POLITECNICO DI TORINO

Masters's Degree in Mathematical Engineering

Masters's Degree Thesis

Quantum Machine Learning: continuous-variable approach with application to Neural Networks

Supervisors:

Prof. Francesco VACCARINO

Dott. Emanuele GALLO

Candidate:

Valeria ENCIN

A.A. 2019-2020

"If you think you understand quantum mechanics, you don't understand quantum mechanics."

Richard Feynman

"In fact, the mere act of opening the box will determine the state of the cat, although in this case there were three determinate states the cat could be in: these being Alive, Dead, and Bloody Furious."

Terry Pratchett

Summary

The work presented in this thesis is focused on quantum computing and in particular on the interplay between quantum devices and machine learning. This interaction gives birth to a relatively recent area of study, called Quantum Machine Learning (QML), which currently represents a hot research field. This work was carried out in collaboration with an important company with home in Turin, DATA Reply S.r.l., whose main focuses are Big Data, Artificial Intelligence, Machine Learning and Quantum Computing.

We start by introducing the fundamental concepts of quantum mechanics, presenting the topic in a formal way from a mathematical point of view while keeping things as simple as possible, as long as they allow us to understand quantum computing basics. After an overview on motivations, possible employments of quantum devices and an analysis of risks and benefits of their development, we proceed to explain quantum computing main concepts, such as qubits and quantum gates. The introduced formalism is the so-called Discrete Variable (DV) formalism, which is the most prominent choice in literature for describing quantum systems. However, as extensively explained during the course of this work, this approach presents several drawbacks: one of the main goals of this thesis is to introduce the alternative formalism of Continuous Variables (CV) and investigate how it can contribute to improve this research area.

Currently, one of the preferred tools for quantum machine learning algorithms are the Variational - or Parametrized - circuits: they allow to train the quantum devices in the same way as a classical neural network and allow to address ML problems, both supervised and unsupervised. After a detailed description of these tools, we proceed to introduce PennyLane, a Python library for QML which is very well suited for dealing with variational circuits.

In the core part of this work, we concentrate on some QML examples, in particular we analyze some applications in which using the CV formalism turns out to be the most convenient choice. Integration, more specifically Monte Carlo integration, is one of those fields which the CV formalism suits best; in a similar way, Gaussian process regression in its continuous variables version provides a great advantage, namely an exponential speedup, as long as some hypothesis are satisfied. We only provide a brief overview of these

applications, since we opted to leave more room to experiments involving variational classifiers and quantum neural networks.

Three use cases are analyzed, all involving the CV formalism. The first one is a variational classifier, employed in a supervised setting for the classification of a simple dataset. The second one is a quantum neural network, used to solve a problem of function fitting. The results and the chosen hyperparameters are presented, together with a formal explanation of the theoretical concepts behind them. The last use case, the most complex and detailed one, focuses on a time series forecasting task, carried out with a quantum neural network, and also presents a comparison with the classical approach.

Sommario

Il lavoro che viene presentato in questa tesi si concentra sul quantum computing ed in modo particolare sull'interazione tra i dispositivi quantistici ed il machine learning. Tale interazione dà vita ad un campo di studi relativamente recente, ovvero il Quantum Machine Learning (QML), che al momento costituisce un settore di ricerca molto attivo. Questo lavoro è stato eseguito in collaborazione con un'importante azienda torinese, DATA Reply S.r.l., i cui principali focus sono Big Data, Artificial Intelligence, Machine Learning e Quantum Computing.

Iniziamo con l'introduzione di alcuni concetti fondamentali della meccanica quantistica, presentando l'argomento tramite un rigido formalismo matematico, cercando al tempo stesso di spiegare i concetti nella maniera più semplice possibile, in modo da rendere comprensibili le nozioni di base riguardanti il computer quantistico. Dopo una panoramica sulle motivazioni, le possibili applicazioni di tali dispositivi ed un'analisi rischi-benefici del loro sviluppo, procediamo alla spiegazione dei principali concetti di quantum computing, come i qubit ed i gate. Il formalismo introdotto è il cosiddetto formalismo delle variabili discrete (DV), che è la scelta prediletta in letteratura per la descrizione dei sistemi quantum. Tuttavia, come verrà ampiamente spiegato nel corso del lavoro, l'adozione di tale formalismo presenta alcuni svantaggi: per questo motivo, uno dei principali obiettivi della tesi è quello di introdurre il formalismo alternativo, che è quello delle variabili continue (CV), e di investigare come tale area di ricerca possa trarne benefici.

Attualmente, uno degli strumenti più utilizzati negli algoritmi di quantum machine learning sono i Variational - o Parametrized - circuits: questi permettono di svolgere il training del dispositivo quantum come se fosse una rete neurale e permettono di affrontare problemi di ML, sia supervisionati che non supervisionati. Dopo una descrizione dettagliata di questi strumenti, procediamo introducendo PennyLane, una libreria di Python per il QML che è particolarmente adatta per interfacciarsi con tali circuiti.

Nella parte centrale di questo lavoro, ci concentriamo su alcuni esempi di QML, in modo particolare analizziamo alcune applicazioni in cui il formalismo delle variabili continue si rivela essere la scelta più conveniente da fare. L'integrazione, ed in modo particolare l'integrazione tramite metodi

Monte Carlo, è uno di quei campi in cui il formalismo CV si adatta molto bene; analogamente la regressione tramite processi Gaussiani nella versione basata sulle variabili continue fornisce un grande vantaggio, in particolare un miglioramento esponenziale in termini di velocità computazionale, a patto che alcune ipotesi siano soddisfatte. Forniamo solo una breve panoramica su questi due argomenti, preferendo lasciare più spazio agli esperimenti che riguardano i variational classifiers e le reti neurali quantistiche.

Analizziamo tre casi di studio, tutti basati sul formalismo CV. Il primo è un variational classifier, utilizzato in un contesto supervisionato per la classificazione di un semplice dataset. Il secondo è costituito da una rete neurale quantistica, che viene utilizzata per risolvere un problema di function fitting. Vengono presentati i risultati e la scelta degli iperparametri, senza trascurare una spiegazione formale dei concetti alla base di essi. L'ultimo caso di studio, il più articolato ed approfondito, si concentra su un algoritmo basato su una rete neurale quantum per la predizione di una serie temporale e presenta inoltre un confronto con l'approccio classico.

Acknowledgements

Desidero ringraziare per primi il relatore di questa tesi, prof. Francesco Vaccarino, e i tutti i ragazzi di Data Reply che ho avuto al mio fianco durante lo svolgimento di questo lavoro ed, ancora prima, durante il periodo di tirocinio. Il vostro supporto, la vostra disponibilità ed i preziosi consigli stati per me fondamentali: a voi va il mio più profondo ringraziamento.

Il pensiero va anche a tutti i docenti e colleghi del mio corso di studi, che tanto mi hanno insegnato e che hanno contribuito a farmi crescere sia come studentessa che come persona. In particolare mi fa piacere citare Andrea, Sara e Sofia, con cui ho speso molto del mio tempo e che sono persone che ho imparato sinceramente a stimare.

Un ringraziamento particolare, poi, va a Roberta e Mara, che, nonostante gli ostacoli e nonostante la distanza che ultimamente ha reso tutto più complicato, mi sono state vicine e di supporto, credendo sempre in me e festeggiando per i miei successi.

Un pensiero affettuoso va all'Istituto Flora, la mia casa torinese, in cui ho passato con serenità questi ultimi cinque anni di studi. Grazie alle Educatrici Apostole, per averci sempre fatto sentire come a casa, e grazie a tutte le ragazze che ho avuto il piacere di conoscere: vi porto sempre nel cuore, sapendo che anche voi fate lo stesso.

Infine, un grande grazie ai miei genitori ed a tutta la mia famiglia, per aver finanziato i miei studi e per aver sempre creduto in me: a tutti voi dedico questo lavoro con immensa gratitudine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General ideas about quantum computing

Quantum computing is the study of a non-classical computational model, based on the laws of quantum mechanics, representing the branch of physics studying the smallest objects of our physical world. In this setting, the laws of classical mechanics are no more applicable: this implies the use of new technologies and algorithms and the possibility to discover new ways in which computers can improve our lives. Quantum computers, in particular, exploit some effects of quantum mechanics, like entanglement and superposition, to solve some classes of problems in a more efficient way with respect to classical computers [Kopczyk, 2018]. As stated in [Calude and Calude, 2017], the quantum potential advantage consists in having faster computations with respect to the classical case. Furthermore, quantum computers are not meant to outperform their classical counterparts in every aspect: researchers are still working to identify which problems are suited for quantum speed-ups and in which fields these devices can be employed in the most successful way [Wittek, 2014].

In the latest decades, researchers have wondered if quantum computers will be able to perform tasks classical computers cannot solve in an efficient way: this is the quest for *quantum supremacy*, which, in the latest years, have involved big companies such as Google or IBM. Quantum supremacy is achieved when a quantum device manages to perform a given task which cannot be performed with any known algorithm on any classical machine in a reasonable time. This definition means that quantum supremacy is reached if a quantum computer can outperform **any** classical machine on one specific problem, even if the classical computers can perform all the remaining tasks better than a quantum device. It is important to remark the fact that the achievement of quantum supremacy does not end in making classical computers obsolete: in fact, as we will better see in this work, quantum and classical machines can work in synergy, exploiting the capacities of each

of them. In October 2019 Google announced the achievement of quantum supremacy: they used a processor of 53 qubits to sample an instance of a quantum circuit 10^6 times in only 200 seconds. Performing this task on a classical machine would take approximately 10^4 years ([Arute et al., 2019]).

Although we will see in more detail most of these concepts, we see here a brief overview of the main features of quantum computing, i.e. what's new with respect to the classical ones.

- **Superposition or coherence:** While objects obeying to laws of classical mechanics cannot exist in two different states at the same time (e.g. the flipping of a coin cannot give as a result head and tails at the same time), subatomic particles, like electrons, are allowed to have two states at once. The resulting particle is said to be in superposition of these two states, meaning that our "quantum coin", once laid on a table, is showing head with a certain probability p and, at the same time, tails with probability $1 - p$. This is a fundamental property at the basis of quantum mechanics and it will be useful to familiarize with the concept of qubit.

- **Measurements:** Given our quantum particle in a superposed state, one thing we are interested in is to discover its actual state: this introduces the concept of quantum measurement. The act of measuring or observing a quantum particle implies the decoherence of it, i.e. no matter if it was in a superposition of states, after measurement the particle will assume one of its possible states.

- **Entanglement:** This is one of the most interesting and perhaps counter-intuitive properties in quantum mechanics. In short, once a particle is entangled to another one, they form a whole system and any action performed on one of them will inevitably affect also the other one. The most mind-blowing fact, although, is that the same is still true even if the two particles are separated and brought at the two extremes of universe. This property is exploited in a large number of quantum algorithms.

1.2 Quantum computing in the NISQ era

Quantum computing is at an early stage but, despite this, several classes of algorithms have been developed in the last decades. However, theoretical results alone do not do all the work, because we currently dispose of a quite large number of algorithms which had birth when quantum hardware had not yet been developed. The actual realization of quantum hardware has been achieved only in the last years, therefore a lot of theoretical results were based on "ideal" (i.e. noise-free) devices. On the contrary, practical evidence shows that quantum devices are inherently noisy and, as a consequence,

algorithms developed on them should take this aspect into account. In order to avoid as much as possible the effect of noise on qubits, quantum computers must be kept at temperatures very near to the absolute zero, which is a not a very convenient solution.

Furthermore, it is not so obvious how to handle with devices with a large number of qubits: nowadays, quantum computers have a relatively small number of qubits and, for this reason, their computational capacity is limited. What sounds promising is qubit growth in the latest years: as we can find in [Ball, 2020], in 2016 IBM constructed a 5-qubit quantum device, in the first months of 2019 Google and IBM announced their 20-qubits devices, while now both companies run quantum computers having from 30 to 50 qubits. However, we cannot be so confident of a continuous growth in the near future, due to the necessity to keep the whole system cool for a sufficient time to perform a computation: this is a challenge that gets harder when numbers grow up.

For these reasons, we can state we currently live in the Noisy, Intermediate-Scale Quantum (or NISQ) era, a term coined by the quantum information theorist John Preskill of the California Institute of Technology in Pasadena. *Intermediate-Scale* indicates the size, in terms of qubits, of the current and near-future devices, i.e. from 50 qubits to a few hundreds. *Noisy*, instead, stands for the inherent noise that undergoes each quantum device and that is responsible for the decoherence of states. As Preskill states in [Preskill, 2018], NISQ era should not be seen as an end point, but as a step towards new technologies we will develop in the future. In fact, it is the term "Intermerdiate-Scale" itself that conveys this message: we are working towards much larger devices, with a huge number of qubits and error correction, often referred to as the fault-tolerant regime. We can be confident quantum technologies will have a positive impact on future societies, but we cannot predict how much close we are to this future.

1.3 Potential advantages

As already pointed out, it is still not so clear which problems could benefit from the development of a powerful quantum computer and one on the main reasons is that the great part of research was conducted only from a theoretical perspective. Nevertheless, from this research some interesting fields have emerged:

- **Cryptanalysis** A lot of current cryptographical systems - for example RSA, to name only one of them - rely on the fact that even for the most powerful supercomputer at our disposal, it is extremely expensive to factorize a large number in its prime factors. One of the first quantum computing algorithms is Shor's algorithm, which

shows how the employment of a quantum device can reduce exponentially the time it takes to factor a large number. If we disposed of a sufficiently powerful device, we could break the great part of current cryptosystems.

- **Simulation** For years classical machines have contributed to expand our knowledge of quantum systems, but the underlying complexity have forced researchers to opt for approximations which did not provide great advantages in extracting useful information. Quantum computers can really represent a crucial point in this field, since they are intrinsically suited to simulate quantum mechanical systems and, for this reason, study areas like quantum chemistry, materials science or nuclear physics could really benefit from them. For example, a quantum device with 50 qubits can encode the wave function of the water molecule, which on a classical device would require to diagonalize an Hamiltonian of the order of $\sim 10^{11}$ [Xanadu, 2019].

- **Machine Learning** The importance that of Machine Learning algorithms have nowadays makes Quantum ML a compelling area of study. This thesis focuses exactly on this relatively recent research field and has the purpose of investigating whether the quantum component can benefit ML and to which extent.

1.4 Risks and benefits of quantum computing

At the light of most recent discoveries, the achievement of noise-free large devices looks closer and closer, but we cannot safely state if and when this goal will be reached. Moreover, to achieve good results we will not only have to build physical devices but also develop algorithms converging from many branches of science, from chemistry to material science, from mathematics to physics. What must be said, however, is that research in the fields of quantum computing has bought to the discovery of new results in classical branches, for example physics or computer science, thus contributing to bring new knowledge and developments in this area ([National Academies of Sciences et al., 2019]).

On the other hand, despite the potential benefits research in this field could likely bring, quantum computing might represent a threat for national security. As we outlined in the previous section, any organization having a sufficiently powerful and stable quantum device in its hands could break current asymmetric cryptosystems, a scary issue that makes flashes apocalyptic pictures before our eyes. It is therefore necessary to be aware of such risks and begin to develop cryptosystems that will be immune to quantum cryptanalysis. The issue of information security is only the tip of an iceberg if we talk about the impact that quantum computers can possibly have on

our society. In the last decades, we have witnessed the huge impact classical computers had had on our world and we can only imagine how quantum devices could change our lives, influencing both economy and technology.

Chapter 2

Theoretical concepts

2.1 Quantum mechanics notation

In this chapter we are going to introduce some fundamental concepts at the core of quantum mechanics, which will turn out to be useful in the following. The two main references are [Kopczyk, 2018] and [Wittek, 2014].

2.1.1 State representation

In quantum mechanics, the two fundamental concepts are the state of a system, which contains statistical information about the system, and observables, which are the physical quantities we want to measure.

The state vector is an element of a Hilbert space $\mathcal{H}$, most often $\mathbb{C}^n$. To have a detailed overview about Hilbert spaces, the reader can have a look to [Klipfel, 2009]. The accepted notation is Dirac's bra-ket notation: a vector (or *ket*) is denoted by $|\psi\rangle$ while a covector in the dual Hilbert space is denoted by a *bra*: $\langle\psi|$. Therefore, $\langle\psi| = |\psi\rangle^\dagger$.

A state before measurement could be in a mixture of states $\{|x_1\rangle, \ldots, |x_n\rangle\}$: therefore a general state $|\psi\rangle$ can be represented as a linear combination of the *basis states* $\{|x_1\rangle, \ldots, |x_n\rangle\}$, having as coefficients the so-called *probability amplitudes*:

$$|\psi\rangle = \alpha_1 |x_1\rangle + \ldots + \alpha_n |x_n\rangle = \alpha_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \ldots + \alpha_n \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix}. \quad (2.1)$$

Stating that the quantum state is a linear combination of the basis states is equivalent to say that the state is in a *quantum superposition* of basis states.

The probability amplitudes have a fundamental role in determining what happens after the measurement of a state: in particular, the square norm $|\alpha_i|^2$ represents exactly the probability that the system will be in state $|x_i\rangle$

after measurement. Since we deal with probabilities, it is fundamental to require that their norm is equal to 1:

$$\sum_{i=1}^{n} \alpha_i = 1.$$

In summary, the superposition of a quantum system encapsulates the idea that the system is in all its possible states simultaneously. Only when a measurement is performed, the system collapses in one of the candidate states with a probability given by the square norm of the corresponding probability amplitude.

More generally, the probability of finding a particle $|\psi\rangle$ in state $|\phi\rangle$ is $\langle\phi|\psi\rangle$, where the previous notation represents the inner product of the two vectors.

As already stated, the second fundamental concept to be introduced is that of observables. Briefly, observables O are operands that, applied to a quantum state, return a real quantity. Keeping in mind that our state $|\psi\rangle$ is a vector in a Hilbert space, observables can be represented by matrices acting on such vectors. After a measurement the result can only be one of the basis states $|x_i\rangle$: for this reason, the observable matrix we are looking for has eigenvector $|x_i\rangle$ with eigenvalue x_i - these eigenvalues are the only values observables can take after being measured.

The following is the action of observable O on a state:

$$O\,|\psi\rangle \rightarrow x_i\,|x_i\rangle\,,$$

which in words means that the measurement of $|\psi\rangle$ performed by the observable collapses the state in superposition to state $|x_i\rangle$, with corresponding eigenvalue x_i. Such operation is irreversible, i.e. it is not possible to restore the superposed form of the collapsed state. Since eigenvalues must be real, the matrix O must be Hermitian and, as a consequence, eigenvectors of distinct eigenvalues are orthogonal. It follows that by choosing an orthogonal basis for each eigenspace, O holds n orthonormal eigenvectors, which constitute an orthogonal basis for $\mathbb{C}^n$. Given an orthogonal basis $|x_i\rangle$ of eigenvectors and a sequence of real numbers x_i, we can write a Hermitian matrix as $O = \sum_{i=0}^{n-1} x_i\,|x_i\rangle\,\langle x_i|$. In the basis of the $|x_i\rangle$ vectors, O is simply a diagonal matrix:

$$O = \begin{pmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & x_n \end{pmatrix}.$$

The measurement of a particular outcome x_i is highly influenced by the probability $|\alpha_i|^2$: the most suitable approach to get a reliable result is to perform multiple measurements of the system in the same state $|\psi\rangle$ and

measure the expectation value of observable O, that is defined as $\langle O \rangle = \langle \psi | O | \psi \rangle$.

To summarize:

- A measurement is an irreversible operation that collapses the state $|\psi\rangle$ in an eigenvector of the observable O;
- The result of the measurement is always the eigenvalue x_i corresponding to the aforementioned eigenvector. The probability of obtaining such eigenvalue x_i is given by $|\langle x_i|\psi\rangle|^2$, where $|x_i\rangle$ is the eigenvector corresponding to x_i. Such formula is also known as Born rule.

2.1.2 Density matrix representation

An alternative representation of quantum states is the *density matrix*, which is an operator obtained with the outer product of a state vector:

$$\rho = |\psi\rangle\langle\psi| .$$

A quantum state that can be written in such from is said to be *pure*. The idea under this definition is that by only knowing state $|\psi\rangle$ we can describe a system having all its particles in the same physical configuration. On the contrary, a *mixed state* is represented by the following density matrix:

$$\rho_{\text{mixed}} = \sum_{i=1}^{n} p_i |\psi_i\rangle\langle\psi_i| ,$$

where p_i represents the probability to find the mixed state in state $|\psi_i\rangle$.

