning task in practice is whether the learner has to only *identify* a hypothesis that is close to the concept generating the examples, or whether the learner also has to *evaluate* the hypothesis on unseen examples later on. Moreover, these differences in tasks have implications for the role of quantum computers in achieving separations. This difference in tasks is reflected in two aspects within the definitions discussed in this section.

Firstly, this difference in tasks is reflected in the difference between CC/QQ and CC/QC separations. In particular, it is reflected in the task that requires a quantum computer (i.e., what task needs to be classically intractable yet efficient on a quantum computer). On the one hand, for a CC/QC separation, one has to show that only a quantum algorithm can *identify* how to label unseen examples using a classical algorithm. On the other hand, for a CC/QQ separation, one also needs to show that only a quantum algorithm can *evaluate* the labels of unseen examples. In Section 6.1.3, we provide an example of a CC/QC separation (contingent on a plausible though relatively unexplored hardness assumption), where the classical hardness lies in *identifying* an hypothesis matching the examples, since the concepts are efficiently evaluatable classically.

Secondly, the difference in tasks is also reflected in the difference between the setting where the learner is allowed to output arbitrary hypothesis, or whether it can only output hypotheses from a fixed hypothesis class. In the arbitrary hypothesis class setting, one has to demand that the hypotheses are efficiently evaluatable (i.e., see Appendix D.1), which allows the learner to efficiently *evaluate* the hypotheses on unseen examples. In the fixed hypothesis class setting, the hypotheses need not be efficiently evaluatable, and the learner is only required to *identify* the correct hypothesis without having to evaluate it on unseen examples. In Sections 6.1.3 and 6.1.4, we provide examples of separation in the setting where the learner is constrained to output hypotheses from a fixed hypothesis class. Note that the classical hardness in these separation lies identifying the hypotheses, as we do not require the learner to evaluate the hypothesis on unseen examples afterwards.

### 2.5.2 Complexity theory

In this section we provide a short overview of the areas of complexity theory that we will refer to when discussing separations in the PAC learning framework. In particular, we focus on the computational hardness assumptions that one can leverage to establish a learning separation.

We turn our attention to the definition of the PAC learning framework (see Definition 3) and make some observations that will be relevant later. First, we note that the hypothesis that the learning algorithm outputs is only required to be correct with probability $\epsilon$ over the target distribution. In complexity theory, this is related to the notion of heuristic complexity classes (for more details see [37]). To define heuristic complexity classes, we first need to incorporate the target distribution as a part of the problem, which is done by considering distributional problems.

**Definition 10** (Distributional problem [37])**.** *A distributional problem is a tuple* $(L, \mathcal{D})$*, where* $L \subseteq \{0,1\}^*$ *is a language*[10] *and* $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ *is a family of distributions such that* $\mathrm{supp}(\mathcal{D}_n) \subseteq \{0,1\}^n$*.*

A distributional problem at its core remains a decision problem, wherein the inputs follow a specific distribution. It is important not to confuse this with a *sampling problem*, in which the goal is to generate samples from a designated distribution. Having defined distributional problems, we now define the relevant heuristic complexity classes.

**Definition 11** (Heuristic complexity [37])**.** *A distributional problem* $(L, \mathcal{D})$ *is in* HeurBPP *if there exists a polynomial-time randomized classical algorithm* $\mathcal{A}$[11] *such that for all* $n$ *and* $\epsilon > 0$[12]*:*

$$\mathsf{Pr}_{x\sim\mathcal{D}_n}\Big[\mathsf{Pr}\big(\mathcal{A}(x, 0^{\lfloor 1/\epsilon\rfloor}) = L(x)\big) \geq \frac{2}{3}\Big] \geq 1-\epsilon, \tag{2.42}$$

*where the inner probability is taken over the internal randomization of* $\mathcal{A}$*.*

*Analogously, we say that a distributional problem* $(L, \mathcal{D})$ *is in* HeurBQP *if there exists a polynomial-time quantum algorithm* $\mathcal{A}$ *that satisfies the property in Eq.* (2.42)*.*

A related and perhaps better known area of complexity theory is that of *average-case* complexity. The main difference between average-case complexity and heuristic complexity, is that in the latter one is allowed to err, whereas in the former one can never err but is allowed to output "don't know". Note that an average-case algorithm can always be converted into a heuristic algorithm by simply outputting a random result instead of outputting "don't know". Similarly, if there is a way to efficiently check if a solution is correct, any heuristic algorithm can be turned into an average-case algorithm by outputting "don't know" when the solution is incorrect. Even though they are closely related, in the PAC learning framework one deals with heuristic complexity.

While heuristic-hardness statements are not as common in quantum computing literature, many cryptographic security assumptions (such as that of RSA and Diffie-Hellman) are in fact examples of heuristic-hardness statements. These heuristic-hardness statements are generally derived from *worst-case to average-case reductions*,

[10] Throughout this thesis, we also use an equivalent definition of a language $L \subseteq \{0,1\}^*$ by instead calling it a *problem* and defining it as a function $L : \{0,1\}^* \to \{0,1\}$ such that $L(x) = 1$ if and only if $x \in L$.

