UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Department of Physics
Bachelor degree

Minimization strategies in Variational Quantum Algorithms

Supervisor:
Prof. Stefano Carrazza

Co-supervisor:
Adrián Pérez Salinas
(Universitat de Barcelona,
Barcelona Supercomputing Center)

Candidate:
Nicole Zattarin
Matr. 931646

A.Y. 2020/2021

Contents

Introduction

Quantum Computing (QC) is a novel computational paradigm based on the features of quantum mechanics, such as superposition and entanglement. This framework integrates several notions that are, at least from a functional point of view, the quantum counterpart of classical concepts. For instance, central elements of the quantum architecture are qubits and gates. A qubit is a two-levels quantum state, thus it represents the analogous of a classical bit, even though it is subjected to the rules of quantum mechanics. Moreover, quantum gates are employed to perform operations on qubits, playing the same role of classical logic gates in traditional computation. Therefore, this framework opens new and different ways to process information.

Nowadays, quantum computation represents a leading research topic both in physics, computer science, and industry, since there are promising experimental results suggesting that this approach may lead to an advantage over classical computation. Theoretical research pursues problems and algorithms in which a quantum speed-up can be reached as a manner to widen the range of applicability of quantum computing. Remarkable results in this sense are represented by Shor's algorithm [15] for factorization and Grover's algorithm [19] designed for unstructured search problems. Nevertheless, the theoretical achievement must also be confirmed experimentally both from a software and hardware point of view, possibly designing specific implementations capable of addressing such computations optimally.

Indeed, a mathematical model of computation is an idealization: we design algorithms on the assumption that the execution will not be affected by errors. This assumption is fundamentally mistaken, since hardware facilities that implement an abstract model are imperfect, both in classical and quantum computing. However, while we overcame such concern for classical devices, it still remains an open problem in QC. In particular, the main concern in this field is that a QC model is affected by inexact implementations of the required logical operations. As a consequence, small quantum computers can successfully be built, but scaling these up to computers that are large enough to yield useful computations presents two main difficulties: decoherence and noise.

At the state-of-art, it is generally said the we are living in the "Noisy Intermediate-Scale Quantum" (NISQ) era, since our quantum devices are substantially imperfect: limited in size and noise affected. In order to overcome such issues, we are pursuing to build a fault-tolerant framework, that allows qubits to be protected from quantum errors introduced by poor control or environmental interactions. However, this Fault-Tolerant Quantum Computing paradigm has yet to be reached. Indeed, different protocols have been developed to avoid decoherence and inaccuracy, see for instance Quantum Error Correction (QEC), but the application of such procedures on current problems is still unfeasible experimentally.

Despite these limitations the quantum advantage over classical computation, Quantum Supremacy, has been reached in the NISQ era. Indeed, Quantum Supremacy is a goal feasible by Near-Term quantum devices, since it does not require high quality QEC computers, which is long-term objective. It is worth to mention the experimental proofs of quantum advantage on NISQ hardwares, since these represent a milestone in QC research: Google claim in 2019 [33], and the result of the researchers of the University of Science and Technology of China in 2020 [34].

A leading strategy to obtain quantum advantage on NISQ devices is offered by Variational Quantum Algorithms (VQA). These methods are quantum-classical hybrid optimization schemes that employ a classical minimizer to train a parameterized quantum circuit. In this model a given state is prepared, it evolves through the circuit, and eventually measurements are performed on the system. Such measures are used to evaluate a loss function, that is then minimized by a classical optimizer.

The main benefit of employing VQAs deals with their capability to provide *good* results in a *reasonable amount* of time. Indeed, VQAs lower the amount of quantum resources needed to perform a computation, even though they are capable of approaching the optimum with a satisfying level of accuracy. In this sense, it is worth to highlight that the quality of the approximation depends both on the number of parameters involved, and their allocation within the circuit, that is the design of the ansatz. Another property that makes these algorithms widely employed is their versatility: VQAs provide a general scheme that may be applied to different problems, from physical systems to Machine Learning.

The classical optimization process is the key element of this procedure, since it takes in input all the elements of the VQA, that are elaborated in order to finally give back the result of the whole routine. As a consequence, resolving a given problem strongly depends on the capability of the optimizer to efficiently explore the possibilities provided by the ansatz. Thus, an in-depth understanding of their behaviour is crucial to design a well performing VQA.

This work focuses on the role played by minimizers in the optimization subroutine of a VQA. Our purpose is to identify the family of optmization algorithms that is best suited for different VQAs. Thus, we provide a detailed description of the possible strategies that may be chosen to minimize a given loss function. The first fundamental property of optimization methods that must be taken into account concerns wether they are designed to compute gradient and higher order derivatives. Gradient-based minimizers make use of first- or second-order derivatives, thus loss functions are required to exhibit a certain regularity for the optimizer to perform properly. In addition, if the search space is too flat or if it presents a huge quantity of local minima, the gradient does not provide all the information necessary to reach convergence, and derivatives-based algorithms may fail. On the other hand, gradient-free methods usually exploit heuristic search strategies, geometrical structures or a combination of both of them, thus these algorithms are more versatile, but they do not take any particular advantage of the mathematical structure of the landscape. An example of derivatives-free methods is offered by Genetic Algorithms (GA), a metaheuristic approach inspired by the process of natural selection.

The first part of our work regards benchmarking a quantum algorithm designed to find the ground state of a given Hamiltonian: the Variational Quantum Eigensolver (VQE). This model is of most importance since VQE has many applications in fields such as physics of matter and quantum chemistry, in which finding the ground energy of a system is fundamental.

Another promising application of the quantum paradigm is Quantum Machine Learning (QML), that consists of the integration of Machine Learning (ML) techniques within the quantum framework. The purpose of such research field is to find new ways to speed-up classical ML models, and to design specific learning schemes. For instance, QML procedures are expected to provide advantages in comparison to the classical counterpart in data analysis. Let us consider a common ML model employed in this field: a binary classifier. It is possible to design a QML binary classifier based on the paradigm of VQAs, a Variational Quantum Classifier (VQC), that exploits the properties of both the variational quantum approach and the classical ML algorithm. We study the behaviour of minimizers on this specific model.

First of all, we evaluate the performances of the VQC on the scholastic problem of distinguishing handwritten digits, employing the MNIST [72] dataset. Moreover, another possible application of a classifier deals with particle identification in high-energy physics, to face this problem *jet tagging* was introduced. This procedure consists of identifying the original collision from the jets created and

measured. Given a set of experimental results, a VQC can be employed to distinguish jets coming from different particles collisions. In particular, we test two datasets for jet tagging that are the results of measurements performed at Large Hadron Collider (LHC): one consists of preprocessed data, while the other is made up by jet images realized with a calorimeter.

This work is organized as follows. In Chapter 1 we introduce the reader to the theoretical basis of Quantum Computing, describing the leading paradigms for computation. Moreover, we discuss the main results of quantum Complexity Theory, and we give a brief overview of the state-of-the-art hardware facilieties. Chapter 2 is dedicated to a detailed description of the structure of VQAs, with a specific focus of VQE. We also provide a full and rigorous portray of the most relevant minimizers. In Chapter 3 we present VQE benchmarks, then we identify which family of optimizers is better-suited for the given problem. Finally, Chapter 4 presents the results of benchmarks on the VQC implemented for binary classification. Thus, in the conclusions we interpret and explain the outcomes.

Chapter 1

Introduction to Quantum Computing

Classical computation has achieved remarkable goals in the last few decades: both technological and theoretical progresses have been exponential and brought nowadays to a completely digitalized world. This growth has been predicted in 1965 by Gordon Moore, who stated the the so-called *Moore's law* [1]. The law is an empirical prediction which asserts that, from that time on, the number of transistors in a dense integrated circuit would double every two years. Such exponential development has been observed for a few decades, anyway, Moore's law failure was also expected, since physical components cannot be reduced in size infinitely. Once the scale of phenomena becomes smaller and smaller only quantum mechanics provides a proper mathematical description, thus it must be exploited to pursue more powerful computational facilities.

The first quantum revolution, that mostly regards hardware and its manipulation, has already led to significant progresses. Indeed, many novel technological systems have been realized thanks to the study of quantum properties of matter, which are, for instance, transistors, lasers, GPS, and semi-conductor devices. Moreover, our capability to control microscopic systems has increased. The second revolution is characterized by the development of new technologies based on the intrinsic qualities of quantum mechanics instead. The framework of Quantum Computing (QC) aims to take an advantage over classical computation by exploiting exactly these theoretical properties: superposition, entanglement, and no-cloning principle. As a consequence, in order to understand QC and its role in the empowering of our computational capabilities, it is necessary to introduce the intrinsic quantum properties that allow us to rethink of information theory in a totally new way.

In this chapter we introduce the main features of QC, focusing on its theoretical basis. Moreover we give the reader an idea of quantum technology limits and prospectives both in terms of hardware implementation and applications.

1.1 Theoretical foundations

Let us consider a 2-dimensional Hilbert space, where the basis, usually refered as *computational basis*, consists of $\{|0\rangle, |1\rangle\}$. We associate to these quantum states two column vectors as follows:

$$|0\rangle \to \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle \to \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.1}$$

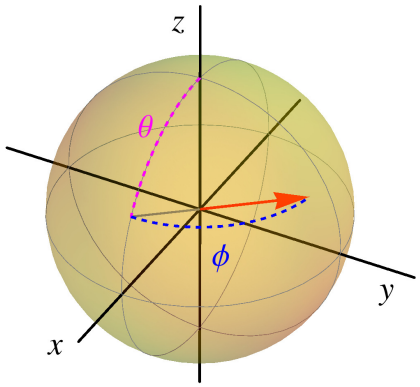


Figure 1.1: The Bloch Sphere allows us to represent geometrically a vector of a 2-dimensional Hilbert space in $\mathbb{R}^3$. The north and the south poles define orthogonal vectors, thus they represent the elements of the computational basis. A generic qubit is univocally defined by two angles θ and ϕ, according to the possible parameterization, see Equation (1.3).

1.1.1 Quantum bits: qubits

A state in a two-dimensional Hilbert space is the fundamental element of quantum computation: a *quantum bit*, more commonly called *qubit*, therefore a qubit is written as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \begin{pmatrix}\alpha\\ \beta\end{pmatrix}, \tag{1.2}$$

where $\alpha, \beta \in \mathbb{C}$, and the state is normalized, i.e. $|\alpha|^2 + |\beta|^2 = 1$.

As the name suggests, a qubit is the quantum counterpart of a classical bit. Nonetheless, a bit has a value of either 0 or 1, while a qubit has the properties of a quantum state, thus it carries much more information than a classical bit, since it is the superposition of two states.

The Bloch Sphere A useful geometrical representation of a qubit is by means of a point in the surface of the Bloch Sphere [2], named after the physicist Felix Bloch. Let us define two angles ϕ and θ, a useful parameterization of the coefficients in Equation (1.2) reads:

$$\alpha = \cos\frac{\theta}{2}, \quad \text{and} \quad \beta = e^{i\phi}\sin\frac{\theta}{2}, \quad \text{with} \quad \theta \in [0,\pi], \phi \in [0, 2\pi]. \tag{1.3}$$

These parameters univocally identify a point (x, y, z) on the surface of the unit sphere in $\mathbb{R}^3$, once we have defined the components of the 3-dimensional vector as:

$$\begin{pmatrix}x\\ y\\ z\end{pmatrix} = \begin{pmatrix}\sin\theta\cos\phi\\ \sin\theta\sin\phi\\ \cos\theta\end{pmatrix}. \tag{1.4}$$

In particular $|0\rangle$ is mapped in $(1, 0, 0)$, whereas $|1\rangle$ in $(0, 0, -1)$, in Figure 1.1 we show how every qubit is a vector on this surface identified by the coordinates reported in Equation (1.4).

Multiple qubits states

Once we have defined what a qubit is, we provide a formal representation of a multiple-qubit system. Indeed, applications need to exploit the whole information that can be carried by a multi-qubit state,

furthermore the most important technological investments are towards assembling devices with high number of qubits, see Section 1.4.

If we consider n qubits, the corresponding state lives in a 2^n-dimensional Hilbert space. All the 2^n possible combinations are:

$$|00\dots00\rangle, |00\dots01\rangle, |00\dots10\rangle, \dots |11\dots1\rangle, \tag{1.5}$$

that, for sake of simplicity, we represent as:

$$|0\rangle, |1\rangle, \dots |2^n-1\rangle. \tag{1.6}$$

Kets in Equation (1.6) are the basis of the Hilbert space to whose a n-qubit state belong. Indeed we can represent a generic state in this space as follows:

$$|\psi_n\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle_n \quad \text{with} \quad \sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1. \tag{1.7}$$

1.1.2 The no-cloning theorem

Now that we have given a formal definition of a quantum state in a two-levels system it is necessary to highlight a theoretical result that gives us a deeper insight on the quantum nature of such systems: the no-cloning theorem [3].
Let us give the statement of this theorem, in the specific case of a multiple-qubit state, and we will then produce its physical explanation.

Theorem (No-cloning theorem)**.** Consider a n-qubit pure state as follows:

$$|\psi\rangle = |\phi\rangle_1 \otimes |x\rangle_2 \otimes \dots \otimes |x\rangle_n \tag{1.8}$$

and suppose that it is possible to build a quantum cloning device, thus a unitary evolution, that transforms $|\psi\rangle$ in the state:

$$|\chi\rangle = |\phi\rangle_1 \otimes |\phi\rangle_2 \otimes \dots \otimes |\phi\rangle_n \tag{1.9}$$

The cloning device has copied the state $|\phi\rangle$ from the first subsystem into all the others. It is possible to prove [4] that such device does not exist.

The outcome is that it is not impossible to create an independent and identical copy of an arbitrary unknown quantum state. The consequences are extremely important in quantum mechanics applications, and especially in quantum computation. For instance this theorem prevent us from employing quantum entanglement, see Section 1.1.4, to transmit classical information, neither to convert a quantum state into a sequence of classical bits.

1.1.3 Density matrix

The information of a quantum system is encoded in its density matrix, this operator allows to typify the entangling properties of a system, thus it plays a central role in quantum computing, in which entanglement is fundamental to reach quantum advantage, see Section 1.4.4.

Definition (Density Matrix)**.** Let us consider a statistical ensemble $\{p_i, |\psi_i\rangle\}$ of quantum states $|\psi_i\rangle$, that occur with probability p_i. The density matrix is the linear operator obtained by the following expression [5]:

$$\hat{\rho} = \sum_i p_i |\psi_i\rangle \langle\psi_i|, \tag{1.10}$$

where $\hat{\rho}$ verifies the conditions:

$$\sum_i p_i = 1, \quad \mathrm{Tr}[\hat{\rho}] = 1. \tag{1.11}$$

An important property of the density matrix is that, once it is calculated, it makes easy to distinguish between pure states and mixtures. We refer to a pure state as a configuration which can be described by a single ket vector in a Hilbert space, while a mixed state is a statistical ensemble of pure states. Indeed $\hat{\rho}$ defines a pure state if and only if $\hat{\rho}^2 = \hat{\rho} \Rightarrow \mathrm{Tr}\left[\hat{\rho}^2\right] = 1$, whereas for a mixture we always obtain $\mathrm{Tr}\left[\hat{\rho}^2\right] < 1$.

Now that we have introduced the qubit and the notion of density matrix, we aim to quantify the content of information in a message. In the wake of Shannon's information entropy [6], a suitable quantity for this purpose is given by the Von Neumann entropy:

$$S[\hat{\rho}] = -\,\mathrm{Tr}[\hat{\rho}\log\rho] = -\sum_i \rho_i \log \rho_i. \tag{1.12}$$

The second equivalence holds in a basis in which $\hat{\rho}$ is diagonal, thus ρ_i are the eigenvalues of the density matrix. The entropy has the property: $S[\hat{\rho}] = 0$ if $\hat{\rho}$ represents a pure state, while it is maximum for a maximally mixed state. In the case of a two-levels system it can be seen as a measure of entanglement, see Section 1.1.4.

1.1.4 Entanglement

We have briefly recalled the definition of density matrix, thus we now introduce the main features of entanglement, focusing on the entanglement of a two-levels system.

Before expressing analytically the definition of an entangled state, let us give the reader a general idea of entanglement, which is one of the core features of quantum mechanics, and an essential element for computation.

The main characteristic of an entangled system deals with the correlation between different subsystems, hence measurements of physical properties performed on entangled particles can be perfectly correlated. Nevertheless, this behaviour highlights that for a two-levels system quantum mechanics does not admit an interpretation in terms of a local realistic theory.
Indeed, the principle of locality states the limits within which two systems may be casually correlated. According to the special theory of relativity, a given system at point A can cause a result in a system at point B in a time that is at most equal to the time needed by light to go from A to B [3]. This assumption has been a crucial point in the development of quantum mechanics as a complete and coherent theory, since it has aroused a debate between eminent scientists, with both theoretical [7] and experimental [8, 9, 10] contributions. We refer to these works for a deep insight of the problem.

The consequence of abandoning local realism is that, for entangled particles, a measurement on a single subsystem affects the entangled system as a whole.

Let us now define what an entanglement state is.

Definition (Entangled states)**.** Consider two Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$ and a two-qubit pure state in the space $\mathcal{H}$ such that $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$. An entangled state cannot be written as a tensor product of two single-qubit states, while a factorized state takes the form $|\psi_A\rangle\,|\phi_B\rangle$.

The four maximally entangled two-qubit states are called Bell's states, these form an orthonormal basis of the Hilbert space, and they take the following form:

$$\left|\psi^+\right\rangle = \frac{|0_A\rangle\,|0_B\rangle + |1_A\rangle\,|1_B\rangle}{\sqrt{2}}, \tag{1.13}$$

$$\left|\psi^-\right\rangle = \frac{|0_A\rangle\,|0_B\rangle - |1_A\rangle\,|1_B\rangle}{\sqrt{2}}, \tag{1.14}$$

$$\left|\phi^+\right\rangle = \frac{|1_A\rangle\,|0_B\rangle + |0_A\rangle\,|1_B\rangle}{\sqrt{2}}, \tag{1.15}$$

$$\left|\phi^-\right\rangle = \frac{|1_A\rangle\,|0_B\rangle - |0_A\rangle\,|1_B\rangle}{\sqrt{2}}. \tag{1.16}$$

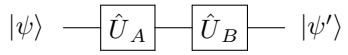


Figure 1.2: Example of a quantum circuit where two unitary gates are applied to a quantum state $|\psi\rangle$ to obtain a final state $|\psi'\rangle = \hat{U}_B \hat{U}_A |\psi\rangle$.

1.2 Quantum computation paradigms

Two main approaches to quantum computation may be considered: quantum annealing and circuit-based processing. In this section we provide the theoretical background to understand these two strategies.

1.2.1 Quantum circuits: gates

Since we are working on a 2-dimensional Hilbert space it is possible to represent quantum operators as matrices. A first approach to quantum computing is the so called gate-based quantum computing, in which operations are carried out applying unitary operators, the gates, to a quantum state.

In particular a quantum logic gate corresponds to a linear unitary transformation, which is associated to a unitary operator $\hat{\mathcal{U}}$. As a consequence the normalization of the qubit is preserved during the transformation, and the operation is reversible. Therefore, in this context, a circuit consists of a series of gates applied to a quantum state, which takes the form of Equation (1.7). Gates operate in sequence on a state, see for instance a simple example in Figure 1.2.

We now provide an overview of the most common quantum gates, that find wide application in quantum algorithms.

Single-qubit gates

In this specific case unitary transformations are represented by 2×2 matrices, where the coefficients are computed in the computational basis. The most common gates read [5]:

- **Pauli matrices gates**: X, Y and Z gates are represented by the three Pauli matrices σ_x, σ_y and σ_z:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.17}$$

  In particular, X gate is the quantum counterpart of a classical NOT gate, since it swaps $|1\rangle \to |0\rangle$ and $|0\rangle \to |1\rangle$, while other quantum Pauli gates do not have a classical counterpart;

- **Hadamard gate**: creates a superposition of states mapping the computational basis as follows:

$$|0\rangle \to |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad |1\rangle \to |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{1.18}$$

  where the matrix that performs such transformation reads:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \tag{1.19}$$

- **Phase shift gate**: acts adding a relative phase shift 2ϕ between the states of the computational basis, recalling that we can neglect the global phase factor and just consider the relative one, the matrix follows:
$$\Upsilon = \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\phi} \end{pmatrix}. \tag{1.20}$$
Fixing $\phi = \pi/8$ we obtain the so called T gate:
$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}, \tag{1.21}$$
which is the basis to build the S gate and to reconstruct σ_z:
$$S = T^2 = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix}, \qquad \sigma_z = T^4; \tag{1.22}$$

- **Rotations**: every state is a point on the surface of the Bloch Sphere, therefore a quantum unitary gate produces a rotation on this surface. Once we have defined rotation gates as follows:
$$R_X(\theta) = e^{-i\frac{\theta}{2}\sigma_x} = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{1.23}$$
$$R_Y(\theta) = e^{-i\frac{\theta}{2}\sigma_y} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \tag{1.24}$$
$$R_Z(\theta) = e^{-i\frac{\theta}{2}\sigma_z} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, \tag{1.25}$$
we can represent every unitary transformation as decomposed in rotations around the y and z axes:
$$U = R_z(\alpha)R_y(\beta)R_z(\gamma), \tag{1.26}$$
where α, β and γ are fixed angles.

Multiple-qubit gates

Two-qubit gates and, more in general, multiple-qubit gates play a central role in quantum computing, since they perform a connection between qubits, that may induce entanglement.

- **Conditional gates**: two-qubit gates that perform a generic one-qubit operator $\hat{\mathcal{U}}$ on a target if and only if the state of the first qubit, i.e. the control, is $|1\rangle$. Otherwise the target does not change. In general a controlled gate is written as:
$$c\mathcal{U} = \begin{pmatrix} id & 0 \\ 0 & \hat{\mathcal{U}} \end{pmatrix}. \tag{1.27}$$
An important gate of this kind is the CNOT, that performs an X gate on the target if control state is $|1\rangle$. Therefore the matrix is written as follows:
$$\text{CNOT} = \begin{pmatrix} 1 & 0 \\ 0 & \sigma_x \end{pmatrix}. \tag{1.28}$$
In Table 1.1 we report the truth table for this gate.

  To better understand the role played by this gate, in Figure 1.3 we show how it can be used to create Bell states, in particular the application of a CNOT allows us to create maximally entangled qubits;

Input		Output	
Control	Target	Control	Target
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Table 1.1: Table of truth of a CNOT gate: if the control qubit is $|0\rangle$ the target does not change, otherwise if the control is $|1\rangle$ the target qubit switches $|0\rangle \to |1\rangle$ or $|1\rangle \to |0\rangle$.

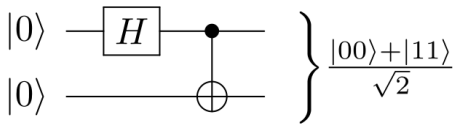


Figure 1.3: Circuit that creates a maximally entangled Bell's state. Combining a CNOT and a H it is possible to build a maximally entangled state with a simple circuit. The given circuit is made up by a H gate applied to the first qubit, then a CNOT gate where target is the second qubit and control is the first qubit. This example shows how CNOT is essential in quantum computing to exploit the features of entanglement.

- **Toffoli**: a three-qubit gate that can be represented as a 8×8 matrix. Toffoli is a double controlled CNOT gate, as Figure 1.4 shows. If both the control qubits are $|1\rangle$, the target switches, otherwise it remains unchanged. Such gate is universal for classical logic, which means that any classical circuit can be simulated with a quantum circuit, while it is not universal for quantum computing.

Universality

Of course there is potentially an infinite set of quantum gates, but it is possible to prove that exists a finite number of gates through which we can approximate the behaviour of any unitary operator [11]. This standard set of gates consists of H, T^2, CNOT and T, even thought T^2 is not strictly necessary to perform universal quantum computation. Indeed, it is introduced since it allows to approximate the circuit fault-tolerantly.

Measurements

In general measurements are a crucial point in quantum mechanics, both because of the collapse of the wave function and of the probabilistic outcome. In quantum computation when we measure a qubit we find a single state $|1\rangle$ or $|0\rangle$, therefore our measurement causes a loss of information. As a result a measurement gate is irreversible, contrary to the other quantum gates.

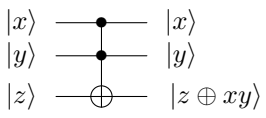


Figure 1.4: Quantum circuit that performs a Toffoli gate has two controlling qubits and a target. The target switches only if x and y are both $|1\rangle$.

1.2.2 Adiabatic Quantum Computation

Up to this point we have exploited a gate-based approach to quantum computing, however there is another approach, to whom we refer as Adiabatic Quantum Computation (AQC), based on adiabatic evolution. Here the problem is encoded into a given Hamiltonian and the solution is given by the ground state of the Hamiltonian itself.

Let us first introduce the adiabatic theorem [5], that is the core of this computational strategy.

Theorem (Adiabatic theorem)**.** Consider a Hamiltonian H_0 whose ground state $|\psi_0\rangle$ is well known, and thus easy to prepare. Assume that it is possible to associate a Hamiltonian H_P with a problem P, so that the ground state of this Hamiltonian $|\psi_p\rangle$ encodes the solution. Define then a slowly-varying time-dependent Hamiltonian $\tilde{H}(t)$, that respects the conditions:

$$\tilde{H}(0) = H_0 \qquad \text{and} \qquad \tilde{H}(T) = H_P, \tag{1.29}$$

where T is a parameter that represents the maximum time of evolution, and it controls the rate at which $H(t)$ changes.
Once we define $s = t/T$, another possible parameterization reads:

$$H(s = t/T) = \tilde{H}(t) \qquad \text{with} \quad s \in [0, 1]. \tag{1.30}$$

The corresponding instantaneous n-th excited eigenvalue is $E_n(s)$. Finally if $E_1(s) > E_0(s)\ \forall s \in [0, 1]$ and if T is large enough, then, during the evolution, the state of the system $|\psi(t)\rangle$ remains very close to the instantaneous ground state of $\tilde{H}(t)\ \forall t \in [0, T]$.

For instance, a common-used interpolating Hamiltonian is:

$$H(s) = (1 - s)H_0 + sH_P. \tag{1.31}$$

Computationally such evolution must be performed through discrete steps, thus we usually define an interval $[0, T]$ and a step size dt. In general the probability of reaching a good result depends on such gap: a smaller step size corresponds to a slower evolution, which means more accurate results. The idea behind adiabatic computation is now clear: a problem is encoded in a generally difficult to study Hamiltonian H_P, then we define a time-dependent lowly-varying Hamiltonian $\tilde{H}(t)$ that respect the hypothesis of Adiabatic theorem. The Hamiltonian H_0 is then adiabatically transformed in H_P, thus at the end the system ends up in the ground state of the problem Hamiltonian. For most Hamiltonians, finding the ground state is an NP-hard problem that classical computers cannot solve efficiently, therefore this approach allows us to find the minimum energy of such Hamiltonians if are able to convert, slowly enough, a well known Hamiltonian to the one that encodes the problem.

1.3 Complexity theory

Classical complexity theory focuses on classifying computational problems according to their resource usage, especially time and memory. We refer to a computational problem as a task that might be solved by a computer through an algorithm.