2.1.3 Composite systems

We now want to consider a more complex and interesting setting, in which we are no longer interested in a single state $|\psi\rangle$ in a Hilbert space $\mathcal{H}$, but in a state in the composite system $\mathcal{H}_A \otimes \mathcal{H}_B$, where $\otimes$ is the tensor product between the two spaces. A vector in this composite space will be denoted as $|\psi\rangle_{AB} = |\psi\rangle_A \otimes |\psi\rangle_B = |\psi\rangle_A |\psi\rangle_B$. The possibility to combine two or more Hilbert spaces leads to one of the most useful and fascinating properties of quantum mechanics: *entaglement*. In short, composite states that can be written as the product of two states are called *separable*, the others are said to be *entangled*. From a more intuitive point of view, quantum entanglement of a composite system consists in having the states composing the system related to each other and such relationship doesn't depend on distance. Even at the two extremes of the Universe, the two entangled states have a strong bond: the measurement outcome of one of them simultaneously dictates the

outcome of the other one. The most common example of entangled state is Bell state, defined as

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

Such composite state cannot be written as the product of two states: the proof of this fact is straightforward and can be found for example in [Kopczyk, 2018].

2.2 Quantum computing

2.2.1 Qubit notation

Just as in classical computing, in which information is encoded by means of bits, in quantum computing the unit of information is given by a *qubit*. While in a classical computer a bit can only be in one of the two well-defined states, 0 or 1, in a quantum machine qubits obey to the laws of quantum mechanics and in particular they are governed by superposition. For this reason, each qubit can be written as a linear combination of two basis states $|0\rangle$ and $|1\rangle$:

$$\alpha_1 |0\rangle + \alpha_2 |1\rangle ,$$

where conventionally

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

and α_1 and α_2 are complex numbers. As we can see, the basis states are vectors of a two dimensional Hilbert space, most often $\mathbb{C}^2$.

From a physical point of view, a qubit is a two-level system like the two spin states $\pm 1/2$ of a particle, or like the horizontal and vertical polarization of a single photon.

Moving to higher-dimensional spaces, we can write for example the general state of two qubits as

$$\alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle ,$$

i.e. as a four dimensional vector. Generalizing a bit more, a state of n qubits is specified as a 2^n- dimensional complex vector.

Going on with our classical/quantum parallel, while a classical machine stores information as strings of bits like 01100011, a quantum computer uses the aforementioned tensor product notation to represent the same string. For this example, it obtains a vector with $2^8 = 256$ components, because in general a state for a 8 qubits system can be written as

$$|\psi\rangle = \alpha_1 |00000000\rangle + \ldots + \alpha_{256} |11111111\rangle .$$

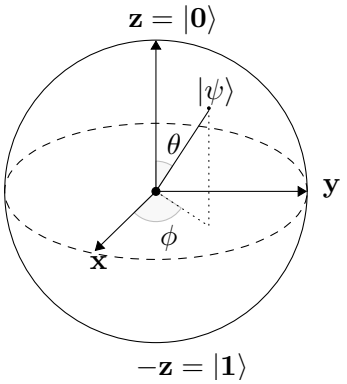


Figure 2.1: Representation of Bloch sphere

Therefore, in a quantum system of 8 qubits it is necessary to use 256 complex numbers, while only 8 are used in a classical computer. On the other hand, however, superposition of basis states generates an interesting property known as *quantum parallelism*, which arises from the fact that each quantum memory register can exists in a superposition of basis states. If we dispose of n qubits in the register, we can explore all 2^n possible combinations at once and each component of this superposition can be seen as a single argument of a function. Therefore, applying only once a function on the register, we get an output for each component of the superposition. This constitutes one of the main advantages of quantum computing, which allow to achieve an exponential speedup with respect to classical devices.

2.2.2 The Bloch sphere

We now see from a more formal point of view how qubit states are represented. It is a common convention to represent the general pure state of a qubit as

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

with $0 \leq \theta \leq \pi$, $0 \leq \phi \leq 2\pi$ and $\gamma \in \mathbb{R}$. The factor $e^{i\gamma}$ is referred to as the *global phase*: states differing only in the global phase are indistinguishable and for this reason such factor will be omitted from now on. With the two inequality constraints, the vector can be represented as a point on the surface of a sphere, which is called the *Bloch sphere*. Apart from the north and south pole, which represent $|0\rangle$ and $|1\rangle$, all other points of the surface of the sphere are inaccessible to classical bits, but not to pure qubits. On the contrary, mixed states represent vectors lying inside the Bloch sphere.

It is now convenient to introduce a concept that will extensively be used in the following: *Pauli matrices*, which have the property to be 2×2 unitary

and Hermitian matrices.

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

Adding also the identity matrix I, we get the basis of the Hilbert space of 2×2 complex matrices.

With this in mind, we can see that the computational basis is given by the Z axis of the Bloch sphere or, in other words, by the eigenvectors of σ_z. In fact,

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Actually, the possible basis are in infinite number: for completeness, we mention here also the basis formed by the eigenvectors of matrix σ_y, i.e. $|+i\rangle$ and $|-i\rangle$, where

$$|+i\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ i \end{pmatrix}, \qquad |-i\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -i \end{pmatrix},$$

and the basis formed by the eigenvectors of matrix σ_x, namely $|+\rangle$ and $|-\rangle$. These two states are defined as

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

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

and they can be derived from the application of Hadamard gate to the elements of the computational Z-basis, as we will better see in the following section. Figure 2.2 shows the representation of the described states on the Bloch sphere.

2.2.3 Quantum gates

Having defined quantum bits, we now need a way to manipulate them in order to obtain useful results. As in classical computers, also in quantum machines we apply transformations on qubits by means of *quantum gates* [Abraham Asfaw, 2020]. Maybe the greatest difference between quantum gates and classical ones is that the former are always reversible; for this reason, the number of input and output bits is always the same. From a mathematical point of view, quantum gates can be represented by matrices acting on state vectors; in particular, these matrices are unitary ones. In the following, we will see some of the most common gates.

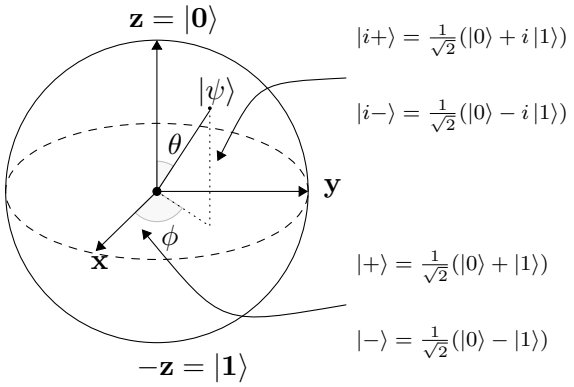


Figure 2.2: Representation of Bloch sphere

NOT gate

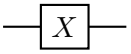
Also called X-gate, it is represented by the Pauli matrix σ_x. To see the effect of this gate, it is sufficient to apply it to the basis state vector:

$$\sigma_x |0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

and conversely

$$\sigma_x |1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle\,.$$

We can easily see that this gate simply swaps the amplitudes of states $|0\rangle$ and $|1\rangle$. Generally, the notation for this gate is the following:



Hadamard gate

It transforms the elements of the computational basis to a superposition of states. The corresponding matrix is

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

and we can see the effect of this gate on $|0\rangle$ and $|1\rangle$:

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

and

$$H |1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle\,.$$

Generally, the notation for this gate is the following:

Z gate

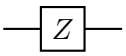
The Z gate leaves qubit $|0\rangle$ invariant and changes the sign of $|1\rangle$; its matrix is the Pauli Z matrix σ_z.

$$\sigma_z |0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

and

$$\sigma_z |1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle .$$

Generally, the notation for this gate is the following:

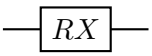


RX gate

The RX gate corresponds to a rotation around the Bloch sphere x-axis of the input qubit by a given angle. The only required parameter is indeed the rotation angle ϕ. The corresponding matrix is

$$RX(\phi) = e^{-i\phi\sigma_x/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix} .$$

In the following, we will denote this gate with this notation:



RY gate

The RY gate corresponds to a rotation of an input qubit around the y-axis of the Bloch sphere by a given angle. The only required parameter is indeed the rotation angle ϕ. The corresponding matrix is

$$RY(\phi) = e^{-i\phi\sigma_y/2} = \begin{pmatrix} \cos(\phi/2) & \sin(\phi/2) \\ \sin(\phi/2) & \cos(\phi/2) \end{pmatrix} .$$

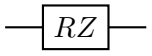
We will denote this gate with the following notation:

RZ gate

The RY gate corresponds to a rotation of an input qubit around the z-axis of the Bloch sphere by a given angle. The only required parameter is indeed the rotation angle ϕ. The corresponding matrix is

$$RZ(\phi) = e^{-i\phi\sigma_z/2} = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$$

We will denote this gate with the following notation:



CNOT gate

The complete name is Controlled-NOT gate. It takes two input bits, one is called control qubit, the other is the target qubit. After the application of CNOT gate, the control qubit remain unchanged, while a NOT gate is applied on the target depending on the value of control. The matrix is

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

Let's see how such gate modifies the vectors of the product basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$:

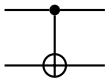
$$CNOT\,|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle$$

$$CNOT\,|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |01\rangle$$

$$CNOT\,|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |11\rangle$$

$$CNOT\,|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle$$

In a more compact way, $CNOT\,|A, B\rangle = |A, B \oplus B\rangle$. Generally, the notation for this gate is the following:

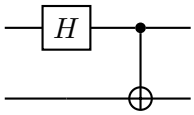


A CNOT gate combined with a Hadamard gate produces the already known Bell state:

$$H\,|0\rangle \otimes |0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}}$$

$$CNOT\frac{|00\rangle + |10\rangle}{\sqrt{2}} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}.$$

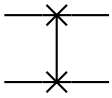
What we have seen is the first example of combination of gates, which applied one after the other on the $|0\rangle$ state, modify it to produce an output vector. Such combination of gates in called *quantum circuit*.



SWAP gate

The combination of three CNOT gates creates the SWAP gate, which, literally, swaps the role of first and second qubit in a two-dimensional basis vector:

$$|A, B\rangle \to |A, A \oplus B\rangle \to |A \oplus (A \oplus B), A \oplus B\rangle =$$
$$|B, A \oplus B\rangle \to |B, (A \oplus B) \oplus B\rangle = |B, A\rangle\,.$$



Combining this gates, but also many others we omitted in this section, it is possible to build a quantum circuit, i.e. an ordered sequence of gates and measurements.

We finally introduce the concept of universality in quantum circuits, since we will use it later in this thesis. Referring to [Abraham Asfaw, 2020], "*a set of quantum gates is said to be* universal *if any unitary transformation of the quantum data can be efficiently approximated arbitrarily well as sequence of gates in the set.*" In simple words, it means that any unitary operation can be written as a finite sequence of gates from this set.

2.3 Continuous variable quantum computing

2.3.1 Motivation and general ideas

Currently, quantum processors are implemented with a multiplicity of physical systems, such as photons, ions, atoms, solid state and superconductive devices [Andersen et al., 2015], [Bromley et al., 2020]. Implementation of fault-tolerant quantum computers is still a great challenge, because of the intrinsic fragility of quantum states. For this reason, the used physical systems must be fully isolated from the external environment, by keeping the systems at near-absolute zero temperatures or in vacuum environments. A valid alternative to this approach - employed by Xanadu team [Xanadu, 2020] - is to use photons instead of electrons to carry information and perform calculations: the reason is that photons are more stable and less affected by random noise from heat, since they have weak interactions with external environment, thus preventing decoherence. This allows to use a set of room-temperature operations, paving new paths to a world of more accessible quantum computers. However, the negative side is that photons do not interact with each other, making the implementation of two-qubit gates a challenge [Takeda and Furusawa, 2019].

Whatever the physical implementation, quantum information processing can be found in two different types, according to the observable - or the degree of freedom - used to encode information. If the observable has a discrete nature, meaning that its eigenvalues are discrete, we refer to discrete variable computation. This is the kind of approach we focused on in the previous section. On the contrary, if the observable has a continuous spectrum, we talk about continuous variable computation. In this section, we focus on the latter, since it can turn to be useful in a large number of situations. In fact, many quantum systems are intrinsically continuous, for example electromagnetic fields or light, to cite only some of them. This kind of computation has at its root crucial differences with respect to the DV one: the CV setting implies using an infinite dimensional Hilbert space, which changes a lot the way things are seen. Despite these differences, which will be better analyzed in the following, it is possible to embed qubit-based computations into the CV picture, so the CV model is as powerful as its qubit counterparts [Killoran et al., 2019b].

One way to practically implement continuous variable information pro-

tocols is to exploit the techniques of quantum optics. To be more precise, information is encoded in single photons - particularly in their degrees of freedom, such as amplitude and phase - and the state of photonic qubits can be described in the discrete photon-number basis. An alternative approach consists in representing the unit of information as a superposition of any continuous real value: this implies using continuous degrees of freedom of light, like the amplitude and phase quadratures of a field mode. Both these approaches will be explained in more detail later. Storing information in qumodes instead of qubits allows to carry a much larger amount of information, since photons propagate at the speed of light and offer large bandwidth for a high data transmission capacity [Xanadu, 2020].

An interesting and fundamental property of photons is that they satisfy the Plank-Einstein relation, which expresses the photon energy E as a function of the wave frequency f:

$$E = hf,$$

where h is the Plank constant $h = 6.62606896 * 10^{-34} J \cdot s$. In contexts where angular frequency is used, it is customary to use the reduced Planck constant $\hbar = h/2\pi$, which by convention is set in this work to 2, exactly as in [Xanadu, 2020].

Before analyzing in a more detailed way the continuous variable formalism, we can have a look to an interesting overview, highlighting the main differences between the DV and CV approaches: Table 2.1 contains this comparison.

	CV	**DV**
Basic element	Qumodes	Qubits
Relevant operators	Quadrature operators $\hat{x}$,$\hat{p}$ Mode operators $\hat{a}$,$\hat{a}^\dagger$	Pauli operators $\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z$
Common states	Coherent states $\|\alpha\rangle$ Squeezed states $\|z\rangle$ Number states $\|n\rangle$	Pauli eigenstates $\|0/1\rangle, \|\pm\rangle, \|\pm i\rangle$
Common gates	Rotation, Displacement, Squeezing, Beamsplitter, Cubic Phase	Phase Shift, Hadamard, CNOT, T Gate
Common measurements	Homodyne $\hat{x}_\phi$, Heterodyne $Q(\alpha)$, Photon-counting $\|n\rangle\langle n\|$	Pauli-basis measurements $\|0/1\rangle\langle 0/1\|, \|\pm\rangle\langle\pm\|, \|\pm i\rangle\langle\pm i\|$

Table 2.1: Comparison between CV and DV formalism, taken from [Xanadu, 2020].

2.3.2 Formalism

To begin with such complex topic, we start analyzing some concepts from theory of infinite dimensional vector spaces [Torre, 2017].

A vector space is a set of vectors

$$V = \{|\alpha\rangle, |\beta\rangle, \ldots\}$$

with a set of scalars $(a, b, \ldots)$ equipped with an addition rule for two vectors

$$|\alpha\rangle + |\beta\rangle = |\gamma\rangle,$$

such that the sum of two vectors is another vector, and a multiplication by scalar rule

$$a\,|\alpha\rangle = |\beta\rangle,$$

such that this product generates another vector. There exists a multiplicative identity element that leaves any vector unchanged when applied to it. The addition rule is commutative and associative, and the scalar multiplication is distributive over addition but also associative. Finally, there is a zero vector, which is an additive identity, so for every vector there exists its additive inverse vector such that they sum to zero.

Wave function space

In classical mechanics, the state of a system is completely described by a point (x, p) representing the position and momentum. However, in quantum mechanics it is impossible to know exactly at the same time both position and momentum of a particle: in fact, a fundamental rule is the position-momentum uncertainty principle, we will have the opportunity to analyze later in this work. While position and momentum are not able to completely describe a quantum system, the wave function ψ performs this task. The space of wave functions can be viewed as forming a vector space, the vector space of states. The set is the collection of all square-integrable, complex valued functions ψ, i.e.

$$\int_{-\infty}^{+\infty} |\psi(x)|^2 dx < \infty.$$

The scalars in the wave function space will be complex numbers, while the zero function will be $\psi(x) = 0$. Every vector space admits a basis, a subset of vectors $|e_i\rangle$ such that any vector can be written in a unique way as

$$|\psi\rangle = \sum_i c_i\,|e_i\rangle,$$

where c_i are scalar coefficients. Focusing on the vector space of wave functions, it has the property to be infinite dimensional and moreover every square-integrable function can be expressed as a linear combination - or superposition - of harmonic oscillatory stationary states ([Torre, 2017]):

$$\psi(x) = \sum_{n=0}^{\infty} c_n \psi_n(x).$$

Moreover, for a vector space of square-integrable, complex-valued functions we can define the inner product as

$$\langle\psi|\phi\rangle = \int_{-\infty}^{\infty} \psi^*(x)\phi(x)dx.$$

Let us introduce another fundamental concept we will use in the following. An operator T on a vector space is a rule that assigns to any input vector an output vector, denoted by $T|\psi\rangle$. We call T a linear operator if the output vector is a linear function of the input vector, i.e.

$$T(a|\alpha\rangle + b|\beta\rangle) = aT|\alpha\rangle + bT|\beta\rangle .$$

If the vector space is the space of square-integrable functions, then a linear operator T is a linear method of making a new square-integrable function $T\psi$ from any given function ψ. Interesting examples are the *position operator*

$$\hat{x}\psi(x) = x\psi(x)$$

and the *momentum operator*

$$\hat{p}\psi(x) = \frac{\hbar}{i}\frac{d}{dx}\psi(x),$$

which will be extensively used in the following.

Given a linear operator T, the set of all the eigenvalues of T is called its *spectrum*. If it consists of a discrete set of numbers, we say that the operator has a discrete spectrum. Instead, it is possible for some operators that their eigenvalues form a continuum, in other words they have a continuous spectrum. It is the case, for example, of the *momentum operator* $\hat{p}$. If we want to find its eigenvalues, we should solve $\hat{p}\psi = \lambda\psi$: we get

$$\lambda\psi = \frac{\hbar}{i}\frac{d}{dx}\psi(x) \qquad \rightarrow \qquad \frac{d}{dx}\psi(x) = \frac{i}{\hbar}\lambda\psi,$$

which is a separable first-order differential equation having as solution

$$\psi(x) = Ae^{\frac{i}{\hbar}\lambda x}.$$

Another example of an operator with a continuous spectrum is the *position operator*. We want a function $\psi(x)$ and a constant λ such that $\hat{x}\psi(x) = \lambda\psi(x) \quad \forall x$. The solution turns out to be, for any choice of λ and constant A,

$$\psi(x) = A\delta(x - \lambda).$$

Using Fourier analysis, we can build a delta function satisfying the above eigenvalue equation:

$$\delta(x - \lambda) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dk e^{ik(x-\lambda)}.$$

2.3.3 Continuous variable systems

As already said, we can describe a continuous variable system as a system whose fundamental degrees of freedom are related to operators having continuous spectra. The eigenstates of such operators form bases for the infinite-dimensional Hilbert space $\mathcal{H}$ of the system. Such a system can be modeled as a set of non-interacting quantum harmonic oscillators with different frequencies; each oscillator is referred to as a *mode* of the system [Adesso et al., 2014]. Then, the CV system of N modes is described by the composite Hilbert space $\mathcal{H} = \otimes_{k=1}^{N} \mathcal{H}_k$, where $\mathcal{H}_k$ is an infinite-dimensional Hilbert space associated to a single mode.

Operators

The most elementary CV system is the bosonic harmonic oscillator, defined via the canonical mode operators $\hat{a}$ and $\hat{a}^\dagger$, named *bosonic annihilation and creation operators*, which describe the creation and annihilation of energy quanta. The operators corresponding to two separate modes, $\hat{a}_1$ and $\hat{a}_2$, satisfy the following relations:

$$[\hat{a}_1, \hat{a}_1^\dagger] = [\hat{a}_2, \hat{a}_2^\dagger] = 1,$$

$$[\hat{a}_1, \hat{a}_1] = [\hat{a}_1, \hat{a}_2^\dagger] = [\hat{a}_1, \hat{a}_2] = [\hat{a}_2, \hat{a}_2] = 0,$$

or, more in general,

$$[\hat{a}_k, \hat{a}_{k'}^\dagger] = \delta_{kk'},$$

$$[\hat{a}_k, \hat{a}_{k'}] = [\hat{a}_k^\dagger, \hat{a}_{k'}^\dagger] = 0.$$

It is possible to equivalently describe the system in terms of the self-adjoint *quadrature operators*, which depend on the creation and annihilation operators as

$$\hat{x} = \sqrt{\frac{\hbar}{2}}(\hat{a} + \hat{a}^\dagger), \qquad \hat{p} = -i\sqrt{\frac{\hbar}{2}}(\hat{a} - \hat{a}^\dagger),$$

known as position and momentum operators, respectively. They fulfill the canonical commutation relation

$$[\hat{x}_k, \hat{p}_l] = i\hbar\delta_{kl}, \quad k, l = 1 : N$$

and they satisfy the eigenvector equations

$$\hat{x} \left|x\right\rangle_x = x \left|x\right\rangle_x, \qquad \hat{p} \left|p\right\rangle_p = p \left|p\right\rangle_p.$$

The previous expressions mean that $\left|x\right\rangle_x$ and $\left|p\right\rangle_p$ are the eigenstates of operators $\hat{x}$ and $\hat{p}$ respectively, with eigenvalues x and $p \in \mathbb{R}$. The eigenvectors $\left|x\right\rangle_x$ and $\left|p\right\rangle_p$ represent a set of orthogonal states spanning an infinite-dimensional Hilbert space: $\left|x\right\rangle_x$ forms a basis, while $\left|p\right\rangle_p$ is its conjugate.