[11] More precisely, a Turing machine.

[12] Here $0^{\lfloor 1/\epsilon\rfloor}$ denotes the bitstring consisting of $\lfloor 1/\epsilon\rfloor$ zeroes (i.e., it is a unary specification of the precision $\epsilon$).

which show that being correct with a certain probability over a specific input distribution is at least as difficult as being correct on all inputs. Problems that admit a worst- to average-case reduction are called *random self-reducible* (for a formal definition see [79]). It is worth noting that despite the term "average-case", these reductions can also yield heuristic hardness statements. Specifically, if one can efficiently check whether a solution is correct, then a worst-case to average-case reduction also results in a heuristic hardness statement when the worst-case is hard. For instance, a worst-case to average-case reduction by Blum and Micali [36] demonstrates that for the discrete logarithm problem being correct on any $\frac{1}{2} + \frac{1}{\text{poly}(n)}$ fraction of inputs is as difficult as being correct for all inputs (notably, modular exponentiation allows for efficient checking of the correctness of a discrete logarithm solution).

Finally, there is the notion of the example oracle. The fact that access to the example oracle radically enhances what can be efficiently evaluated is related (though not completely analogous, as we will explain below) to the notion of "advice" complexity classes such as $\mathsf{P/poly}$.

**Definition 12** (Polynomial advice [24])**.** *A problem $L : \{0,1\}^* \to \{0,1\}$ is in $\mathsf{P/poly}$ if there exists a polynomial-time classical algorithm $\mathcal{A}$ with the following property: for every $n$ there exists an advice bitstring $\alpha_n \in \{0,1\}^{\text{poly}(n)}$ such that for all $x \in \{0,1\}^n$:*

$$\mathcal{A}(x, \alpha_n) = L(x). \tag{2.43}$$

*Analogously, we say that a problem $L$ is in $\mathsf{BQP/poly}$ if there exists a polynomial-time quantum algorithm $\mathcal{A}$ with the following property: for every $n$ there exists an advice bitstring $\alpha_n \in \{0,1\}^{\text{poly}(n)}$ such that for all $x \in \{0,1\}^n$:*

$$\mathsf{Pr}\big(\mathcal{A}(x, \alpha_n) = L(x)\big) \geq \frac{2}{3}, \tag{2.44}$$

*where the probability is taken over the internal randomization of $\mathcal{A}$.*

Equivalently, $\mathsf{P/poly}$ can also be defined as the class of problems that can be solved by a *non-uniform* family of polynomial-size Boolean circuits. Specifically, for each instance size, a polynomially-sized circuit that solves the problem exists, though there does not need to be a polynomial-time algorithm that constructs these circuits from the instance size. Since in the PAC learning framework we deal with randomized learning algorithms one may want to consider $\mathsf{BPP/poly}$ instead, however by [15] we have that $\mathsf{BPP} \subseteq \mathsf{P/poly}$, and so $\mathsf{BPP/poly} = \mathsf{P/poly}$. On the other hand, from the perspective of the PAC learning framework, it is both natural and essential to allow the algorithm that uses the advice to err on a fraction of inputs, which is captured by the complexity class $\mathsf{HeurP/poly}$.

**Definition 13** (Heuristic complexity with polynomial advice)**.** *A distributional problem $(L, \mathcal{D})$ is in $\mathsf{HeurP/poly}$ if there exists a polynomial-time classical algorithm $\mathcal{A}$ with the following property: for every $n$ and $\epsilon > 0$ there exists an advice string $\alpha_{n,\epsilon} \in \{0,1\}^{\text{poly}(n,1/\epsilon)}$ such that:*

$$\mathsf{Pr}_{x \sim \mathcal{D}_n}\left[\mathcal{A}(x, 0^{\lfloor 1/\epsilon \rfloor}, \alpha_{n,\epsilon}) = L(x)\right] \geq 1 - \epsilon. \tag{2.45}$$

*Analogously, we say that $(L, \mathcal{D})$ is in* HeurBQP/poly *if there exists a polynomial-time quantum algorithm $\mathcal{A}$ such that: for every $n$ and $\epsilon > 0$ there exists an advice string $\alpha_{n,\epsilon} \in \{0,1\}^{\mathrm{poly}(n,1/\epsilon)}$ with the following property:*

$$\mathsf{Pr}_{x\sim\mathcal{D}_n}\left[\mathsf{Pr}\left(\mathcal{A}(x, 0^{\lfloor 1/\epsilon \rfloor}, \alpha_{n,\epsilon}) = L(x)\right) \geq \frac{2}{3}\right] \geq 1 - \epsilon, \tag{2.46}$$

*where the inner probability is taken over the internal randomization of $\mathcal{A}$.*

Note that in the PAC learning framework, the advice that the learning algorithm gets is of a specific form, namely that obtained through queries to the example oracle. This is more closely related to the notion of "sampling advice" complexity classes such as BPP/samp [109] defined below. In [109] it is shown that BPP/samp $\subseteq$ P/poly, i.e., sampling advice is not more powerful than the standard notion of advice.