1.3.1 Quantum Turing Machine

We briefly introduce, without any pretence of being exhaustive, the idea of a Turing Machine (TM) and of its quantum counterpart, since these are useful tools to study the complexity of a computational problem without specifying any hardware device.

A TM [12] is the simplest example of a universal quantum computer. It consists in a theoretical machine, thus it works as a mathematical model, that manipulates symbols on a tape strip. The

operations are taken from a finite set of elementary instructions, even thought it is possible to replicate the logic associated with any algorithm. Thus the elements of a TM are: a potentially infinite *tape* divided into adjacent cells containing a *symbol*, a *head* that can read and write symbols and move the tape, a *state register* that stores the state of the TM, and a finite table of *instructions*. The original TM was deterministic, while there exists a classical machine that exploit a probabilistic approach, which is called Probabilistic Touring Machine (PTM). But both PTM and Deterministic Touring Machine (DTM) are based on classical physics.

The quantum generalization of a TM is called a Quantum Turing Machine (QTM). In this case the internal states become pure or mixed states, tape alphabet symbols are replaced by a Hilbert space, while operations are carried out by unitary transformations that map the Hilbert space into itself.

1.3.2 Classical and quantum classes of algorithms

Quantum computing may offer an effective alternative point of view on computation, therefore it is useful to clarify which are the limits of classical computation and which advantages are carried by the quantum approach. Indeed, several discoveries suggest that quantum and classical computers yield differing notions of computational hardness. In particular we recall the achievements of researchers such as Richard Feynman [13], David Deutsch [14] and Peter Shor [15].

Therefore, in order to better understand the role played by quantum algorithms in the landscape of hardness, let us define some complexity classes for classical algorithms [16]:

- **P – Polynomial time**: problems that can be solved in polynomial time, hence algorithms that have a time complexity that requires at most $\mathcal{O}(n^a)$, with $a \in \mathbb{N}$;
- **NP – Non-deterministic Polynomial time**: a problem is in NP if its solution can be guessed and verified in polynomial time, irrespective of the difficulty of obtaining a correct solution;
  - **NP-Complete**: a problem which is NP and all other NP problems are polynomial-time reducible to it;
  - **NP-Hard**: a problem such that all NP problems are polynomial-time reducible to it, but it is not necessary a NP problem itself;
- **PSPACE – Polynomial space**: a problem which sets a limit on memory resources. All these questions are solvable by some algorithms whose total space usage, in all instances, can be upper-bounded by a polynomial in the instance size;
- **BPP – Bounded-error probabilistic polynomial time**: a class of problems that focuses on the idea that randomized algorithms may be faster than the deterministic ones. This class contains P and, more in general, all the problems whose algorithms are allowed to make random choices and for which there exists a polynomial-time algorithm that solves every instance with a success probability of at least 2/3.

Note that it is actually an open problem, included in the seven Clay Mathematics Institute Millennium Problems [17], whether the complexity class P is equivalent to the class NP. In Figure 1.5 we report the Venn diagrams of P and NP relationships, both in the case of P=NP and P≠NP.

Les us now define some classes of quantum algorithms:

- **BQP – Bounded-error Quantum Polynomial time**: the quantum analogue for a BPP problem, this class includes P complexity;
- **EQP – Exact Quantum Polynomial time**: class of quantum algorithms that can solve a task for certain in a polynomial time;

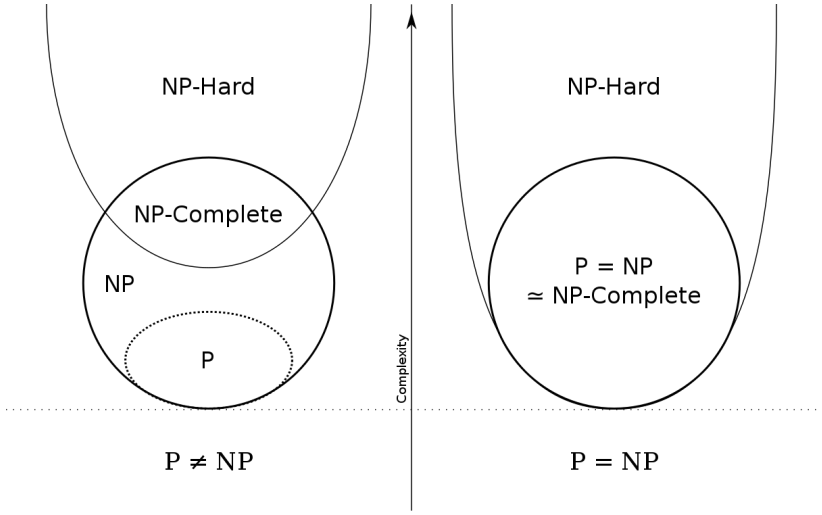


Figure 1.5: Classes of classical problems whether NP=P or NP≠P. If NP does not correspond to P it means that there are problems in NP that cannot be solved in polynomial time, but the result can be verified in polynomial time. This is, at the moment, the highly believed option, but there is not any proof yet.

- **QMA – Quantum Merlin-Arthur**: in this problems there is a prover (Merlin), who is computationally unbounded but untrustworthy, who sends a message to a verifier (Arthur), who uses a BQP machine. The algorithms of such class are the set of all decision problems that can be solved with a quantum algorithm such that, if the answer is "Yes", then the verifier can prove it with probability greater than 2/3, with a proof that can run in polynomial time on a quantum device. Otherwise, if the answer is "No", then the verifier accepts it in no more than 1/3 of the cases.

Note that BQP is low[1] for itself, since $BQP^{BQP} = BQP$, which means that BQP can call an undefined number of subroutines that perform a BQP algorithm without increasing its complexity. Thus polynomial time algorithms are closed under composition, i.e. composition of polynomial time routines is still polynomial.

1.3.3 Quantum vs Classical

Let us now discuss the relationship between BQP and classes of classical algorithms. BQP contains P, and BQP is contained in PSPACE. It is generally thought that classes such as BQP contain problems which are considered intractable on classical computers, but are thought to be tractable in QC.

The relationship between BQP and NP is not trivial at all, and it is not been proved yet, even thought it is conjectured that BQP solves hard problems outside of P. Of course this problem is still related to the unsolved question $P = NP$. It is proved that $NP \subsetneq BQP$, an example of problem that is NP, but not BQP, is given by Grover's algorithm [19]. This routine is designed to solve unstructured search problems: classical algorithms' complexity is $\mathcal{O}(N)$, while the quantum approach allows to reduce execution time to $\mathcal{O}(\sqrt{N})$, where N is the size of the input instance. Anyway, we are confident that quantum algorithms can solve problems that are unfeasible for a

[1] In computational complexity theory, a complexity class B is said to be *low* for a complexity class if and only if $A^B = A$. Which means that if a A calls a subroutine in B, complexity is still in A.

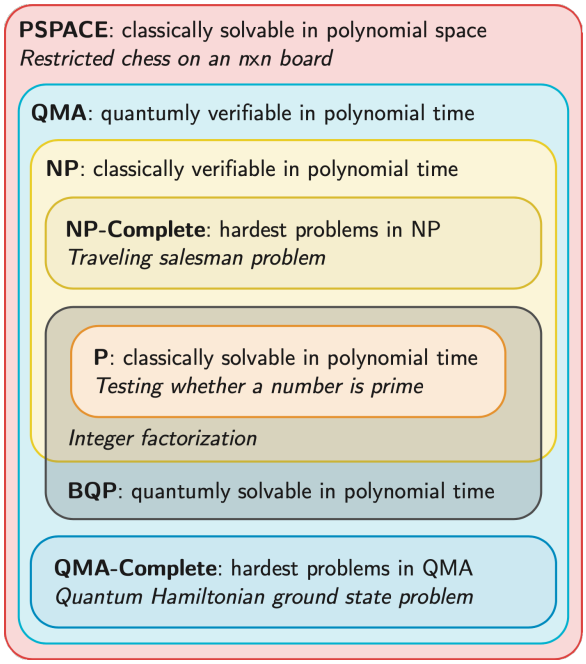


Figure 1.6: The widely believed containment relations for the most relevant classes of computational problems in quantum and classical algorithms. Note that these relations have not been mathematically proved for many of the complexity classes in the Figure, as we have already discussed for P and NP. Source [18].

classical computer to approach. There are indeed several suggestions to the thesis that NP hard problems and BQP may be addressed with a quantum approach, for instance see Shor's algorithm [15] for factorization and Harrow-Hassidim-Lloyd (HHL) algorithm [20].

These algorithms represent paradigmatic examples of problems in which quantum advantage over classical computation is observed. Here we present some of them, focusing on time complexity.

Grover's algorithm

A common task in computation concerns unstructured search problems, which consists in finding a targeted element in a given list of items, such problem is NP-hard. Grover proposed [19] an algorithm capable of producing a quadratic speed-up over the classical counterpart, it is indeed a perfect example of quantum advantage in terms of speed. Moreover, this method can be employed as a general trick or subroutine to obtain quadratic improvements for different algorithms.

Consider a given list of N items, the problem is to find a marked element is such list. Addressing this problem classically requires on average $N/2$ operations, N in the worst case, thus time complexity reads $\mathcal{O}(N)$. Grover's approach can face the same problem with a run time that is $\mathcal{O}(\sqrt{N})$, furthermore it is generical, since it does not depend on the structure of the elements.

Shor's algorithm

Integers factorization, that is the problem of finding the unique decomposition of an integer into a product of primes, is hard to approach computationally with a classical device. As a consequence, the speed-up guaranteed by Shor's algorithm [15] represents an important step in the understanding of the quantum approach capabilities. Consider an integer N, Shor's algorithm [15] solves the problem in a time that is polynomial in $\log N$, for completeness complexity reads:

$\mathcal{O}((\log N)^2(\log\log N)(\log\log\log N))$. Nevertheless, the most efficient classical algorithm for factorization, General Number Field Sieve, requires an execution time that is exponential in the instance size. This is probably the most dramatic example of how the paradigm of quantum computing changed our perception of which problems should be considered tractable. Indeed, integer factorization problems can be efficiently solved with Shor's method on a quantum computer, thus the complexity class is BQP.

Harrow-Hassidim-Lloyd (HHL) algorithm

A problem which has wide applications in almost all sciences concerns solving linear systems of equations, Harrow-Hassidim-Lloyd (HHL) algorithm [20] was designed to compute the result of a scalar measurement on the solution vector to a given linear system of equations.

Consider a linear system with N variables, represented by a sparse matrix, and with a low condition number[2], HHL shows a logarithmic scale in $\log N$. Classically, for a generic matrix, the most efficient algorithm to solve linear systems is Gaussian Elimination, which requires $\mathcal{O}(N^3)$. If the matrix is required to be sparse and positive semi-definite Conjugate Gradient method computes the solution in $\mathcal{O}(Nsk)$, and eventually when only a summary statistic of the solution vector is needed, the algorithm runs in $\mathcal{O}(N\sqrt{k})$. This last case is the one to be compared to quantum algorithm for linear systems of equations. The original HHL runs in $\mathcal{O}(k^2\log N/\epsilon)$ on sparse matrices, where $\epsilon > 0$ stands for the error, while an extension for dense matrices has been proposed [21] whose complexity is $\mathcal{O}(\sqrt{N}\log Nk^2)$.

1.3.4 The Church-Turing Thesis

Now that we have introduced all the necessary tools, we present the most important theoretical achievements in terms of computability. The first big result was developed by Alonzo Church and Alan Touring (CTT) [16], it states:

Theorem (CTT)**.** If an algorithm can be performed on any device, then there is an equivalent algorithm for a Universal Turing Machine (UTM) which performs exactly the same algorithm.

However this statement does not take into account the problem of time complexity, which is, as we have already discussed, a key factor in algorithms theory. The result of this deeper analysis leads to the Strong Church-Turing Thesis (SCTT):

Theorem (SCTT)**.** Any algorithmic process can be simulated *efficiently* using a UTM.

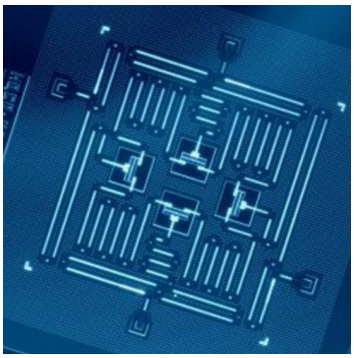
Nonetheless, counterexamples [22] to SCTT occur if we consider randomness. Hence UTM can be updated, in order to take into account probabilistic machines too, the outcome is the Extended Church–Turing thesis ECTT:

Theorem (ECTT)**.** Any algorithmic process can be simulated efficiently using a Probabilistic Turing Machine (PTM).

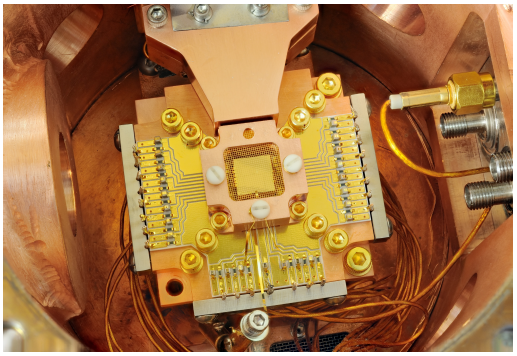
Let us now concentrate on the contribution of quantum computation: quantum devices are supposed to solve certain problems exponentially faster than a classical computer. This leads to the quantum version of the thesis, the Quantum Extended Church–Turing thesis QECTT:

Theorem (QECTT)**.** Any realistic physical computing device can be efficiently simulated by a fault-tolerant quantum computer.

[2]Informally, a condition number of a problem measures the sensitivity of the solution to small perturbations in the input data. In the specific case of a linear system it provides a bound on how inaccurate the solution will be after approximation.



(a) Superconducting device assembled by IBM



(b) Chip based on trapped ions technoology

Figure 1.7: Images of physical realizations of superconducting qubits and trapped ions. In (a) a recent superconducting device fabricated at IBM. Source [23]. In (b) a chip ion trap for quantum computing from 2011 at NIST.

Up to this point we offered an overview on the theoretical computational problem of finding algorithms that can perform a task in a reasonable time. Clearly, solving a computational problem requires not only a solid theory, but also hardware facilities. Indeed, in Section 1.4 we will cover such field.

1.4 Hardware

Before giving a brief overview of the main architectures of quantum devices it is useful to underline the role of QC in a typical workflow: we can think of it as a subroutine in a much larger computation that is performed on classical computers. For instance, we develop our quantum circuit protocols in high-level languages on a classical computer, these instructions can then control the quantum system.

1.4.1 Physical realization of qubits

Several realizations of QC are considered, each of which exploits different physical processes. We are going to give and idea of the main possibilities, even though a full description of these architectures goes beyond the purposes of this work. The most important aspect to keep in mind is that, at the state-of-art, small quantum computers can successfully be built, but scaling these up to computers that are large enough to yield useful computations presents two main difficulties: decoherence and noise.

The leading architectures for QC consist of superconducting qubits and trapped ions. Indeed, in the last few years we have achieved the result [24] of controlling trapped ions [25] or superconducting circuits [26], so that the error rate per gate for two-qubit gates is below the 0.1% level.

Trapped ions The trapped ions approach consists in ionizing atoms, that are later confined and suspended in free space using electromagnetic fields. Qubits are stored in stable electronic states of each ion. Operations are carried out by lasers, that induce coupling between qubits or entangle them. The measurement is performed by an additional laser, that couples only one of the states.

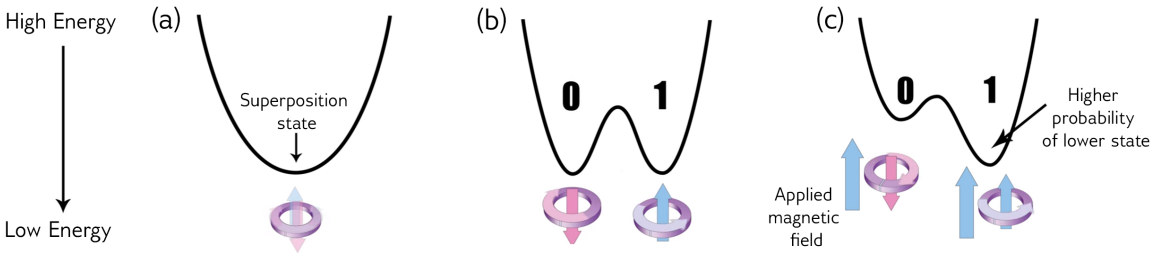


Figure 1.8: Energy diagram that represents the evolution from an unbiased superposition state to a biased state. In the former case (a) the plot presents a single energy well, while annealing goes on (b) superposition is lost and the two wells of the diagram refer to the two individuals states $|0\rangle$ and $|1\rangle$. In (c) a magnetic field induces an asymmetry in the system, therefore the probability of measuring $|1\rangle$ becomes higher.

Superconducting qubits Superconducting circuits are assembled on silicon or sapphire substrates, and they are designed in the radio-frequency spectrum. Operations are performed by sending specific microwave pulses for a controlled amounts of time into the physical qubit. The entire apparatus must be cooled below 100mK to operate. Moreover, the system is shielded from magnetic fields and other factors that may induce decoherence.
Research in superconducting quantum computing is conducted by companies such as Google [27], IBM [28], Rigetti [29], and Intel [30]. D-Wave Systems [31] are also investing in developing more than two thousands superconducting qubits, even thought these implement quantum annealing, see Section 1.4.2, instead of a gate-based QC model.

Another factor that should be taken into account, aside from the number of qubits and the error rate, is the time necessary to make a computation. In this sense it is important to highlight that superconducting circuits are about a thousand times faster than ion trap quantum processors.

For sake of completeness, we recall other QC technologies that are leading the sector, such as neutral atom devices, architectures that exploit nuclear magnetic resonance (NMR) or photonics, and silicon-based spin-qubits [16].

1.4.2 Quantum annealing

Quantum Annealing (QA) is a quantum approach to computation based on the adiabatic theorem. In QA the system starts in the ground state of a well known Hamiltonian and, as it anneals, it adiabatically moves to the ground state of the problem Hamiltonian.

In practice annealing is possible as long as there are not transitions from the ground state to the excited ones. Transitions may occur if the evolution is not slow enough, furthermore they may be induced by thermal fluctuations and interference from outer energy sources. Anyway perfect isolation is hard to achieve, thus experimentally nothing ensures us from jumping to an excited state. Even thought such physical limitations, the low-energy state found are still very useful, and QA can be thought as the real-world counterpart to AQC.

At present time D-Wave is massively investing in QA devices. The architecture is based on superconducting qubits, implemented as a circulating current, with a corresponding magnetic field.

For instance, consider the diagram in Figure 1.8. At the beginning, see (a), $|0\rangle$ and $|1\rangle$ are in a superposition and the energy has a single global minimum. While annealing goes on, the energy diagram becomes a double-well potential if it is not affected by external perturbation. This corresponds to a 50% probability to measure $|0\rangle$ and $|1\rangle$. The computational advantages turn out when *biases* and *coupling* are introduced. A bias is a quantity that controls an external magnetic field, this field modifies the depth of potential wells as shown in (c) (Figure 1.8): the deepest well

corresponds to the most probable state to measure. Furthermore coupling is realized by a coupler device, that induces entanglement between qubits. We can conclude that the core of QA consists of biases and couplings that define an energy landscape, thus D-Wave quantum computer finds the minimum energy of that landscape.

1.4.3 NISQ and Near-Term devices

The most challenging issues related to the physical realization of quantum devices deal with protecting qubits from environmental noises that may induce decoherence. To avoid such concerns, Quantum Error Correction (QEC) protocols are considered. However, most of the proposed algorithms require millions of physical qubits to be performed through QEC, which is still unfeasible experimentally. Indeed, existing quantum devices contain about 100 qubits, and an error-corrected quantum computer with millions of physical qubits may take decades.

At the state-of-art, we have realized quantum computers that are substantially imperfect, indeed we address them as "Noisy Intermediate-Scale Quantum" (NISQ). *Noisy* because we do not have enough qubits to spare for error correction, and so we will need to directly use the imperfect qubits at the physical layer. *Intermediate-Scale* because of their intermediate number of qubits.

Moreover, we aim to develop devices that can run quantum algorithms in order to reach the long-expected quantum advantage over classical computation. Quantum technologies are pursuing the objective of realizing such architectures; we then refer to the devices that will be developed in the next few years as "Near-Term" quantum computers. So the NISQ era corresponds to the period when only a few hundred of noisy qubits are available, while the Near-Term era involves any quantum computation performed in the next few years.

1.4.4 Quantum Supremacy

As we have discussed, quantum computers are introduced in order to solve problems that a classical device cannot address in any feasible amount of time or, more in general, to execute computational tasks with better performances in terms of accuracy and resources employed. We refer to such quantum advantage over classical devices as "Quantum Supremacy" [32, 24]. Therefore, our aim is to show that a programmable quantum device can solve a problem that is unfeasible for a classical device to approach. We can then conclude our discussion on Complexity Theory and quantum devices highlighting that quantum supremacy must be reached both on the hardware and on the theoretical point of view. Indeed, it involves both engineering tasks and theoretical task of finding a problem that can be solved by that quantum computer, and that has a superpolynomial speed-up over the best possible classical algorithm for that problem.

Quantum Supremacy is a goal feasible by Near-Term quantum devices, since it does not require high quality QEC computers, which is long-term objective. A remarkable example is offered by Google's claim in 2019 [33]. Their processor with 53 programmable superconducting qubits has performed an instance in about 200 seconds, while a classical supercomputer would take approximately 10.000 years to solve the same problem. Another promising result was achieved by Chao-Yang Lu and Jian-Wei Pan of the University of Science and Technology of China (USTC) in Shanghai, they performed a technique called Gaussian boson sampling with their quantum computer, named Jiŭzhāngby. The result [34] was 76 detected photons, that lead to a sampling rate about 10^{14}-fold faster than the capabilities of classical supercomputers.

Chapter 2

Optimization in Variational Quantum Algorithms

2.1 Variational Quantum Algorithms

Variational Quantum Algorithms (VQA) dominate gate-based quantum computation, since they have emerged as the leading strategy to obtain quantum advantage on NISQ devices. One of the main advantages of VQAs is their versatility: they provide a general framework that can be employed to solve a wide range of problems.

However there are some common elements that VQAs share. These are the *loss function*, a *parameterized quantum circuit* (PQC), that is the quantum circuit whose parameters are manipulated in the minimization process, a *measurement scheme*, which extracts the expectation values to evaluate the loss function. The final ingredient, the focus topic of this work, is a *classical optimizer*, that is necessary to find the minimum of the loss function. In Figure 2.1 we report a diagrammatic representation of a VQA workflow [18].

Loss function A crucial aspect of a VQA is encoding the problem into a loss function, which maps values of the trainable parameters of the PQC into a real number:

$$f : \boldsymbol{\theta} \to \mathbb{R}. \tag{2.1}$$

The specific form of the function depends on the specific application, thus it is extremely important to design a function that meets some requirements [35]. First of all, its minimum must correspond to the solution of the problem, furthermore, in terms of computability, it is also fundamental for f to be efficiently estimated by performing measurements on the circuit and to be trainable through classical optimization processes.

Parametrized quantum circuit Another important element of a VQA is the quantum circuit that prepares the state that best meets the objective, we will refer to this also as the *ansatz* of the problem. There are two approaches to define the structure of an ansatz: problem-inspired and problem-agnostic. The former, more commonly used, is employed when the PQC is designed to suit the specific problem. On the other hand, the latter is generic and it can be used when no relevant information is readily available.

Formally the ansatz is applied to an initial state $|\psi_0\rangle$. Once we have defined the unitary operation performed by the whole circuit as $U(\boldsymbol{\theta})$, the final state reads:

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})\,|\psi_0\rangle\,. \tag{2.2}$$

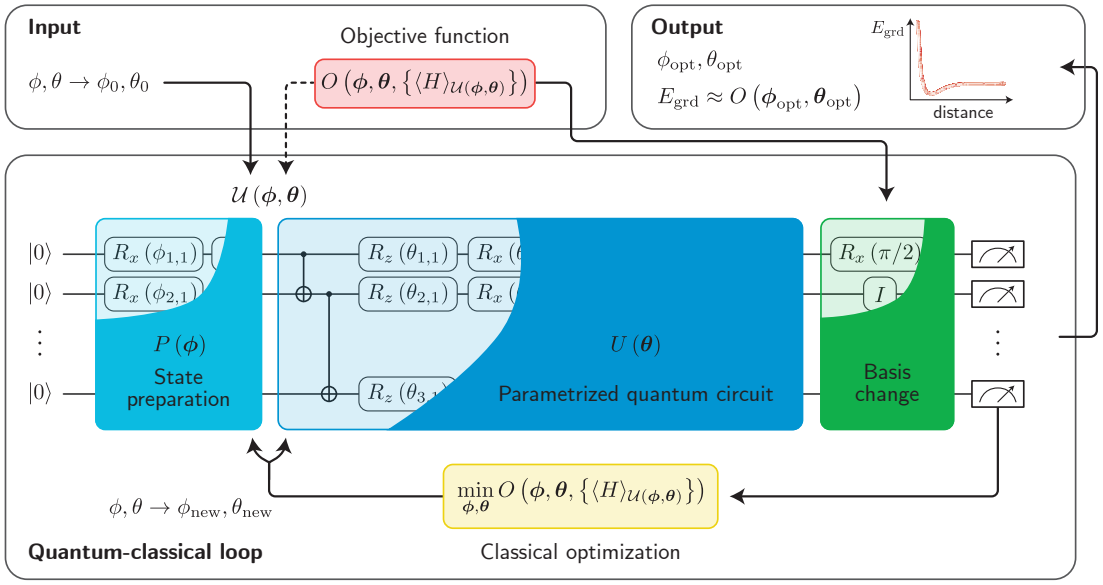


Figure 2.1: Structure of a VQA. The workflow can be seen as divided into four main components: i) the objective function O that encodes the problem to be solved; ii) the ansatz, whose parameters must be tuned; iii) the measurement scheme, which performs the basis changes and measurements needed to compute the objective; and iv) the classical optimization. The inputs of such procedure are: the circuit ansatz and the initial parameters ϕ_0 and θ_0. Outputs include the parameters that minimize the loss, and the corresponding loss function value.

The initial state depends on the specific problem, it may be chosen as the state $|0\rangle^{\otimes n}$, for a n-qubit circuit. The choice of the initial parameters for the circuit may depend on the task, but these are more commonly assigned randomly. In Figure 4.2 we report a schematic diagram of an ansatz: we can think of it as a sequence of layers of unitary transformations.

Measurement scheme To estimate the loss function, measurements on the system are required. This objective can be theoretically reached by testing whether we find $|0\rangle$ or $|1\rangle$ while measuring a qubit. In practice this is not straightforward. However, a detailed description of this process goes beyond the aims of our discussion, see [18].

Classical optimizer The optimization process plays a central role in this workflow, since its efficiency and reliability are fundamental for VQAs to be successful. Moreover, quantum algorithms encounter difficulties that are peculiar to the quantum approach, see for instance Section 4.4. This work focuses on such complications, in order to better circumscribe the problem and to find a proper strategy for optimizing quantum-based loss functions. In next chapters we are going to exploit such techniques, from gradient-based to heuristic algorithms.

2.1.1 Variational Quantum Eigensolver

One of the most exemplary and useful application of VQAs consists in estimating the ground state of a given system, once its Hamiltonian is known. In terms of hardness such problem is not expected to be solvable by quantum computers in a reasonable time [36]. To face such issue on NISQ devices Variational Quantum Eigensolver (VQE) is considered.