Talking about the operators, instead, $\hat{p}$ is the generator of positive translations in position and $-\hat{x}$ is the generator of positive translations in momentum. A general pure quantum state $|\phi\rangle$ of a CV system can be written as a superposition of either $|x\rangle_x$ or $|p\rangle_p$.

It is often convenient to work with a vector grouping together the canonical operators:

$$\hat{\mathbf{R}} = (\hat{x}_1, \ldots, \hat{x}_N, \hat{p}_1, \ldots, \hat{p}_N)^T,$$

which allows us to write in a more compact form the commutation relation:

$$[\hat{R}_k, \hat{R}_l] = i\hbar\Omega_{kl}, \quad k, l = 1 : 2N$$

where $\mathbf{\Omega}$ is the $N-$mode symplectic form

$$\mathbf{\Omega} = \begin{pmatrix} 0 & \boldsymbol{I}_N \\ -\boldsymbol{I}_N & 0 \end{pmatrix}.$$

$\mathbf{\Omega}$ is therefore a $2N \times 2N-$dimensional matrix, real valued, invertible and antisymmetric, for which it holds $\mathbf{\Omega}^{-1} = \mathbf{\Omega}^T = -\mathbf{\Omega}$. This symplectic form will turn to be essential to translate the description of the system from the Hilbert space to a phase space formalism.

The space $\mathcal{H}_k$ for each mode k can be spanned by the Fock basis $\{|n\rangle_k\}$, $n \in \mathbb{N}$ of eigenstates of the *number operator* $\hat{n} = \hat{a}^\dagger\hat{a}$. Hence, the number operator $\hat{n}$ satisfies the following equation

$$\hat{n}\,|n\rangle = n\,|n\rangle\,,$$

where $|n\rangle$ are the eigenstates with relative eigenvalue n, and

$$[\hat{n}, \hat{a}^\dagger] = \hat{a}^\dagger, \quad [\hat{n}, \hat{a}] = -\hat{a}.$$

Moreover, it holds:

$$\hat{a}^\dagger\,|n\rangle = \sqrt{n+1}\,|n+1\rangle\,, \qquad \hat{a}\,|n\rangle = \sqrt{n}\,|n-1\rangle\,.$$

It is clearer from these equations the reason why $\hat{a}$ and $\hat{a}^\dagger$ are called annihilation and creation operators: they subtract and add a particle to the system. The eigenstates of the number operator, $|n\rangle$, are called number states and they form a discrete countable basis for the states of any single qumode.

For each mode k there exists a vacuum state $|0\rangle_k \in \mathcal{H}_k$, characterized by the absence of particles, such that

$$\hat{a}_k\,|0\rangle_k = 0.$$

Then the vacuum state for the complete system of N modes will be denoted by $|0\rangle = \otimes_k\,|0\rangle_k \in \mathcal{H}$.

An alternative to the base made of number states is the one constituted by *coherent states*, which are the right-eigenstates of the annihilation operator $\hat{a}_k$. Coherent states $|\alpha\rangle_k$ are obtained with the application of the single-mode Weyl *displacement operator* $\hat{D}_k$ to the vacuum: $|\alpha\rangle_k = \hat{D}_k |0\rangle_k$, where

$$\hat{D}_k(\alpha) = e^{\alpha \hat{a}_k^\dagger - \alpha^* \hat{a}_k}$$

and the coherent amplitude $\alpha \in \mathbb{C}$ satisfies $\hat{a}_k |\alpha\rangle_k = \alpha |\alpha\rangle_k$. Tensor products of coherent states for N different modes are obtained by applying the N-mode Weyl operators $\hat{D}(\boldsymbol{\xi})$ to the vacuum state $|0\rangle$. We can also define the operators $\hat{D}(\boldsymbol{\xi})$ in terms of the canonical operators $\hat{\boldsymbol{R}}$:

$$\hat{D}(\boldsymbol{\xi}) = e^{i\hat{\boldsymbol{R}}^T \boldsymbol{\Omega} \boldsymbol{\xi}}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2N}.$$

Therefore, $|\boldsymbol{\xi}\rangle = \hat{D}_{\boldsymbol{\xi}} |0\rangle$.

Phase space description

The states of a CV system are a set of positive semidefinite operators $\{\rho\}$ on the Hilbert space $\mathcal{H} = \otimes_{k=1}^N \mathcal{H}_k$. It could be difficult to deal with infinite-dimensional matrices: an equally complete and more convenient description of any quantum state ρ can be provided by the s-ordered *characteristic functions*

$$\chi_\rho^s(\boldsymbol{\xi}) = Tr[\rho \hat{D}(\boldsymbol{\xi})] e^{s||\boldsymbol{\xi}||^2/2}, \quad \boldsymbol{\xi} \in \mathbb{R}^{2N}.$$

The vector $\boldsymbol{\xi}$ belongs to the real $2N-$ dimensional space $\Gamma = (\mathbb{R}^{2N}, \boldsymbol{\Omega})$, called the *quantum phase space*. We know from Heisenberg uncertainty principle that in the quantum case it is not possible to describe the state of a system in terms of a single phase space point: for this reason, phase space regions are usually adopted to represent a particular state. The definition of the characteristic functions leads to an interesting fact: in the phase space picture, the tensor product structure is replaced by a direct sum structure, therefore the $N-$mode phase space decomposes as $\Gamma = \oplus_k \Gamma_k$, where $\Gamma_k = (\mathbb{R}^2, \omega)$ is the local phase space associated with mode k. Instead, by employing complex Fourier transform, it is possible to derive an alternative set of descriptions of the states of a continuous variable system; it is constituted by the real quasi-probability distributions W_ρ^s, which are able to completely describe such states:

$$W_\rho^s(\boldsymbol{r}) = \frac{1}{(2\pi)^{2N}} \int_{\mathbb{R}^{2N}} \chi_\rho^s(\xi) e^{-i\boldsymbol{r}\boldsymbol{\Omega}\boldsymbol{\xi}} d^{2N}\xi.$$

The $2N$ real arguments $\boldsymbol{r}$ of the function are the eigenvalues of the quadrature operators from $\hat{\boldsymbol{R}}$. These distributions are referred to as quasi-probability distributions because they sum up to unity, but they differ a bit from regular probability distributions. In particular, there could exist infinitely many

quantum states ρ characterized by a function W_ρ^s not being a regular probability distribution for some values of s, since it can present negative values or singular points in the phase space.

Gaussian states

A Gaussian state is defined as any state whose characteristic functions and quasi-probability distributions are Gaussian functions on the quantum phase space Γ. A general multi-mode Gaussian function has the form

$$f(\boldsymbol{x}) = C \exp\left(-\frac{1}{2}\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{x}\right),$$

where $\boldsymbol{A}$ is a positive definite $N \times N$ matrix.

Our starting point is the vacuum state $|0\rangle$. Other states can be created by evolving the vacuum state according to

$$|\psi\rangle = e^{-itH}|0\rangle,$$

where H is a bosonic Hamiltonian (i.e., a function of the operators $\hat{a}_i$ and $\hat{a}_i^\dagger$) and t is the evolution time. States where the Hamiltonian is at most quadratic in the operators $\hat{a}_i$ and $\hat{a}_i^\dagger$ (equivalently, in $\hat{x}_i$ and $\hat{p}_i$) are called Gaussian.

Generally speaking, a Gaussian state is completely characterized by its first and second canonical moments and nothing else. Defining $\left\langle \hat{O} \right\rangle_\rho = Tr[\rho\hat{O}]$ as the mean of operator $\hat{O}$ evaluated on state ρ, we can derive the expressions of these moments. The first moments $\boldsymbol{d}$ of a state ρ are defined as

$$d_j = \left\langle \hat{R}_j \right\rangle_\rho$$

and the second moments $\boldsymbol{\sigma}$ are

$$\sigma_{ij} = \frac{1}{2}\left\langle \Delta R_i \Delta R_j + \Delta R_j \Delta R_i \right\rangle, \quad \Delta\hat{\boldsymbol{R}} = \hat{\boldsymbol{R}} - \left\langle \hat{\boldsymbol{R}} \right\rangle_\rho,$$

which form the so-called covariance matrix $\boldsymbol{\sigma} = (\sigma_{ij})$. For N-mode Gaussian states ρ the quasi-probability function is completely characterized by $\boldsymbol{d}$ and $\boldsymbol{\sigma}$ and specifically has the form

$$W(\boldsymbol{r}) = \frac{\exp\left(-\frac{1}{2}(\boldsymbol{r}-\bar{\boldsymbol{r}})\boldsymbol{\sigma}^{-1}(\boldsymbol{r}-\bar{\boldsymbol{r}})\right)}{(2\pi)^N\sqrt{\det(\boldsymbol{\sigma})}},$$

having defined $\bar{\boldsymbol{r}} = \left\langle \hat{\boldsymbol{R}} \right\rangle_\rho$.

Coherent states are instances of Gaussian states and they have the property of being states with minimum Heisenberg uncertainty:

$$\Delta\hat{x}_k \Delta\hat{p}_k = \frac{\hbar}{2}.$$

Proof of this fact can be found in Appendix A. They are completely characterized by their first and second moments respectively:

$$\bar{\boldsymbol{r}} = 2\sqrt{\frac{\hbar}{2}}\left(\mathrm{Re}(\alpha), \mathrm{Im}(\alpha)\right), \quad \boldsymbol{\sigma} = \frac{\hbar}{2}\boldsymbol{I}.$$

Another class of states satisfying this property is that of *squeezed states*: they have unbalanced variances on the two quadratures for each modes, i.e. when they have a large variance on position, they have a small variance on momentum and vice-versa. The single-mode squeezing operator is defined as

$$\hat{S}(\zeta) = \exp\left[\frac{1}{2}(\zeta^* \hat{a}_k^2 - \zeta \hat{a}_k^{\dagger\,2})\right], \quad \zeta = se^{i\theta}.$$

The most general Gaussian pure state $|\psi\rangle_k$ is the *displaced squeezed state*, obtained applying on the vacuum state the displacement operator and the squeezing operator:

$$|\psi_{\alpha,s,\theta}\rangle_k = \hat{D}_k(\alpha)\hat{S}_k(\zeta)\,|0\rangle_k\,.$$

Here the parameters defining the state are the displacement vector $\alpha \in \mathbb{C}$, the squeezing degree $s \in \mathbb{R}^+$ and the squeezing phase $\theta \in [0, 2\pi]$.

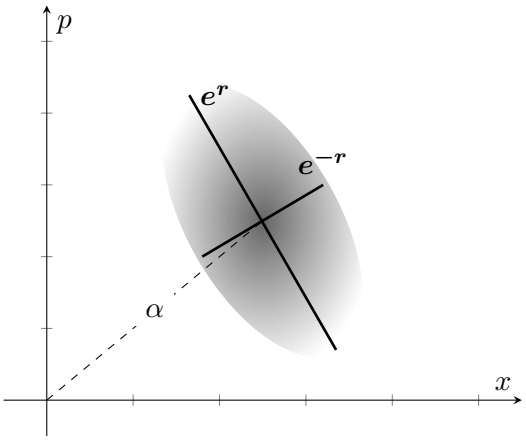


Figure 2.3: Simple representation of a Gaussian state for a single mode. Shape and orientation are determined by the displacement parameter α and squeezing parameter $\zeta = re^{i\theta}$.

We have seen so far that despite the infinite dimension of the associated Hilbert space, the complete description of an arbitrary Gaussian state ρ is given by the $2N \times 2N$ covariance matrix $\boldsymbol{\sigma}$. A question we can ask at this point is: given a $2N \times 2N$ real symmetric matrix, how can we check that it is a valid covariance matrix? It turns out that such matrix $\boldsymbol{\sigma}$ corresponds to a Gaussian state if and only if

$$\boldsymbol{\sigma} + i\frac{\hbar}{2}\boldsymbol{\Omega} \geq 0,$$

where the matrix inequality is understood in the sense that the eigenvalues of the quantity are non-negative. More in general, the previous condition must hold for the covariance matrix of any continuous variable state, which in the most general case can present moments of any order.

In many situations it turns out to be convenient to visualize Gaussian states as hyperellipsoids in the phase space corresponding to all the points within one deviation of the mean with respect to $\boldsymbol{\sigma}$. For example, for a single-mode system a Gaussian state can be seen as an ellipse in a 2-dimensional phase space with center in $(\langle\hat{x}\rangle, \langle\hat{p}\rangle)$ and with axis orientation depending on $\boldsymbol{\sigma}$ [Grimmer et al., 2018].

Fock states

Fock states - or number states - are the discrete counterpart of Gaussian states: in fact, they form a discrete countable basis for qumode systems [Killoran et al., 2019b]. They are denoted by $|n\rangle$, where n is a non negative integer and they are the eigenvalues of the number operator $\hat{n}$. Any Gaussian state can be expressed in the Fock basis; for example, a coherent state in this basis will have form

$$|\alpha\rangle = \exp\left(-\frac{|\alpha|^2}{2}\right) \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle .$$

Gaussian unitaries and the symplectic group

Having defined the basic quantities that represent a Gaussian state, one may wonder: how can we represent unitary transformations? The answer is constituted by a map from unitary transformations on a Hilbert space to real symplectic transformations on the first and second moments, defined as

$$\rho' = \hat{U}\rho\hat{U}^\dagger \rightarrow \begin{cases} \boldsymbol{d'} = \boldsymbol{S}\boldsymbol{d} \\ \boldsymbol{\sigma'} = \boldsymbol{S}\boldsymbol{\sigma}\boldsymbol{S}^T \end{cases} ,$$

where $\boldsymbol{S}$ is a symplectic matrix corresponding to the action of $\hat{U}$ on state $\hat{\rho}$. Gaussian quantum information has at its basis these symplectic transformations. The group of real symplectic matrices can be defined by

$$\boldsymbol{S}\boldsymbol{\Omega}\boldsymbol{S}^T = \boldsymbol{\Omega}.$$

It can be shown that any symmetric positive-definite matrix can be written in a diagonal form by means of a symplectic transformation. An interesting use of this result consists in finding the so-called *symplectic eigenvalues* of an arbitrary Gaussian state characterised by a covariance matrix $\boldsymbol{\sigma}$. In fact, the following theorem holds:

Theorem 2.3.1 *Given $\boldsymbol{\sigma}$, a $2N \times 2N$ positive definite matrix, there exists $\boldsymbol{S}$ in the group of real symplectic matrices diagonalizing $\boldsymbol{\sigma}$ as*

$$\boldsymbol{\sigma} = \boldsymbol{S} \begin{pmatrix} \boldsymbol{\nu} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\nu} \end{pmatrix} \boldsymbol{S}^T.$$

Collecting the eigenvalues in a single vector $\boldsymbol{\nu} = diag(\nu_1, \ldots, \nu_N)$, we get the so-called *symplectic spectrum* of $\boldsymbol{\sigma}$. For a physical state the symplectic eigenvalues must satisfy $\nu_k \geq 1,\ k = 1 : N$.

To sum up, we have constructed a new space description, alternative to the Hilbert space description: a comparison between the latter and the new phase space description can be seen in Table 2.2.

Property	Hilbert space $\mathcal{H}$	Phase space Γ
Dimension	∞	$2N$
Structure	$\otimes$	$\oplus$
Description	ρ	$\boldsymbol{d}, \boldsymbol{\sigma}$
Validity condition	$\rho \geq 0$	$\boldsymbol{\sigma} + i\boldsymbol{\Omega} \geq 0$
Unitary operations	$\hat{U}:\ \hat{U}^\dagger \hat{U} = \mathbb{I}$	$\boldsymbol{S}:\ \boldsymbol{S\Omega S}^T = \boldsymbol{\Omega}$
Spectra	$\hat{U}^\dagger \rho \hat{U} = \underset{0 \leq \lambda_i \leq 1}{\text{diag}} \{\lambda_j\}_{j=1}^{\infty}$	$\boldsymbol{S^T \sigma S} = \underset{1 \leq \nu_k < \infty}{\text{diag}} \{(\nu_k, \nu_k)\}_{k=1}^{\infty}$

Table 2.2: Comparison between Hilbert space description and phase space description.

2.3.4 Qumode-based computation

As seen in the previous sections, CV systems are characterized by modes, which are the main information-carrying units of CV quantum computers. Moreover, each mode is associated to the canonical mode operators $\hat{a}$ and $\hat{a}^\dagger$. It is sufficient to combine a certain number of modes and to modify them with a suitable sequence of gates to obtain a general CV quantum computation.

CV gates

Unitary operations are always linked to a generating Hamiltonian H via the following rule:

$$U = \exp(-itH).$$

The common approach is to classify each unitary by the degree of the generating Hamiltonian. We can build an N-mode unitary by applying a sequence of gates from a universal gate set, each gate acting only on one or two modes. As stated in [Killoran et al., 2019b], "*a CV quantum computer is said to be universal if it can implement, to arbitrary precision and*

with a finite number of steps, any unitary which is polynomial in the mode operators." Following the above classification, unitaries will be divided in Gaussian and non-Gaussian gates: the former are one or two-modes gates which are quadratic in the mode operators, the latter are single-mode gates with degree 3 or higher. We now see some of the most important gates.

- **Displacement gate** It is defined from the displacement operator we introduced in the previous section:

$$\hat{D}(\alpha) = e^{\alpha\hat{a}^\dagger - \alpha^*\hat{a}}.$$

The effect of applying this operator in a similarity transformation of the canonical mode operators results in their displacement:

$$\hat{D}^\dagger(\alpha)\hat{a}\hat{D}(\alpha) = \hat{a} + \alpha,$$

$$\hat{D}(\alpha)\hat{a}\hat{D}^\dagger(\alpha) = \hat{a} - \alpha.$$

Similarly, the action of this gate on position and momentum operators is the following:

$$\hat{D}^\dagger(\alpha)\hat{x}\hat{D}(\alpha) = \hat{x} + \sqrt{2\hbar}\mathrm{Re}(\alpha),$$

$$\hat{D}^\dagger(\alpha)\hat{p}\hat{D}(\alpha) = \hat{p} + \sqrt{2\hbar}\mathrm{Im}(\alpha).$$

The pure position and momentum displacement operators are defined as:

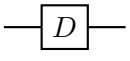
$$X(x) = \hat{D}(x/\sqrt{2\hbar}) = \exp(-ix\hat{p}/\hbar), \qquad X^\dagger(x)\hat{x}X(x) = \hat{x} + x,$$

$$Z(p) = \hat{D}(ip/\sqrt{2\hbar}) = \exp(ip\hat{x}/\hbar), \qquad Z^\dagger(p)\hat{p}Z(p) = \hat{p} + p.$$

$X(x)$ displaces a state in phase space by x in position, while $Z(p)$ displaces a state in phase space by p in momentum. It easy to show that the displacement operator acts in the following way on coherent states:

$$\hat{D}(\alpha)\left|\beta\right\rangle = \left|\alpha + \beta\right\rangle.$$

From now on, we will denote the displacement gate with the following:



- **Rotation gate** One of the simplest gates is the rotation or phase shift gate, defined as

$$\hat{R}(\phi) = e^{i\phi\hat{a}^\dagger\hat{a}}.$$

The application of such gate in a similarity transformation of $\hat{a}$ is

$$\hat{R}^\dagger(\phi)\hat{a}\hat{R}(\phi) = \hat{a}e^{i\phi},$$

while it rotates the position and momentum quadratures to each other:

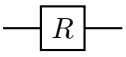
$$\hat{R}^\dagger(\phi)\hat{x}\hat{R}(\phi) = \hat{x}\cos(\phi) - \hat{p}\sin(\phi),$$

$$\hat{R}^\dagger(\phi)\hat{p}\hat{R}(\phi) = \hat{p}\cos(\phi) + \hat{x}\sin(\phi).$$

In words, the rotation gate rotates a state counterclockwise in phase space by an angle ϕ. The phase shifting operator also shifts the phase of the coherent state:

$$\hat{R}(\phi)\,|\alpha\rangle = \left|\alpha e^{i\phi}\right\rangle.$$

In the following, we will denote the rotation gate with:



- **Squeezing gate** This gate is defined as

$$\hat{S}(\zeta) = \exp\left[\frac{1}{2}(\zeta^*\hat{a}^2 - \zeta\hat{a}^{\dagger\,2})\right].$$

The application of such gate in a similarity transformation of $\hat{a}$ is

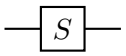
$$\hat{S}^\dagger(\zeta)\hat{a}\hat{S}(\zeta) = \hat{a}\cosh s - \hat{a}^\dagger e^{i\phi}\sinh s,$$

$$\hat{S}^\dagger(\zeta)\hat{a}^\dagger\hat{S(\zeta)} = \hat{a}^\dagger\cosh s - \hat{a}e^{-i\phi}\sinh s,$$

where $\zeta = se^{i\phi}$, $r \geq 0$ and $\phi \in [0, 2\pi)$. The squeeze gate affects the position and momentum operators as

$$\hat{S}^\dagger(\zeta)\hat{x}_\phi\hat{S}(\zeta) = e^{-s}\hat{x}_\phi, \qquad \hat{S}^\dagger(\zeta)\hat{p}_\phi\hat{S}(\zeta) = e^{s}\hat{p}_\phi.$$

It squeezes the position quadrature by a factor of s, while stretching the conjugate quadrature by $1/s$. In the following, we will denote the squeezing gate with:



- **Beam-splitter gate** For the annihilation and creation operators of two modes, i.e. $\hat{a}_1$ and $\hat{a}_2$, the beam-splitter is defined as

$$\hat{B}(\theta,\phi) = \exp\Big(\theta(e^{i\phi}\hat{a}_1\hat{a}_2^\dagger - e^{-i\phi}\hat{a}_1^\dagger\hat{a}_2)\Big).$$

The beam-splitter will transform the annihilation and creation operators according to

$$\hat{B}^\dagger(\theta,\phi)\hat{a}_1\hat{B}(\theta,\phi) = \hat{a}_1\cos\theta - \hat{a}_2 e^{-i\phi}\sin\theta = t\hat{a}_1 - r^*\hat{a}_2,$$

$$\hat{B}^\dagger(\theta,\phi)\hat{a}_2\hat{B}(\theta,\phi) = \hat{a}_2\cos\theta + \hat{a}_1 e^{i\phi}\sin\theta = t\hat{a}_2 + r^*\hat{a}_2,$$

having defined $t = \cos\theta$ and $r = e^{i\phi}\sin\theta$, which represent the transmittivity and reflectivity amplitudes of the beam-splitter. Therefore, the beam-splitter transforms in the following way a coherent state: $\hat{B}(\theta,\phi) = |\alpha,\beta\rangle = |\alpha',\beta'\rangle$ where

$$\alpha' = \alpha\cos\theta - \beta e^{-i\phi}\sin\theta,$$

$$\beta' = \beta\cos\theta + \alpha e^{i\phi}\sin\theta,$$

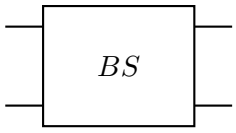
which is still a coherent state.