**Definition 14** (Sampling advice [109])**.** *A problem $L : \{0,1\}^* \to \{0,1\}$ is in* BPP/samp *if there exists polynomial-time classical randomized algorithms $\mathcal{S}$ and $\mathcal{A}$ such that for every $n$:*

- *$\mathcal{S}$ generates random instances $x \in \{0,1\}^n$ sampled from the distribution $\mathcal{D}_n$.*
- *$\mathcal{A}$ receives as input $\mathcal{T} = \{(x_i, L(x_i)) \mid x_i \sim \mathcal{D}_n\}_{i=1}^{\mathrm{poly}(n)}$ and satisfies for all $x \in \{0,1\}^n$:*

$$\mathsf{Pr}\big(\mathcal{A}(x, \mathcal{T}) = L(x)\big) \geq \frac{2}{3}, \tag{2.47}$$

  *where the probability is taken over the internal randomization of $\mathcal{A}$ and $\mathcal{T}$.*

Having related notions in the PAC learning framework to different areas of complexity theory, we are now ready to determine what computational hardness assumptions one can leverage to establish that no classical learner is able to learn a given concept class. More specifically, how hard must evaluating the concepts be for the concept class to not be classically learnable? Since the learning algorithm is a *randomized* algorithm that *heuristically* computes the concepts when provided with *advice* in the form of samples from the example oracle, the existence of a polynomial-time learning algorithm puts the concepts in a complexity class that we call HeurBPP/samp.

**Definition 15.** *A distributional problem $(L, \mathcal{D})$ is in* HeurBPP/samp *if there exists classical randomized algorithms $\mathcal{S}$ and $\mathcal{A}$ such that for every $n$:*

- *$\mathcal{S}$ generates random instances $x \in \{0,1\}^n$ sampled from the distribution $\mathcal{D}_n$.*
- *$\mathcal{A}$ receives as input $\mathcal{T} = \{(x_i, L(x_i)) \mid x_i \sim \mathcal{D}_n\}_{i=1}^{\mathrm{poly}(n)}$ and for every $\epsilon > 0$ satisfies:*

$$\mathsf{Pr}_{x\sim\mathcal{D}_n}\left[\mathsf{Pr}\big(\mathcal{A}(x, 0^{\lfloor 1/\epsilon \rfloor}, \mathcal{T}) = L(x)\big) \geq \frac{2}{3}\right] \geq 1 - \epsilon, \tag{2.48}$$

  *where the inner probability is taken over the internal randomization of $\mathcal{A}$ and $\mathcal{T}$.*

More precisely, if the concepts lie outside of HeurBPP/samp, then the concept class is not classically learnable. We can connect the class HeurBPP/samp to other complexity classes by adopting a proof strategy similar to that of [109] (we defer the proof to Appendix D.1.3).

**Lemma 4.** $\mathsf{HeurBPP/samp} \subseteq \mathsf{HeurP/poly}$.

By the above lemma, we find that any problem not in HeurP/poly is also not in HeurBPP/samp. Consequently, to show the non-learnability of a concept class, it is sufficient to show that the concept class includes concepts that are not in HeurP/poly.

Having discussed the related notions from computational learning theory and complexity theory, we are set to investigate how one establishes learning separations. First, in Section 6.1, we will analyze how existing learning separations have used efficient data generation, and we generalize this construction to (*i*) establish a learning separation (contingent on a plausible though relatively unexplored hardness assumption) with efficiently evaluatable concepts, and (*ii*) establish a learning separation in the setting where the learner is constrained to output an hypothesis from a fixed hypothesis class. Afterwards, in Section 6.2, we discuss the additional constructions required to prove separations in tune with the folklore that quantum machine learning is most likely to have it advantages when the data generated by a "genuine quantum process". For an overview of the learning separations discussed throughout this thesis see Table 2.1.

| **First proposed** | **Concepts based on** | **Separation** | **Complexity concepts** |
|---|---|---|---|
| [126] | Discrete logarithm | CC/QQ | $\notin \mathsf{BPP}$ |
| [173] | Discrete cube root | CC/QC | $\in (\mathsf{P/poly}) \setminus \mathsf{BPP}$ |
| Section 6.1.3 | Modular exponentiation | CC/QC | $\in \mathsf{P}$ |
| Section 6.1.4 | Discrete cube root | $\mathsf{C}_{\mathcal{H}}/\mathsf{Q}_{\mathcal{H}}$ | $\in \mathsf{P}$ |
| Section 6.2.1 | Quantum process | CC/QQ | $\notin \mathsf{HeurP/poly}$ but $\in \mathsf{BQP}$ |

Table 2.1: The learning separations discussed in Sections 6.1 and 6.2.