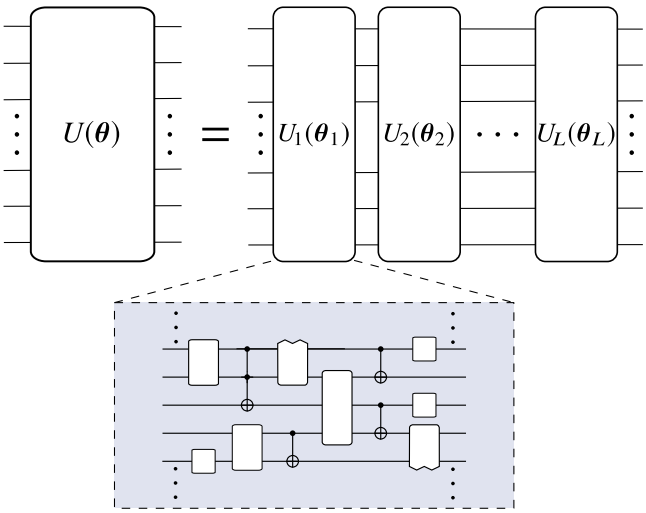


Figure 2.2: The unitary operator $U(\boldsymbol{\theta})$ can be seen as a sequence of L transformations $U_\ell(\boldsymbol{\theta_\ell})$. Moreover, every $U_\ell(\boldsymbol{\theta_\ell})$ consists of a sequence of parameterized and unparameterized gates. Source [35].

Consider a Hamiltonian H, whose ground state is unknown, and a PQC, whose corresponding unitary transformation is $U(\boldsymbol{\theta})$. VQE loss function is given by:

$$f(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})| H |\psi(\boldsymbol{\theta})\rangle = \langle\psi_0| U^\dagger(\boldsymbol{\theta}) H U(\boldsymbol{\theta}) |\psi_0\rangle , \tag{2.3}$$

where $|\psi(\boldsymbol{\theta})\rangle$ is the state obtained after the application of the ansatz on a certain initial state $|\psi_0\rangle$. Usually the Hamiltonian H is represented as a linear combination of products of Pauli matrices. Moreover, since practical physical systems are often described by sparse Hamiltonians, the evaluation can be carried out in a time that is usually $\mathcal{O}(n^\alpha)$.

By minimizing f we obtain a set of parameters $\boldsymbol{\theta}_{best}$, which allow us to evaluate the best approximation for the ground state of H as:

$$|\psi_g\rangle = |\psi(\boldsymbol{\theta}_{best})\rangle . \tag{2.4}$$

The corresponding eigenvalue is the approximated energy of the ground state, and it coincides with the objective evaluated in $\boldsymbol{\theta}_{best}$, thus the approximate eigenvalue $\tilde{E}_g$ reads:

$$\tilde{E}_g = f(\boldsymbol{\theta}_{best}) = \langle\psi(\boldsymbol{\theta}_{best})| H |\psi(\boldsymbol{\theta}_{best})\rangle . \tag{2.5}$$

Note that it is guaranteed that the approximation $\tilde{E}_g$ is an upper bound to the real value E_g, thus $E_g \leq \tilde{E}_g$.

Several extensions of VQE have been proposed to exploit different properties of such model, we focus on the so-called Adiabatically Assisted Variational Quantum Eigensolver (AAVQE)[37].

Adiabatically Assisted Variational Quantum Eigensolver

A key point in VQE is approaching the problem with a proper optimizer, since for certain Hamiltonians it is common to converge to local minima instead of the global optimum.

A possible solution to this issue is proposed in [37]. The idea is to combine the quantum annealing approach and a VQE model, in order to exploit the advantages of both strategies. Indeed, VQE does

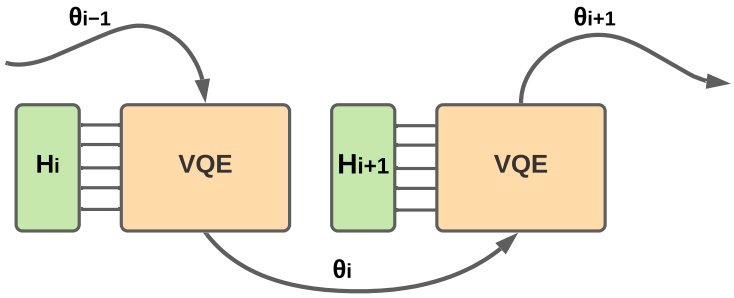


Figure 2.3: Diagrammatic flux of the AAVQE. At each iteration the input parameters of VQE are the ones obtained as results of the previous step. Moreover each variational algorithm finds the ground state of a different Hamiltonian, that is updated through a discrete adiabatic evolution.

not ensure us of finding the real minimum, while AQC algorithms always find their way to the solution, as long as the hypotheses of the Adiabatic Theorem are satisfied. We can take advantage of the structure of the variational approach, that employs parameterized ansatzes and classical optimizers, and combine it with the theoretical guarantees offered by AQC.

The algorithm works by applying VQE at every step of a discrete adiabatic evolution, see Section 1.2.2, from a well known Hamiltonian to the one of the problem. In other words we consider each discrete step as a new problem to be solved through the variational algorithm. At every iteration the Hamiltonian is updated with a rule that is the discretization of Equation (1.2.2), with step Δs, such that:

$$s_i = s_{i-1} + \Delta s, \qquad s_0 = 0. \tag{2.6}$$

Once the Hamiltonian is defined, a minimization process occurs, $\boldsymbol{\theta}_{best}$ are found, and then assigned to the ansatz for the next step. Indeed, every minimization returns the ground state energy and the circuit parameters of the corresponding Hamiltonian. Then the whole process can be thought as a training of the parameters, since the input parameters of each step are the results of the previous minimization. In Figure 2.3 we report a schematic workflow of AAVQE.

Now that we have extensively discussed VQA, with specific in-depth analysis of the VQE, we want to focus on the optimization problem. Indeed, as we have already introduced, minimization plays a central role in such class of algorithms, then our discussion will deal with finding the proper family of optimizers that best suits different variational quantum problems. In the next sections we will introduce the classes of optimizers that will be the topic of our benchmarks. In particular, in this context, our aim is to discuss such minimization methods theoretically, leaving a further discussion concerning the implementation for Chapter 3.

2.2 Optimizers

In this section we provide a detailed overview of the optimization algorithms that are going to be the topic of our discussion. A first fundamental distinction that concerns both performances and reliability of a minimizer is between gradient-based and gradient-free methods. Indeed, if derivatives are not to be computed, functions are allowed to exhibit less regularity, while if gradient or Hessian are needed the function must be at least differentiable. As a result, there are different approaches to minimization, that exploit different properties of a given function, and thus require different levels of regularity of the loss function.

2.2.1 Gradient-based

An important class of gradient-based algorithms is made up by the so-called Quasi-Newton methods, inspired by Newton's method. The latter assumes that the function can be locally approximated at the second-order around the minimum, thus it considers both gradient and Hessian matrix. Anyway, computing an Hessian is not straightforward in terms of time loss, to solve such issue Quasi-Newton methods are then introduced. Every Quasi-Newton method proposes a different approximation of the Hessian, that is not computed, but updated through successive gradient evaluations.

Consider a function f and its second-order approximation around a certain point $\boldsymbol{x}_k$:

$$f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T \tilde{H} \Delta\boldsymbol{x}, \tag{2.7}$$

where $\tilde{H}$ stands for an approximation of the Hessian matrix. The gradient of the approximated function reads:

$$\nabla f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) \approx \nabla f(\boldsymbol{x}_k) + \tilde{H}\Delta\boldsymbol{x}. \tag{2.8}$$

Then, to find a minimum, the condition $\nabla f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) = 0$ must be verified. This way we find the so-called *Newton step*, that is the updating condition on $\boldsymbol{x}_k$:

$$\Delta\boldsymbol{x} = -\tilde{H}^{-1}\nabla f(\boldsymbol{x}_k). \tag{2.9}$$

Up to this point the whole discussion is still valid for a generic Newton's method, Quasi-Newton algorithms require a proper approximation of the Hessian matrix instead. Once $\Delta\boldsymbol{x}$ is the one in Equation (2.9), the approximated matrix must satisfy the *secant equation*, that follows:

$$\nabla f(\boldsymbol{x}_k + \Delta\boldsymbol{x}) = \nabla f(\boldsymbol{x}_k) + \tilde{H}\Delta\boldsymbol{x}. \tag{2.10}$$

Note that in a 1-dimensional space solving Equation (2.10) for $\tilde{H}$ and applying the Newton's step with the updated value is equivalent to the secant method, while in more dimensions Quasi-Newton methods differ in the choice of the solution to the secant equation. Here we present some commonly used Quasi-Newton's methods, that will be benchmarked in Chapter 3.

Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)

A commonly-used Quasi-Newton method is the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [38]. To understand its behaviour we have to derive the specific update condition of the Hessian matrix for this method.
Eventually Equation (2.10) can be written in a different form, in order to highlight the update condition on $\tilde{H}$:

$$\tilde{H}_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k). \tag{2.11}$$

Let us now define two useful quantities, in order to make notation easier:

$$\begin{aligned} \boldsymbol{y}_k &= f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \\ \boldsymbol{s}_k &= \boldsymbol{x}_{k+1} - \boldsymbol{x}_k. \end{aligned} \tag{2.12}$$

The key point of BFGS concerns the fact that the Hessian matrix must be positive definite, thus the condition $\boldsymbol{s}_k^T \boldsymbol{y}_k > 0$ must be verified. Moreover, to provide a proper update condition for the Hessian it is required to enforce this condition explicitly, this is done introducing two rank-one matrices U_k and V_k, whose sum is a rank-two matrix. Then the Hessian update condition reads:

$$\tilde{H}_{k+1} = \tilde{H}_k + U_k + V_k \tag{2.13}$$

Imposing the secant equation and taking into account the symmetry properties of such matrices, we finally obtain the BFGS update condition:

$$\tilde{H}_{k+1} = \tilde{H}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\tilde{H}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \tilde{H}_k^T}{\boldsymbol{s}_k^T \tilde{H}_k \boldsymbol{s}_k}. \tag{2.14}$$

Once we have found the update condition for the Hessian matrix, we have all the ingredients needed to run the algorithm.

L-BFGS-B

In order to preserve memory resources during optimization, an extension of the classical BFGS was developed [39]: the Limited-memory BFGS (L-BFGS). Indeed, BFGS requires the storage in memory of n^2 approximated values of the inverse matrix, being n the number of variables of the given function. L-BFGS is designed to only need a linear amount of memory.

A further extension is well suited for bounded problems: L-BFGS-B [40]. This routine works identifying at each iteration fixed and free variables, and then calling L-BFGS as subroutine to optimize the free ones.

Truncated-Newton method

Hessian-free optimization, also referred as Truncated-Newton [41], is a method for constrained optimization, that suits problems involving non-linear functions with large numbers of variables. The basis of this approach is, once again, Newton's method second-order approximation, see Equation (2.7). Anyway, it differs from others Quasi-Newton algorithms, because it does not make any approximation to the Hessian. Let us now introduce the main ingredients of this method.

Consider a n-dimensional vector $\boldsymbol{u}$, and the Hessian H of a given loss function, then $H\boldsymbol{u}$ can be easily computed using finite differences at the cost of a single extra gradient evaluation, as follows:

$$H\boldsymbol{u} = \lim_{\epsilon \to 0} \frac{\nabla f(\boldsymbol{\theta} + \epsilon \boldsymbol{u}) - \nabla f(\boldsymbol{\theta})}{\epsilon}. \tag{2.15}$$

We also recall that there is a well known effective algorithm for optimization of quadratic objectives: the Conjugate-Gradient (CG) [38].

Truncated-Newton consists of an iterative application of a truncated CG to solve Newton's method. CG is truncated since in the worst case it requires n iterations to converge, thus waiting for it to completely converge is extremely time consuming. Computation of the Hessian matrix is based on Equation 2.15.

Note that no approximation of the Hessian is done, since $H\boldsymbol{u}$ can be computed accurately by stable algorithms. The only difference between the classical Newton's method and Hessian-free is that it performs an incomplete optimization, through CG, instead of inverting the whole matrix. Variations to such algorithm have been proposed, we highlight the Truncated-Newton for constrained optimization (TNC).

Sequential Least SQuares Programming optimizer

Sequential Quadratic Programming (SQP) is a family of methods for constrained nonlinear optimization. Its power is due to the fact that methods of this kind can handle any degree of non-linearity including non-linearity in the constraints. On the other hand the main disadvantage is that these algorithms incorporates several derivatives, which likely need to be analytically solvable.

Note that if the problem is unconstrained, then the method reduces to Newton's, otherwise it consists of a Quasi-Newton applied to a Lagrange function, that takes into account the loss function, and equality- and inequality-constraints.

In general a constrained nonlinear problem with equality- and inequality-constraints reads:

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \\ \text{subject to:}\quad &b(\boldsymbol{x}) \geq 0 \\ &c(\boldsymbol{x}) = 0 \end{aligned} \tag{2.16}$$

whose corresponding Lagrangian follows:

$$\mathcal{L}(\boldsymbol{x}, \alpha, \beta) = f(\boldsymbol{x}) - \alpha b(\boldsymbol{x}) - \beta c(\boldsymbol{x}), \tag{2.17}$$

where α and β are the Lagrange multipliers. Constrained optimization may be performed solving the system $\nabla\mathcal{L} = 0$, whose we refer to Karush-Kuhn-Tucker conditions (KKT):

$$\nabla\mathcal{L} = \begin{bmatrix} \partial\mathcal{L}/\partial\boldsymbol{x} \\ \partial\mathcal{L}/\partial\alpha \\ \partial\mathcal{L}/\partial\beta \end{bmatrix} = \begin{bmatrix} \nabla f(\boldsymbol{x}) - \alpha\nabla b(\boldsymbol{x}) - \beta\nabla c(\boldsymbol{x}) \\ -b(\boldsymbol{x}) \\ -c(\boldsymbol{x}) \end{bmatrix} = 0. \tag{2.18}$$

We now recall that optima of the loss function are also minima of the Lagrangian and vice versa, since as KKT are verified $\mathcal{L}(\boldsymbol{x}, \alpha, \beta) = f(\boldsymbol{x})$. The idea is then to apply Newton's method to optimize the Lagrangian, but the improvement direction $\boldsymbol{d}$ for Newton's is found in an indirect way, through a quadratic minimization subroutine that is solved using quadratic algorithms. Thus, at the $k-$th iteration, a basic SQP procedure defines a search direction $\boldsymbol{d}$ that is the solution of the quadratic programming sub-problem:

$$\begin{aligned} \min_d \Big\{ f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T\nabla^2_{xx}\mathcal{L}(\boldsymbol{x}_k, \alpha_i, \beta_i)\boldsymbol{d} \Big\} \\ \text{subject to:}\quad b(\boldsymbol{x}_k) + \nabla b(\boldsymbol{x}_k)^T\boldsymbol{d} \geq 0 \\ c(\boldsymbol{x}_k) + \nabla c(\boldsymbol{x}_k)^T\boldsymbol{d} = 0. \end{aligned} \tag{2.19}$$

A well-known algorithm of this family is the Sequential Least SQuares Programming (SLSQP) optimization [42].

Trust-constrained

Trust-constrained is a trust-region method for constrained minimization implemented in SciPy [43]. Classical line-search problems are based on first- or second-order approximation of the loss function, as in Equation 2.7. Such approximation is well-posed only if the step is small enough, condition that is not guaranteed to be satisfied for unbounded problems. Indeed, this defect will always occur for linear approximations, and for quadratic models where the Hessian is indefinite.

Trust-region methods solve such issue by introducing a trust-region constraint, thus they define a region around the current iterate within which they trust the model to be an adequate representation of the loss function. They also fix a step $\mathcal{S}$, whose direction changes whenever the size of the trust region is altered. For a given step $\mathcal{S}$, the condition reads:

$$\|\mathcal{S}\| \leq \Delta_k, \tag{2.20}$$

where Δ_k is a suitable scalar radius for the k-th step. Therefore, the problem is similar to a typical constrained model as SLSQP, with an additional requirement given by Equation 2.20. As a consequence the corresponding problem follows:

$$\begin{aligned} &\min_x f(x) \\ \text{subject to:}\quad &c^\ell \leq c(x) \leq c^u \\ &x^\ell \leq x \leq x^u. \end{aligned} \tag{2.21}$$

The implementation of this specific algorithm is based on [44] for equality-constraint problems and on [45] for problems with inequality constraints, thus it is well-suited for large scale problems.

Quasi-Newton's methods are not the only family of gradient-based optimizers, different techniques may be considered. Here we present two approaches: Simultaneous Perturbation Algorithm for Stochastic Approximation (SPSA), based on a gradient approximation, and Migrad, which implements second-derivatives approximations.

Simultaneous Perturbation Algorithm for Stochastic Approximation (SPSA)

Simultaneous Perturbation Algorithm for Stochastic Approximation (SPSA) [46, 47] algorithm is a gadient-based optimization technique well suited for problems in which it is difficult or impossible to directly obtain a gradient of the objective function with respect to the parameters. This method is then suggested when a closed-form solution to the problem is not available and where loss function may be contaminated with noise, that is the case of many loss functions in Near-Term VQAs.

The key idea of SPSA's approach is to provide an approximation of the gradient, relying on measurements of the loss function only. To be more specific, SPSA uses a stochastic approximation of the gradient, by simultaneously perturbing, with a specific vector $\boldsymbol{\Delta}^{(k)}$, all parameters in a random direction. A suggested choice for $\boldsymbol{\Delta}^{(k)}$ is to use a Bernoulli distribution with a probability $p = 0.5$ for each ± 1 outcome, note that uniform and normal distributions are not allowed.

Let us consider a given function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize, where n is the number of variables, thus the vector of parameters is $\boldsymbol{\theta} \in \mathbb{R}$. Given an initial point $\boldsymbol{\theta}^{(0)}$ and a *small* learning rate $\eta > 0$, the update condition of the parameters at the $k-$th iteration reads:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla f(\boldsymbol{\theta}^{(k)}). \tag{2.22}$$

Where, fixed a small expansion step size ϵ, the gradient is approximated as follows:

$$\nabla f(\boldsymbol{\theta}^{(k)}) \approx \frac{f(\boldsymbol{\theta}^{(k)} + \epsilon \boldsymbol{\Delta}^{(k)}) - f(\boldsymbol{\theta}^{(k)} - \epsilon \boldsymbol{\Delta}^{(k)})}{2\epsilon} \boldsymbol{\Delta}^{(k)}, \tag{2.23}$$

$\boldsymbol{\Delta}^{(k)}$ is the perturbative vector that may be chosen following the discussed criteria.

In addiction, at every iteration η and ϵ must be updated. Let us define γ, α and A, at the $k-$th step, parameters are calculated as follows:

$$\epsilon = \frac{\epsilon_0}{(k+1)^\alpha}, \tag{2.24}$$

$$\eta = \frac{\eta_0}{(k+1+A)^\gamma}, \tag{2.25}$$

where ϵ_0 and η_0 are the initial step and learning rate. An effective choice for the coefficients is $\gamma = 0.602$ and $\alpha = 0.101$. For a more detailed description of the parameters involved and their update conditions we refer to [47]; in general it is possible to provide reasonable initial parameters, but to perfectly suit the problem tuning is suggested.

The computational advantage in terms of time is clear: to approximate the gradient at each step only two function evaluations are required. The stop condition may be controlled studying how parameters change during minimization or through a brute-force approach with a given maximum number of iterations.

Migrad

Migrad [48] from iMinuit [49] library is an algorithm of almost exclusive usage in high-energy physics, indeed it is included in Cern's ROOT library [50].

The algorithm implemented is Fletcher's [51] variation of the original Davidon-Fletcher-Powell [52] [53], and it makes use of first derivatives, while it computes approximated derivatives, in order to achieve quadratic convergence near the optimum. Thus its main weakness is that it depends on the knowledge of first derivatives, and fails if they are computed inaccurately. As a consequence, it is extremely fast near a minimum, while it is slower if the function is ill behaved.

Let us now discuss more widely its mechanism. The algorithm starts from a fixed set of parameters $\boldsymbol{\theta}_0$, thus it computes a good approximation for the gradient evaluated in that point: $\nabla f(\boldsymbol{\theta}_0)$. Moreover, a covariance matrix $\boldsymbol{\sigma}_0$ is assigned, even though it can be a generic diagonal matrix, not necessary related to the specific instance. At every k-th iteration a linear search along the direction individuated by a vector $\boldsymbol{u}_k$ is performed, thus given:

$$\boldsymbol{u}_k = \boldsymbol{\theta}_{k-1} - \nabla f(\boldsymbol{\theta} k - 1)\boldsymbol{\sigma}_{k-1}, \tag{2.26}$$

at each step the method finds α that minimizes:

$$f\big(\boldsymbol{\theta}_{k-1} - \alpha \nabla f(\boldsymbol{\theta}_{k-1})\boldsymbol{\sigma}_{k-1}\big). \tag{2.27}$$

Such result leads to the updating rule:

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \alpha \nabla f(\boldsymbol{\theta}_{k-1})\boldsymbol{\sigma}_{k-1}. \tag{2.28}$$

The new gradient is $\nabla f(\boldsymbol{\theta}_k)$ and the covariance matrix is updated with a generic form:

$$\boldsymbol{\sigma}_k = \boldsymbol{\sigma}_{k-1} - G(\boldsymbol{\sigma}_{k-1}, \boldsymbol{\theta}_{k-1}, \boldsymbol{\theta}_k, \nabla f(\boldsymbol{\theta}_{k-1}), \nabla f(\boldsymbol{\theta}_k)), \tag{2.29}$$

where G is called updating function. G can take different forms, either the original Davidon [52] or Fletcher's dual formula [51], depending on Fletcher's "switching" criterion [51]. Stopping criterion is based on a measure of the estimated distance to the minimum, thus it reads:

$$\nabla f(\boldsymbol{\theta}_k)^T \boldsymbol{\sigma} \nabla f(\boldsymbol{\theta}_k) < 10^{-6}\epsilon, \tag{2.30}$$

where as default value it is set $\epsilon = 1$.

2.2.2 Traditional gradient-free approaches

Most gradient-based optimizers have problems dealing with noisy and discontinuous functions, that are common in variational quantum problems. To minimize these classes of functions other techniques, with less regularity requirements, are introduced. In this section we describe some commonly used algorithms.

Nelder-Mead algorithm

The Nelder-Mead (NM) method or Simplex Search algorithm, originally published in 1965 [54] is an heuristic search algorithm for multidimensional unconstrained optimization. This approach is derivative-free and it employs only function evaluations, thus it suits problems whose loss function has a derivative that is not well-defined in the whole space.

NM makes use of simplexes, where a simplex S in $\mathbb{R}^n$ is defined as the convex hull[1] of $n+1$ vertices $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n \in \mathbb{R}^n$. A simplex-based direct-search method begins with a set of $n+1$ points that are vertices of the working simplex S, and the corresponding set of function values at the

[1] A convex hull of a given shape is the smallest convex set that contains it.

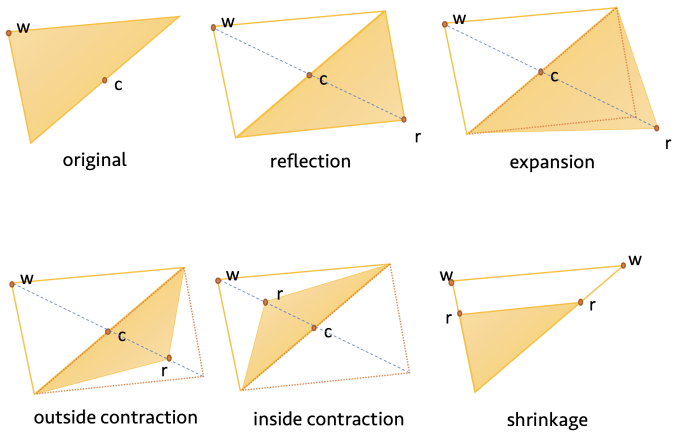


Figure 2.4: Operations performed on the simplex in a 2-dimensional application of Nelder-Mead's algorithm

vertices. Therefore, the main ingredients are the simplex and loss evaluations in the vertices of such simplex:

$$\begin{aligned} &\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n \in \mathbb{R}^n, \\ &f_0 = f(\boldsymbol{x}_0), \ldots, f_n = f(\boldsymbol{x}_n). \end{aligned} \tag{2.31}$$

In addition, to start the minimization the initial working simplex has to be nondegenerate, i.e. the points $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n$ must not lie in the same hyperplane.

At each step one or more test points are chosen and the corresponding function values are evaluated. The method then performs a sequence of transformations of the working simplex S, in order to substitute the old vertices with new ones, for which $f(x)$ has a lower value. The simplest approach implements a greedy expansion, and it is based on replacing the worst point with a point reflected through the centroid[2] of the remaining points. If in such region the function value is lower than the best current point, then we can try stretching the simplex out along this line exponentially. Otherwise, if this new evaluation does not provide better results in comparison to the previous one, then the algorithm shrinks the simplex towards a better point.

The stopping condition occurs when the simplex becomes *sufficiently* small or when all the values f_i are close enough. In Figure 2.4 we report a graphical representation of transformations applied to the simplex in a 2-dimensional NM.

Nelder-Mead algorithm usually requires one or two function evaluations for each iteration, this property makes it faster than other methods, at least in those cases when function evaluation is very expensive or time-consuming. Nonetheless convergence is not theoretically guaranteed for many families of functions, therefore the algorithm can take an enormous amount of iterations being faraway from the minimum.

Powell method

Powell algorithm [55] is a conjugate direction method [56] for bound-constrained minimization, it is well suited for functions that are not differentiable, since no derivatives are taken. It performs series of 1-dimensional minimizations along each vector of the a given set of directions, which is updated at each iteration of the main loop.

[2]The centroid or geometric center of a plane figure is the arithmetic mean position of all the points in the figure.

At the beginning of the routine three main ingredients are required: an initial point, a fixed number of real-valued inputs, and a set of initial search vectors $\{\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{n-1}\}$, where n is the number of variables. Minimization is performed by a bi-directional search along each search vector, the minima obtained during each bi-directional line read:

$$\left\{\boldsymbol{x}_0 + \alpha_0 \boldsymbol{s}_0,\ \boldsymbol{x}_0 + \alpha_0 \boldsymbol{s}_0 + \alpha_1 \boldsymbol{s}_1, \ldots,\ \boldsymbol{x}_0 + \sum_{i=0}^{n-1} \alpha_i \boldsymbol{s}_i \right\}, \tag{2.32}$$

where $\boldsymbol{x}_0$ is the starting point and α_i are the scalar coefficient on the $i-$th direction of the current set. Thus the starting point of the next iteration reads :

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \sum_{i=0}^{n-1} \alpha_i \boldsymbol{s}_i. \tag{2.33}$$

The set of directions is now updated: the linear superposition $\sum_{i=1}^{N} \alpha_i \boldsymbol{s}_i$ is added at the end of the list, while the search vector whose contribute in the linear combination is maximum is removed. The algorithm performs a series of iterations until it becomes stable on a certain result.