On the other hand, it is possible to see how the beam-splitter transforms the quadrature operators:

$$\begin{cases} \hat{B}^\dagger(\theta,\phi)\hat{x}_1\hat{B}(\theta,\phi) = \hat{x}_1\cos\theta - \sin\theta(\hat{x}_2\cos\phi + \hat{p}_2\sin\phi) \\ \hat{B}^\dagger(\theta,\phi)\hat{p}_1\hat{B}(\theta,\phi) = \hat{p}_1\cos\theta - \sin\theta(\hat{p}_2\cos\phi - \hat{x}_2\sin\phi) \end{cases}$$

$$\begin{cases} \hat{B}^\dagger(\theta,\phi)\hat{x}_2\hat{B}(\theta,\phi) = \hat{x}_2\cos\theta + \sin\theta(\hat{x}_1\cos\phi - \hat{p}_1\sin\phi) \\ \hat{B}^\dagger(\theta,\phi)\hat{p}_2\hat{B}(\theta,\phi) = \hat{p}_2\cos\theta + \sin\theta(\hat{p}_1\cos\phi + \hat{x}_1\sin\phi) \end{cases}$$

The beam-splitter gate is represented as



- **Quadratic phase gate** Its definition is

$$P(s) = \exp\Big(i\frac{s}{2\hbar}\hat{x}^2\Big)$$

and it acts on the annihilation operator as

$$P^\dagger(s)\hat{a}P(s) = \hat{a} + i\frac{s}{2}(\hat{a} + \hat{a}^\dagger).$$

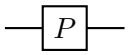
On the other hand, we can see from

$$P^\dagger(s)\hat{x}P(s) = \hat{x},$$

$$P^\dagger(s)\hat{p}P(s) = \hat{p} + s\hat{x}.$$

that it preserves position and shears the phase space.

Its representation is



- **Cubic phase gate** It is defined as

$$V(\gamma) = \exp\Big(i\frac{\gamma}{3\hbar}\hat{x}^3\Big).$$

Its action on the annihilation operator is

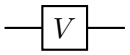
$$\hat{V}^\dagger(\gamma)\hat{a}V(\gamma) = \hat{a} + i\frac{\gamma}{2\sqrt{2/\hbar}}(\hat{a}+\hat{a}^\dagger)^2,$$

while when applied on the quadrature operators it returns

$$V^\dagger(\gamma)\hat{x}\hat{V}(\gamma) = \hat{x},$$

$$\hat{V}^\dagger(\gamma)\hat{p}\hat{V}(\gamma) = \hat{p} + \gamma\hat{x}^2.$$

Its representation is



- **Kerr gate** The Kerr interaction is given by the Hamiltonian

$$H = \hat{n}^2,$$

which is diagonal in the Fock basis. We can therefore define the Kerr gate as

$$K(\kappa) = \exp\big(i\kappa\hat{n}^2\big).$$

It is represented by

We can see that the first six gates are Gaussian, since they are quadratic in the mode operators; instead the last two are non-Gaussian. Using a suitable combination of Displacement, Rotation, Squeezing, and Beamsplitter Gates, it is possible to implement any multimode Gaussian gate, making these operators sufficient for quadratic unitaries.

CV measurements

In analogy with CV states and gates, measurements can be classified in Gaussian and non-Gaussian. The Gaussian class consists of two continuous types: *homodyne* and *heterodyne* measurements, while the central non-Gaussian measurement is given by photon counting. The "-dyne" measurements always preserve the Gaussian property of the initial state, both when acting conditionally and unconditionally: this is why they are called Gaussian [Serafini, 2017].

- **Homodyne measurement** Ideal homodyne detection is defined as a projective measurement onto the eigenstates of the position operator $\hat{x}$. By their nature, these states form a continuum, so homodyne measurements are intrinsically continuous, returning real values. In formulae, the state is projected onto the states

$$|x_\phi\rangle \langle x_\phi| ,$$

so the relative Hermitian operator is

$$\hat{x}_\phi = \cos\phi\hat{x} + \sin\phi\hat{p}. \tag{2.2}$$

When this operator is measured the outcome probabilities are

$$p(x_\phi) = \langle x_\phi|\rho|x_\phi\rangle$$

where $|x_\phi\rangle$ is an eigenvector of $\hat{x}_\phi$. For this reason, homodyne detection is also called quadrature detection. This means that performing a homodyne measurement consists in rotating the state by $-\phi$ and performing an $\hat{x}$-homodyne measurement. If we have a multimode Gaussian state and we perform a homodyne measurement on one of the modes, the conditional state on the remaining modes stays Gaussian.

Let's see now how homodyne detection is practically performed in optical set-ups [Tyc and Sanders, 2004]. A light field can be written as a superposition of two quadrature oscillations:

$$\hat{x}\cos(\omega t) + \hat{p}\sin(\omega t),$$

where ω is the wave frequency. Homodyne measurement consists in extracting the quadrature information of the field. To do so, homodyne detection matches the input signal with a strong coherent quadrature reference $|\alpha\rangle$ (where $\alpha = |\alpha|e^{i\phi}$, $|\alpha| \gg 1$) of a local oscillator: in this way it is possible to derive photon statistics depending on the phase φ of the local oscillator. The adjective "homodyne" is given by the fact that the input field is mixed with a laser field at the same frequency.

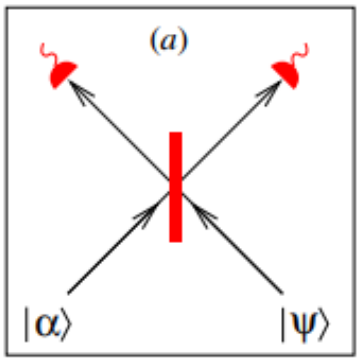


Figure 2.4: Homodyne detection scheme: the input state $|\Psi\rangle$ is mixed with the coherent state $|\alpha\rangle$ from a local oscillator and photons are counted at the two output ports. Source: [Tyc and Sanders, 2004].

By letting φ vary, it is possible to derive phase dependent properties of the signal state. In order to obtain non-classical properties of light, it is necessary to perform a phase sensitive measurement with respect to the quadrature (2.2).

In practice, to mix the signal fields with the local oscillator, a Beam-Splitter is used and the two output fields are then photodetected: the resulting statistics are analyzed to infer the quadrature statistics (See Figure 2.4). Good results are obtained using a 50:50 Beam-Splitter - to mix the optical signal and the coherent state - and the difference between the two photon counts at the output port - as the output statistics. In this case we talk about of balanced homodyne detection. The connection between photon statistics and quadrature phase can be proven by means of computations involving characteristic functions or quasi probability distributions: the interested reader can find more details in [Tyc and Sanders, 2004].

- **Heterodyne measurement** While the above described measurement involves only $\hat{x}$, heterodyne measurement can be considered as a simultaneous measurement of both the quadrature operators. Because these operators do not commute, they cannot be simultaneously measured without some degree of uncertainty. Heterodyne measurement consists in projecting the state onto the coherent states

$$\frac{1}{\pi} |\alpha\rangle \langle\alpha| .$$

The outcomes of the detection are labelled by α and the outcome probability on a state of a single mode ρ is given by

$$\frac{\langle\alpha|\rho|\alpha\rangle}{\pi}.$$

The practical way of implementing heterodyne detection in optical set-up is very similar to the scheme of homodyne detection, but the input mode is mixed with the coherent state at a different frequency: from here the adjective "heterodyne".

- **Photon counting** Photon counting is a non-Gaussian projective measurement given by
$$|n_i\rangle \langle n_i| .$$
It is an alternative measurement with respect to the previous two and it brings to light the "particle-like" nature of qumodes, instead of the "wave-like". Excluding outcome $n = 0$, this kind of measurement performed on a single mode of a multimode Gaussian state will result in making the left modes to become non-Gaussian. For this reason, photon-counting can be exploited to implement non-Gaussian gates.

Finally, we want to see what happens to a Gaussian state when a portion of the system modes is measured through -dyne detections [Serafini, 2017]. Given the initial Gaussian state of a system partitioned in subsystems A and B, with covariance matrix and first moments

$$\boldsymbol{\sigma} = \begin{pmatrix} \boldsymbol{\sigma}_A & \boldsymbol{\sigma}_{AB} \\ \boldsymbol{\sigma}_{AB}^T & \boldsymbol{\sigma}_B \end{pmatrix}, \qquad \bar{\boldsymbol{r}} = \begin{pmatrix} \bar{\boldsymbol{r}}_A \\ \bar{\boldsymbol{r}}_B \end{pmatrix},$$

let us determine the final covariance matrix and first moments of the $n-$mode subsystem A given a -dyne measurement outcome on the $m-$mode subsystem B. We need to evaluate the overlap between the initial state ρ of subsystem B and the general-dyne Gaussian state with displacement vector $\boldsymbol{r}_m$ and covariance matrix $\boldsymbol{\sigma}_m$. As stated in [Serafini, 2017], the $n-$mode subsystem A will have the following first and second moments:

$$\boldsymbol{\sigma}_A \mapsto \boldsymbol{\sigma}_A - \boldsymbol{\sigma}_{AB} \frac{1}{\boldsymbol{\sigma}_B + \boldsymbol{\sigma}_m} \boldsymbol{\sigma}_{AB}^T,$$

$$\bar{\boldsymbol{r}}_A \mapsto \bar{\boldsymbol{r}}_A + \boldsymbol{\sigma}_{AB} \frac{1}{\boldsymbol{\sigma}_B + \boldsymbol{\sigma}_m} (\boldsymbol{r}_m - \bar{\boldsymbol{r}}_B).$$

From this, we can note a couple of interesting things. First, the conditional evolution of the second moments does not depend on the measurement outcome. Second, if no correlations are present, i.e. if $\boldsymbol{\sigma}_{AB} = 0$, the above map reduces to the identity, meaning that measuring subsystem B will have no effect on subsystem A if the two are not correlated.

Chapter 3

Variational circuits for Quantum Machine Learning

3.1 Quantum Machine Learning

Machine learning is a branch of Artificial Intelligence that consists in making a computer automatically learn from experience without being explicitly programmed. Its purpose is that of recognising patterns and learn from data thus providing predictions. Machine learning algorithms process a large amount of data and perform tasks that are natural to the human brain, like image recognition or pattern identification. The basic task in machine learning is that of minimizing a constrained multivariate function to obtain as a result a decision function that maps input points to output points.

The term *learning* refers to three main branches of machine learning: supervised, unsupervised and reinforcement learning [Schuld et al., 2015]. In supervised learning, a computer is given a set if input-output pairs and has to learn a function that correctly maps a given input to an output. An example of this is pattern classification, in which vectors of input data have been assigned to different classes. Differently, unsupervised learning consists in detecting patterns in data without prior knowledge of it. A straightforward example of this kind of learning is clustering, which consists in grouping the input objects in such a way that objects belonging to the same cluster are more similar to each other than to those in other clusters. Finally, reinforcement learning has the goal to determine how a software agent should take actions in an environment of rules and goals in order to maximize its reward. The general idea is to punish or reward the agent on the basis of its strategy in order to teach him how to win.

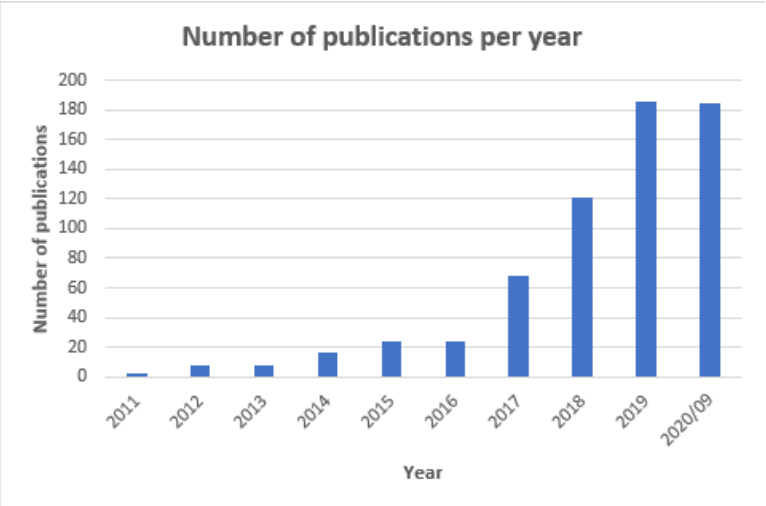


Figure 3.1: Number of publications containing "quantum machine learning" in the title in the last ten years. Please take into account that information about 2020 is obviously incomplete. Source: Google Scholar.

During latest years, researchers have started to study how performance of machine learning algorithms can be improved by the use of quantum computers, thus giving birth to a new branch of study: *quantum machine learning*. Quantum mechanics enters in the machine learning universe and gives birth to endless number of ways to improve machine learning algorithms. However, quantum machine learning is very young, in the sense it is not yet clear which commercial applications we can expect from it.

Nevertheless, a lot of companies have decided to invest on it and lately, quantum machine learning, together with quantum optimization, have become a hot field of research, with a large amount of work developed in this area: Figure 3.1 shows this increment. From this fact, we can easily understand there is are "positive vibrations" about this topic in scientific world. Quantum computers are indeed very well suited for machine learning tasks, since they handle *quantum information*, corresponding to vectors in high dimensional spaces. The use of sufficiently large and fault-tolerant computers could actually speed-up linear algebra operations, which constitute the core of many machine learning tasks. In fact, a very likely advantage from the introduction of quantum could be computational speed: quantum computers, even in these early stages of development, have much higher performances than their classical counterparts in solving some kinds of problems. It is therefore reasonable to hope that quantum machine learning could lead to a quadratic or even exponential speedup of already existing methods. However, execution time is not the only aspect of learning algorithms: for example, storage capacity is another field of interest. As stated in [Wittek, 2014], the quantum Hopfield networks are capable of storing exponentially more

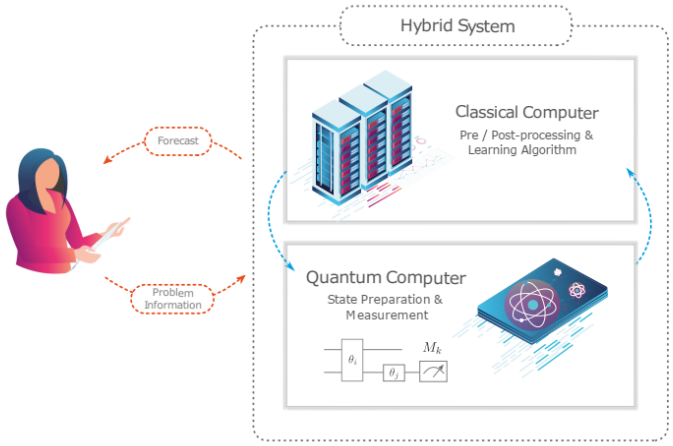


Figure 3.2: Intuitive representation of hybrid-quantum mechanism, taken from [Benedetti et al., 2019]

patterns than their classical analogues. These and many other aspects have motivated research in this field, opening a large number of roads towards a new era of computer science.

3.2 Parametrized quantum circuits

Parametrized quantum circuits, also known as *variational circuits*, are quantum algorithms typically composed of fixed gates, such as CNOT gates, and of gates depending on free and adjustable parameters, like Rotation transformations. The main approach for solving machine learning problems is to formalize such problems as variational optimization problems and use hybrid quantum-classical approaches to find approximate solutions. As stated in [Benedetti et al., 2019], the hybrid approach in the most general form consists of three components: the human, the classical computer and the quantum computer. The task of the human is to understand the problem and to select a suitable model to represent it. Data preprocessing is performed on the classical computer, which provides these inputs to the quantum machine. The latter reads the processed input data, prepares the quantum state according to the structure of the variational circuit and performs a measurement. These outcomes are postprocessed by the classical computer which updates the parameters in order to minimize a given cost function. The adjusted parameters are then provided again to the quantum computer in a closed loop.

We now see the three ingredients which compose a supervised learning model based on a variational circuit:

- Preparation of the initial state, usually the zero state;

- An encoder circuit $U_{\phi(x)}$ that takes the input and encodes it in the quantum circuit;
- A variational circuit $U_{\boldsymbol{\theta}}$ which constitutes the core of the model. It is followed by the measurement of an observable which consists in the estimation of expectation values.

3.2.1 Information encoding

There are several ways to encode data in a quantum circuit, all related to kernel methods, whose main goal is to embed data into a higher dimensional feature space where our problem can be solved in an easier way [Schuld and Killoran, 2019]. Given n qubits, the common approach is to start from the product state $|0\rangle^{\otimes n}$. Then, an encoder circuit $U_{\phi(x)}$ is applied to the input state: this can be interpreted as a feature map

$$x \to U_{\phi(x)} |0\rangle^{\otimes n}$$

to the high-dimensional Hilbert space of n qubits. Data can now be analyzed in this feature Hilbert space. If we take the inner product of two points in such space, we obtain a kernel function that allows to compute the distance between data points. Different kernels correspond to different models of pattern recognition: the technique consisting in switching between different kernels is said to be the *kernel trick* which, in short, consists in changing the data encoding.

Let us now see in a more detailed and formal way the theory behind kernel methods and data encoding. For a generality purpose, let us consider a classical dataset of M records and N features

$$\mathcal{D} = \{x^{(1)}, \dots, x^{(m)}, \dots, x^{(M)}\},$$

where $x^{(m)}$ is a $N-$ dimensional vector from a certain input set $\mathcal{X}$. Our goal is to embed data into a quantum system of n qubits. Kernel methods consist in using a distance measure $\kappa(x, x')$ for any input $x, x' \in \mathcal{X}$ with the purpose of constructing models which capture the properties of a data distribution. Such distance measure is connected to inner products in the so-called *feature space*.

Definition 3.2.1 *Let $\mathcal{F}$ be a Hilbert space, more specifically our feature space, and let $\mathcal{X}$ be the input set with x its generic entry. Then, we can define a* feature map *as a map $\phi : \mathcal{X} \to \mathcal{F}$ which maps input x to a vector $\phi(x) \in \mathcal{F}$, which is called* feature vector.

We can now introduce an additional concept, i.e. that of kernel functions.

Definition 3.2.2 *A function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is said to be a* kernel function *if the Gram matrix K, defined as $K_{m,m'} = \kappa(x^m, x^{m'})$, is positive semidefinite, namely if for any finite subset $\{x^{(1)}, \dots, x^{(M)}\}$ of $\mathcal{X}$ and for any $c_1, \dots, c_M \in \mathbb{C}$, it holds*

$$\sum_{m,m'=1}^{M} c_m c^*_{m'} \kappa(x^m, x^{m'}) \geq 0.$$

Given these definitions, we can write the inner product of two points in the feature space as

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}, \tag{3.1}$$

which defines a kernel $\kappa(x, x')$. Therefore, if we want to encode an input x into a feature vector $|\phi(x)\rangle$ of the Hilbert space, it is sufficient to exploit the feature map $\phi : \mathcal{X} \to \mathcal{F}$ to derive a kernel κ from equation (3.1).

Let us go on with some further definitions.

Definition 3.2.3 *Consider a Hilbert space $\mathcal{R}$ of functions $f : \mathcal{X} \to \mathbb{C}$ and let $\langle \cdot, \cdot \rangle$ be an inner product defined on $\mathcal{R}$. $\mathcal{R}$ is said to be a* Reproducing Kernel Hilbert Space *(RKHS) if every point evaluation is a continuous functional $F : f \to f(x)$ for all $x \in \mathcal{X}$. Equivalently, there exists a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ such that*

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

with $\kappa(x, \cdot) \in \mathcal{R}$ and $\forall f \in \mathcal{H}$, $x \in \mathcal{X}$.

Considering an input map $\phi : \mathcal{X} \to \mathcal{F}$ giving rise to a kernel $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}$, the corresponding RKHS has the form

$$\mathcal{R}_\kappa = \{f : \mathcal{X} \to \mathbb{C} : \ f(x) = \langle u, \phi(x) \rangle_{\mathcal{F}}, \forall x \in \mathcal{X}, u \in \mathcal{F}\}.$$

The functions $\langle u | \cdot \rangle$ in the RKHS associated with the feature map can be seen as linear models for which $u \in \mathcal{F}$ defines a hyperplane in the feature space.

A fundamental theoretical result is the following theorem.

Theorem 3.2.1 (Representer Theorem) *Let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a kernel, $\mathcal{D}$ a dataset made of pairs $(x^m, y^m) \in \mathcal{X} \times \mathbb{R}$ and $f : \mathcal{X} \to \mathbb{R}$ a class of model functions in the RKHS $\mathcal{R}_\kappa$. Moreover, assume we dispose of a cost function $\mathcal{C}$, quantifying the distance of the predicted outputs $f(x^m)$ from the targets y^m, which has a regularization term of the form $g(||f||)$ where $g : [0, \infty) \to \mathbb{R}$ is a strictly monotonically increasing function. Then any function $f^* \in \mathcal{R}_\kappa$ minimizing the cost function can be written as*

$$f^*(x) = \sum_{m=1}^{M} \alpha_m \kappa(x, x^m),$$

for some $\alpha_m \in \mathbb{R}$.

This theorem implies that instead of minimizing over an infinite dimensional RKHS we can simply solve the problem of finding parameters α_m of the previous expression.

One may now wonder how to combine the explained theory of kernels and the construction of variational circuits. The approach we follow consists in associating a quantum Hilbert space with a feature space and derive a kernel from the inner product of quantum states. From the previous results, this procedure will automatically give us a RKHS on which we can apply kernel theory. Specifically, given a feature map $\phi : \mathcal{X} \to \mathcal{F}$ which allows to encode an input x into a vector $|\phi(x)\rangle$, it is straightforward to derive its kernel from equation (3.1). Then, from the previous results, the derived kernel is the reproducing kernel of the RKHS $\mathcal{R}_\kappa$, whose functions are the inner product

$$f(x; u) = \langle u|\phi(x)\rangle \,,$$

with $|u\rangle \in \mathcal{F}$, which defines a linear model.

Going back to our circuit-based approach, as already stated the quantum feature map $x \to |\phi(x)\rangle$ can be seen as an encoder circuit $U_{\phi(x)}$ acting on the ground state $|0\rangle^{\otimes n}$:

$$U_{\phi(x)} |0\rangle^{\otimes n} = |\phi(x)\rangle \,.$$

The aforementioned linear model consists in the inner product between $|\phi\rangle$ and a general quantum state $|u\rangle \in \mathcal{F}$. It is therefore necessary to consider a second circuit U for which it holds $U |0\rangle^{\otimes n} = |u\rangle$: it specifies the hyperplane of a linear model in the Hilbert space. If the feature state $|\phi(x)\rangle$ is orthogonal to $|u\rangle$ then x lies on the decision boundary, if the inner product is positive, then x lies on the left of the hyperplane and on the contrary if the product is negative, the point is on the right of the plane.

In the following, we will use U as a variational circuit which will be parametrized, so $U = U(\theta)$: a hybrid training of θ will learn the optimal model $|u(\theta)\rangle = U(\theta) |0\rangle^{\otimes n}$, or, taking into account also the encoding circuit, $U(\theta)U_\phi |0\rangle^{\otimes n}$. From this state, a measurement will determine the output of the model. Depending on the measurement, this is not necessarily a linear model in the feature Hilbert space.

We present here some of the most common ways to encode classical information into quantum circuits (they are taken from [Hubregtsen et al., 2020] and [Xanadu, 2019]).

Basis encoding

It consists in encoding n binary features into a basis state of n qubits. Input data, i.e. classical information, must be a binary string: then the embedding is simply the bit-wise translation of classical data into the relative states in

the quantum system. In mathematical terms, we have $x^{(m)} \in \{0,1\}^N$ and our dataset can be written as a superposition of the basis states:

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left| x^{(m)} \right\rangle .$$

Taking for example a dataset with $M = N = 2$, with $x^{(1)} = 01$ and $x^{(2)} = 10$, we have

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}} |01\rangle + \frac{1}{\sqrt{2}} |10\rangle .$$

The generic kernel for this encoding is the Kronecker delta

$$\kappa(x, x') = \langle i|j\rangle = \delta_{ij}.$$

Amplitude encoding

It encodes 2^n features into the amplitude vector of n qubits. If the dimension of the feature vector is smaller than the number of amplitudes, it is possible to pad it with a constant for the missing dimensions. A vector of dimension n will be encoded in quantum form using $\log_2(n)$ qubits or, equivalently, a classical 2^n-dimensional vector is represented by the amplitudes of a n qubit quantum state $|\psi_x\rangle$. The feature map $x \to |\psi_x\rangle$ thus provides an exponential advantage in terms of memory.

$$|\psi_x\rangle = \sum_{i=1}^{2^n} \alpha_i |x_i\rangle ,$$

where we have denoted with $|x_i\rangle$ the $i-$th computational basis state and with α_i the $i-$th component of vector x (see also equation (2.1)).

Note that, keeping in mind the hypothesis at the basis of (2.1), the amplitudes α_i must be normalized, i.e. $\sum_i |\alpha_i|^2 = 1$.

This type of encoding provides an advantage with respect to the previous one, since now data can also be integer or floating point.

We now want to see what happens to our example dataset if we perform amplitude encoding to it. As suggested in [Xanadu, 2019], we should concatenate all input records to obtain a single vector:

$$\alpha = C_{norm}\{x_1^{(1)}, \ldots, x_N^{(1)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)}\},$$

where C_{norm} is the necessary normalization constant. The representation of our dataset will therefore be

$$|\mathcal{D}\rangle = \sum_{i=1}^{2^n} \alpha_i |x_i\rangle .$$

As an example, if we want to encode the vector $x = (3.3, 0.0, 1.2, 0.0) \in \mathbb{R}^n, n = 4$ with amplitude embedding, we should do the following. First, we normalize it, obtaining $x_{norm} = \frac{1}{\sqrt{12.33}}(3.3, 0.0, 1.2, 0.0)$ and then we use two qubits to represent it as

$$|\psi_{x_{norm}}\rangle = \frac{1}{\sqrt{12.33}}(3.3\,|00\rangle + 1.2\,|10\rangle).$$

In this case, the kernel is linear:

$$\kappa(x, x') = \langle\psi_x|\psi_{x'}\rangle = x^T x'.$$

Product encoding

Also called angle encoding, it can be constructed using a single rotation of angle x_i for each qubit and can encode n features with n qubits. For example, x_i is encoded as

$$|\phi(x_i)\rangle = \cos(x_i)\,|0\rangle + \sin(x_i)\,|1\rangle\,, \qquad i = 1 : N.$$

This corresponds to a feature embedding circuit with the effect

$$U_\phi : x \in \mathbb{R}^N \to \begin{pmatrix}\cos(x_1)\\ \sin(x_1)\end{pmatrix} \otimes \ldots \otimes \begin{pmatrix}\cos(x_N)\\ \sin(x_N)\end{pmatrix}$$

and implies a cosine kernel

$$\kappa(x, x') = \prod_{i=1}^{N} \cos\big(x_i - x_i'\big).$$

3.2.2 The variational circuit

It is always possible to model any given problem with a parametrized quantum circuit: the issue is that such circuit could be infeasible to use due to a too large number of qubits or to an excessive depth. The aim of the study and construction of variational circuits is to get a circuit with suitable depth and with a feasible number of parameters.

The variational circuit $U_{\boldsymbol{\theta}}$ takes as input the set of free parameters $\boldsymbol{\theta}$ together with a set of fixed parameters: all of these enter in the circuit as the gates arguments. This allows to convert classical information into quantum information $U_{\boldsymbol{\theta}}\,|0\rangle$. Then, after the circuit processing, quantum data is again converted to classical data thanks to the evaluation of the expectation value of an observable $\hat{B}$:

$$f_i(x; \boldsymbol{\theta}) = \left\langle \hat{B} \right\rangle = \langle 0|U(x; \boldsymbol{\theta})^\dagger \hat{B} U(x; \boldsymbol{\theta})|0\rangle\,.$$

In Figure 3.3 we can see the structure of a variational circuit in its entirety as described above. First the zero state is prepared and, after this, an encoder circuit is applied. The concatenation of the encoder and the following model circuit allows to convert our data into quantum information. Finally, a measurement is performed, returning a classical result which contributes to the update of circuit parameters, which are provided again to the quantum component, generating a loop.

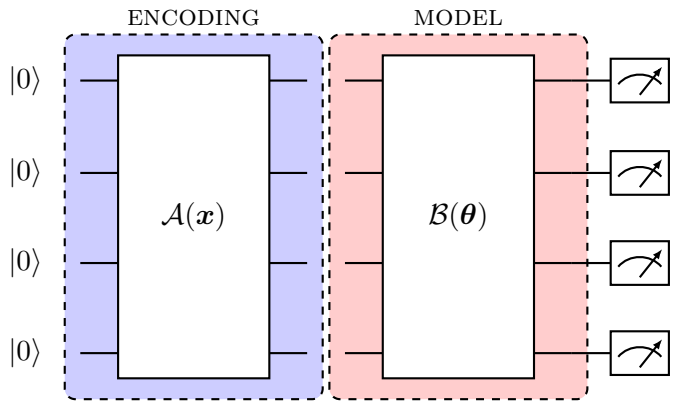


Figure 3.3: Schematic representation of a variational circuit in its most general form.

3.2.3 Circuit learning

As already seen, parametrized quantum circuits are used to solve data-driven tasks. The goal is to learn a given function and this problem is mathematically summarized by the task of minimizing a cost function $C(\Theta)$, which represents our objective function, with respect to the parameters Θ. The set Θ includes both the free parameters $\boldsymbol{\theta}$ and the input x. The problem of updating these parameters while minimizing the objective function is not a trivial one and can be addressed in different ways. In the next section, we present a Python library for quantum optimization which facilitates the task of hybrid optimization of variational circuits.

3.3 PennyLane

PennyLane is a hardware-friendly Python 3 library designed for quantum optimization, automatic differentiation and machine learning, which allows to perform hybrid quantum-classical computations [Bergholm et al., 2018]. Moreover, it is the first library for quantum machine learning, taking advantage of quantum hardware to calculate gradients and perform tasks such as machine learning and optimization. In fact, using PennyLane it is possible

to run machine learning algorithms directly on available quantum hardware platform, such as IBM Q.

3.3.1 Parametrized quantum circuits in PennyLane

Consider again the setting of the previous section, in which we have a parametrized quantum circuit having as arguments Θ. Our task is to minimize a predefined cost function $C(\Theta)$, which quantifies the quality of our solution, using the already explained hybrid quantum-classical approach. In general, we can imagine quantum and classical ingredients as *classical* and *quantum nodes*. These nodes are combined together according to the structure of a Directed Acyclic Graph (DAG), meaning that information flows from one node to its successors, with no possibility of incurring in loops. Each node may involve a number of input and output variables represented by the incoming and outgoing edges. Apart from this basic rule, the way gates are arranged, i.e. the *circuit architecture*, is arbitrary.

In PennyLane, a quantum node is the encapsulation of a function $f(x; \boldsymbol{\theta})$ that is executed on a quantum device, which can be quantum hardware or a classical simulator.

PennyLane is able to automatically compute gradients of each node and therefore it can compute the derivative of the final node with respect to the input variables. Each node in the variational circuit can depend on adjustable parameters $\boldsymbol{\theta}$ and can receive an input x from the previous nodes; finally it produces an output $f(x; \boldsymbol{\theta})$. Going through the circuit, information about derivatives is accumulated following the rules of *automatic differentiation*: in this way it is possible to compute the gradient of the objective function in order to minimize it using a gradient descent approach. In a nutshell, automatic differentiation is the ability for software to automatically compute derivatives of code with respect to free parameters. It is common to implement automatic differentiation by means of the backpropagation algorithm, which relies on the chain rule from calculus: in the next section we are going to see how it works in a more detailed way.

3.3.2 Gradient computation

PennyLane grounds its optimization on the gradient-descent algorithm and its variants. In order to approach the minimum of the objective function, at every step the single variables μ in Θ are updated according to the following rule:

Algorithm 1 Gradient descent step

```
1: procedure GRADIENT-DESCENT STEP
2:     for μ ∈ Θ do
3:         μ^(t+1) = μ^(t) − η^(t) ∂_μ C(Θ)
```

$\eta^{(t)}$ is the learning rate and can be adapted at each step t. The computation of the partial derivative $\partial_\mu C(\Theta)$ is necessary to compute the gradient of the cost function with respect to all variables Θ, $\nabla_\Theta C(\Theta)$. This computation is done by means of automatic differentiation techniques, such as backpropagation, a classical algorithm which has been adapted to the case in which computations are done in presence of quantum nodes.

There are three ways to compute derivatives of quantum nodes: analytical, numerical, or device-provided. By default, PennyLane uses the device or analytical derivatives whenever it can.

Analytical derivatives

The idea of this approach is based on the fact that the gradient of a quantum function $f(\boldsymbol{\theta})$ can often be expressed as a linear combination of other quantum functions, which use the same circuit and differ only in a small shift of the argument:

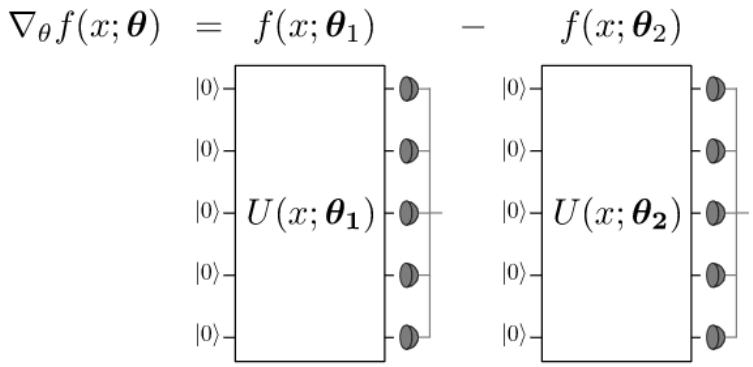


Figure 3.4: Visual representation of the fact that the gradient of a quantum circuit can be decomposed in a linear combination of quantum circuit functions from [Xanadu, 2019].

Denoting with $f(x;\boldsymbol{\theta}) = f(\mu)$ the output of a quantum node, we have

$$\partial_\mu f(\mu) = c(f(\mu + s) - f(\mu - s)),$$

where c, s are parameters typical for each kind of gate. This is called the *parameter shift* differentiation rule and gives exact gradients. We should note that it is different from finite differences: first, the shift s is usually a macroscopic shift, while in a finite difference method the increment is infinitesimal; second, the parameter shift formula gives exact derivatives, while the finite difference method only provides an approximation.

Let's now see in a more detailed way the mathematical computations behind this rule. The unitary transformation of our circuit can be decomposed

into the product of unitaries:

$$U(x;\boldsymbol{\theta}) = U_N(\theta_N)U_{N-1}(\theta_{N-1})\dots U_i(\theta_i)\dots U_1(\theta_1)U_0(x),$$

where x is the input and $\boldsymbol{\theta}$ are the parameters.

We can focus for a first moment on the case we have only two gates, $U_i(\theta_i)$ and $U_0(x)$. Then, the quantum circuit function will be

$$f(x;\theta_i) = \left\langle \hat{B} \right\rangle = \langle 0|U_0^\dagger(x)U_i^\dagger(\theta_i)\hat{B}U_i(\theta_i)U_0(x)|0\rangle = \langle x|U_i^\dagger(\theta_i)\hat{B}U_i(\theta_i)|x\rangle .$$

Setting $U_i^\dagger(\theta_i)\hat{B}U_i(\theta_i) = \mathcal{M}_{\theta_i}(\hat{B})$, we can compute the gradient of the quantum circuit function as

$$\nabla_{\theta_i} f(x;\theta_i) = \langle x|\nabla_{\theta_i}\mathcal{M}_{\theta_i}(\hat{B})|x\rangle .$$

In many cases we can express this gradient as a linear combination of the same transformation $\mathcal{M}$, but with different parameters:

$$\nabla_{\theta_i}\mathcal{M}_{\theta_i}(\hat{B}) = c(\mathcal{M}_{\theta_i+s}(\hat{B}) - \mathcal{M}_{\theta_i-s}(\hat{B})),$$

where c, s are parameters depending on the transformation $\mathcal{M}$ and not on the θ_i.

To complete our study, let us see what happens if we consider the case in which we have many gates. We can absorb any gate applied before i as

$$|\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1})\dots U_1(\theta_1)U_0(x)\,|0\rangle$$

and any gate applied after i as

$$\hat{B}_{i+1} = U_{i+1}^\dagger(\theta_{i+1})\dots U_N^\dagger(\theta_N)\hat{B}U_N(\theta_N)\dots U_{i+1}(\theta_{i+1})$$

thus obtaining

$$f(x;\theta_i) = \langle\psi_{i-1}|U_i^\dagger(\theta_i)\hat{B}_{i+1}U_i(\theta_i)|\psi_{i-1}\rangle = \langle\psi_{i-1}|\mathcal{M}(\hat{B}_{i+1})|\psi_{i-1}\rangle .$$

Therefore, the gradient will be

$$\nabla_{\theta_i} f(x;\theta_i) = \langle\psi_{i-1}|\nabla_{\theta_i}\mathcal{M}_{\theta_i}|\psi_{i-1}\rangle ,$$

which is the same as the simpler case, replacing $|x\rangle$ with $|\psi_{i-1}\rangle$ and $\hat{B}$ with $\hat{B}_{i+1}$. This means that in order to compute the gradient with respect to parameter i, it is sufficient to leave all other gates as they are and to modify only $U(\theta_i)$.

Numerical derivatives

Using this approach, the partial derivative estimation is computed by evaluating the output node $f(\mu)$ at several values which are close to μ. Therefore, the approximation of the derivative will be

$$\partial_\mu f(\mu) \approx \frac{f(\mu + \Delta\mu) - f(\mu)}{\Delta\mu}$$

if we use a forward finite-differences method and

$$\partial_\mu f(\mu) \approx \frac{f(\mu + 0.5\Delta\mu) - f(\mu - 0.5\Delta\mu)}{\Delta\mu}$$

if we use a centered finite-differences method.

Device derivatives

In this approach, we query the device for derivatives, if they are known. This clearly constitutes a speedup with respect to the previous methods, since information required to compute derivatives is already stored and reused from the forward circuit evaluation.

Chapter 4

Applications and experiments

4.1 Applications of continuous variable formalism

In this section we study in a more detailed way some of the most interesting employments of continuous variable formalism of quantum mechanics to the fields of Machine Learning and Mathematics. In particular, our goal is to understand in which fields and in which ways the continuous variable model can improve already existing classical or quantum discrete variable algorithms. The general idea is quite simple and intuitive: continuous variables offer a significant improvement, with respect to the discrete case, in those cases in which continuous quantities are involved. Discretizing such quantities can lead to lower performances and information loss: the use of a continuous model helps to avoid these drawbacks and allows to write more suitable and efficient models.

The first four applications are taken from the literature; in particular, sections 4.1.1 and 4.1.2 only present two examples of applications, but we did not try to implement these algorithms and to reproduce the presented results. Sections 4.1.3 and 4.1.4, instead, represent a step further, since we tried to implement the algorithms presented in the reference papers and we report here the obtained results. Finally, next section, 4.2, presents an independent study focused on an application of continuous variable quantum neural networks to the problem of time series forecasting.

4.1.1 Monte Carlo integration

Integration is one of those fields which the CV formalism suits best, since discretization could affect its performance in a negative way. In particular, following the paper [Rebentrost et al., 2018], we focus on a CV version of quantum Monte Carlo integration for the evaluation of single and multi-

dimensional integrals. As stated in the cited paper, under some conditions, this algorithm can provide quadratic speedups with respect to the classical case.