Constrained Optimization by Linear Approximation (COBYLA)

Constrained Optimization by Linear Approximation (COBYLA) [57, 58] is a gradient-free approach to minimization capable of handling nonlinear inequality constraints. The idea behind this algorithm is to define a trust region and then to evaluate the constrains and the objective function at the vertices of such region.

At each step, being n the number of variables of the function, a trust region with $n+1$ vertices is defined, and the corresponding objective function values are calculated at the vertices of the shape. Both a linear interpolation of such values and a linear interpolation of the inequality constraints are then performed. Iteration by iteration, the trust region is modified, in order to focus on the pae landscape where it is more likely to find the optimum, when this region is *small enough* convergence is supposed to be reached. Note that COBYLA implements an approach akin to the one of NM: they both map the space making use of a simplex with $n + 1$ vertices.

2.2.3 Genetic algorithms (GA)

Genetic Algorithms (GA), included in the Evolutionary Algorithms (EA), are a class of meta-heuristic models inspired by the natural selection process. They have wide applications in different fields, especially in optimization, as we are going to discuss in this section.

The main idea behind these procedures is analogue to Darwin's evolution theory: the algorithm starts with a set of solutions, represented by *chromosomes* or *individuals*, that is the so-called *population*. At every generation a new population, inspired by the previous one, is created. This is motivated by the hope that the offspring will be better than the predecessors. The criterion to define which is the best individual in a population is given by the *fitness function*: the lower the fitness the more suitable to survive the individual.

The first aspect to take into account when designing a GA is the encoding of chromosome. There are various encodings, that are well suited for different problems:

- **Binary**: every chromosome is a string of bits;
- **Permutation**: every chromosome is a string of numbers in a sequence;
- **Value**: every chromosome is a string of values, of any datatype;
- **Tree**: every chromosome is a tree of some objects.

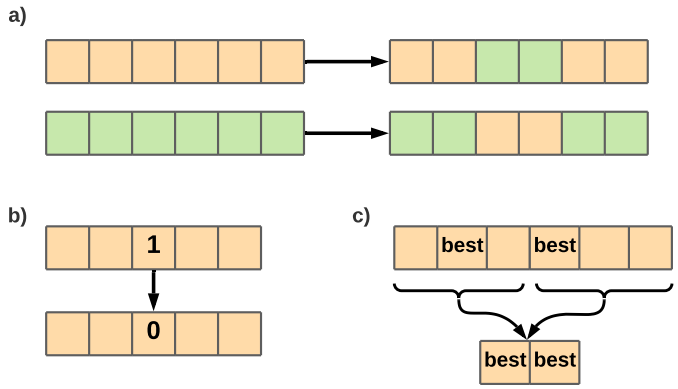


Figure 2.5: Schematic representation of operations performed on a chromosome. Two-points crossover (a): two points are picked randomly from the parent chromosomes. The bits in between the two points are swapped between the parent organisms. Bit Flip mutation (b): we select one or more random bits and flip them. This is used for binary encoded GAs. Tournament selection (c): several tournaments among a fixed number of individuals, three in this case, are runned. The winner of each tournament, according to its fitness, is selected for crossover.

Every generation evolution is performed by applying different operators: *selection*, *crossover* and *mutation*. Selection is, unsurprisingly, the process of selecting the best individuals in a population. Crossover selects genes from parent chromosomes and creates a new offspring. After crossover is performed, mutation take place. Mutation changes randomly the new offspring, in order to prevent falling all solutions of a population into a local optimum.

There are several parameters that are used to control GAs, the most important ones are *crossover probability* and *mutation probability*. The former defines how often will the crossover be performed, while the latter says how often parts of chromosomes will be mutated. Other significant parameters are used to define the whole process of evolution: the *population size* defines the wideness of the landscape of possible solutions, while the maximum *number of generations* limits the algorithm in terms of time and iterations.

Let us now focus on the application of GAs in optimization. There are plenty of possible configurations for a GA, we will now describe an example of genetic optimizer that will be the object of our benchmarks. We have to minimize a given objective function $f(\boldsymbol{\theta})$, depending on a list of parameters $\boldsymbol{\theta} \in \mathbb{R}^n$. Let this function be the fitness function for the evolutionary strategy and n be the number of variables. A possible choice is to represent the individuals, with a value encoding, as lists of n parameters, exactly those parameters that are needed to evaluate $f(\boldsymbol{\theta})$. This way the best individuals in a population are those whose fitness value is minimum, that is, in other words, those parameters that minimize the objective. Eventually, different crossover, mutation and selection approaches are available, in Figure 2.5 we report a graphical representation of examples of how such operators work.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [59] consists of a EA, that combine GAs approach with a more quantitative study of the covariance matrix, in order to guide the evolution. It does not require derivatives calculation, thus it is well suited for non regular functions.

At each generation evolutionary algorithms collect new candidate solutions according to a certain distribution in $\mathbb{R}^n$. The covariance matrix adaptation (CMA) is a method to update the covariance matrix of this distribution. It exploits two main ideas: maximum-likelihood principle and the record-

ing of two evolutionary paths of the distribution mean.

Maximum-likelihood principle [60] is employed as update criterion both for the covariance matrix and the mean of the distribution. The mean is updated in order to maximize, at a certain iteration, the likelihood of previously successful candidate solutions. Similiarly, the covariance matrix is incrementally adapted to the problem evolution, such that likelihood of the previously successful search steps is increased. The two paths are used one for the covariance matrix adaptation procedure, and the other to introduce another control on the step size, thus they provide informations about the correlation between consecutive steps. Therefore, these act in order to avoid premature convergence yet allowing fast convergence to an optimum. In Appendix A we provide the mathematical explanation of CMA-ES, for a more detailed description of the algorithm we refer to more specific works, for instance see [59].

CMA-ES is characterized by specific invariance properties, such as scale invariance, invariance under rotations, and invariance under order-preserving transformations of the objective function value. These features make this method similar to Nelder-Mead, while most optimizers usually exhibit only invariance under translations.

2.2.4 Hyperparameter Optimization Algorithms (HOA)

Hyperparameter optimization or tuning is a procedure that consists of optimizing hyperparameters in Machine Learning (ML) problems. We refer to *hyperparameters* as those parameters that are not updated during the learning and are used to configure the model.

Our approach to optimization is the one implemented in Optuna [61], a framework for ML tuning. Once an objective function is defined, the procedure is based on evaluating it a huge number of times with different hyperparameters. Every evaluation of the objective function is called *trial*, thus optimization performs multiple trials in order to find the best configuration of parameters.

The idea is to map the space, therefore we need algorithms that drive the search to the optimum: *samplers*, to sample hyperparameters, and *pruners*, to prune unpromising trials.
Samplers study the search space with the informations provided by parameters values and objective function evaluations. The aim is to identify the optimal landscape which leads to better objective values. In particular, two types of sampling strategies are considered: relative sampling and independent sampling. Relative sampling focuses on the relationship between parameters, therefore these algorithms evaluate multiple parameters simultaneously. Independent sampling determines a value of a single parameter, without considering its correlations with the others.
Pruners are used as stopping criterion, in order to prevent algorithms from being stuck in unpromising search regions.

We focus on a specific choice of such algorithms: a pruner-based on the median stopping rule, that we are addressing as *Median Pruner*, and a sampler using Tree-structured Parzen Estimator (TPE) algorithm.

It is straightforward to describe Median Pruner's behaviour: it prunes if the best intermediate result of a trial is worse than median of intermediate results of previous trials at the same step. Let us now describe TPE's approach to minimization.

Tree-structured Parzen Estimator (TPE)

Tree-structured Parzen Estimator (TPE) [62] is based on independent sampling. The main idea behind this approach is that on each trial, for each parameter, TPE selects the best objective values and fits the corresponding parameters with a Gaussian Mixture Mode[3] (GMM) $\ell(x)$. It then does the same for the remaining parameter values with another GMM $h(x)$, and finds the proper x that minimizes the ration $\ell(x)/h(x)$.

[3]A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.

Consider an objective function $f : \chi \to \mathbb{R}$ costly to evaluate, model-based algorithms approximate the loss, in order to make it cheaper to evaluate. The optimum of such surrogate becomes the proposal point for where the true loss function should be evaluated.
To give a more detailed explanation of this mechanism we employ the criterion of Expected Improvement (EI) [62]. Thus it is useful to give a formal definition of such quantity: EI is defined as the expectation value under some model M of $f : \chi \to \mathbb{R}^n$ that $f(\boldsymbol{x})$ will exceed negatively some threshold y^*, in formula we have:

$$EI_{y^*} \doteq \int_{-\infty}^{\infty} \max(y^* - y, 0)\, p_M(y|x)dy. \tag{2.34}$$

We refer to χ as the *configuration space*, that is a graph-structured generative process for drawing valid samples.

TPE provides $p(x|y)$ transforming the configuration space: distributions of the configuration prior are replaced with non-parametric densities. For instance, uniform distributions are transformed into truncated Gaussian mixtures, log-uniform into exponentiated truncated Gaussian mixture, and categorical into re-weighted categorical.

Once that a finite set of observations, i.e. $\{x_1, \ldots, x_k\}$ in the non-parametric densities space, is defined, these transformations represent a learning algorithm that can produce a variety of densities over the configuration space χ. Two of these densities are used to define $p(x|y)$, as follows:

$$p(x|y) = \begin{cases} \ell(x) & \text{if} \quad y < y^* \\ h(x) & \text{if} \quad y \geq y^* \end{cases} \tag{2.35}$$

where $\ell(x)$ is the density produced by all the observations $\{x_i\}$ such that the corresponding loss function values are $f(x_i) > y^*$, while $h(x)$ is defined with the remaining observations.
The optimization of EI reads:

$$EI_{y^*} = \int_{-\infty}^{y^*} (y^* - y)\, p(y|x)dy = \int_{-\infty}^{y^*} (y^* - y)\, \frac{p(x|y)p(y)}{p(x)}dy \tag{2.36}$$

where Bayes' theorem [63] has been used. By defining $\gamma = p(y < y^*)$ and $p(x) = \int_{\mathbb{R}} p(x|y)p(y)dy = \gamma\ell(x) + (1-\gamma)h(x)$ we find:

$$\int_{-\infty}^{y^*} (y^* - y)\, p(x|y)p(y)dy = \ell(x) \int_{-\infty}^{y^*} (y^* - y)\, p(y)dy = \gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy. \tag{2.37}$$

So EI shows a specific trend:

$$EI_{y^*} = \frac{\gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma\ell(x) + (1-\gamma)h(x)} \propto \left(\gamma + \frac{h(x)}{\ell(x)(1-\gamma)}\right)^{-1}. \tag{2.38}$$

Therefore, to maximize improvement, we need to minimize $\ell(x)/h(x)$, which means that we are pursuing to have points with high probability under $\ell(x)$. In practice at each iteration the algorithm selects the point x^* with the greatest EI.

Chapter 3

VQE benchmarks

In this chapter we test different minimization strategies, exploiting all the algorithms discussed in Chapter 2 and further approaches based on them. We study performances of optimizers both in terms of convergence and execution time, in order to find a proper trade-off between computational resources and accuracy of the results.

3.1 Setup employed in simulations

In this section we provide a detailed description of the model employed for the benchmarks, moreover we discuss both hardware and software facilities. This way we can look at the results being aware of the resources involved, and we can draw conclusions about algorithmic performances.

3.1.1 Hardware and software resources

Before approaching the simulations, it is necessary to introduce the reader to which facilities have been used. Therefore, all the results provided in next sections must be considered strictly depending on the framework and the devices employed.

Hardware

Since we are addressing the problem of finding a proper trade-off between time and accuracy, we will specify which devices have been used to run the simulations. Here we provide a brief description of the available resources:

- **Galileo:** University of Milan cluster for simulations, with a total amount of 920 cores, divided on different CPUs, for a total memory of 1150 GB; every node has a maximum of 48 cores. The scheduling mechanism and the high number of devices involved make this a stable machine. See `https://lcm.mi.infn.it/farm/la-struttura/` for a detailed explanation of the hardware resources and of the software used to manage jobs;
- **Montblanc:** server with 64 threads and 125GB of memory. Faster than Galielo, but less stable, since execution time is more likely to be affected by simultaneous excecutions;
- **Local:** 8 threads machine (2.3 GHz Quad-Core Intel Core i5), with 8GB of memory.

All simulations have been addressed with the most proper resource, always maintaining an internal coherence. The hardware will be always specified in order to provide the reader with a clear vision of the results.

Framework for simulation: Qibo

To perform classical simulations of quantum systems we employ Qibo [64], a framework that specifically addresses this task. Qibo provides an API for quantum circuit design, AQC and a clean design to implement classical-quantum hybrid algorithms, a key feature is that it exhibits high-performances thanks to hardware acceleration tools.

Qibo already implements quantum models, as the VQE, Hamiltonian objects, and provides method to evaluate eigenvalues and eigenvectors of a given system. Therefore, it makes easier to simulate a VQE on a well known problem and to compare the results obtained from the algorithm with the theoretical value.

3.1.2 Target model

We aim to benchmark accuracy and execution time of VQE with a finite-depth ansatz as proposed in [65]. Here we propose a full description of all the main ingredients employed in the procedure.

Ansatz

Let us consider a special class of quantum circuits, made up by several layers of unitary transformations, this approach allows us to observe if there is a relationship between performances and the number of layers. Since more layers correspond to a higher number of parameters, we would intuitively expect accuracy to be directly correlated with the depth of a circuit. We will thoroughly discuss that this intuition is partially true, since circuits manifest a finite flexibility: after a certain number of parameters, adding new layers does not affect the convergence.

We build the circuit using only a finite number of gates, single-qubit rotations and two-qubit controlled operators. Therefore, we are approaching the problem of finding the state that best estimates the ground state, assembling a finite number of elementary gates. This approach, even thought it may be not straightforward to implement, is warranted to be effective by a theoretical result: Solovay-Kitaev theorem. Such theorem guarantees that an arbitrary unitary transformation acting on n qubits can be approximated with precision ϵ by using at most $\Theta(\log^c(1/\epsilon))$ elementary gates chosen appropriately from a universal set of quantum gates closed under inversion [65]. However it does not give any suggestion concerning how to build such transformation, thus other considerations must be taken into account do design a proper ansatz.

We follow the approach proposed in [65], inspired by perturbation theory. In Figure 3.1 we report the architecture of the ansatz, that is made up by single-qubit rotations $R_Y(\boldsymbol{\theta})$ and control-Z gates (CZ) that act on two contiguous qubits. CZ are particularly important elements since they induce entanglement between the pair of qubits they act on.

Once the ansatz is choosen, let us focus on how many parameters are involved in the minimization process. Such property is a key point in our discussion since it affects the flexibility of a circuit, and consequentially the performances of VQE. Consider a circuit with N_{Lay} layers and N_Q qubits, the total amount of parameters N_{par} is given by:

$$N_{par} = 2\,N_{Lay}\,N_Q + N_Q. \tag{3.1}$$

Therefore every new layer introduces $2\,N_Q$ new parameters to the circuit. In our simulations initial parameters are always set as a vector of pseudo-random values uniformly distributed in $[0, 2\pi]$, since they must represent rotations. All the results discussed in Section 3.2 refer to numbers generated randomly with seed 0, in Section 3.4 we study how the choice of a specific seed affects results.

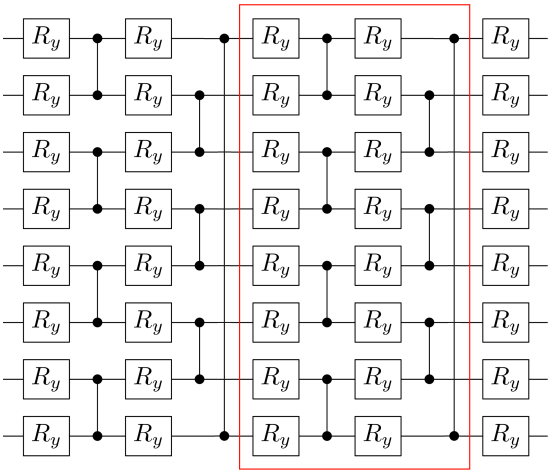


Figure 3.1: Parameterized quantum circuit used as variational ansatz for VQE, in the case of a 8-qubit system. Each layer is represented by gates within the red box. The whole circuit is only composed by CZ gates between adjacent qubits and rotations.

Hamiltonian

Let us consider the Heisenberg XXZ chain, that is a quantum statistical model employed to study critical points and phase transitions of magnetic systems. Its Hamiltonian reads:

$$H_{XXZ} = \sum_i \left(\sigma_x^i \sigma_x^{i+1} + \sigma_y^i \sigma_y^{i+1} + \delta \sigma_z^i \sigma_z^{i+1}\right), \tag{3.2}$$

where δ stands for the spin anisotropy, in our simulations $\delta = 0.5$. A discussion over the physical meaning of this system is not the purpose of this work, we choose such paradigmatic model because it is exactly solvable, in order to have a precise target which we can compare our results with.

3.1.3 Procedure

We still have to describe the procedure implemented to benchmark different algorithms, and the metrics defined to evaluate their performances. First of all we measure the execution time t, necessary both to create the model in Qibo and to run the simulation, thus to perform the whole minimization subroutine of VQE.
As a measure of accuracy α we introduce the logarithmic scale and define:

$$\alpha \doteq \log_{10}(1/\epsilon) \qquad \text{with} \qquad \epsilon = \left|\tilde{E}_g - E_g\right|, \tag{3.3}$$

where $\tilde{E}_g$ is the ground state obtained by VQE and E_g is the expected value. This way the displacement ϵ is:

$$\epsilon = 10^{-\alpha}. \tag{3.4}$$

Thus, accuracy measures how many orders of magnitude the result is distant from the real ground state.

Eventually, we repeat the simulations for different number of qubits and different number of layers. Tests are done on systems with less than 15 qubits, so we perform a single-thread execution, because in such cases the parallelization overhead may decrease performance.

3.2 Benchmark of different algorithms on a classical VQE

In this section we present benchmarks of all the algorithms considered in Section 2.2 with a classical VQE approach. Code available at `https://github.com/nicolezattarin/VQE-minimization-strategies`.

3.2.1 Gradient-based

To benchmark gradient-based algorithms on the model discussed in Section 3.1.2 we refer both to Scipy's minimizers [1][43], and to iMinuit library [49]. The SPSA tested is implemented by us instead. To be more precise BFGS, L-BFGS, SLSQP, Trust-constr, and TNC are supported by Scipy, while migrad is offered in the Python interface for the Minuit2 C++ library, maintained by CERN's ROOT team.

SPSA implementation The main optimization process is theoretically described in Section 2.2.1, here we discuss the implementation, especially focusing on the issues related to initial parameters tuning. Implementation of SPSA follows the steps proposed in [47], to adapt the algorithm to the specific problem two possible approaches may be chosen: calibration of the learning rate and tuning.

We implement the possibility to calibrate the starting learning rate η, similarly to the approach recommended in [66], in order to make the algorithm more flexible and capable of adapting to different loss functions.

Once we have defined the maximum number of iterations MaxIter for the whole process of minimization, calibration is obtained with N_{cal} repetitions:

$$N_{cal} = \min(25, \max(1, \text{MaxIter} \,//\, 5)). \tag{3.5}$$

At every iteration of calibration a perturbative vector $\boldsymbol{\Delta}^{(j)}$ is fixed, as discussed in Section 2.2.1, and the following quantity, let us refer to it as DeltaLoss, is computed:

$$\text{DeltaLoss}^{(j)} = \frac{\left| f(\boldsymbol{\theta}^{(0)} + \epsilon\boldsymbol{\Delta}^{(j)}) - f(\boldsymbol{\theta}^{(0)} - \epsilon\boldsymbol{\Delta}^{(j)}) \right|}{N_{cal}}. \tag{3.6}$$

Thus the sum of all the computed values of $\text{DeltaLoss}^{(j)}$ is used to evaluate a proper correction to η_0 before minimization starts:

$$\eta = 2\epsilon(A+1)\frac{\eta_0}{\sum_j \text{DeltaLoss}^{(j)}}. \tag{3.7}$$

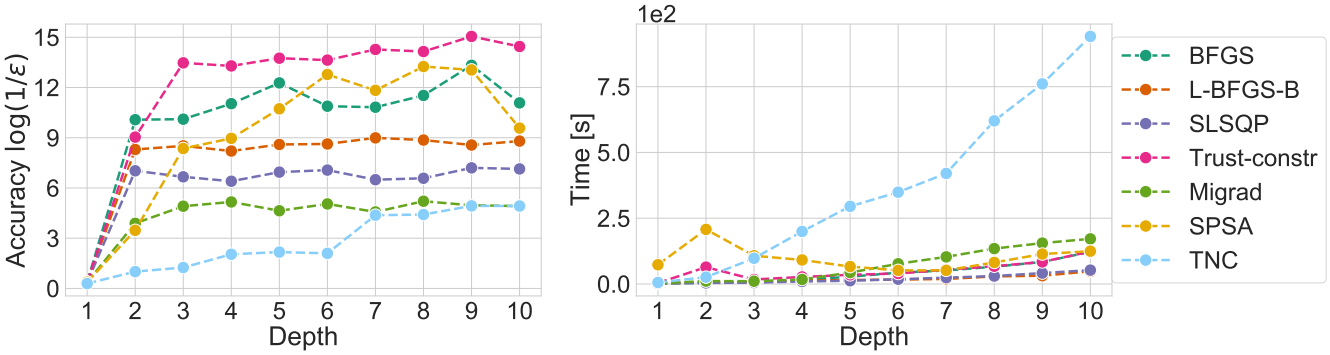
Eventually one may adapt η_0 and ϵ_0 to the specific problem with hyperparameters optimization. We tune such parameters with Optuna [61] in a proper interval, drawing inspiration by default values fixed in Qiskit algorithm [66]. Thus we set $\eta_0 = 0.4$ and $\epsilon_0 = 0.1$ as default values, since this choice represents a proper trade-off between simulations with different number of qubits, and we benchmark SPSA with the most suitable pair of parameters for every specific run.

Let us now focus on the stopping criterion of the whole SPSA run: iterations go on until the maximum number of iteration MaxIter is reached or until parameters do not change significantly. Stop condition reads:

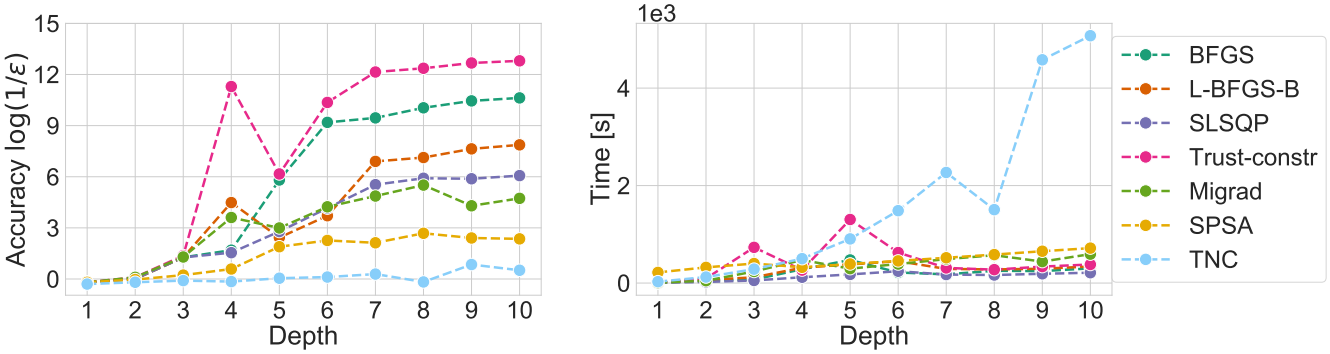
$$\frac{\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}\|}{\|\boldsymbol{\theta}_{i-1}\|} \leq \text{precision} \tag{3.8}$$

where precision is a fixed value, in our simulation it is set to 10^{-10}.

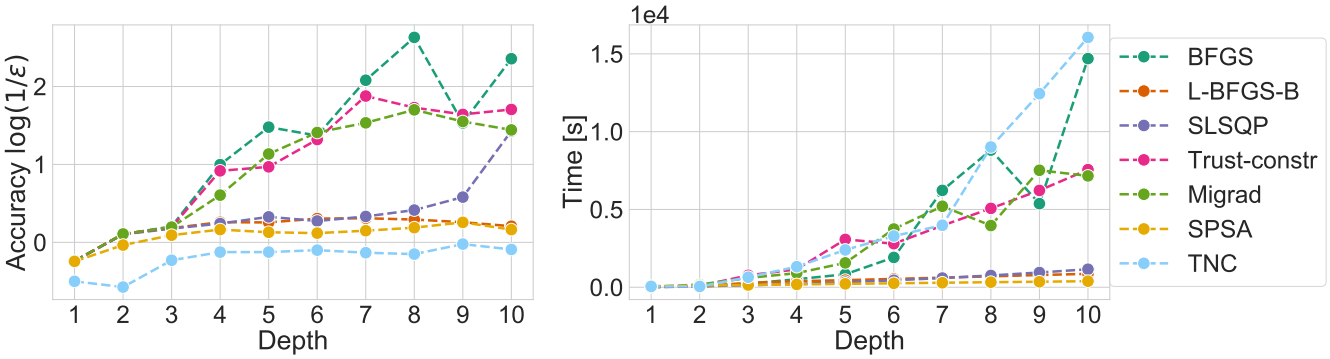
[1] See documentation at `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize` .



(a) Simulation performed with a 4-qubit system. Performances, in terms of accuracy, increase as the ansatz become deeper. However, after approximately the fourth layer accuracy presents a plateau, as a sign that the circuit is not flexible enough. Time increases, with a few exceptions, with the number of layers, since more parameters must be optimized. Note that TNC needs a significant amount of time, while it does not ensure satisfying performances.



(b) Simulation performed with a 6-qubit system. Accuracy increases as the circuit becomes deeper until the 7th or 8th layer. Time generally increases, but a significant exception to this trend is offered by Trust-constr, that exhibits an irregular behaviour. Note that TNC needs a significant amount of time, while it does not ensure satisfying performances.



(c) Simulation performed with a 8-qubit system. Many algorithms, as SLSQP, SPSA, and L-BFGS-B, exhibit an accuracy that is a little more than zero for most of values of depth. In all the cases accuracy is less than three, leading to performances that are far less satisfying than in 4- or 6-qubit systems.

Figure 3.2: Accuracy and execution time for different number of layers, for all the gradient-based algorithms taken into account. We clearly see that accuracy decreases with the number of qubit, while time generally increases both with the number of qubits and depth. Simulations are run on Galileo, which is the most stable hardware resource available, single thread execution with one CPU per task is performed.

In Figure 3.2 we show both accuracy and execution time for different number of layers, i.e. depth of the circuit, for all the gradient-based algorithms taken into account.
For a 4-qubit system, Figure 3.3a, performances increase as the ansatz becomes deeper. Nevertheless, layers added after the fourth seem not to be effective: we would expect accuracy to increase indefinitely, but it presents a sort of plateau. This observation is a sign that the circuit is not flexible enough, thus even increasing the number of parameters and entangling gates, minimizers cannot converge more precisely to the optimum.
A similar behaviour is observed for a 6-qubit system, Figure 3.3b, and a 8-qubit ansatz, Figure 3.3c. However, in these cases a higher number of layers is needed to saturate the flexibility of the circuit.