Let us first see a brief overview of Monte Carlo integration. Assume we are given a $n-$dimensional function $g(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$, whose integral over a region $\mathcal{R}$ of $\mathbb{R}^n$ will be

$$\mathcal{I} = \int_{\mathcal{R}} d\boldsymbol{x} g(\boldsymbol{x}).$$

In most cases the evaluation of this integral is hard and it could be a good idea to opt for MC sampling to find an approximate solution. In particular, the basic idea is to represent the integral as an expected value:

$$\mathcal{I} = \int_{\mathbb{R}^n} d\boldsymbol{x} p(\boldsymbol{x}) f(\boldsymbol{x}),$$

where $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function describing a random variable of the outcomes and $p(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ is a multidimensional probability distribution. Therefore, our integral can be approximated with MC using N_C samples as

$$\mathcal{I} \approx \tilde{\mathcal{I}} = \frac{1}{N_C} \sum_{i=1}^{N} f(x_i),$$

where $x_i \sim p(x)$. The probability to have an error greater than ϵ is given by

$$\mathcal{P}(|\mathcal{I} - \tilde{\mathcal{I}}| \geq \epsilon) \leq \frac{\sigma^2}{N_C \epsilon^2},$$

where σ^2 denotes the variance of $p(\boldsymbol{x})$. Hence, in order to have a constant error probability it is enough to choose $N_C = \mathcal{O}(\sigma^2/\epsilon^2)$.

The proposed CV quantum Monte Carlo algorithm allows a quadratic speedup with respect to the classical one in terms of the number of steps: it is enough to pick $N_C = \mathcal{O}(1/\epsilon)$. The high level description of the algorithm can be seen in Figure 4.1, representing the circuit used to implement QMC. The procedure consists of two stages:

- The first stage prepares the first mode using a unitary $\mathcal{G}$ such that its position wavefunction matches $\sqrt{p(\boldsymbol{x})}$. The second and third modes are squeezed in the position eigenbasis as much as possible and then $\mathcal{H}$ imprints $f(\boldsymbol{x})$ using the CV analog of a controlled rotation. The probability of performing a successful postselection on these two modes then gives the integral. This step does not provide a speedup with respect to the classical analog: on the contrary, the CV version of amplitude estimation used in the next stage allows to achieve better results.

- The second stage is to perform a CV version of amplitude estimation. Multiple applications of the three-mode unitary $\mathcal{Q}$ amplifies the amplitude corresponding to the projection. Adding a squeezed ancilla mode and instead applying the controlled unitary $\mathcal{Q}_c$ imprints the integral into the final mode. Measuring the final mode in the position eigenbasis then gives the integral as an expectation value. The total number of applications of $\mathcal{Q}_c$ can be $\mathcal{O}(1/\epsilon)$ for an approximation error ϵ.

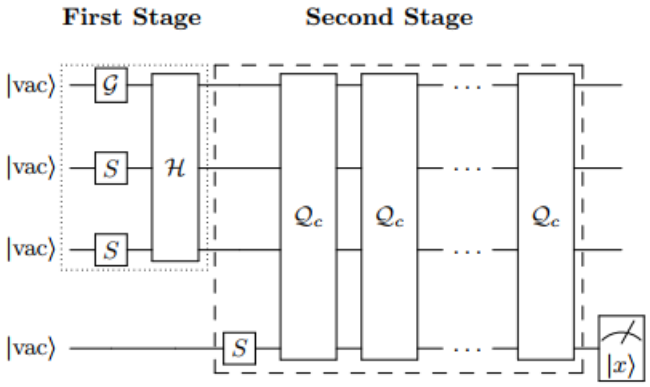


Figure 4.1: Quantum circuit diagram for one dimensional integration using CV QMC. Source: [Rebentrost et al., 2018].

The presented algorithm has evident advantages over its discrete counterpart, since it allows to perform multidimensional integration using a fixed number of modes, in contrast with the discrete case, which requires discretization and the use of a number of qubits that increases with the desired accuracy. However, some requirements must be satisfied for the algorithm to work:

- The function $f(\boldsymbol{x})$ must be bounded, i.e. $0 \leq f(\boldsymbol{x}) \leq 1$ for all $\boldsymbol{x}$;
- There exists a polynomial $h(\boldsymbol{x})$ which is related to the function $f(\boldsymbol{x})$ via $f(\boldsymbol{x}) = 1/|h(\boldsymbol{x})|^2$ with pointwise error at most ϵ_h on a compact set. Outside of the set, the integral is vanishingly small;
- There exists a unitary $\mathcal{G}$ that allows to efficiently prepare a quantum state which encodes $\sqrt{p(\boldsymbol{x})}$ in its amplitudes;
- An efficient continuous-variable gate sequence related to the polynomial function $h(\boldsymbol{x})$ can be constructed;
- A reflection around the computational initial state and the state defining the projective measurement can be efficiently implemented.

4.1.2 Gaussian process regression

Another interesting field in which the continuous variable formalism provides good results is that of Gaussian process regression. In [Das et al., 2018], the authors claim that the CV version of this model can provide an exponential speedup, thus reducing in a sensible way the computational time. The setting is that of supervised learning, in particular regression: the goal is to predict a continuous output, given an input training dataset.

As a general idea, Gaussian process regression is a Bayesian approach to regression: assuming a linear relation $\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$ (where $\boldsymbol{x}$ is the input vector, f is the function value, $\boldsymbol{y}$ is the observed target value and ϵ is a bias), we start specifying a prior distribution $P(f)$. Then, the posterior distribution can be easily derived from the Bayes formula

$$P(f|\boldsymbol{y}, X) = \frac{P(\boldsymbol{y}|X, f)P(f)}{p(\boldsymbol{y}|X)}$$

and includes both the information about the prior distribution and the dataset. In the following section, we see in a more detailed and formal way the mentioned concepts.

Classical Gaussian process regression

Let's start with a given dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}$ of N $d-$dimensional inputs $\boldsymbol{x}_i$ and scalar outputs y_i. We assume also that the outputs are noisy, i.e.:

$$y_i = f(\boldsymbol{x}_i) + \epsilon,$$

where f is the latent function and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ (i.i.d. Gaussian noise). A Gaussian process is a collection of random variables, any number of which have a joint Gaussian distribution [Rasmussen and Williams, 2016]. It is completely characterized by its mean and covariance function (also called kernel), which for a real process $f(\boldsymbol{x})$ is defined by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))],$$

where $m(\boldsymbol{x})$ is the mean function of $f(\boldsymbol{x})$. We can denote the process as

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')).$$

In the context of regression, our purpose is to predict the distribution of $f_* = f(\boldsymbol{x}_*)$ of a given new test point $\boldsymbol{x}_*$, .

Exploiting some nice properties of the normal distribution, we are able to write the joint distribution of the target values and the function value in the test point:

$$\begin{pmatrix} \boldsymbol{y} \\ f_* \end{pmatrix} \sim \mathcal{N}(\boldsymbol{0}, \mathbb{K}), \qquad \mathbb{K} = \begin{pmatrix} k(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \mathbb{I} & k(\boldsymbol{x}, \boldsymbol{x}_*) \\ k(\boldsymbol{x}_*, \boldsymbol{x}) & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{pmatrix},$$

where, without loss of generality, we have set the mean vector to zero.

Now, using again the properties of multivariate normal distribution we are able to write the conditional probability

$$P(f_*|\boldsymbol{y}) \sim \mathcal{N}(\bar{f}_*, \Sigma_*),$$

where

$$\bar{f}_* = k(\boldsymbol{x}_*, \boldsymbol{x})[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \mathbb{I}]^{-1} \boldsymbol{y}$$
$$\Sigma_* = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, \boldsymbol{x})[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \mathbb{I}]^{-1} k(\boldsymbol{x}, \boldsymbol{x}_*).$$

Therefore, in order to predict the distribution of f_*, it is sufficient to compute the mean $\bar{f}_*$ and the variance Σ_*.

Quantum Gaussian process regression

In order to efficiently compute the mean $\bar{f}_*$ and the variance Σ_*, the authors follow a process based on a quantum SVD algorithm for non-sparse low rank matrices. Therefore, let's start outlining the main steps of this method.

- Assume that $N = 2^n$ and call $K = k(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \mathbb{I} \in \mathbb{R}^{N \times N}$.
- Encode K as $\hat{K} = \begin{pmatrix} K & \\ & \mathbb{I} \end{pmatrix}$.
- To access $\hat{K}$, build a one-sparse Hermitian matrix H having only one non-vanishing element in each row: this can be approximated by a matrix $\tilde{H}$, having only even integers as entries.
- $\tilde{H}$ can be decomposed as the sum of matrices having eigenvalues ± 1, encoding efficiently a good approximation to all the information in the matrix $\hat{K}$.
- Next, we need to prepare states containing encodings of $|\boldsymbol{y}\rangle$ and $|\boldsymbol{k}_*\rangle$, where $\boldsymbol{k}_* = k(\boldsymbol{x}, \boldsymbol{x}_*)$. This can be achieved by applying a series of rotations to the vacuum state $|0 \dots 0\rangle$.

The routine to compute mean and variance is inspired by the algorithm above. First, our input state can be written in terms of the states $|\boldsymbol{y}\rangle$ and $|\boldsymbol{k}_*\rangle$ defined in the last bullet point. To this we append two CV resource modes in a squeezed state with squeezing parameter ξ, which should be kept as small as possible. Next, we apply the unitary $e^{i\gamma \frac{\hat{K}}{4N} \hat{N} p_R \tilde{p_R}}$, where $\hat{N} = \frac{\mathbb{I} - Z}{2}$ and γ is a parameter that can be adjusted at will. The next steps are a bit technical and we will not focus on them in this work, but it is worth to mention that, repeating the entire process M times, the error is $\epsilon \lesssim \frac{\gamma^2}{M^2 \xi^4} ||\hat{K}||^2$. We choose a small enough squeezing parameter and a large enough γ in such a way the error is ϵ, i.e. $\gamma \sim \frac{\xi^2}{\epsilon}$. Forcing the parameters ξ and γ to be in the specified range simplifies the computation of the mean and the variance.

4.1.3 Neural networks

Classical neural networks

Neural networks are one of the most successful models in Machine Learning and they are used for many tasks, such as regression and classification, to cite only a couple of them. The typical structure of a neural network can be seen in Figure 4.2, showing in particular a feedforward net.

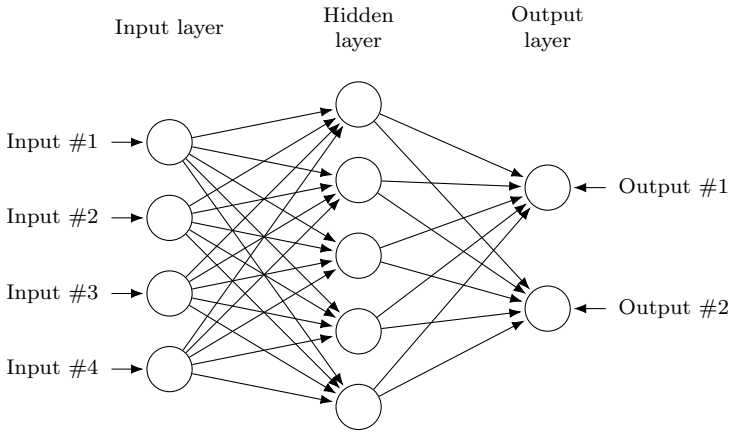


Figure 4.2: General structure of a neural network

The network begins with an input layer of real-valued neurons, which feed forward onto a series of one or more hidden layers. If the n neurons at one layer are represented by vector $\boldsymbol{x} \in \mathbb{R}^n$, then the m neurons in the subsequent layer are

$$\mathcal{L}(\boldsymbol{x}) = \varphi(W\boldsymbol{x} + \boldsymbol{b}),$$

where $W \in \mathbb{R}^{m \times n}$ is the weight matrix, $\boldsymbol{b} \in \mathbb{R}^m$ is the bias vector and φ is a non-linear activation function. Finally, the output layer performs an affine transformation on the last hidden layer and this time the activation function can be linear, or nonlinear, for example a function such as *softmax*.

Generally, both the weight matrix and the bias vector are characterized by free parameters θ_W and θ_b, while the activation function contains no free parameters and acts element-wise on its inputs. Each layer will have its own set of parameters: collecting all the parameters in vector $\boldsymbol{\theta}$, an N-layer neural network model is given by

$$\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{L}_N \circ \ldots \circ \mathcal{L}_1(\boldsymbol{x})$$

and maps an input $\boldsymbol{x}$ to a final output $\boldsymbol{y}$.

All these things are made possible by some universality results, which assure that, given a sufficient number of free parameters, any continuous function on a closed and bounded subset of $\mathbb{R}^n$ can be approximated by

feedforward neural networks to an arbitrary degree of accuracy. This is a fundamental result proving the power of neural networks, but it says nothing about how to find the approximation of our function and how much this procedure is efficient. In order to adjust the network parameters so that it fits the input data, it is customary to use variations of the gradient descent algorithm on a cost function characterizing the similarity between outputs of the neural network and training data.

Quantum neural networks in CV framework

In their paper [Killoran et al., 2019a], the authors propose a CV version of the described neural network, claiming it provides several advantages over the discrete-variable model. The most prominent one is that the CV model naturally fits the continuous, or *analog*, nature of neural networks. Moreover, it is able to easily apply non-linear transformations using the phase space picture, a task which qubit-based models struggle with.

In analogy with the classical case, we can define the concept of universality in the CV model, which is defined as the ability to approximate arbitrary transformations of the form

$$U_H = exp(-itH),$$

where the generator $H = H(\hat{x}, \hat{p})$ is a polynomial function of $(\hat{x}, \hat{p})$, the position and momentum operators. As the reader should remember from section 2.3.4, a set of gates is universal if it can be used to build any U_H through a polynomial-depth quantum circuit. Any circuit of this kind is composed of Gaussian transformations and non-Gaussian transformations (like the Kerr gate or the cubic phase gate) which correspond to non-linear functions in the phase space. More specifically, the universal gate set must contain Rotations, Displacements, Squeezing and Beam-Splitter transformations, combined with any non-Gaussian gate.

The scheme for quantum neural networks is based on two main concepts: first, the structure of classical networks, which have proven to be very efficient in a high number of situations, and second the variational circuit, which is one of the preferred ways to approach quantum algorithms on near-term quantum devices.

A general CV quantum neural network is built up as a sequence of layers, where each layer contains every gate from the universal gate set. In particular, a layer $\mathcal{L}$ consists of the successive gate sequence shown in Figure 4.3:

$$\mathcal{L} = \Phi \circ \mathcal{D} \circ \mathcal{U}_2 \circ \mathcal{S} \circ \mathcal{U}_1, \tag{4.1}$$

where:

- $\mathcal{U}_i(\boldsymbol{\theta}, \boldsymbol{\varphi})$ are general $N-$port linear inferometers containing beam-splitters and rotations;

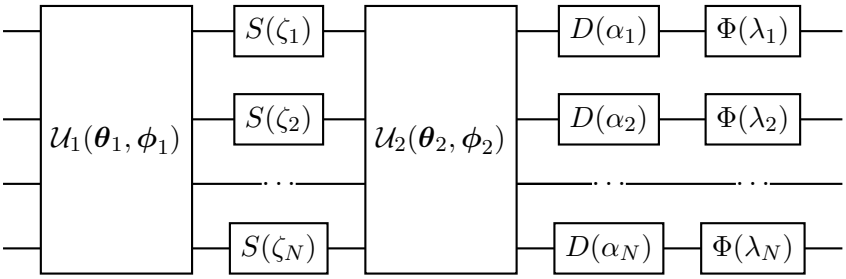


Figure 4.3: General structure for a single layer $\mathcal{L}$ of a CV neural network. The first four components carry out an affine transformation, followed by a final nonlinear transformation.

- $\mathcal{D} = \otimes_{i=1}^{N} \hat{D}(\alpha_i)$ is a collection of displacement operators, each acting independently on each mode;
- $\mathcal{S} = \otimes_{i=1}^{N} \hat{S}(s_i)$ is a collection of squeezing operators, in analogy with the previous bullet point;
- $\Phi = \Phi(\boldsymbol{\lambda})$ is some non-Gaussian gate.

The collection of gate variables $(\boldsymbol{\theta}, \boldsymbol{\varphi}, \boldsymbol{r}, \boldsymbol{\alpha}, \boldsymbol{\lambda})$ represents the free parameters of the network.

The expression (4.1) describes a unitary affine transformation on N qumodes; the application of a non-Gaussian gate Φ has a double effect, namely adding a non-linearity term and the achievement of universality. Using $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{p})$, we can write an expression for a single layer,

$$\mathcal{L}(\boldsymbol{z}) = \Phi(M\boldsymbol{z} + \boldsymbol{\alpha}),$$

which reminds the expression of a layer of the classical feedforward network.

This architecture can also receive classical inputs: to do this, it is sufficient to replace some of the free parameters with input data. Finally, the output of the network can be obtained by performing measurements and/or computing expectation values.

Having seen a high level description of both classical and quantum neural network structures, it is interesting to see how our quantum layer can embed the transformation $\mathcal{L}(\boldsymbol{x}) = \varphi(W\boldsymbol{x} + \boldsymbol{b})$ characterizing the classical layer. First of all, we need to encode classical data, in the form of a $N-$ dimensional vector, into position eigenstates as

$$\boldsymbol{x} \leftrightarrow |x\rangle = |x_1\rangle \otimes \ldots \otimes |x_N\rangle \,.$$

With our purpose in mind, i.e. performing the transformation $\mathcal{L}(\boldsymbol{x}) = \varphi(W\boldsymbol{x} + \boldsymbol{b})$, we can start focusing only on $W\boldsymbol{x} + \boldsymbol{b}$. In particular, we can write the singular value decomposition of the weight matrix, namely

$W = O_2 \Sigma O_1$, with O_1 and O_2 orthogonal matrices and Σ positive diagonal. This transformation can be carried out by a pair of inferometers, together with a squeezing gate. The required inferometers are characterized by only phaseless beam-splitters and are employed to perform the orthogonal transformations:

$$U_k(\boldsymbol{\theta}_k, \mathbf{0}) \left|\boldsymbol{x}\right\rangle = \left|O_k \boldsymbol{x}\right\rangle .$$

On the other hand, the squeezing transformation has the task of reproducing the effect of the positive diagonal matrix $\Sigma = \mathrm{diag}(\{s_i\}_{i=1}^N)$:

$$\otimes_{i=1}^N S(\zeta_i) \left|\boldsymbol{x}\right\rangle \propto \left|\Sigma \boldsymbol{x}\right\rangle ,$$

with $\zeta_i = \log(s_i)$. Finally, the displacement transformation has the role of adding the bias vector $\boldsymbol{b} = \{\alpha_i\}_{i=1}^N$, $\alpha_i \in \mathbb{R}$:

$$\otimes_{i=1}^N D(\alpha_i) \left|\boldsymbol{x}\right\rangle = \left|\boldsymbol{x} + \boldsymbol{b}\right\rangle .$$

Hence, the "inner part" of our transformation is given by the sequence of CV gates $\mathcal{D} \circ \mathcal{U}_2 \circ \mathcal{S} \circ \mathcal{U}_1$; what's missing is the non linear function φ. The latter can be achieved thanks to a certain type of non-Gaussian gates (see [Killoran et al., 2019a] for details) acting on each qumode:

$$\otimes_{i=1}^N \Phi(\lambda_i) \left|\boldsymbol{x}\right\rangle = \left|\varphi(\boldsymbol{x})\right\rangle .$$

Therefore, we have seen how transformation (4.1) represents the classical neural network layer on position eigenstates.

A fundamental property of quantum physics, which, as already stated, contributes to make quantum devices so advantageous, is that we can act not only on some fixed basis states, e.g., the states $\left|\boldsymbol{x}\right\rangle$, but also on superpositions of them, $\left|\psi\right\rangle = \int \psi(\boldsymbol{x}) \left|\boldsymbol{x}\right\rangle d\boldsymbol{x}$, where $\psi(\boldsymbol{x})$ is a multimode wavefunction. The general CV neural network provides a greater improvement with respect to its classical counterpart since it takes advantage of the power of universal quantum computation. For this reason, in general we expect that a CV quantum NN cannot be easily and efficiently simulated on a classical computer: as stated in [Killoran et al., 2019a], for many applications a classical device would require an exponential number of resources to approximate the quantum net.

On the other hand, it is interesting to remark that a classical neural network can be embedded into the quantum formalism. In fact, the quantum version can be used to run the classical version, by using the former in such a way it does not generate any quantum behaviour, such as entanglement or superposition.

Training of continuous variable quantum neural networks can be performed following two different approaches: through classical simulation or directly on quantum hardware. Classical simulation consists in evaluating the cost functions and the gradients with respect to the parameters of

each layer. However, as the size of the network grows, this task becomes less tractable: for this reason, direct evaluation on hardware will likely be necessary for large scale networks. The PennyLane library provides tools for training hybrid quantum-classical machine learning models, using both simulators and real-world quantum hardware.

An application to function fitting

One of the most straightforward and simple applications of neural nets is the problem of curve fitting: in this paragraph we will focus on the implementation of a CV quantum neural network to reproduce the action of a simple function $f(x)$ on a one-dimensional input vector x. The employed network is the one described in the previous section, but since the input vector is real valued, only one qumode is needed, thus simplifying a lot the structure. A single layer of the resulting network is therefore the following:

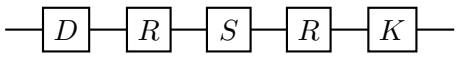
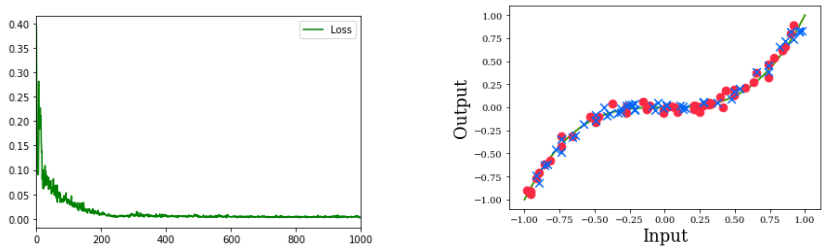


Figure 4.4: Single layer of the CV QNN for function fitting.

First of all, input data x is encoded in our circuit by means of the application of the displacement gate $\hat{D}(x)$ on the vacuum state $|0\rangle$. Then, the resulting state is fed forward the core of the algorithm, i.e. the variational circuit above characterized by free parameters to be trained. Finally, we measure the expectation value of the position operator $\hat{x}$, which is desirably very near, or even equal, to the value of $f(x)$. For this reason, we define a loss function as the mean squared error between the circuit output and the target function values: we aim to minimize this function, since in the ideal case there is exact correspondence between predicted and actual values and the loss is 0. However, it turns out to be convenient to minimize a cost function constituted by the loss defined above plus a penalty term. The latter is needed when performing numerical simulations of quantum circuits, since each qumode is truncated to a given cutoff dimension in the Fock space. During training it is possible that some gate parameters grow excessively, thus pushing the output state outside the given Fock space. For this reason, we penalize such unnormalized states having trace different from one. In this way, during training the loss function is kept as much as possible near to zero, while checking that the state trace is one.

Figure 4.5 shows the result of this algorithm for the fitting of function $f(x) = x^3$. The training was done for 1000 epochs on a 6 layers circuit, built as in Figure 4.3; since the system is constituted by a single mode, the two inferometers simply reduce to a Rotation gate. The optimization algorithm which turned out to perform best in this setting is Adam (see Appendix B for a description of this procedure) and the minimum attained loss is equal



(a) Loss (b) Plot

Figure 4.5: Loss function and plot for the function $f(x) = x^3$ on $[-1, +1]$ domain. The red circles represent training data, while the blue crosses the predictions.

to 0.002.

4.1.4 Variational classifiers

In this section we extend and further explore some of the concepts we introduced in Section 3.2 as found in [Schuld and Killoran, 2019] and see an application to a real problem. In particular, we will see how a parametrized quantum circuit can be employed for an elementary classification task in the continuous variable framework, thus becoming a *CV variational classifier*. Given a simple dataset, we want our machine to learn the structure of data and predict the correct label of each point.

As the reader should remember, the basic idea for building a variational circuit is very similar to that of kernel methods: we define a feature map $\phi : \mathcal{X} \to \mathcal{F}$ from the input space to the higher dimensional Hilbert space $\mathcal{F}$ where points in $\mathcal{X}$ can be classified in an easier way. The use of the feature map implies the definition of a kernel, which measures the distance between points in the feature space: each kernel is related to a different information encoding approach. Such data embedding method from $\mathcal{X}$ to the feature Hilbert space can be written as a state preparation circuit $U_\phi(x)$ acting on the vacuum state. It is then followed by the so-called model circuit, $U(\theta)$, which contains both non-parametrized and parametrized gates, thus resulting in the following transformation:

$$U(\theta)U_\phi \left|0\right\rangle^{\otimes n} .$$

The final step consists in some measurement of the resulting state, which, in the case of variational classifiers, can represent either the final class label (0/1 or -1/1) or the probability of measuring such outcomes. In order to make the circuit learn the structure of our input data it is necessary to define a cost function which quantifies the difference between the expected output and the predicted one. The set of circuit parameters are updated at each training iteration in order to approach the minimum of the cost function and

the final choice of these parameters provides a set of labels (or probabilities) which is hopefully as close as possible to the real ones.

In the remaining part of this section, we will see two different experiments exploiting the feature map: the first one consists in deriving a kernel from its definition and using it in a classical classification algorithm. On the other hand, the second experiment uses the feature map - in particular the squeezing feature map - as an information encoding tool in a variational circuit.

A classical classifier with squeezing feature map

In this paragraph we are going to see how a classification problem can be solved using a classical approach, namely a Support Vector Classifier, while using a custom kernel deriving from a quantum feature map.

Let us start by defining a squeezed vacuum state:

$$|z\rangle = \frac{1}{\sqrt{\cosh r}} \sum_{n=0}^{\infty} \frac{\sqrt{(2n)!}}{2^n n!} (-e^{i\varphi} \tanh r)^n \, |2n\rangle \,,$$

where $\{|n\rangle\}$ is the Fock basis and $z = re^{i\varphi}$ is the complex squeezing factor. It is convenient to use the notation $|z\rangle = |(r, \varphi)\rangle$ and to define a feature map from $\mathbb{R}$ to the Fock space as $x \to |(c, x)\rangle$, where c represents the strength of squeezing. The resulting feature map is called the *squeezing feature map with phase encoding* and its corresponding kernel is

$$\kappa(\boldsymbol{x}, \boldsymbol{x}'; c) = \prod_{i=1}^{N} \left\langle (c, x_i) | (c, x_i') \right\rangle ,$$

where

$$\left\langle (c, x_i) | (c, x_i') \right\rangle = \sqrt{\frac{\operatorname{sech} c \operatorname{sech} c}{1 - e^{i(x_i' - x_i)} \tanh c \tanh c}}. \tag{4.2}$$

This kernel is easy to compute on a classical computer: for this reason, we can use it in our Support Vector Classification algorithm.

The problem we want to solve is that of binary classification of points from the *moons* dataset of `sklearn` ([Pedregosa et al., 2011]). The training of the algorithm was performed using SVC with a custom kernel implementing equation (4.2). As stated and proven in [Schuld and Killoran, 2019], the squeezing feature map makes data linearly separable in the Fock space, so we are able to obtain a perfect fit on training data. Further experiments were performed also using other popular kernel already implemented in `sklearn`, but only the `rbf` one was able to provide satisfying results. Figure 4.6 shows the outcomes of the experiments.

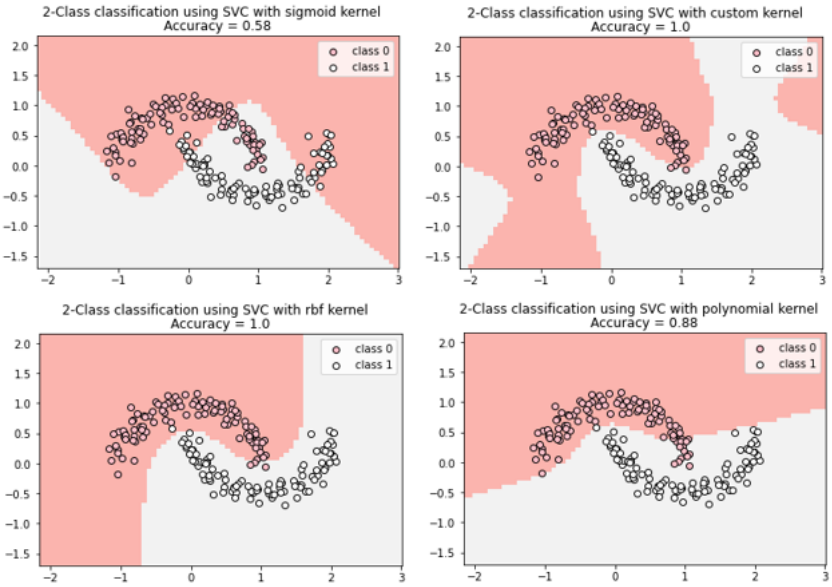


Figure 4.6: Overview of classification results on *moons* dataset using SVC with 4 different kernels.

A CV quantum variational classifier

It is now the moment to see how the already seen feature map can be combined with a parametrized circuit with continuous variable gates to build a CV variational classifier with the purpose of classifying a bi-dimensional dataset. The used encoding function is the already introduced squeezing feature map: we therefore start with two vacuum modes $|0\rangle \otimes |0\rangle$ and we map the input point $\boldsymbol{x} = (x_1, x_2)$ to a quantum state $|(c, \boldsymbol{x})\rangle = |c, x_1\rangle \otimes |c, x_2\rangle$ by squeezing each of the quantum modes. The model circuit shown in Figure 4.7 is then applied on the resulting quantum state: it is constituted by a sequence of a Beam-Splitter, Displacement, Quadratic and Cubic Phase gates.

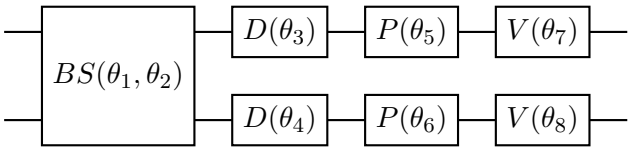


Figure 4.7: Model circuit of the variational classifier with Beam-Splitter (BS), Displacement (D), Quadratic (P) and Cubic (V) Phase gates.

Finally, the output of our continuous variable model will be the probability $p(n_1, n_2)$ of measuring a certain Fock state $|n_1, n_2\rangle$. The authors of the paper preferred to measure instead the probabilities $p(n_1 = 2, n_2 = 0)$

and $p(n_1 = 0, n_2 = 2)$ thus obtaining vector (o_0, o_1). This array can then be normalized

$$\frac{1}{o_0 + o_1} \begin{pmatrix} o_0 \\ o_1 \end{pmatrix} = \begin{pmatrix} p(y=0) \\ p(y=1) \end{pmatrix},$$

to obtain the probabilities to predict class 0 and 1 respectively.

We built a 4-layers structure obtained from the repetition of the circuit of Figure 4.7 and defined a square loss function to measure the distance between the actual class label and the probability of measuring the positive class:

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - p(y=1))^2,$$

where y_i is the real class label.

We tested our algorithm on two different datasets, `sklearn`'s *moons* and a bi-dimensional dataset constituted of two overlapping "blobs". By varying the cutoff dimension of the Fock space, the squeezing strength parameter c, the optimization algorithm, the number of epochs and the gate initial parameters, we obtain different results, which can be more or less satisfying.

- **Moons dataset**: The best results (i.e. minimum loss equal to 0.096) were obtained with the following choice of parameters:

cutoff	c	optimizer	n_epochs	init_params
7	1	Adagrad	2000	`Theta1`

Table 4.1: Hyperparameters and optimizer for the Moons dataset

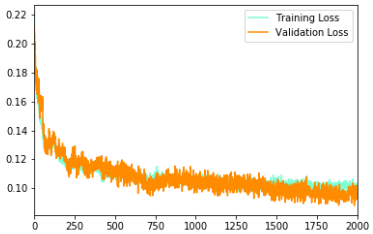
  Figure 4.8 shows in its first row the obtained results for this dataset. 4.8a shows on the same figure both training and validation loss: they are approximately equal and both converging, so we can safely state that we are not overfitting nor underfitting. On the other hand, in 4.8b we can see our dataset representation, together with some shaded areas representing the probability of predicting the positive class.

- **Blobs dataset**: Good results were obtained using the following combination of parameters:

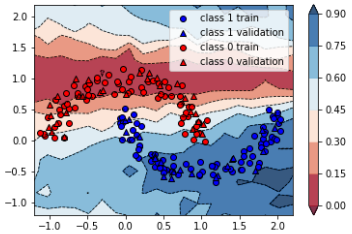
cutoff	c	optimizer	n_epochs	init_params
7	1	Adagrad	2000	`Theta1`

Table 4.2: Hyperparameters and optimizer for the Blobs dataset

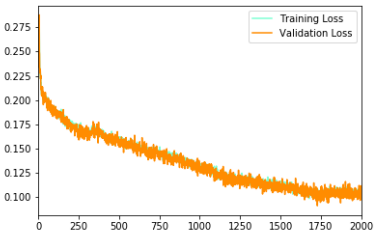
  The minimum attained error in this case is equal to 0.099. The second row of Figure 4.8 shows the results for this dataset: as before, training and validation loss functions (4.8c) are very close to each other, which



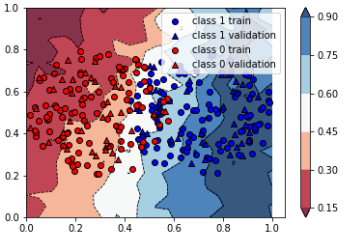
(a) Loss for Moons



(b) Decision boundary for Moons



(c) Loss for Blobs



(d) Decision boundary for Blobs

Figure 4.8: Overview of the CV variational classifier results for moons dataset (first row) and blobs dataset (second row). Both datasets have a test fraction of 0.25.

is an indicator of a satisfying result. In 4.8d we see our dataset together with some shaded areas representing the probabilities of predicting class 1.

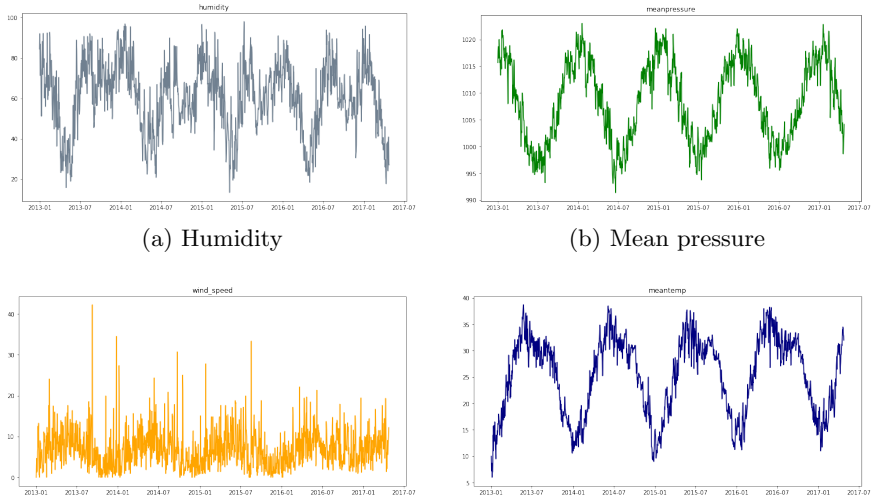
Above we defined

```
Theta1 = (0.05*np.random.rand(n_layers,2,4),0.05*np.random.rand())
```

and the interested reader can find some notes about Adagrad optimizer in Appendix B.

4.2 CV quantum neural networks for time series forecasting

This section presents a detailed analysis on a time series dataset, with the goal of forecasting. We will take two different approaches, one based on feedforward neural networks (FFNN), the other based on recurrent networks (RNN) and for both of them we will make a comparison between a classical setting and a quantum one. While theoretical concepts about FFNNs has already been presented in the previous part, we give here a theoretical overview about RNNs, in order to deeply understand how they work and why they are so useful for time series prediction.



(a) Humidity (b) Mean pressure

(c) Wind speed (d) Mean temperature

Figure 4.9: Plot of features and target in our dataset, obtained from merging training and test sets and after cleaning data.

4.2.1 Dataset

The chosen dataset, [Weather Underground API, 2019], is a quite simple one in which, given three features - mean pressure, wind speed and humidity - we want to fulfill the task of temperature prediction in the city of Delhi, India. From an exploratory analysis, we can see data have a daily frequency and the involved time period ranges from 1st January 2013 to 24th April 2017; furthermore, the provided dataset is split in a training set with 1461 observations and a test set with 114 rows.

After data cleaning, a plot of the given features and target shows foreseeable patterns in data, in particular a strong seasonal component due to the nature of our dataset. In Figure 4.9 we can see these patterns, together with a strong negative correlation between target and mean pressure: this correlation turns out to be equal to -0.88.

4.2.2 FeedForward networks

Quantum version

The first experiment regards the quantum version of a FFNN, built in a similar fashion as in the relative theoretical part. The difference is that this time we do not employ it to fit a curve, but to predict the value of Mean temperature, using the other three features. As a consequence, the employed layer will present three qumodes and it will have the following structure:

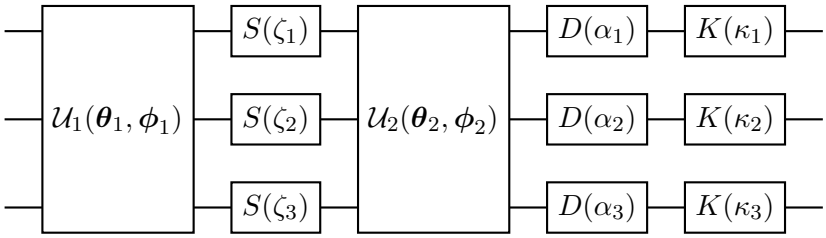


Figure 4.10: General structure for a single layer $\mathcal{L}$ of our 3-modes CV neural network.

First, normalized data are encoded in the quantum circuit by means of Displacement embedding: qumode i is displaced by quantity x_i, $i = 1, 2, 3$, where each x_i represents one of the features. Then, layer of Figure 4.10 is applied a suitable number of times: the depth of the circuit is a hyperparameter of our model and it represents a tradeoff between better results (higher depth) and low computational time (low depth). In fact, stacking different layers one after the other increases the number of trainable parameters: this results in longer computational times. Our choice fell upon 2. Finally, a measurement is performed on one of the wires; since the circuit structure is symmetric, none of them is preferred. The measured quantity is the expectation value of the position operator $\hat{x}$, i.e. a single scalar.

The most convenient choice of Loss function is given by the mean squared error between the measured value, representing the prediction of Mean temperature, and the actual value of the target variable. In order to prevent overfitting, it turns out to be convenient to add to the Loss function a L2-regularization term, defined as

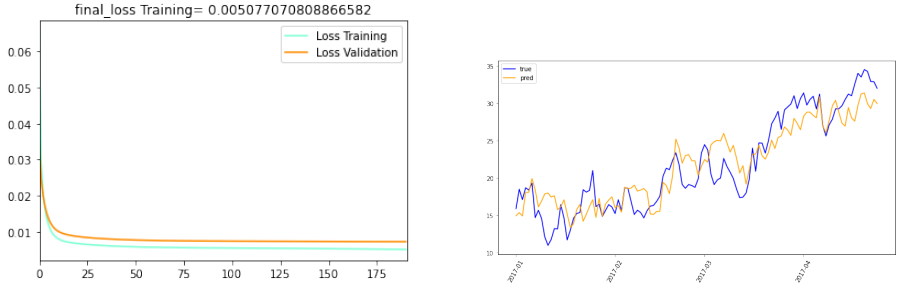
$$\eta * 0.5 * w^T w,$$

where η is the regularization strength and w is a vector obtained from the concatenation of input parameters of active gates, namely Squeezing, Displacement and Kerr, which can change the energy of the system.

After training our model for a suitable number of steps, we obtain the optimal set of parameters, providing the prediction shown in Figure 4.11 with a mean squared error of 7.73.

Classical version

It is now interesting to see how the classical counterpart of the above quantum neural network performs on the same data. Again, we normalize input data to have range in $[0, 1]$ and feed them into a structure built with TensorFlow [Abadi et al., 2015] in such a way it resembles the quantum FFNN as much as possible. In particular, since the quantum version has three qumodes and a depth of two, we build a net constituted by two Dense Layers, with three neurons each, followed by a single neuron with a sigmoid



(a) Loss (b) Plot

Figure 4.11: Figure on the left shows both training and validation loss for the quantum FFNN. In the picture on the right we see real and predicted values on test set.

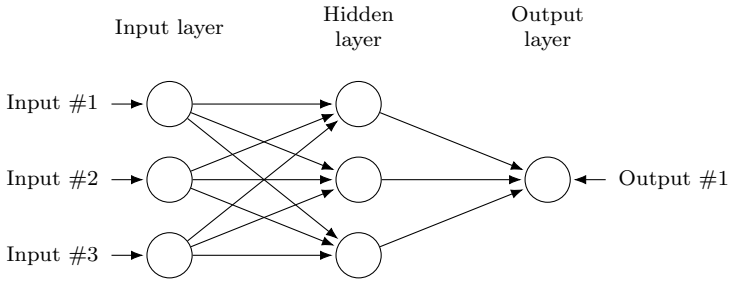


Figure 4.12: Graphical representation of the employed classical neural network, obtained from the concatenation of three Dense layers.