Comparing the plots provided in Figure 3.2 we clearly see that, as the number of qubits increases, accuracy decreases, while the required execution time becomes higher. In order to have an idea of the orders of magnitude involved, consider Trust-constr applied on a 4-qubit circuit: accuracy reaches approximately 15, thus, recalling Equation (3.4), it means that the ground state eigenvalue found through this algorithm is about 10^{-15} distant from the exact value. On the other hand, consider as example SPSA and L-BFGS-B acting on a 8-qubit system, if accuracy tends to zero, then the gap $\epsilon \to 1$. In general a 8-qubit system reaches approximately a maximum accuracy of 2.6, corresponding to a displacement from the expected value that occurs at the third decimal place.
It is now clear how accuracy is affected by the size of the circuit in terms of qubits: a 8-qubit ansatz provides a result that is, in the best case, equal to the expected one up to the second decimal place, while a 4-qubit ansatz up to the 14th.

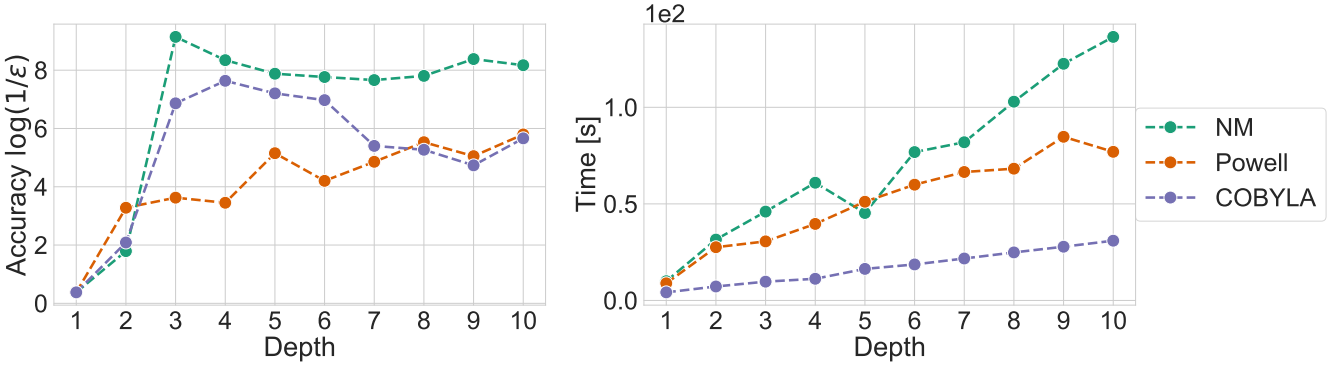
3.2.2 Gradient-free

Let us now focus on gradient-free algorithms, in this case we employ methods in Scipy's minimizers [43], that implements COBYLA, Nelder-Mead, and Powell.

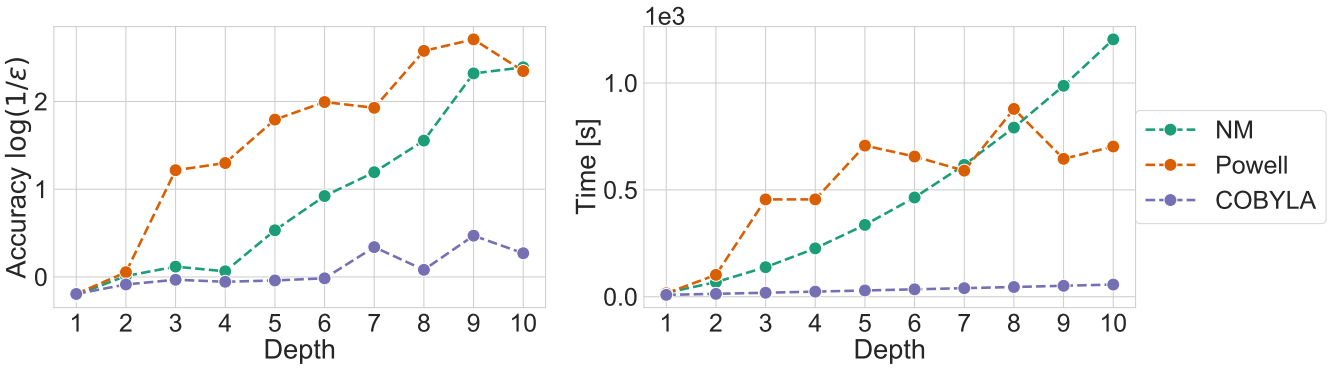
We recall that Nelder-Mead and COBYLA use a geometrical approach to minimization, as they take advantage of the properties of a simplex. Powell is a conjugate direction method instead.

In Figure 3.3 we report results for gradient-free optimizers. Comparing plots, the patterns observed are similar to what we have already obtained regarding gradient-based methods in Figure 3.2. To be more precise, in a 4-qubit system, Figure 3.3a, accuracy increases as layers are added to the circuit. Nevertheless, for what concerns Nelder-Mead, after a certain number of layers the growth approximately stops and accuracy oscillates around a fixed value, 8 in this case. Focusing on Powell, its accuracy does not saturate, it gradually increases instead; to conclude COBYLA exhibits an irregular behaviour.
In Figure 3.3b, we provide results for a 6-qubit circuit. In this case accuracy gradually, and irregularly, increases with the number of layers for what concerns Powell and Nelder-Mead. The performances of COBYLA are extremely poor if compared with other algorithms.
Even worse performances are shown in a 8-qubit ansatz, Figure 3.3c. Indeed, COBYLA's accuracy is always negative, and it decreases to -1. Note that a negative value in this metric corresponds to $\epsilon \geq 1$, thus the error is not negligible at all, and the result cannot be considered a proper approximation of the real value.

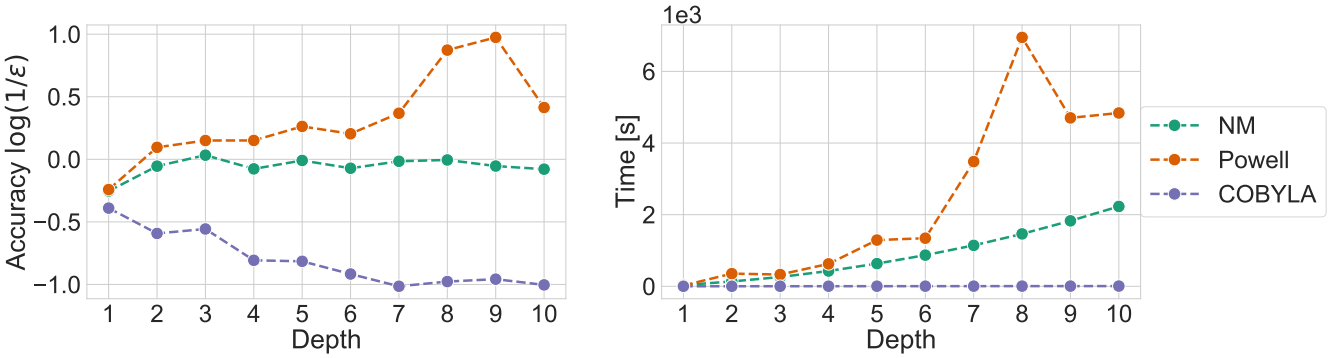
We conclude that derivative-free algorithms show less satisfying performances, in terms of accuracy, than the derivatives-based. This result suggests that the loss function is regular enough to be optimizerd by gradient-based methods. Indeed, the main advantage in approaching a minimization task with derivative-free methods is that such algorithms are suitable for a larger class of functions. If the given function already has the regularity required, this versatility is not exploited.



(a) Simulation performed with a 4-qubit system. Accuracy increases as layers are added to the circuit. Nevertheless, for Nelder-Mead, after a certain number of layers the growth approximately stops and accuracy oscillates around a fixed value. Powell's accuracy does not saturate in 10 layers, thus its grows is slow but continuous. COBYLA shows a more irregular behaviour, since it increases until a fixed number of layers and then oscillates while decreasing. Time scales with the number of layers.

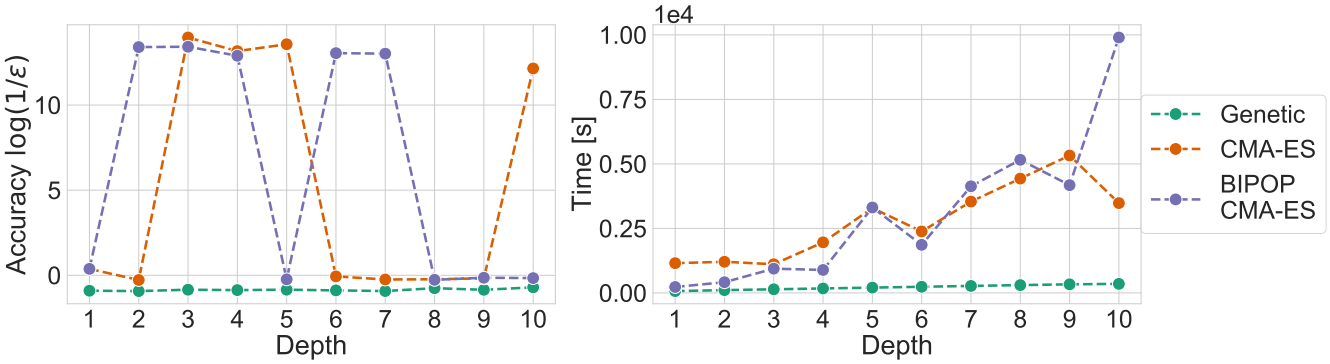


(b) Simulation performed with a 6-qubit system. Accuracy scales with depth, in this case 10 layers are not enough to saturate the flexibility of the circuit, so we do not observe a plateau as in other cases. Time scales with the number of layers.

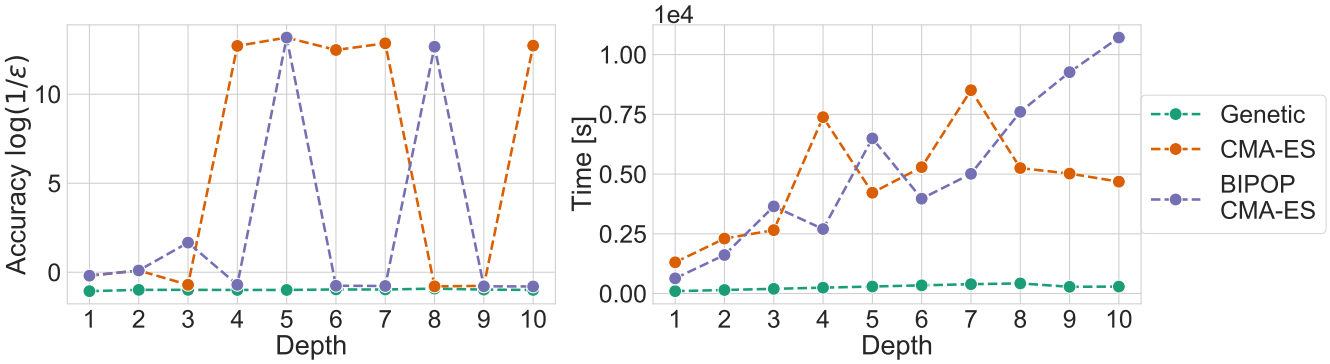


(c) Simulation performed with a 8-qubit system. COBYLA's accuracy decreases as depth increases, Nelder-Mead's is approximately constant, while Powell's increases, but still remains low.

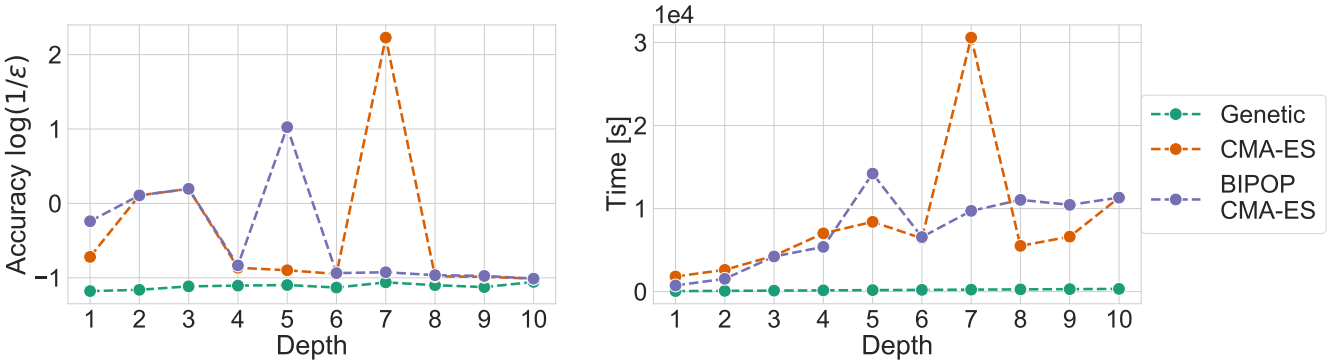
Figure 3.3: Accuracy and execution time for different number of layers, for all the gradient-free algorithms taken into account. Results show that accuracy decreases with the number of qubits, while time increases both with the number of qubits and depth of the circuit. Simulations are run on Galileo, which is the most stable hardware resource available, single thread execution with one CPU per task is performed.



(a) Simulation performed with a 4-qubit system. Genetic algorithm is not high-performance at all, while BiPop and classical CMA-ES are occasionally well suited, but exhibit in general too many irregularities to be reliable.



(b) Simulation performed with a 6-qubit system. Genetic algorithm does not shows an acceptable accuracy, while BiPop and classical CMA-ES present an oscillating behaviour.



(c) Simulation performed with a 8-qubit system. Accuracy is not acceptable in any case, since it is negative or almost zero everywhere, exception made for a fixed value of depth for CMA-ES and BiPop.

Figure 3.4: Accuracy and execution time for different number of layers, for all the evolutionary algorithms taken into account. In general accuracy presents too many irregularities, thus we consider such methods unreliable for the given problem. Execution time of CMA-ES is much more than the one of the GA, and in general of all the gradient-based and gradient-free methods considered. Simulations are performed on Galileo, which is the more stable hardware resource available.

Parameter	Value
Population size	150
Maximum number of generations	300
Crossover probability	0.05
Mutation probability	0.08
Independent flip probability	0.4

Table 3.1: Parameters of the GA fixed after tuning.

3.2.3 Genetic algorithms

Let us now focus on the evolutionary approach to minimization, we exploit two strategies: our implementation of a GA, and CMA-ES.

GA implementation We implement a GA optimizer, as discussed in Section 2.2.3, with Deap [67], an evolutionary computation framework. Every individual is a list of all the parameters of the circuit, while two-point crossover, bit flip mutation and tournament selection of size three are employed. Parameters are fixed after tuning with Optuna [61], their values are reported in Table 3.1.

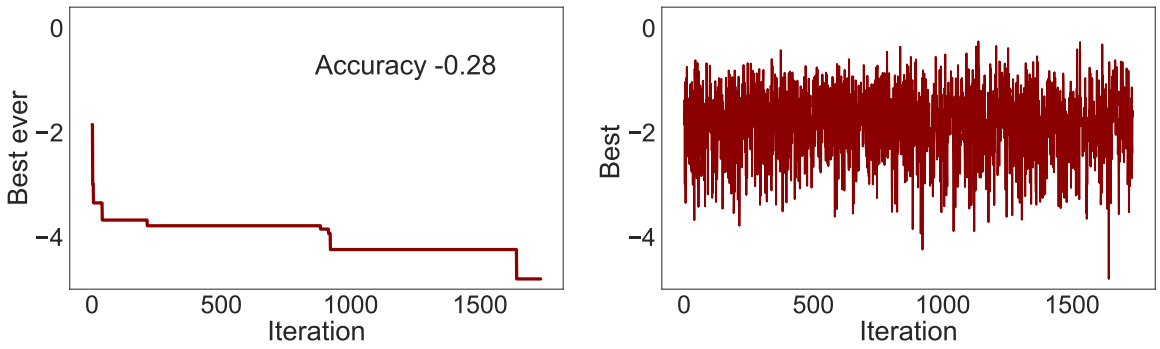
Let us now focus on CMA-ES, we benchmark it both with default options fixed in Pycma [68], and with BiPop option set. In order to solve multi-modal optimization problems, CMA-ES should be used in a restart setting with increasing population sizes, BiPop [69] is an option available for CMA-ES to implement such multi-start approach. First of all, algorithm run with standard population size, then two interlaced multi-start regimes are applied: one with a smaller population and the other with an increased one. Every regime is provided with a budget function that keeps count of the number of function evaluations: depending on which budget value is smaller, a complete run of either one or the other strategy is launched.

In Figure 3.4 we report benchmark results of evolutionary algorithms. A first observation concerns the irregular behaviour of CMA-ES. Indeed, for a 4- and 6-qubit system, accuracy varies between a maximum value, comparable to Trust-constr performance, and almost zero.
A 8-qubit circuit leads to similar results, but in this case accuracy tends, in general, to be negative, with a few values of depth corresponding to a non-negative accuracy. Moreover the genetic algorithm implemented with Deap does not get satisfying results in any case, since accuracy is always negative. A remark on execution time must be done: GAs need more time to run than other methods already discussed. Thus, they need many computational resources and, in this case, the loss is not balanced by high performances in terms of convergence.

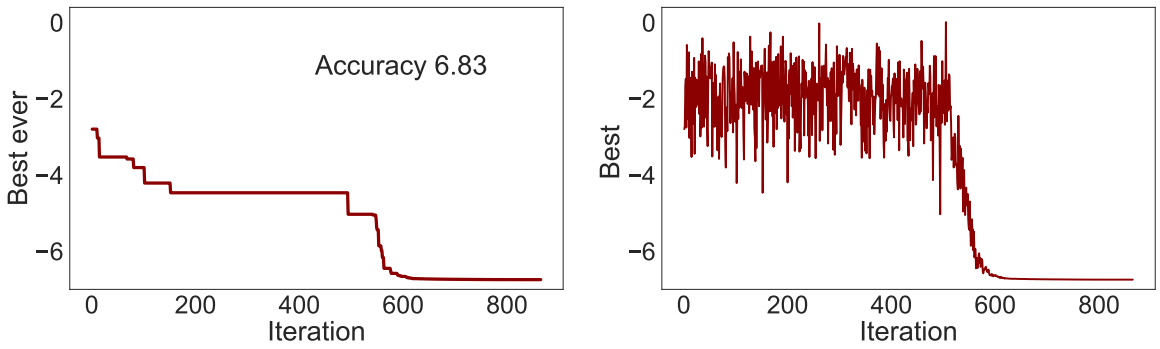
To better understand CMA-ES behaviour, let us focus on its way to approach the optimum and reach convergence. In Figure 3.5 we report both the best value over the whole minimization and the best value at each iteration for two runs of a classical CMA-ES. When accuracy is negative, Figure 3.5a, the algorithm stops afters a high number of iterations, even though it has not achieved the optimum. Indeed, the best oscillates as if the search were still random. Otherwise, consider for instance a 4-qubit and 2-layer circuit as shown in Figure 3.5b, after a certain number of iterations the algorithm finds a proper way to the optimum and follows that way until it converges. Such behaviour leads to a higher accuracy.

3.2.4 Hyperparameter optimization

Another possible strategy of minimization makes use of tuning as a pure optimizer. We implement such approach in Optuna [61], a hyperparameter optimization framework to automate hyperparam-



(a) Simulation performed with a 4-qubit system, 4-layer ansatz. The best of each iteration oscillates as if the search were completely random, indeed convergence is not reached and accuracy is negative.



(b) Simulation performed with a 4-qubit system, 2-layer ansatz. The best of each iteration oscillates until the algorithm finds a way to converge and follow that path, thus accuracy is 6.83.

Figure 3.5: Evolution of the best value over the whole optimization and the best value at each iteration for two runs of a CMA-ES. The algorithm leads to negative values of accuracy 3.5a when the best oscillates as if the search were completely random, thus it stops only when a high number of iterations is reached. Otherwise, when higher values of accuracy 3.5b are obtained, at a certain point the algorithm finds a proper way to the optimum and follows that way until it converges. Simulations are performed locally.

eter search. Thus the main idea is to consider the parameters of a loss function as hyperparameters to be optimized.

Since parameters of gates represent rotations the search space is the interval $[0, 2\pi]^n$ where n is the number of parameters. This observation is true for every optimizer taken into account, but a further discussion becomes important in this case. Indeed, this method works by mapping, point by point, the whole space: the larger the space, the harder the mapping.

It is then clear that every new parameter added to the circuit amplifies the search space, making convergence slower and harder to approach. Indeed, tuning relies on testing the loss function for different choices of parameters on every trial. The key point here is mapping the space densely enough, in order to make it possible to find the optimal point. Such mechanism requires a huge number of function evaluations, even if pruners and samplers work to make the search more precise. Our tests are done with 10000 trials and simulations are performed locally, exploiting parallelization with SQLite [70], making it possible to run multiple trials and to save old results.

In Figure 3.6 we report accuracy for different numbers of qubits and different depths. The plot shows that accuracy decreases as the number of qubits increases, while it is not affected by depth.

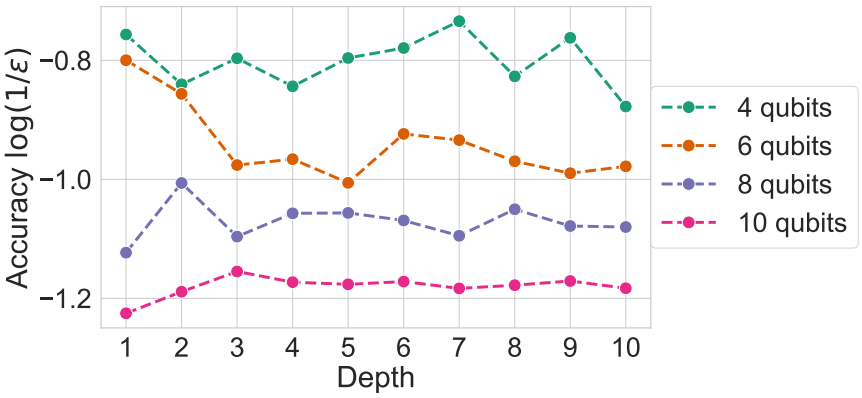


Figure 3.6: Accuracy reached by hyperparameter optimizer, for different number of qubits and different values of depth. It decreases with the numbers of qubits, while it does not show a clear dependence by depth. However in all the cases accuracy is negative, thus this method is not suitable to the problem. Simulations are performed locally by exploiting parallelization with SQLite [70].

In general accuracy is always negative, thus this approach is not a satisfying method to solve our problem. This result may be affected by the number of trials, a further mapping of the space increases the probability to find the optimum. However, this search may take an enormous number of evaluations, and running so many trials requires a huge execution time, thus this approach is not appropriate to solve the problem considered.

3.3 Variations to VQE

Finding the ground state of a given Hamiltonian is a problem that may be addressed with VQE, and with approaches that take advantage of such model into subroutines. In this section we benchmark minimizers on these algorithms, in order to highlight how these different approaches affect performances.

3.3.1 AAVQE

The first strategy that we are taking into account is the AAVQE, see Section 2.1.1.

In AAVQE the system starts from a well known Hamiltonian and then it evolves, through a user-defined scheduling function, to the Hamiltonian of the problem. Here we refer to the scheduling function as the parameterization $s(t)$ proposed in Equation (1.30). We run our simulations with a non-interacting Pauli-X Hamiltonian that describes the starting system:

$$H_0 = -\sum_i \sigma_x^i, \tag{3.9}$$

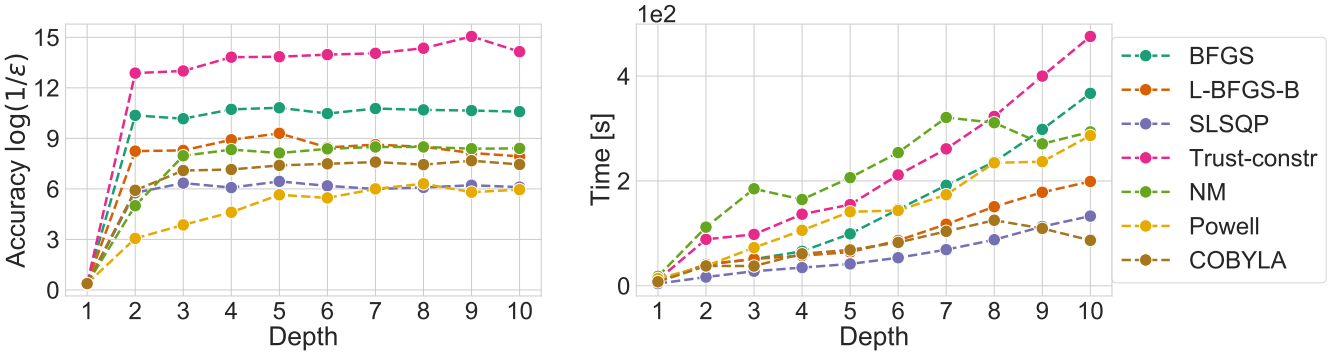
while we choose a linear scheduling function that reads:

$$s(t) = t \quad \text{with} \quad t \in [0, T], \quad T = 1. \tag{3.10}$$

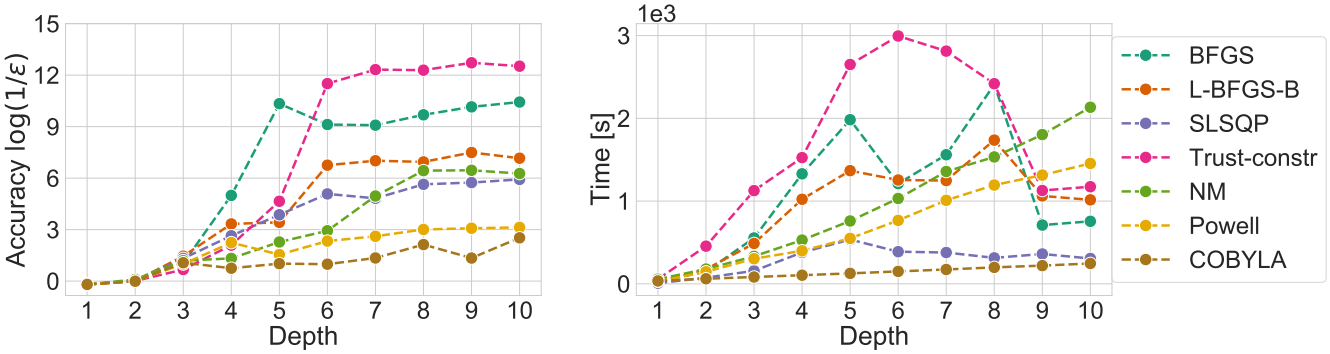
The following simulations are run with 10 iterations, thus with a step $dt = 0.1$.

In Figure 3.7 we show the benchmarks of Scipy's algorithms, run on Galileo with a single thread execution. For those algorithms that were already high-performance, such as Trust-constr, BFGS and others, we observe that accuracy does not increase significantly in its absolute value. Nevertheless, their behaviour is more stable than in the classical VQE, see Figure 3.2 and Figure 3.3. Indeed, this

procedure works as a training of parameters: while standard VQE optimizes a random set of values, at the final step of AAVQE parameters have already been trained on models that are adiabatically evolving to the one under study.
Moreover, it is interesting to observe that there are algorithms, whose performances are very poor on a classical VQE, that reach higher values of accuracy on AAVQE, see for instance COBYLA.
As for execution time, this method calls at every iteration a VQE subroutine, so time significantly increases in comparison to VQE.



(a) Simulation performed with a 4-qubit system.



(b) Simulation performed with a 6-qubit system.

Figure 3.7: Accuracy and execution time for Scipy's algorithms applied to AAVQE. Accuracy shows a behaviour similar to the one observed on a classical VQE, but algorithms seems to be more stable than in the classic case. Required execution time significantly increases in comparison to the classic VQE, since multiple VQE calls are performed.

3.3.2 Training layer by layer

Another possible approach to train parameters is to optimize a layer at a time, fixing the rest of the trainable elements of the ansatz.

This procedure repeats these single-layer optimization cycles until convergence is reached, thus when it is verified the condition:

$$\frac{\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}\|}{\|\boldsymbol{\theta}_{i-1}\|} \leq \text{PRECISION}, \tag{3.11}$$

where PRECISION is a fixed quantity, in our simulations we set 10^{-10}. We choose a high level of precision, in order to exploit all the advantages of this approach. However sometimes this choice

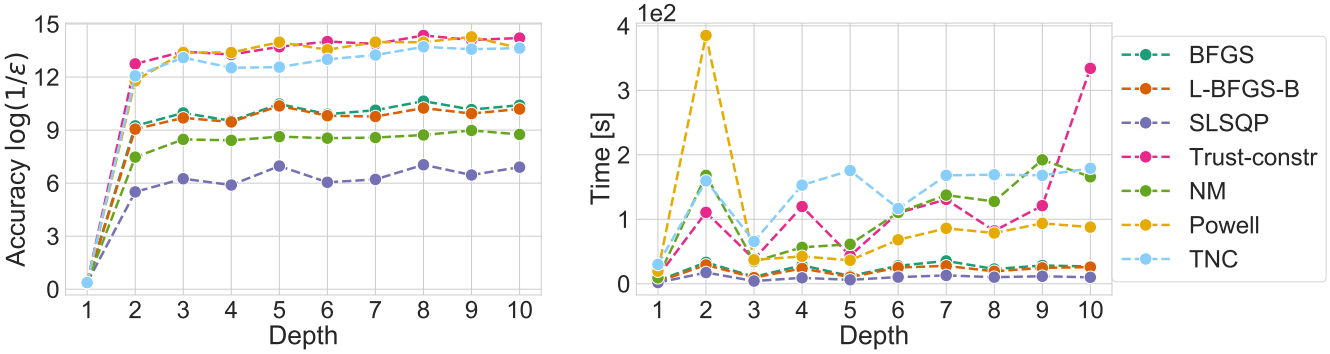


Figure 3.8: Accuracy and execution time for chosen Scipy's algorithms run on a 4-qubit ansatz, and training a layer at a time. Accuracy shows a behaviour similar to the one observed in classical VQE, but algorithms seem to be more stable than in the classic case. A significant exception is represented by Powell, whose accuracy increases from a maximum of about 6 in a classic VQE to a maximum of almost 15 with this training.

makes the convergence impossible to reach, this may be caused both by the size of the circuit and the specific algorithm employed. Indeed, when small ansatzes or not stable methods are involved, parameters may change too much at each iteration and Equation (3.11) can never be verified. This issue is faced applying COBYLA, thus we do not report results for such algorithm.

In Figure 3.8 we show accuracy and execution time for a 4-qubit circuit. As we have already discussed regarding AAVQE, this method generally does not increase performances, but it guarantees more stability. Yet, a significant exception is offered by Powell, whose accuracy increases from a maximum of about 6 on a classic VQE to a maximum of almost 15.

3.4 Final remarks

In this section we clarify some aspects of our discussion, and we provide a final overview of the benchmarks, in order to give an answer to the question: *which is the most suitable family of optimizers for this problem?*

3.4.1 Error and seed dependence

As we have already discussed, the optimizers taken into account, exception made for the tuning approach, need an initial guess for the parameters, i.e. a starting point in the space. To maintain the algorithms as general as possible, random parameters uniformly distributed in $[0, 2\pi]$ have been considered. However, these values are not really random, but pseudo-random. Pseudo-random numbers are employed to approximate the properties of sequences of random numbers, but they are not exactly random since they are univocally determined by an initial parameter, the *seed*. Once a seed is fixed, the pseudo-random sequence is defined, thus different seeds correspond to different sequences, and of course the same seed generates the same sequence.

All the results shown in Section 3.2 and Section 3.3 refer to initial parameters generated with seed 0. We perform the same simulations, for a 4-qubit system, with different seeds, in order to obtain an error band for each algorithm. Results are shown in Figure 3.9.

In Figure 3.9a we see that the most suitable algorithm, Trust-constr, exhibits a little dependence on the seed, with an error band that covers less than ± 1, providing almost constantly the highest performances. Gradient-based algorithms show, in general, a weak dependence by the seed, the

most error-affected methods are BFGS and SPSA, whose bands remain under ±2. Gradient-free methods, Figure 3.9b exhibit a non-trivial dependence by seed. Indeed, both COBYLA and Powell display a band that extends within ±2.5 around the average. The most interesting behaviour, even if it represents a negative result, concerns CMA-ES, see Figure 3.9c. As we have discussed in Section 3.2.3, CMA-ES accuracy oscillates between a maximum, high-performance, and almost zero. However, we cannot individuate a reason that leads to this strange behaviour, in particular we do not know why for those certain values of depth accuracy is high. Testing CMA-ES for different seeds helps us understand the outcome of our simulations, since the algorithm reaches convergence for a certain value of depth depending on the seed. For instance, if with a certain seed accuracy is zero exception made for 2-, 5-, 7- and 10-layer ansatzes, for another seed it may be zero apart from 1-, 3-, 4- and 8-layer ansatzes. As a consequence, error bar may extend from zero to the maximum, leading to an unreliable behaviour.

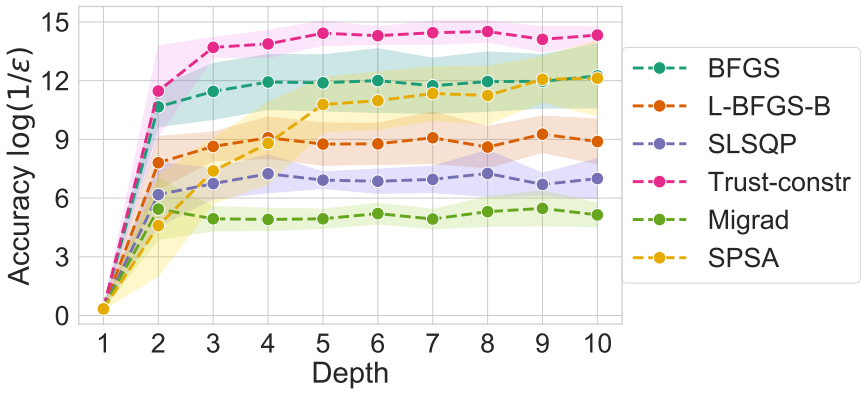
3.4.2 Conclusions

Benchmarks show that gradient-based algorithms are well-suited for the given problem, while gradient-free methods exhibit in general poor performances on a classical VQE.

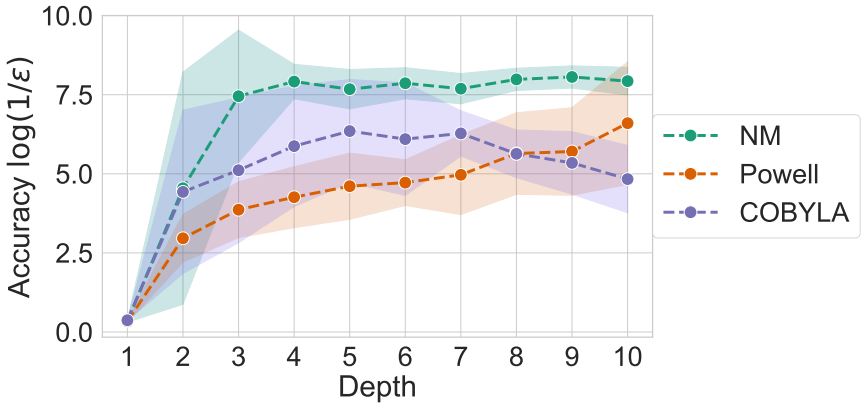
In Section 3.2.1 and Section 3.2.2 we have shown that the most suitable and stable algorithms for the given loss function are BFGS and Trust-constr. In general, gradient-based algorithms are more stable, and they lead to results that are extremely close to the expected one, even thought after a certain number of layers accuracy exhibits a plateau due to a lack of flexibility of the ansatz. Gradient-free methods are less stable and reach lower values of accuracy, thus their performances do not saturate after a certain number of layers.

Section 3.2.3 has shown that genetic algorithms are not reliable, since they exhibit poor performances, as the GA that we have implemented, or an irregular behaviour, especially regarding CMA-ES. Moreover, the tuning approach discussed in Section 3.2.4 is not suitable for the problem because of the high dimensionality of the search space. In addition, GAs and hyperparameter optimization methods require too many computational resources in terms of time, thus time-accuracy trade-off is not acceptable.

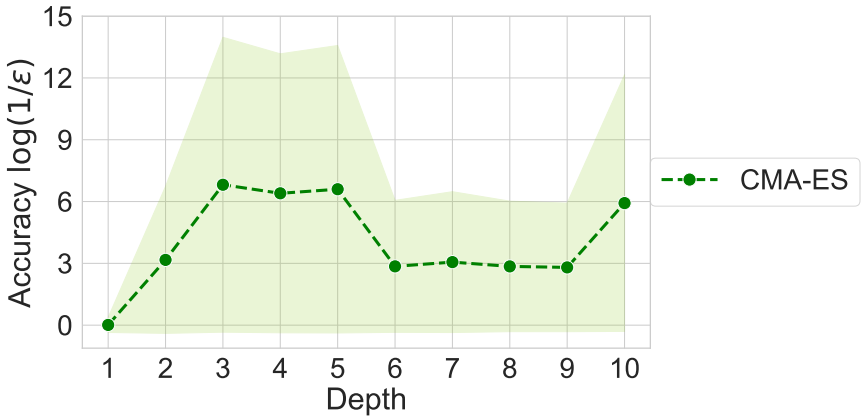
To find the ground state, then to minimize the same loss function of a classical VQE, other approaches that employ VQE as subroutine of a more complex algorithm can be considered. In Section 3.3.1 and Section 3.3.2 we tested two different approaches to train the parameters of an ansatz. The main outcome is that these methods are useful to provide more stable results in dependence on the number of layers. Just a few algorithms exhibit better performances than on a classic VQE, but these approaches do not increase accuracy overall, since they do not exceed the maximum obtained on VQE.



(a) Gradient-based algorithms exhibit a weak dependence by seed, the most error-affected methods are BFGS and SPSA, whose bands do not cover more than ±2.



(b) Gradient-free methods exhibit a non-trivial dependence by seed. Indeed COBYLA and Powell display a band that can extend within ±2.5.



(c) CMA-ES accuracy oscillates, for a given seed, between a maximum, high-performance, and almost zero, changing the seed affects the outcome moving high-performance points to different depth values. Thus the error bar may be very wide, since it goes from the minimum to the maximum, that are almost 14 distant for a 3-,4-, 5-layer circuit.

Figure 3.9: Accuracy of different algorithms in average for a 4-qubit system, the corresponding error bars are provided. Error bars are evaluated repeating the simulations with different seeds to generate initial parameters. Gradient-free methods exhibit a little dependence by the seed, while gradient-based are less stable. CMA-ES shows a peculiar behaviour, since error bars may cover an interval of almost ±7.

Chapter 4

Quantum Machine Learning applications: a Variational Quantum Classifier

In this chapter our purpose is to test the optimization strategies introduced in Chapter 2 on a Quantum Machine Learning problem for binary classification. Thus we first discuss the structure and the elements of a classifier of this kind, then we test optimizers on different dataset to classify.

4.1 Variational Quantum Classifier (VQC)

In Machine Learning a classifier is a supervised learning model that allows to recognize objects and to be able to separate them into categories. As a consequence, binary classification is the task of classifying elements of a set into two groups, i.e. two subsets targeted with 0 and 1. We address the problem of implementing a classical-quantum hybrid strategy, to exploit the potential of VQAs [71].

To solve this task many elements are needed: a procedure to encode the specific dataset into a PQC, a quantum circuit that works as ansatz, a loss function, and, of course, a classical minimization routine. Next sections focus on describing all these elements employed in a VQC, in order to understand how a classifier works and which is the role played by minimizers.

4.1.1 Encoding of the dataset

The encoding of a given data into a quantum circuit is a delicate task, since different datasets require different encodings. This procedure consists in associating every element of a given dataset to a quantum state $|\psi\rangle$, to whom the ansatz is thought to be applied.
We provide two methods to input data into a quantum state, one specifically designed for images, and another one that efficiently encodes small arrays.

Encoding arrays Consider a dataset that consists of n-dimensional arrays:

$$\mathcal{L} = [x_0, \dots, x_{n-1}], \tag{4.1}$$

every array can be mapped into a n-qubit state, thus we can employ a n-qubit circuit as ansatz for the VQC.

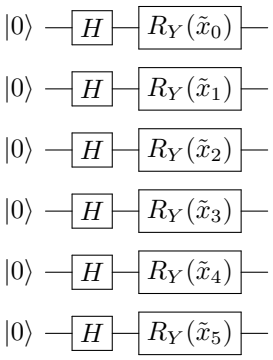


Figure 4.1: First layer of the parameterized quantum circuit used to encode small arrays. This specific design refers to a 6-qubit system, any generalisation is trivial. A first layer of Hadamard gates is applied to the state $|0\rangle^{\otimes n}$, followed by a layer of R_Y rotations. Every R_Y applied to the i-th qubit performs a rotation that is equal to the rescaled value $\tilde{x}_i$, computed from the initial array as defined in (4.2).

To convert the instance into a quantum circuit we first need to normalize its values so that they can represent rotations, thus we build another array $\tilde{\mathcal{L}}$, that reads:

$$\tilde{\mathcal{L}} = [\tilde{x}_0, \dots, \tilde{x}_{n-1}] = \pi \, \frac{[x_0, \dots, x_{n-1}]}{\max[|x_0|, \dots, |x_{n-1}|]}. \tag{4.2}$$

The encoding is then realized by the circuit proposed in Figure 4.1. A first layer of Hadamard gates is applied to the state $|0\rangle^{\otimes n}$, followed by a layer of $R_Y(\theta)$ rotations. Here Equation (4.2) comes to help: the rescaled elements $\tilde{x}_i$ of $\tilde{\mathcal{L}}$ are picked as parameters of each R_Y rotation.

This procedure is well-suited for arrays of dimensions up to $10 \sim 20$ elements, but it is not recommended for larger circuits since their simulation requires too many computational resources.

Encoding images We cannot employ the same approach just explained to encode an image, because the size of an image makes it unfeasible. Indeed, every figure can be associated to an array whose size is equal to the number of pixels. Nevertheless, an image of size 32x32 is mapped to an array of 1024 elements, thus it requires a 1024-qubit circuit, that is almost impossible to simulate. As a consequence, another strategy must be introduced: if the given image is made up by ξ pixels, then we can build a model that encodes it into a n-qubit system, such that $2^n = \xi$ is the dimension of the corresponding Hilbert space.

First of all we associate an image to an array, built concatenating each row of pixels. Then this array is normalized, since we want it to represent a quantum state. Finally, the elements of such array are chosen as the coefficients of a state in a 2^n-dimensional Hilbert space. If the array of pixels is $[\alpha_0, \dots, \alpha_{2^n-1}]$, in formula we have:

$$|\psi\rangle = \alpha_0 \, |0\rangle + \cdots + \alpha_{2^n-1} \, |2^n - 1\rangle \, . \tag{4.3}$$

The advantage is clear: the corresponding circuit is only n-dimensional.

4.1.2 Ansatz and measurements

A key element in a VQA is the parameterized circuit that operates on the initial state. Once we have associated a given dataset to a set of quantum states, we can temporarily forget about the original dataset and focus on the quantum counterpart of it.

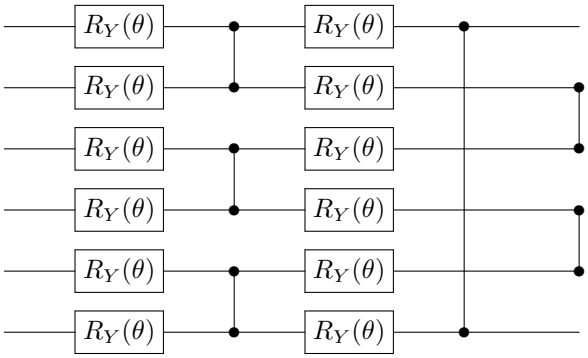


Figure 4.2: Ansatz for a 6-qubit classifier. Rotations $R_Y(\theta)$ represent the trainable elements of the circuit, while entanglement is guaranteed by CZ gates.

In Figure 4.2 we report our choice of the elementary layer of the variational ansatz, depth may be increased adding consequentially more layers of the same kind. Entanglement is guaranteed by CZ gates, while rotations are applied by $R_Y(\theta)$.

Measurements are performed in the last step, thus at the end of the circuit another layer of rotations is applied and the first qubit is measured. The meaning of this measurement is straightforward: the classifier predicts that a given element is labeled with 0 if we measure $|0\rangle$, otherwise the predicted label is 1. Moreover, measurements are employed to evaluate loss functions, see Section 4.1.3.

4.1.3 Loss functions

Loss functions play a central role in ML, since they define an objective against which the performance of the model is evaluated. They are the main ingredient of the whole problem of minimization of course, thus they work as a guide for the algorithm.

In our implementation of a VQC we provide three different loss functions, that exploit different approaches, but whose evaluation is always based on measurements. Indeed, every circuit is executed *nshots* times, each time we measure the first qubit, then we compute the frequencies of measuring 1 or 0, let $\mathcal{F}$ be this frequency.
Here we present the three loss functions employed.

Mean Squared Error (MSE)

As we have introduced, the circuit is executed *nshots* times for every element in a n-dimensional dataset, and the corresponding frequencies are evaluated, both for target 0, whom we refer as $\mathcal{F}^{(0)}$, and 1, referred as $\mathcal{F}^{(1)}$. Then Mean Squared Error (MSE) loss function is defined as follows:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \mathcal{F}_i^{(\{0,1\})}\right)^2, \tag{4.4}$$

where $\mathcal{F}_i^{(\{0,1\})}$ represents the frequency of finding either 0 or 1, according to the *real* targets, thus the probability of labelling correctly the item. For instance, if the real label is 0, $\mathcal{F}_i^{(0)}$ is used, that is the probability of finding 0 for the i-th element. Otherwise if the element should be labeled with 1, $\mathcal{F}_i^{(1)}$ is considered, that is the probability of measuring 1 according to the observed frequencies.
The interpretation is straightforward, since we are measuring a distance from the correct prediction, that is the goal value of $\mathcal{F}_i^{(\{0,1\})}$: the lower the loss, the better the classification.

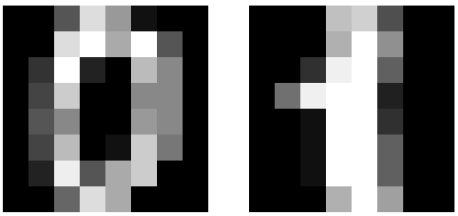


Figure 4.3: Samples taken from MNIST dataset in scikit-learn [72]. Every image represents a hand-written 1 or 0, of size 8x8, thus it has a total amount of 64 pixels.

Cross Entropy (CE)

Another possible approach is to introduce a logarithmic scale, that is often employed in information theory as a measure of uncertainty. We define Cross Entropy (CE) loss function as follows:

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{F}_i^{(\{0,1\})}, \tag{4.5}$$

in which the quantities involved are defined as in MSE. Such definition is well-posed since our VQC pursues to have higher values of $\mathcal{F}_i^{(\{0,1\})}$, which correspond to lower values of $\mathcal{L}_{CE}$.

Limited Cross Entropy (LCE)

Following the same approach introduced with CE, we provide a loss function that focuses on those elements that are classified with a wrong label. Then we define Limited Cross Entropy (LCE) loss function, that takes into account only elements whose frequency associated to the real target is less than 0.55, in formula we have:

$$\mathcal{L}_{LCE} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{F}_i^{(\{0,1\})} \qquad \text{with} \qquad \mathcal{F}_i^{(\{0,1\})} < 0.55. \tag{4.6}$$

4.1.4 Measure of accuracy

Now that we have widely described the main ingredients of the VQC proposed, it is worth to introduce a measure of accuracy that may apply to this problem, in order to make it possible to evaluate its performances.

In this framework we choose a probabilistic approach: accuracy is measured as the probability of labelling an element with the right target. Thus, once we have defined the label of the i-th element as Λ_i, accuracy α reads:

$$\alpha = \frac{1}{N}\sum_{i=1}^{N} |\Lambda_i^{predicted} - \Lambda_i^{real}|, \tag{4.7}$$

where Λ_i can be either 0 or 1.

In the next sections we study the behaviour of optimizers on different problems, all related to binary classification with the VQC proposed. Every benchmark refers to a different input dataset, thus our aim is, as in Chapter 3, to compare gradient-based, gradient-free and genetic algorithms.

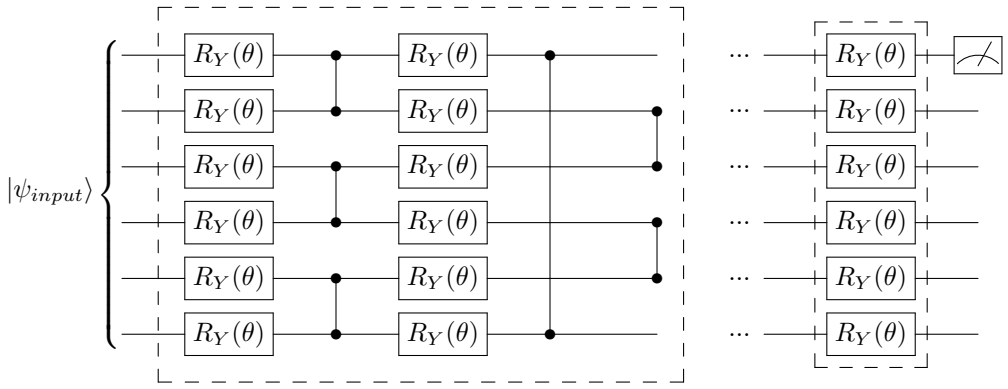


Figure 4.4: Ansatz designed for digit classification. The first set of R_Y rotations and CZ gates within the first dashed box, constitutes a single layer, while the whole circuit consists of a series of layer. At the end of the layers additional rotations are applied and a measurement is performed on the first qubit, see the dashed box on the right.

4.2 Digit classification

In the first problem that we are going to study the binary classifier is supposed to distinguish between handwritten digits, thus to recognise if a given sign is 0 or 1. In Figure 4.3 we report two samples, taken from a dataset with 360 images [72], that is employed in our simulations.
As we have widely discussed in Section 4.1.1, to encode an image with ξ pixel in a quantum circuit we must exploit a n-qubit system where $2^n = \xi$. In this case the images are 8x8, thus a 6-qubit ansatz is required. In Figure 4.4 we show the ansatz employed: it makes use of both rotations, the trainable elements, and CZ gates, to entangle qubits.

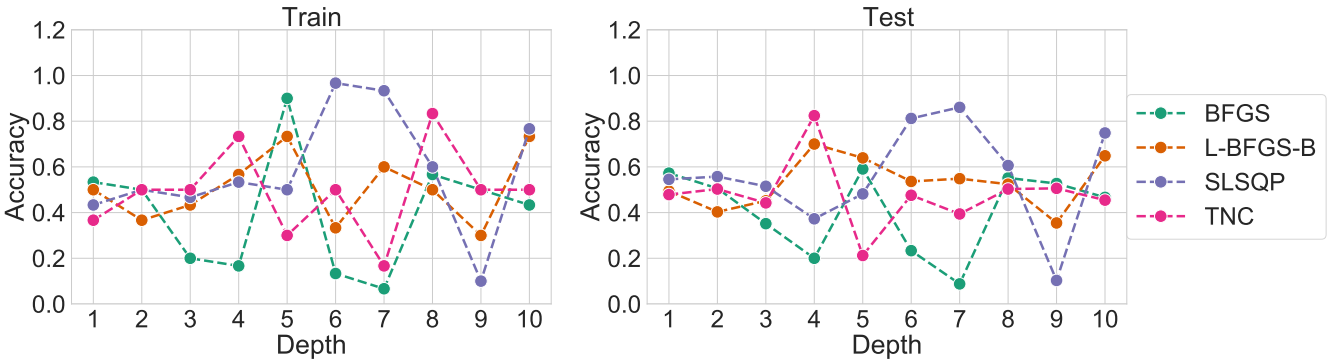
Simulations are performed on Montblanc, with pseudo-random input parameters generated with seed 0, and uniformly distributed in $[0, 2\pi]$.

Some remarkable results of benchmarks are shown in Figure 4.5 for CE loss function, and in Figure 4.6 in reference to MSE. The main outcome is that classic gradient-free algorithms and the GA, see Figure 4.5b and Figure 4.6b, perform well for the given problem. To be more specific, Nelder-Mead, Powell and COBYLA exhibit a stable behaviour with accuracy almost constantly equal to 1 for the train set. The test set shows a less stable trend, but it is still highly performant. The GA exhibits a less stationary trend, with accuracy that decreases for certain values of depth, especially for what concerns the test set.
On the other hand, gradient-based algorithms, see Figure 4.5a and Figure 4.6a, exhibit an irregular behaviour marked by fast variations in dependence on the number of layers. Moreover, accuracy shows such oscillations around 0.5, which is a non-satisfying probability of success for classification. Comparing results for CE and MSE loss functions we observe that Nelder-Mead exhibits a less stable behaviour when it minimizes MSE, but there are not relevant distinctions to highlight.

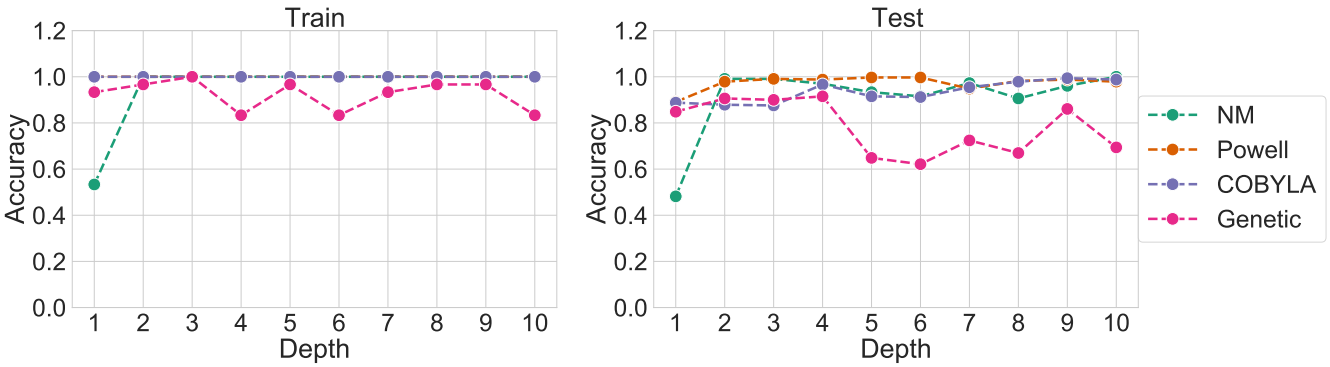
We can conclude that gradient-free algorithms are well-suited for the problem, since they lead to almost 100% success, gradient-based algorithms are not reliable instead. This observation leads to a guess: the search space may be too flat to be approached with gradient-based techniques.

4.3 Jet tagging

In Section 4.2 we benchmarked some gradient-based and gradient-free algorithms on a well-known, scholastic problem; however, a binary classifier finds wide applications in different fields. Here we

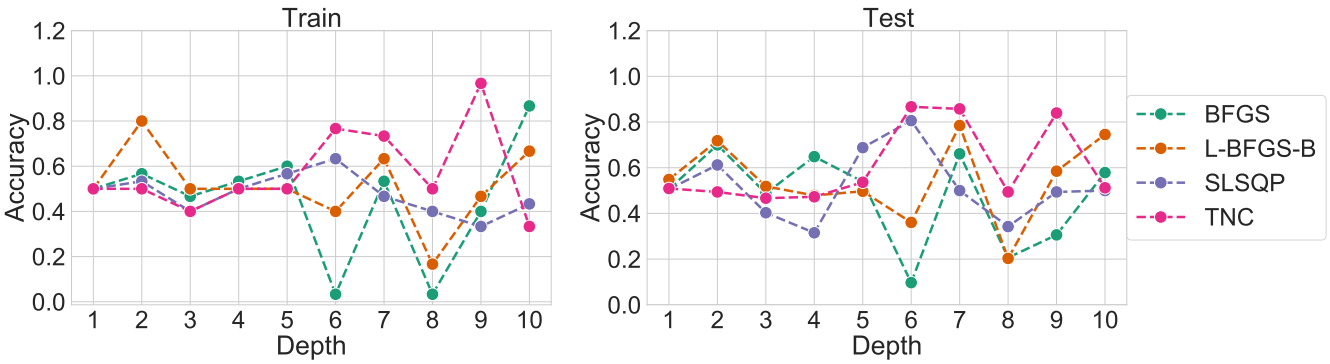


(a) Test and train accuracy for classification with gradient-based algorithms and CE loss function. Algorithms exhibit an irregular behaviour with non satisfying levels of accuracy. Oscillations are around the value 0.5, which is an unacceptable probability of success for a classifier to be used.

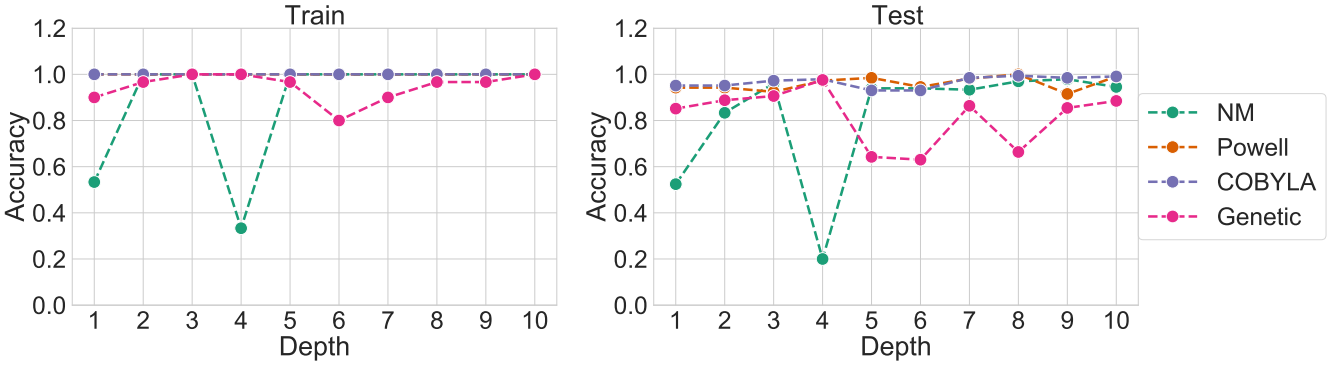


(b) Test and train accuracy for classification with gradient-free algorithms and CE loss function. Nelder-Mead, Powell and COBYLA exhibit a stable behaviour with accuracy almost constantly equal to 1 for the train set and between 0.9 and 1 for test test. The GA exhibits a less stationary trend, especially in the test set, but it still provides satisfying results.

Figure 4.5: Test and train accuracy for classification with gradient-based 4.5a and gradient-free 4.5b optimization of CE loss function. Classical gradient-free algorithms are well-suited for the problem, since they reach almost 100% accuracy both in train and test, while the GA is less stable, but still highly performant for fixed number of layers. Gradient-based algorithms exhibit an irregular behaviour with non satisfying levels of accuracy instead. Simulations are performed on Montblanc, with pseudo-random input parameters generated with seed 0, and uniformly distributed in $[0, 2\pi]$.



(a) Test and train accuracy for classification with gradient-based algorithms and MSE loss function. Algorithms exhibit an irregular behaviour with non satisfying levels of accuracy.



(b) Test and train accuracy for classification with gradient-free algorithms and MSE loss function. Powell and COBYLA exhibit a stable behaviour with accuracy almost constantly equal to 1. Nelder-Mead and the GA exhibit a less stationary trend, but they still provide satisfying results.

Figure 4.6: Test and train accuracy for classification with gradient-based 4.6a and gradient-free 4.6b optimization of MSE loss function. Classical gradient-free algorithms are well-suited for the problem, while the GA is less stable, but still highly performant. Gradient-based algorithms exhibit an irregular behaviour with non satisfying levels of accuracy instead. Simulations are performed on Montblanc, with pseudo-random input parameters generated with seed 0, and distributed uniformly in $[0, 2\pi]$.

present a possible application in High Energy Physics. In this context a possible problem deals with distinguishing different jets that come from particle collisions [73], in order to individuate the original reaction.
The data taken into account are the results of one of the following collisions:

$$\begin{aligned} &\text{target } 0 \quad pp \to W^+W^- \to qqqq, \\ &\text{target } 1 \quad pp \to qq, qg, gg, \end{aligned} \tag{4.8}$$

where p represents a proton, $W^{\pm}$ are vector bosons, g stands for gluon, q is a generic way to refer to quarks. This collisions are usually performed at LHC.
Next sections are dedicated to address the problem of classifying these collisions with different optimizers. Two different kind of datasets are provided: one consists of images and the of arrays.

(a) Examples of images without pile-up. On the left a sample that refers to target 0, while on the right a sample of target 1.

(b) Examples of images with pile-up. On the left a sample that refers to target 0, while on the right a sample of target 1.

Figure 4.7: Samples images with and without pile-up, for both the jets taken into account in Equation (4.8). Images are 32x32 and every pixel can take a value between 0 and 255. These values represent a density of energy, thus they are measured with a calorimeter.

4.3.1 Images

The first dataset that we are going to test consists of 32x32 images, where every pixel takes a value between 0 and 255 [73], these are the results of energy measurements through a calorimeter. Images are divided into two datasets, that we study separately, referred as "with" and "without" *pile-up*, where pile-up represents proton-proton in-time interactions. In Figure 4.7 we provide a few examples of images taken from such dataset.

To encode an image into a quantum state we employ the strategy proposed in Section 4.1.1. In this case a 10-qubit ansatz is required, in Figure 4.8 we provide its representation, that is analogous to the others previously described.

We are interested in comparing the performances of different optimizers, thus we do not need to concentrate in finding the best possible results with specific ML techniques overall, we refer to more detailed studies [71] for a in-depth analysis of the classifier and its limits. Indeed, in this context we choose to report just those results that are useful to highlight how the minimizers behave, and that provide us with suggestions concerning which minimization strategy must be applied.

In Figure 4.9 we report benchmarks results concerning classification on a subset with 25 elements of the dataset with pile-up.
First of all, comparing Figure 4.9a, Figure 4.9b and Figure 4.9a, we observe that different loss functions lead to similar results in terms of accuracy.

It is clear which family of optimizers is more suitable for the given problem: gradient-based algorithms exhibit performances that are constantly around 50% of success, while gradient-free methods provide an accuracy of approximately 75%. As we have already introduced, our discussion does not focus on such numerical results considered individually, however it is important to underline that

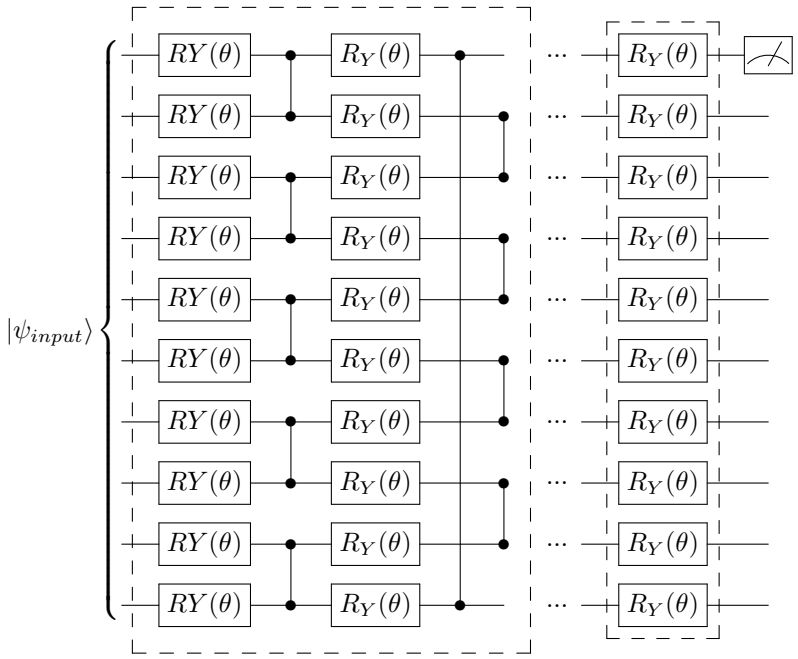


Figure 4.8: Ansatz designed for 32x32 images classification. The fundamental layer is made up by R_Y rotations and CZ gates within the dashed box on the left. At the end of the layers additional rotations are applied and a measurement is performed on the first qubit, see the dashed box on the right.

50% of correct classifications is not acceptable for a classifier.
However, our aim is to understand why gradient-based algorithms show so poor performances. This behaviour may be caused by the structure of the search space itself: derivative-based algorithms rely on following the direction of the gradient, thus if the space is flat or if it presents many local minima, optimizers can easily get stucked in such minima.

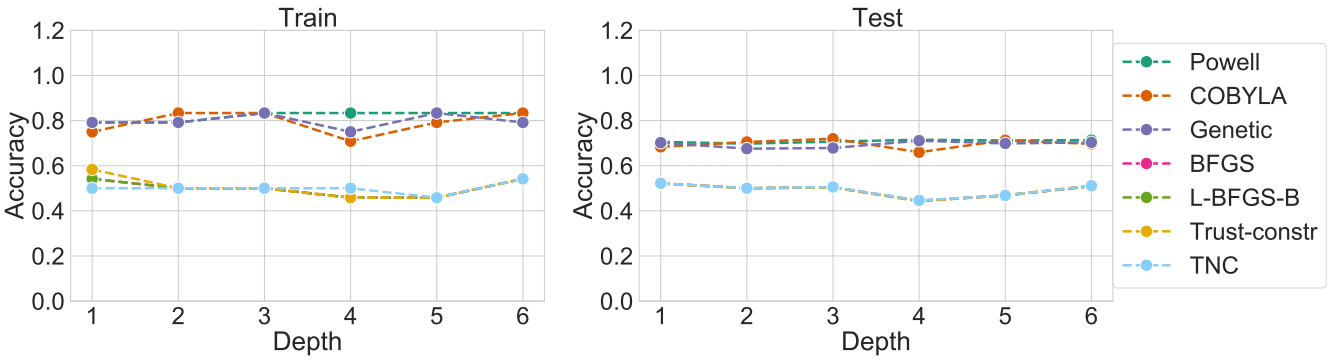
4.3.2 Features

The last dataset that we take into account is made up by arrays of six items each. Every element has a specific physical meaning, for a detailed description of these values we refer to [74]. However, let us present the quantities involved: the invariant mass $\sqrt{s}$ of the trimmed jet, N-subjettiness $\tau_{21}^{\beta=1}$, and the energy correlation functions $C_2^{\beta=1}$, $C_2^{\beta=2}$, $D_2^{\beta=1}$, $D_2^{\beta=2}$. We refer to such elements as features, see Figure 4.12, for an schematic representation of the elements.

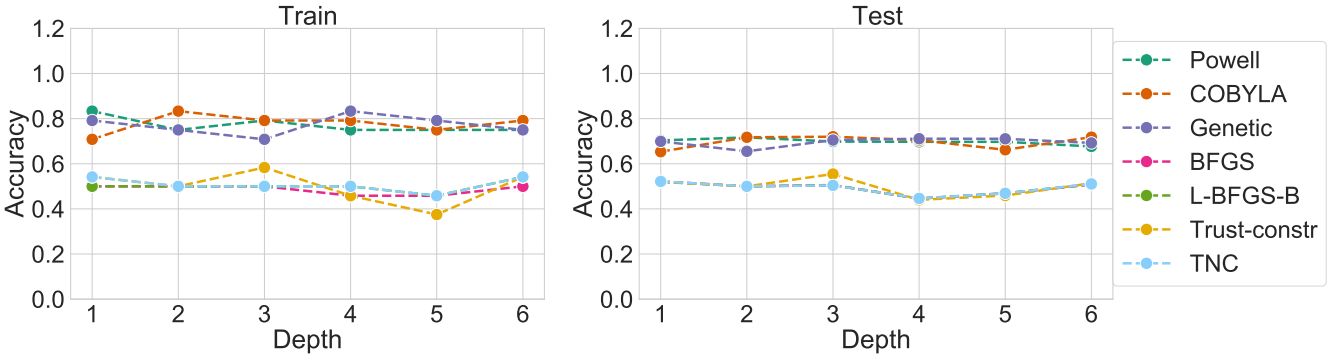
To encode a small size array into a quantum circuit we employ the strategy proposed in Section 4.1.1. Therefore, the ansatz is a 6-qubit circuit, the diagrammatic representation of both the encoding gates and the elementary layer is shown in Figure 4.11.

In Figure 4.12 we report both train and test accuracy regarding classification on a subset with 25 elements of the dataset without pile-up. The main outcome is that both gradient-based and gradient-free methods provide poor performances. To be more specific, test accuracy shows a stable behaviour around 50% of success for all the optimizers, while training accuracy exhibits an irregular trend. However, performances are generally poor, even though it is interesting to highlight that training accuracy of gradient-free algorithms tends to be a little above gradient-based's.

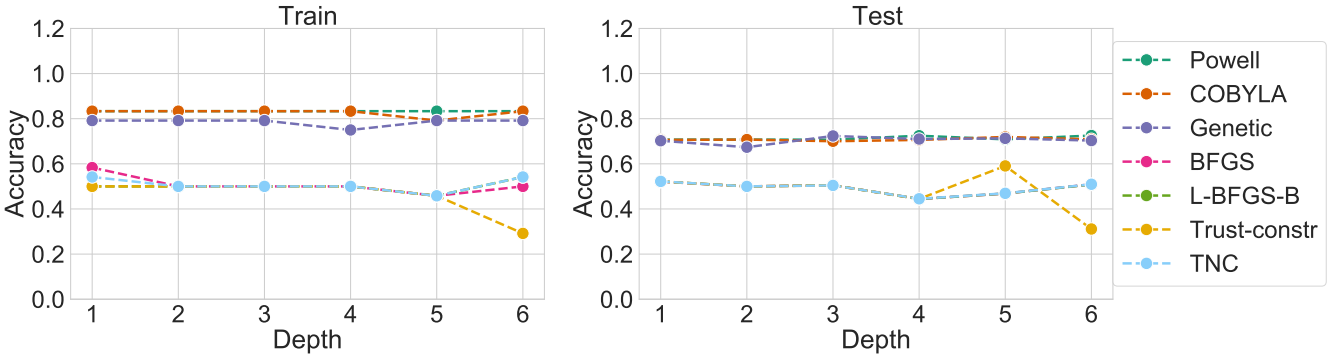
The fact that, in contrast to previous examples, gradient-free and gradient-based methods provide



(a) Test and train accuracy for image classification with gradient-based and gradient-free methods and CE loss function. Gradient-based algorithms exhibit performances that are constantly around 50% of success, while gradient-free methods provide an accuracy of approximately 75%.



(b) Test and train accuracy for image classification with gradient-based and gradient-free methods and LCE loss function. Gradient-based algorithms exhibit performances that are constantly around 50% of success, while gradient-free methods provide an accuracy of approximately 75%.



(c) Test and train accuracy for image classification with gradient-based and gradient-free methods and MSE loss function. Gradient-based algorithms exhibit performances that are constantly around 50% of success, while gradient-free methods provide an accuracy of approximately 75%.

Figure 4.9: Test and train accuracy for image classification on a subset with 25 elements of pile-up dataset. Results are provided both for gradient-based and gradient-free methods. Gradient-based methods show a poor level of accuracy, since they guarantee success for about 50% of the data. Gradient-free algorithms are more suitable instead, even though they reach a maximum of 0.75 in terms of accuracy. Simulations run on Montblanc.

$\sqrt{s}$	$\tau_{21}^{\beta=1}$	$C_2^{\beta=1}$	$C_2^{\beta=2}$	$D_2^{\beta=1}$	$D_2^{\beta=2}$

Figure 4.10: Structure of an element in features dataset. The array is made up by six items, which represent the invariant mass $\sqrt{s}$ of the trimmed jet, N-subjettiness $\tau_{21}^{\beta=1}$, and the energy correlation functions $C_2^{\beta=1}$, $C_2^{\beta=2}$, $D_2^{\beta=1}$, $D_2^{\beta=2}$.

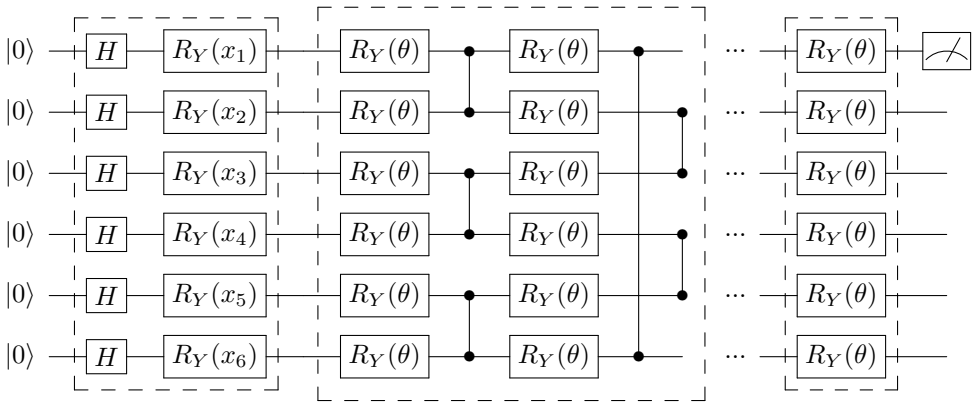


Figure 4.11: Encoding gates and elementary layer of the ansatz employed for features classification. The dashed box on the left represents the gates needed to encode the array into a quantum state. The central box is the elementary layer constituted by CZ gates and R_Y rotations, while at the end of all the layers additional rotations are applied, see dashed box on the right.

similar results, suggests that such poor performances are not due just to the oprimization process. For instance, other reasonable sources of error could be seek in the encoding procedure and in the variational ansatz, that may be affected by a lack of flexibility. So we actually cannot draw conclusions specifically regarding the behaviour of the optimizers in this specific case, thus we leave this discussion to further works.

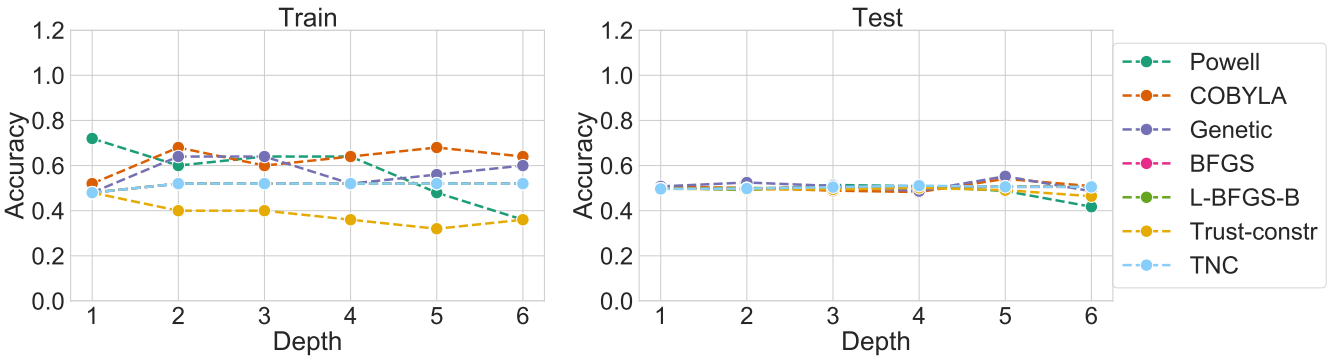
4.4 Barren Plateaus

In this section we provide further comments on the benchmarks, in order to better understand why we observe a certain behaviour, thus why gradient-based algorithms are not well-suited for the given problem.

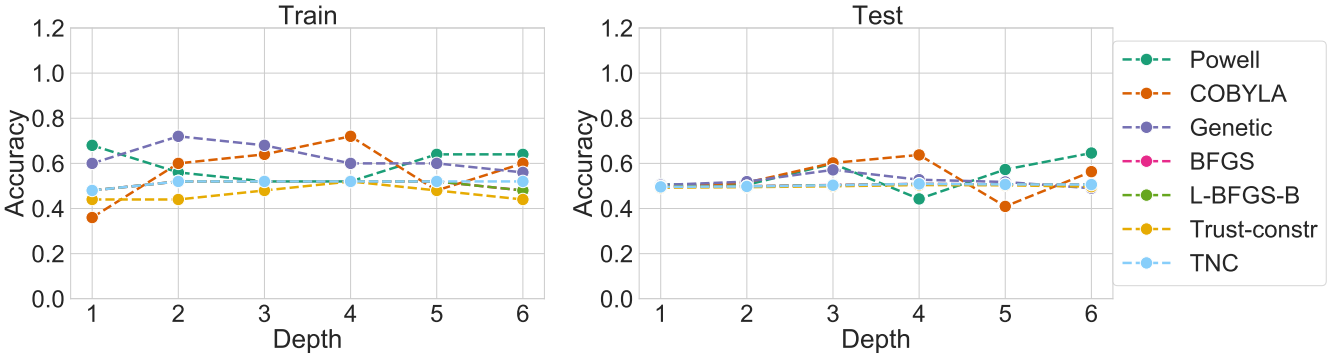
The main outcome of benchmarks discussed in Section 4.2 and Section 4.3 is that gradient-based algorithms fail in approaching the optimum, while gradient-free seem to be well-suited for the problem. The reason of such behaviour must be sought in Barren Plateaus, a common phenomenon for VQA, so it is useful to provide a formal definition of it.

Consider a parameterized circuit $U(\boldsymbol{\theta})$ and a loss function f, the corresponding system is said to exhibit a Barren Plateau if its gradients vanish exponentially with the number of qubits [75]. Recalling Chebyshev's inequality, for a fixed value δ the condition reads:

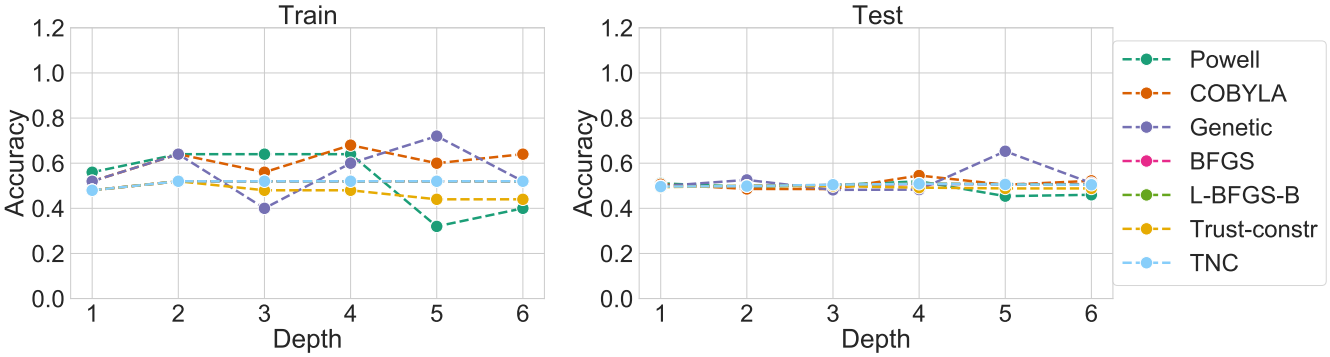
$$P\Big[\Big|\frac{\partial f}{\partial \theta_k}\Big| \geq \delta\Big] \leq \frac{1}{\delta^2}\mathrm{Var}\Big[\frac{\partial f}{\partial \theta_k}\Big]. \tag{4.9}$$



(a) Test and train accuracy for features classification with gradient-based and gradient-free methods and CE loss function. All the methods exhibit poor performances.



(b) Test and train accuracy for features classification with gradient-based and gradient-free methods and LCE loss function. All the methods exhibit poor performances.



(c) Test and train accuracy for features classification with gradient-based and gradient-free methods and MSE loss function. All the methods exhibit poor performances.

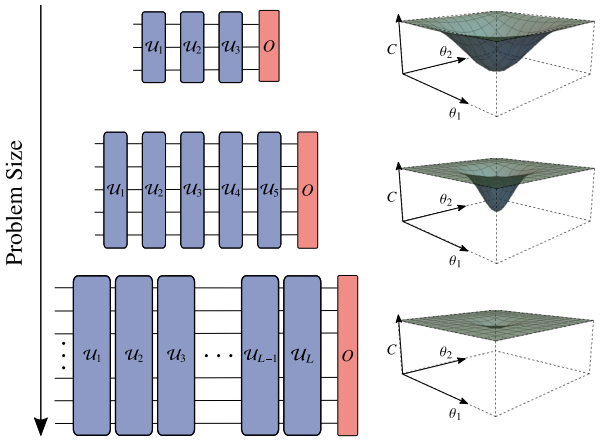
Figure 4.12: Test and train accuracy for features classification on a subset with 25 elements of the dataset without pile-up. Results are provided both for gradient-based and gradient-free methods. Both gradient-based and gradient-free methods exhibits poor performances. Concerning train accuracy gradient-free algorithms occasionally show a better behaviour, while test accuracy is almost constant for all the methods. Simulations run on Montblanc.

Another possible definition, in terms of probability, can be given: a model presents Barren Plateaus when the gradient vanishes exponentially with high probability.
As a consequence, loss functions that exhibit Barren Plateaus need exponentially precise measurements to determine the minimization direction.

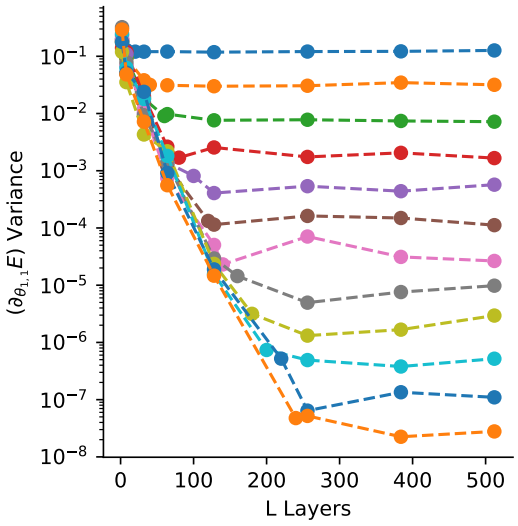
In Figure 4.13a we report a model representation of the phenomenon: as the size of the circuit increases, both in terms of layers and qubits, the search space becomes flatter. As a consequence, the gradient of a given loss function becomes less capable to determinate the path to the global minimum.

A more quantitative way to figure out how the gradient looses information as the circuit becomes larger is shown In Figure 4.13b. The plot exhibits the results of a simulation [76] that clearly explains the dependence of the gradient variance by the number of qubits and the depth of a circuit. For a fixed number of qubits, as the number of layers increases variance decreases until it reaches a fixed value, creating a plateau. On the other hand, comparing results with a different number of qubits, we observe that the height of such plateau changes. Indeed, the limit value of gradient's variance is fixed by the number of qubits: larger circuits lead to flatter search spaces, thus to lower values of variance.

To conclude, in Section 4.3 and Section 4.2 we have observed that gradient-based algorithms are not well-suited for classification with a VQA. Such issue can be explained if we take into account the phenomenon of Barren Plateaus, that often occur in quantum ML problems. As a consequence, when the search space presents Barren Plateaus, gradient-free approaches to minimization must be chosen.



(a) Diagrammatic representation of Barren Plateaus phenomenon: as the size of the ansatz increases, both in terms of number of qubits and layers, the gradient vanishes exponentially. The corresponding search space becomes flatter. Source [77].



(b) Sample variance of the gradient of a two-local Pauli term plotted as a function of the number of layers in the 1D circuit. Different lines correspond to all even numbers of qubits between 2 and 24, with 2 qubits being the top line. For a given number of qubits, as the number of layers increases variance decreases until it reaches a fixed value, creating a plateau. The height of such plateau is fixed by the number of qubits: larger circuits in terms of qubits lead to lower values of gradient's variance. Source [76]

Figure 4.13: In VQAs the phenomenon of Barren Plateaus may occur. When it happens, the gradient of a given loss function vanishes exponentially with the number of qubits. As a consequence, loss functions that exhibit Barren Plateaus need exponentially precise measurements to determine the minimization direction.

Conclusion

Our whole discussion is aimed to analyze the behaviour of different classes of optimizers in VQAs. In particular, we first benchmarked the VQE implemented in Qibo on the Heisenberg XXZ chain model. Eventually, we tested a VQC for binary classification on datasets composed by images of handwritten digits and experimental results of particle collisions. Let us now summarize the main results, in order to conclude which classes of optimizers are better suited for specific families of VQAs.

For what concerns VQE, in Chapter 3 we concluded that gradient-based methods converge to extremely satisfying results. Indeed, in the best case we obtained values equal to the real eigenstate up to the 14th decimal place. Better convergence is forbidden by a lack of flexibility of the circuit employed. In general, derivative-based algorithms exhibit a stable behaviour in terms of dependence on the depth of the ansatz. Moreover, such methods are not significantly affected by seed changes, thus they represent a reliable approach to minimize the given problem. On the other hand, gradient-free algorithms show non satisfying performances if compared with the best derivative-based methods. In particular, accuracy exhibits a non-stable behaviour with non-trivial seed dependence. Moreover, results quickly become poor as the number of qubits increases, for instance the real eigenvalue of a 8-qubits system differs of more than 1 unit from the one obtained applying gradient-free methods on the VQE. We find similar unsatisfying results by benchmarking hyperparameters optimization algorithms, which is not hard to believe because they perform a totally heuristic search. For what concerns EAs, the main outcome is that, even if they occasionally converge, they generally get stucked, and accuracy exhibits a huge dependence on seed changes. Moreover, GAs need many resources in terms of time, but such loss is not balanced by high computational performances, we then conclude that these methods are not reliable to solve the given problem.

All the previous observations suggest that the loss function is regular enough to be optimized by gradient-based methods. Moreover, we can try to draw conclusions concerning the search space: since algorithms that employ the gradient are highly performant, we can guess that the landscape does not present many local minima where the minimizer can easily get stucked in. In other words, the search space is probably curved enough for the gradient to guide the convergence, without leading to incorrect results. Eventually, the fact that gradient-free algorithms and GAs are significantly affected by seed changes can be explained if we remember that they rely on a metaheuristic search. It is not hard to believe that changing the initial point can bring such algorithms to explore different regions of the landscape, that may be occasionally too far from the optimum.

In Section 3.3 we tested two different methods to train parameters through multiple calls to VQE subroutines: Adiabatically Assisted VQE (AAVQE) and a layer by layer training. The main outcome is that these approaches do not provide better performances overall if compared to the performances of a classical VQE. However, it is worth to highlight that employing VQE as a subroutine of a more complex process guarantees more stable results in terms of dependence by the number of layers employed in the circuit.

Finally, in Chapter 4 we addressed the problem of finding a proper minimizer for binary classification with a VQC. To do so we tested three different datasets: MNIST dataset of handwritten digits, and experimental results of particles collisions at LHC, jet images and features.

Both tests performed on digits and jet images lead to the same outcome: gradient-based methods fail to approach the optimum. Indeed, these methods provide a model that is able to correctly classify a given data with only 50% probability of success, while gradient-free approaches and the GA reaches 100% accuracy on digits and about 75% on jet tagging. A possible explanation to the fact that derivative-based approaches fail can be guessed if we focus on the structure of the search space itself. The landscape may present many local minima, and it is with high probability extremely flat. Indeed, this is a common phenomenon in quantum computation and especially in VQAs called Barren Plateaus. A model is said to present Barren Plateaus when the gradient vanishes exponentially as the circuit increases both in terms of number of qubits and layers. In other words, the landscape becomes so flat that the variance of the gradient is almost negligible. Our results can be explained in the framework of VQAs taking into account such phenomenon.

In contrast, tests performed on features dataset provide poor results for all the algorithms taken into account. Since these data have a physical meaning similar to jet images, we would expect to observe the same distinction between gradient-based and gradient-free methods. Thus, we suppose that such poor performances may be not due just to the optimization process. For instance, other reasonable sources of error could be seek in the encoding procedure and in the variational ansatz, but we leave this discussion for further works.

To conclude, we tried to explain all the results obtained by testing different VQAs, but an open question still remains: why do we observe an opposite outcome analyzing a QML model and the VQE? We deepened both why gradient-based methods are well-suited for the VQE and their failure in the VQC model with considerations regarding the search space. Nevertheless, why do these landscapes differ is a non-trivial question. A possible next step could deal with understanding which relationships occur between the search space, the circuit involved and the quantum properties of the state. An in-depth analysis of such features may also help to design high performance QML model and to improve the existing ones.

List of Acronyms

AAVQE Adiabatically Assisted Variational Quantum Eigensolver

AQC Adiabatic Quantum Computation

BFGS Broyden–Fletcher–Goldfarb–Shanno

BPP Bounded-error Probabilistic Polynomial time

BQP Bounded-error Quantum Polynomial time

CE Cross Entropy loss function

CG Conjugate Gradient

CMA Covariance Matrix Adaptation

CMA-ES Covariance Matrix Adaptation Evolution Strategy

COBYLA Constrained Optimization by Linear Approximation

CTT Church-Turing Thesis

DTM Deterministic Turing Machine

EA Evolutionary Algorithm

ECTT Extended Church-Turing Thesis

EI Expected Improvement

EQP Exact Quantum Polynomial time

GA Genetic Algorithm

GMM Gaussian Mixture Model

HHL Harrow-Hassidim-Lloyd algorithm

HOA Hyperparameter Optimization Algorithms

KKT Karush-Kuhn-Tucker conditions

L-BFGS Limited-memory BFGS

L-BFGS-B Bounded Limited-memory BFGS

LCE Limited Cross Entropy loss function

ML Machine Learning

MSE Mean Squared Error loss function

NISQ Noisy Intermediate-Scale Quantum

NM Nelder-Mead

NP Non-deterministic Polynomial time

P Polynomial time

PQC Paramaterized Quantum Circuit

PSPACE Polynomial Space

PTM Probabilistic Turing Machine

QA Quantum Annealing

QC Quantum Computer

QEC Quantum Error Correction

QECTT Quantum Extended Church-Turing Thesis

QMA Quantum Merlin-Arthur

QTM Quantum Turing Machine

SCTT Strong Church-Turing Thesis

SLSQP Sequential Least SQuares Programming optimizer

SPSA Simultaneous Perturbation Stochastic Optimization

SQP Sequential Quadratic Programming

TM Turing Machine

TNC Truncated Newton Constrained

T Toffoli gate

TPE Tree-structured Parzen Estimator

UTM Universal Turing Machine

VQA Variational Quantum Algorithms

VQC Variational Quantum Classifier

VQE Variational Quantum Eigensolver

List of Figures

Appendix A

$(\mu/\mu_w, \lambda)$-CMA-ES algorithm

The most common implementation of CMA-ES is the so-called $(\mu/\mu_w, \lambda)$-CMA-ES algorithm [78], where at each iteration weighted combination with weights w of the best μ parent individuals out of λ generated offsprings is computed. In algorithm 1 we provide a full description of the procedure.

First of all, for a given n-dimensional search space, parameters must be set consequentially, see lines 4-12.

The algorithm is then initialized in lines 14-19: we set the starting distribution mean and current favorite solution $\boldsymbol{m}_0 \in \mathbb{R}$, starting step size σ_0, and the evolutionary paths $\boldsymbol{p}_{\sigma,0}$, $\boldsymbol{p}_{c,0}$. Then the covariance matrix $\boldsymbol{C}_0$ is created and it is set equal to the identity matrix, note that at every iteration it must be a symmetric, and positive-defined n x n matrix. c_σ^{-1} represents backward time horizon for the evolution path $\boldsymbol{p}_\sigma$, while μ_w is the variance effective selection mass. Then d_σ is the damping parameter to control the change of the step size, usually $d_\sigma \approx 1$. If $d_\sigma = \infty$ or c_σ the step size is constant. Analogously, c_c^{-1} is backward time horizon for $\boldsymbol{p}_c$, eventually c_1 and c_μ are the learning rates respectively for the rank-one and rank-μ updates of $\boldsymbol{C}_k$.

In line 21 the main loop starts, λ candidate solutions are updated by adding a random Gaussian mutation defined by $\boldsymbol{C}_t$, and the loss function is then evaluated for all candidate solution:

$$\boldsymbol{x}_{t,k} = \mathcal{N}(\boldsymbol{m}_t, \sigma_t{}^2, \boldsymbol{C}_t) = \boldsymbol{m}_t + \sigma_t \, \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}_t) \tag{A.1}$$

$$\boldsymbol{f}_k = f(\boldsymbol{x}_k) \tag{A.2}$$

where t is the index of the main loop, while $k = 0, \ldots, \lambda$ is the index of the cycle in line 22. Once that the possible solutions are set, the new mean value is computed in line 26:

$$\boldsymbol{m}_{t+1} = \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda} \quad \text{where} \quad \sum_{i=1}^{\mu} \boldsymbol{w}_i = 1, \; w_i > w_j > 0 \;\; \forall i < j. \tag{A.3}$$

In line 27, the update condition of the evolutionary path $\boldsymbol{p}_\sigma$ reads:

$$\boldsymbol{p}_{\sigma,t+1} = \underbrace{(1 - c_\sigma)}_{\text{discount factor}} \boldsymbol{p}_{\sigma,t} + \sqrt{c_\sigma(2 - c_\sigma)} \underbrace{\sqrt{\mu_w} \, \boldsymbol{C}^{t-1/2} \, \overbrace{\frac{\boldsymbol{m}_{t+1} - \boldsymbol{m}_t}{\sigma_t}}^{\text{displacement of } \boldsymbol{m}}}_{\text{distributed as } \mathcal{N}(\boldsymbol{0}, \boldsymbol{id})}. \tag{A.4}$$

Thus step size at the k-th iteration is computed with cumulative step size adaptation [79] in line 28:

$$\sigma_{t+1} = \sigma_t \, \exp\left\{ \frac{c_\sigma}{d_\sigma} \Big(\frac{\|\boldsymbol{p}_{\sigma,t+1}\|}{\mathbb{E}\|\mathcal{N}(\boldsymbol{0}, \boldsymbol{id})\|} - 1 \Big) \right\}, \tag{A.5}$$

where $\mathbb{E}\|\mathcal{N}(\mathbf{0}, \boldsymbol{id})\|$ is the expected value of the distribution. Implementation may introduce an approximation of the expected value, that reads:

$$\begin{aligned}\mathbb{E}\|\mathcal{N}(\mathbf{0}, \boldsymbol{id})\| =& \sqrt{2}\,\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \\ \approx& \sqrt{n}\Big(1 - \frac{1}{4n} + \frac{1}{21n^2}\Big).\end{aligned} \tag{A.6}$$

The covariance matrix update consists of two parts: a rank-one, and rank-μ, see line 32. Rank-one update is realized through the evolutionary path $\boldsymbol{p}_c$ in line 30, similarly to the behaviour of the step size. A trigger h_σ is used to stall the update of $\boldsymbol{p}_c^{t+1}$, in formula we have:

$$h_\sigma = \mathbb{1}_{\|\boldsymbol{p}_{\sigma,t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4+\frac{2}{n+1})\mathbb{E}\|\mathcal{N}(\mathbf{0},\boldsymbol{id})\|}, \tag{A.7}$$

$$\boldsymbol{p}_{c,t+1} = (1 - c_c)\ \boldsymbol{p}_{c,t} + h_\sigma \sqrt{c_c(2 - c_c)}\ \sqrt{\mu_w}\ \frac{\boldsymbol{m}_{t+1} - \boldsymbol{m}_t}{\sigma_t}. \tag{A.8}$$

Rank-μ update is based on the evaluation, in line 31, of a covariance matrix $\boldsymbol{C}_\mu$, that is a weighted sum of covariances of successful steps of μ best individuals:

$$\boldsymbol{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_t)}{\sigma_t} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_t)^T}{\sigma_t}. \tag{A.9}$$

Therefore the global update condition, line 32, is a weighted sum of previously accumulated information, coefficient $1 - c_\mu - c_1$, with new terms given by rank-one and rank-two updates, coefficients c_1 and c_μ. In formula it reads:

$$\boldsymbol{C}_{t+1} = (1 - c_1 - c_\mu)\ \boldsymbol{C}_t + c_1\ \underbrace{\boldsymbol{p}_{c,t+1}\boldsymbol{p}_{c,t+1}^T}_{\text{rank-one update}} + c_\mu\ \underbrace{\boldsymbol{C}_\mu^+}_{\text{rank-}\mu\text{ update}}, \tag{A.10}$$

where $1 - c_1 - c_\mu \geq 0$.

Now that we have described the mechanism of a possible implementation of CMA-ES, note that update conditions for both mean and covariance matrix are supposed to maximize a likelihood. That is exactly the principle that we have already discussed in Section 2.2.3.

Algorithm 1

procedure $(\mu/\mu_w, \lambda)$-CMA-ES $(f, \boldsymbol{\theta}_0)$
  Given $n \in \mathbb{N}^+$ ▷ Dimension of the search space
  Set parameters:
    $\lambda \leq 4 + \lfloor 3 \log n \rfloor$ ▷ Number of candidate solutions
    $\mu = \lfloor \lambda/2 \rfloor$ ▷ Number of best parent individuals
    $w_i = \frac{\log(\mu+1/2) - \log i}{\sum_{j=1}^{\mu} \left(\log(\mu+1/2) - \log j \right)}$ For i=1 to μ
    $\mu_w = \frac{1}{\sum_{j=1}^{\mu} w_i}$
    $c_\sigma = \frac{\mu_w + 2}{n + \mu_w + 3}$
    $d_\sigma = 1 + c_\sigma + 2 \max\left(\sqrt{\frac{\mu_w - 1}{n+1}} \right)$
    $c_c = \frac{4}{n+4}$
    $c_1 = \frac{2 \min(1, \lambda/6)}{(n+1.3)^2 + \mu_w}$
    $c_\mu = 2 \frac{\mu_w - 2 + 1/\mu_w}{(n+2)^2 + \mu_w}$
  Initialization:
    $\boldsymbol{m}_0 \in \mathbb{R}$
    $\sigma_0 > 0$ ▷ Starting mutation step size
    $\boldsymbol{p}_{\sigma,0} = 0$ ▷ Evolutionary path in $\mathbb{R}^n$
    $\boldsymbol{p}_{c,0} = 0$ ▷ Evolutionary path in $\mathbb{R}^n$
    $\boldsymbol{C}_0 = \boldsymbol{id}$ ▷ Covariance matrix
    $t \leftarrow 0$
  Main loop:
  while Stopping criterion is met **do**
    for k=1 to λ **do**
      $\boldsymbol{x}_k = \boldsymbol{m}_t + \sigma_t \, \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}_t)$ ▷ Sampling λ candidates from a multivariate normal distribution
      $\boldsymbol{f}_k = f(\boldsymbol{x}_k)$ ▷ Evaluation of individuals' loss functions
    end for
    $\boldsymbol{m}_{t+1} \leftarrow \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$ ▷ Update mean value, $i : \lambda$ denotes i-th best individual on f
    $\boldsymbol{p}_{\sigma,t+1} \leftarrow (1 - c_\sigma) \, \boldsymbol{p}_{\sigma,t} + \sqrt{c_\sigma(2 - c_\sigma)} \, \sqrt{\mu_w} \, \boldsymbol{C}^{t-1/2} \frac{\boldsymbol{m}_{t+1} - \boldsymbol{m}_t}{\sigma_t}$ ▷ step size path update
    $\sigma_{t+1} \leftarrow \sigma_t \exp\left\{ \frac{c_\sigma}{d_\sigma} \left(\frac{\|\boldsymbol{p}_{\sigma,t+1}\|}{\mathbb{E}\|\mathcal{N}(\boldsymbol{0}, \boldsymbol{id})\|} - 1 \right) \right\}$ ▷ step size update
    $h_\sigma = \mathbb{1}_{\|\boldsymbol{p}_{\sigma,t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4 + \frac{2}{n+1}) \mathbb{E}\|\mathcal{N}(\boldsymbol{0}, \boldsymbol{id})\|}$ ▷ Trigger parameter evaluation
    $\boldsymbol{p}_{c,t+1} \leftarrow (1 - c_c) \, \boldsymbol{p}_{c,t} + h_\sigma \sqrt{c_c(2 - c_c)} \, \sqrt{\mu_w} \, \frac{\boldsymbol{m}_{t+1} - \boldsymbol{m}_t}{\sigma_t}$ ▷ Covariance matrix evolutionary path update for rank-one
    $\boldsymbol{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_t)}{\sigma_t} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_t)^T}{\sigma_t}$ ▷ Matrix evaluated as weighted sum of covariances of successful steps of μ best individuals
    $\boldsymbol{C}_{t+1} = (1 - c_1 - c_\mu) \, \boldsymbol{C}_t + c_1 \underbrace{\boldsymbol{p}_{c,t+1} \boldsymbol{p}_{c,t+1}^T}_{\text{rank-one update}} + c_\mu \underbrace{\boldsymbol{C}_\mu^+}_{\text{rank-}\mu\text{ update}}$ ▷ Covariance matrix update
    $t \leftarrow t + 1$
  end while

Bibliography

[1] Gustafson and John L. *Moore's Law*, pages 1177–1184. Springer US, Boston, MA, 2011.

[2] F. Bloch. Nuclear Induction. *Phys. Rev.*, 70:460–474, Oct 1946.

[3] L. Rottoli S. Forte. *Fisica Quantistica*. 2019.

[4] W.K Wootters and W.H Zurek. A Single Quantum Cannot be Cloned. *Nature*, page 299, 1982.

[5] S. Olivares. *Lecture Notes on Quantum Computing*. 2019.

[6] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, page 379–423, 1948.

[7] A. Einstein, B. Podolsky, and N. Rosen. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Phys. Rev. 47*, pages 777–780, 1935.

[8] A. Aspect, P. Grangier, and G. Roger. Experimental tests of realistic local theories via Bell's theorem. *Phys. Rev. 47*, page 460, 1981.

[9] A. Aspect, P. Grangier, and G. Roger. Experimental realization of Einstein-Podolsky-Rosen gedankenexperiment; a new violation of Bell's inequalities. *Phys. Rev. 47*, page 91, 1982.

[10] A. Aspect, P. Grangier, and G. Roger. Experimental test of Bell's inequalities using time-varying analyzers. *Phys. Rev. 47*, page 1804, 1982.

[11] *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, (1937):669–677, Jun 1995.

[12] A.M. Turing. On Computable Numbers, with an Application to the Entscheidungs problem. 1937.

[13] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, page 467–488, 1983.

[14] D. Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London*, page 97–117, 1985.

[15] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[16] J.D. Hidary. *Quantum Computing: An Applied Approach*. Springer Cham.

[17] J. A. Carlson, A. Jaffe, and A. Wiles. *The Millennium prize problems*. Providence, R.I.: American Mathematical Society., 2006.

[18] K. Bharti, A. Cervera-Lierta, T. Ha Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke, W. Mok, S. Sim, L. Kwek, and A. Aspuru-Guzik. Noisy intermediate-scale quantum (NISQ) algorithms. 2021.

[19] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[20] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15), Oct 2009.

[21] L. Wossnig, Z. Zhao, and A. Prakash. Quantum Linear System Algorithm for Dense Matrices. *Physical Review Letters*, 120(5), Jan 2018.

[22] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 1977.

[23] J.M. Gambetta, J.M. Chow, and Steffen. Building logical qubits in a superconducting quantum computing system. *Nature*, 2016.

[24] J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[25] C.J. Ballance, T.P. Harty, N.M. Linke, M.A. Sepiol, and D.M. Lucas. High-Fidelity Quantum Logic Gates Using Trapped-Ion Hyperfine Qubits. *Physical Review Letters*, 117(6), Aug 2016.

[26] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, and et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, Apr 2014.

[27] Google Research. Google AI quantum, 2017.

[28] IBM Research. IBM quantum experience, 2016.

[29] Rigetti. Rigetti Computing, 2017.

[30] Intel Corporation. Intel Quantum Computing, 2017.

[31] D-Wave Systems. The Quantum Computing Company, 2011.

[32] John Preskill. Quantum computing and the entanglement frontier, 2012.

[33] F. Arute, K. Arya, and R. et al. Babbush. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019.

[34] H. Zhong, H. Wang, Y. Deng, M. Chen, L. Peng, Y. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X. Yang, W. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N. Liu, C. Lu, and J. Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[35] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, Jarrod R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles. Variational Quantum Algorithms. 2020.

[36] J. Kempe, A. Kitaev, and O. Regev. The Complexity of the Local Hamiltonian Problem. *SIAM J. Comput. 35*, 2006.

[37] A. Garcia-Saez and J. I. Latorre. Addressing hard classical problems with Adiabatically Assisted Variational Quantum Eigensolvers, 2018.

[38] J Nocedal and S J Wright. Numerical Optimization. *Springer New York*, pages 120–122, 2006.

[39] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[40] J. Luis Morales and J. Nocedal. Remark on "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained Optimization". *ACM Trans. Math. Softw.*, 38(1), December 2011.

[41] J. Martens. Deep learning via Hessian-free optimization. *Proc. International Conference on Machine Learning.*, 2010.

[42] D. Kraft. A software package for sequential quadratic programming. *Tech. Rep. DFVLR-FB 88-28*, 1988.

[43] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[44] M. Lalee, J. Nocedal, and T. Plantenga. On The Implementation Of An Algorithm For Large-Scale Equality Constrained Optimization. *SIAM Journal on Optimization*, 8:682–706, 1998.

[45] R. Byrd, M. E. Hribar, and J. Nocedal. An Interior Point Algorithm for Large-Scale Nonlinear Programming. *SIAM J. Optim.*, 9:877–900, 1999.

[46] J. Gacon, C. Zoufal, G. Carleo, and S. Woerner. Simultaneous Perturbation Stochastic Approximation of the Quantum Fisher Information, 2021.

[47] J.C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998.

[48] F. James and M. Roos. Minuit - a system for function minimization and analysis of the parameter errors and correlations. *Computer Physics Communications*, 10(6):343–367, 1975.

[49] H. Dembinski and P. Ongmongkolkul et al. scikit-hep/iminuit. Dec 2020.

[50] R. Brun and F. Rademakers. ROOT - An Object Oriented Data Analysis Framework. Sep 1996.

[51] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

[52] R. Fletcher and M. J. D. Powell. A Rapidly Convergent Descent Method for Minimization. *The Computer Journal*, 6(2):163–168, 08 1963.

[53] W.C.Davidon. *SIAMJ.Optim.1.* 1991.

[54] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[55] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 01 1964.

[56] Conjugate-Direction Methods. In: Practical Optimization. Springer, Boston, MA. *Springer, Boston, MA.*, pages 145–173, 2007.

[57] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica 7*, pages 287–336, 1998.

[58] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. *Advances in Optimization and Numerical Analysis, eds. S. Gomez and J-P Hennart, Kluwer Academic (Dordrecht)*, pages 51–67, 1994.

[59] N. Hansen. The CMA Evolution Strategy: A Tutorial, 2016.

[60] W. Haynes. *Maximum Likelihood Estimation*, pages 1190–1191. Springer New York, New York, NY, 2013.

[61] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[62] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[63] T. Bayes and Price. LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

[64] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza. Qibo: a framework for quantum simulation with hardware acceleration, 2020.

[65] C. Bravo-Prieto, J. Lumbreras-Zarapico, L. Tagliacozzo, and J. I. Latorre. Scaling of variational quantum circuit depth for condensed matter systems, 2020.

[66] H. Abraham, AduOffei, and R. Agarwal at al. Qiskit: An Open-source Framework for Quantum Computing, 2019.

[67] F.A. Fortin, F.M. De Rainville, M.A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[68] N. Hansen, Y. Akimoto, and P. Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019.

[69] N. Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In *ACM-GECCO Genetic and Evolutionary Computation Conference*, Montreal, Canada, July 2009.

[70] R. D. Hipp. SQLite, 2020.

[71] L. Confalonieri. Towards the implementation of a quantum classifier, 2021.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[73] P. Baldi, K. Bauer, C. Eng, P. Sadowski, and D. Whiteson. Jet substructure classification in high-energy physics with deep neural networks. *Physical Review D*, 93(9), May 2016.

[74] L. Asquith et al. D. Adams, A. Arce. Towards an Understanding of the Correlations in Jet Substructure, 2015.

[75] Z. Holmes, K. Sharma, M. Cerezo, and P.J. Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus, 2021.

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

[77] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P.J. Coles. Noise-Induced Barren Plateaus in Variational Quantum Algorithms, 2021.

[78] I. Loshchilov. A Computationally Efficient Limited Memory CMA-ES for Large Scale Optimization, 2014.

[79] A. Chotard, A. Auger, and N. Hansen. Cumulative Step-size Adaptation on Linear Functions, 2012.