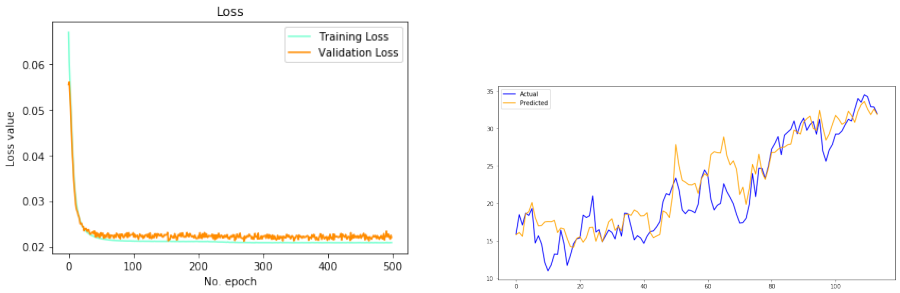
activation function, in order to output only one value in the $[0, 1]$ interval (Figure 4.12).

The described model is quite simple and, fortunately, it returns results which are similar to the ones obtained in the quantum version. In particular, the obtained mean squared error is equal to 7.88 and we can see in Figure 4.13 training and validation loss plot and forecast vs. actual values.

4.2.3 LSTM networks

Traditional neural networks, like the feedforward NN introduced and employed above, are not able to identify a pattern in data having a temporal dependence, like in the case of time series data. Recurrent neural networks [Olah, 2015] come to our aid in these cases: they are networks with loops inside and are built in such a way they are able to remember data structures over time. Actually, they do not differ a lot from a standard neural network, since the loops can be unfolded and the resulting net has as many layers as are the input data points: in this way information is propagated though time (see Figure 4.14 for a better understanding).

Theoretically, this kind of neural network is able to learn both short-term



(a) Loss (b) Plot

Figure 4.13: Figure on the left shows both training and validation loss for the classical FFNN. In the picture on the right we see real and predicted values on test set.

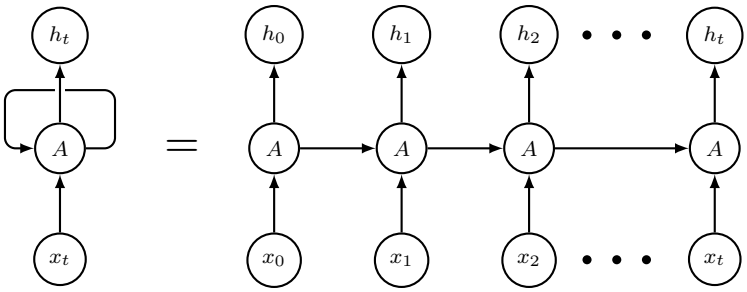


Figure 4.14: Simple scheme of a Recurrent Neural Network. The core of the network, A, takes as input a value x_t and returns h_t as output.

and long-term dependencies; unfortunately, in practical settings the latter is not always guaranteed. Furthermore, recurrent neural networks are affected by the problem of vanishing gradient [Arbel, 2018], which makes the algorithm stagnate and not further improve. For these reasons, while handling time series data, it is convenient to use a particular type of RNN, namely a Long Short Term Memory (LSTM) net. A LSTM network is quite similar to the one already presented, but, while each repeating module of a RNN has a very simple structure, the one of a LSTM is quite complex: Figure 4.15 shows how each layer of such network is built. Information flows from left to right and each module receives two inputs from the previous one, c_{t-1} and h_{t-1}, together with x_t, which represents our input data. Symmetrically, it provides c_t and h_t to the following layer and outputs y_t, which, in the case of time series forecasting, represents the predicted data point.

We now see in a detailed manner each of its components and the role they play in handling information.

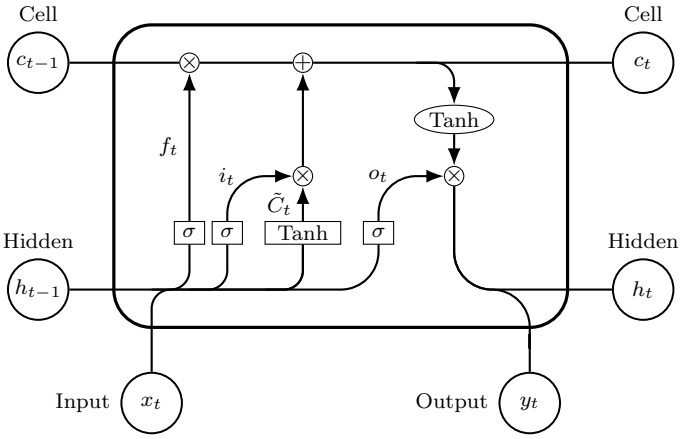


Figure 4.15: LSTM internal structure

Cell state

We start with the horizontal line crossing the layer in its entirety: it is called the *cell state* and it is responsible for handling long-term memory information. For this reason, it has a minimal interaction with the rest of the components, namely some linear transformations. The cell state is shown in Figure 4.16.

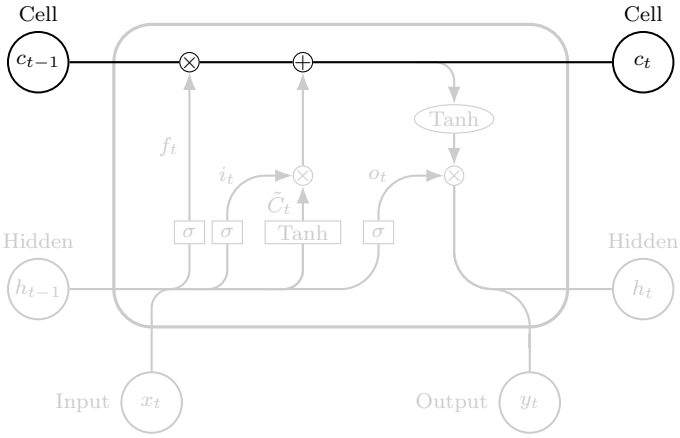


Figure 4.16: Cell state

Forget gate layer

The first task our layer performs is to determine which information retain and which throw away. This is done by taking as input a concatenation of h_{t-1} and x_t, performing on them a linear transformation and applying on the result a sigmoid function. This kind of function returns an output between 0 and 1, which represents how much of the previous information we want to retain: 0 means we completely throw away it, while 1 we keep it entirely. This is the reason why it is called *forget gate layer*. The output of this operation is f_t:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

which is also shown in Figure 4.17.

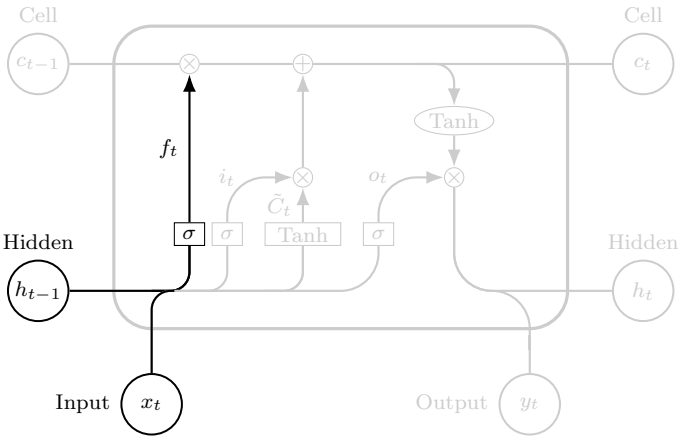


Figure 4.17: Forget gate layer

Input and Tanh layer

This part has the task to determine which new information has to be stored in the cell state. It is composed of two parts, the first is a sigmoid layer, known as *input gate layer*, with the aim to decide which value we update; the second is a Tanh layer, whose output will be combined with that of the previous part to later update the cell state. The two performed operations are

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \text{Tanh}(W_c[h_{t-1}, x_t] + b_c).$$

Figure 4.18 highlights the involved part of the layer.

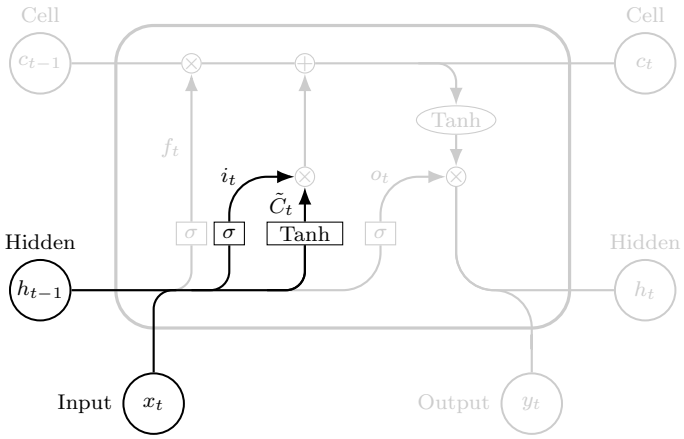


Figure 4.18: Input and Tanh layer

Cell state update

The next step consists in updating the cell state that will be provided to the next layer (Figure 4.19). The performed operation is quite simple:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t,$$

which, in words, means that we forget information from the previous state with the multiplication by f_t and we add the new information by summing the second factor.

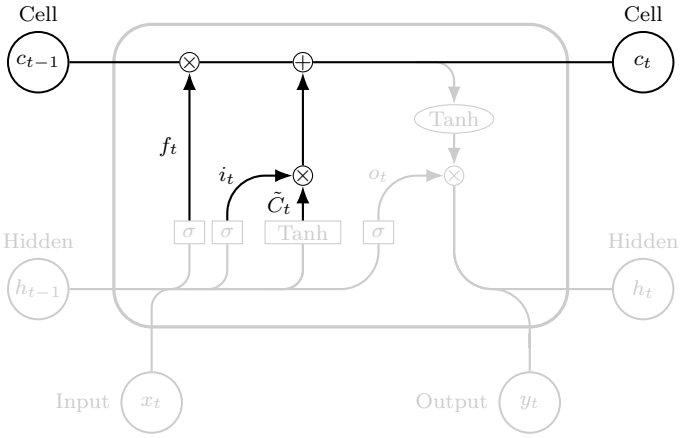


Figure 4.19: Cell state update

Output layer

The final part consists in determining what to output. Referring to Figure 4.20, the highlighted sigmoid layer decides what fraction of the cell state we will output and returns o_t. Then, the cell state passes through Tanh and it is multiplied by o_t:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o),$$

$$h_t = o_t * \text{Tanh}(C_t).$$

This provides h_t, which will be inputted to the next layer as a short-term memory component. If necessary, the prediction y_t can be obtained from h_t with a final linear transformation:

$$y_t = W_v \cdot h_t + b_v.$$

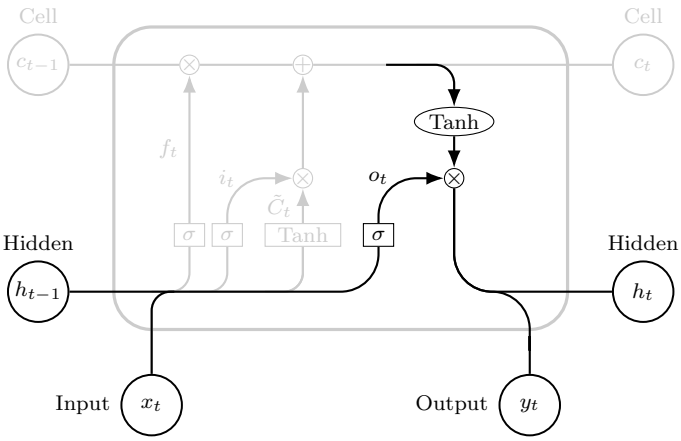


Figure 4.20: Output layer

Quantum part

This part is devoted to the presentation of the results of our experiments involving LSTM nets plus a quantum component. The quantum part enters in the model as a `KerasLayer`, a class from PennyLane, which is a useful tool to transform a quantum node into a Layer of Keras library. Therefore, it is simply necessary to define a convenient quantum circuit and then use it as a layer of Keras Model or Sequential classes to build more complex models. In particular, the employed quantum circuit is chosen to be very similar to the one described in the previous section, with three qumodes, a Displacement embedding and one layer of quantum neural network; the three measurements are once again the expectation values of the position operator $\hat{x}$.

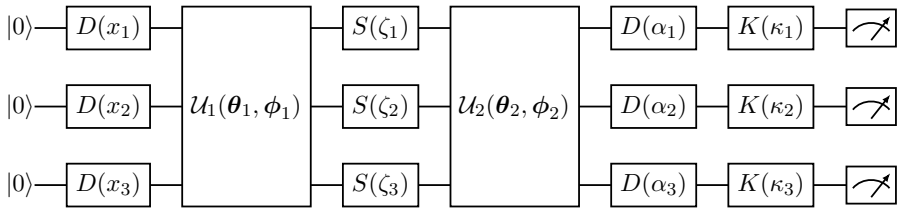


Figure 4.21: Complete quantum circuit to be embedded inside a Keras Layer.

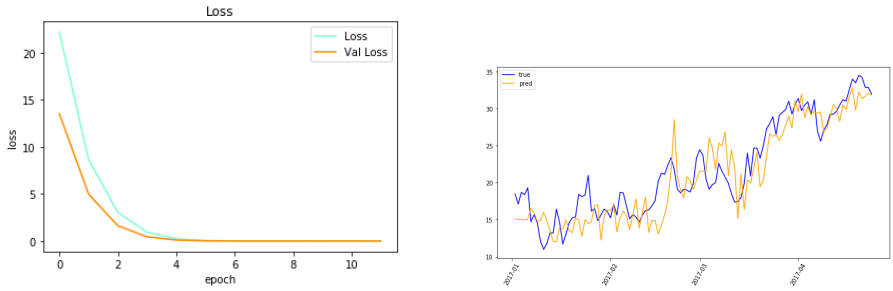
Exploiting this layer, we can build a sequential model stacking a LSTM layer with three neurons, our quantum layer and finally a Dense layer with a single neuron and a sigmoid activation function. This structure is chosen in such a way to be as similar as possible to the feedforward network presented above.

Keras LSTM requires input data to have a specific input structure, so, besides normalizing our input, it is necessary to reshape data to have shape [Samples, Timesteps, Features].

- Samples represents the number of sequences we use;
- Timesteps is the number of obervations for each sample;
- Features stands for the number of input units.

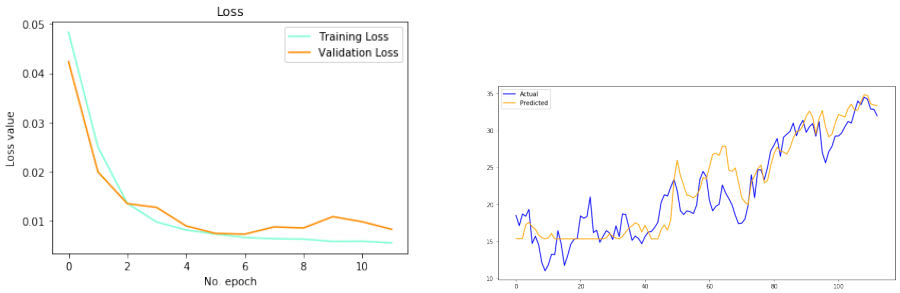
Here we choose timesteps=2, since it provides the most satisfying results.

After training our model, we get the optimal set of weights which we use to compute the predictions on the test set. The reference metric is as usual the mean squared error, which in this case is 8.45; Figure 4.22 shows instead the training and validation loss plots and the predicted values vs. the original ones.



(a) Loss (b) Plot

Figure 4.22: Figure on the left shows both training and validation loss for the quantum version of LSTM model. In the picture on the right we see real and predicted values on test set.



(a) Loss (b) Plot

Figure 4.23: Figure on the left shows both training and validation loss for the classical version of LSTM model. In the picture on the right we see real and predicted values on test set.

Classical part

With the purpose of building a classical model to be compared to the previous one, we normalize and reshape data in the already described way in order to provide them as input to the LSTM net. The resulting model substitutes the quantum layer with a simple Dense layer, therefore the resulting network presents a first LSTM with three neurons, a Dense layer with the same number of units and a final layer with a sigmoid activation function.

Results are satisfying, since the mean squared error is this time equal to 7.76; the plot of loss and predictions can be seen in Figure 4.23.

4.2.4 Conclusions and final considerations

To conclude, it is convenient to see in a tabular form the results we extensively commented above: in particular, we report the value of mean squared error for each experiment.

	Classical	**Quantum**
FFNN	7.88	7.73
LSTM	7.76	8.45

Table 4.3: Tabular summary of the main results of our experiments on time series data. The table shows the value of mean squared error in each single case.

From the point of view of performance error, the four models provide comparable results, which might suggest that choosing a quantum or classical approach is totally indifferent. Actually, this is not so true, since all the experiments involving a quantum component took a lot more time than their classical counterpart. This is due to the fact that we need to simulate three qumodes, which of course takes time: in particular the model involving the `KerasLayer` was quite slow and this is the reason why we trained the model - and the corresponding classical one - for a lower number of iterations. This remark is somehow a confirmation of a concept we already presented in the introductory part of this thesis: quantum machine learning is a quite new research field and researches are still working in order to understand in which fields using a quantum component could benefit the most. Going on making experiments and studies in this field is definitely worth it, since we can only imagine what amazing results we could reach in the very next years.

We now want to rethink of our work and try to analyze it in a critical way, in order to find some aspects to be improved and to detect some starting points for possible further research studies.

- An interesting point from which to start could consist in analyzing in a deeper way the impact the choice of some hyperparameters may have on our model. In particular, focusing for example on the quantum FFNN, we could run our model with different values for the circuit depth while keeping track of mean squared error. We can try to foresee the results, i.e. we will probably see a reduction of the error as the number of layers increases, until a saturation point in which we would see no more improvement. This point will indicate the optimal number of layers, which in this case would be derived from more reliable observations with respect to our approach which was based on trial and error. This example could obviously apply to other hyperparameters and to other experiments, leading not only to more accurate results, but also to a more interesting analysis of our model.

- Another way our four experiments could be improved may consist in employing a rolling window to produce a more reliable forecast. In fact, what we have done is to use the training set as it is and to directly predict the values in our test set. With a rolling window approach, instead, we would predict only one step at a time and each time we

would add the next point to our training set. The size of the window could be constant or not, depending on what turns out to be more convenient, and the downside is that we would have an additional hyperparameter, namely the window size.

- An interesting alternative to the LSTM network with a quantum component could be represented by the following model. It consists in building a quantum neural network in the way we have already seen, this time with some input wires devoted to receiving input data and the others to take the input from the previous layers:

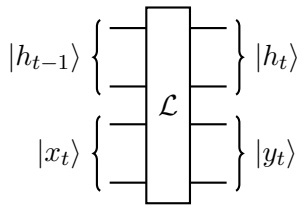


Figure 4.24: General structure for a single layer $\mathcal{L}$ of a hypothetical quantum recurrent network, as suggested in [Killoran et al., 2019a].

This structure, however, could be quite difficult to implement for some reasons: the first is that it would be necessary to increment the number of employed wires, thus leading to a higher computational time. Secondly, it would require to use a circuit with a number of layers equal to the number of input rows: also this would increment the number of trainable parameters an consequently the computational time. Last, we should note that the internal structure of each layer is not so obvious and using the standard neural network could not provide the hoped results: probably, a deeper study of the layer structure would be necessary.

Chapter 5

Conclusions

We started describing quantum computers and giving an overview of the main properties in quantum mechanics, which contribute to making quantum devices so unique and advantageous. We have introduced the concept of quantum supremacy and we have stressed the fact that this kind of machines are not meant to outperform their classical counterparts in every aspect. This leads to the description of the main fields researchers agree will benefit the most from the introduction of quantum devices, like for example machine learning, which constitutes the focus of this work. We have then seen how the period we are now living can be addressed as NISQ era, since current devices have a small number of qubits and are characterized by inherent noise. We therefore remarked how hardware implementation is nowadays a challenge and, to conclude the introductory part, we have briefly analyzed how quantum computers could impact on our lives in the future and what the main challenges will be.

For the theoretical part, we started introducing the concepts of state of a system and of observables, presenting them by means of a rigorous mathematical formalism. We have seen that states can be represented as elements of a finite Hilbert space and that they can be written as a superposition of states: the coefficients of this linear combination contribute to determining the output state we get after a measurement. Similarly, we introduced the concept of observable, which can be written as a matrix acting on a quantum state: this matrix is Hermitian and the measurement will always make the state collapse onto an eigenvector of the observable, while the result of such operation will be represented by the corresponding eigenvalue. The eigenvalues are always real and the eigenvectors of the observables form an orthonormal basis for the Hilbert space. After this, we have defined a qubit as a two-level system (like the spin up or down of an electron), representing the quantum counterpart of a classical bit. We have seen how qubits can be represented on the Bloch sphere and we have introduced the Pauli matrices, which constitute the most common observables in this context. Finally, we

described some of the most used gates, which are unitary matrices acting on qubits and manipulating them: differently from their classical counterparts, quantum gates are always reversible and have the same number of input and output bits. Besides, we introduced the concept of circuit, obtained from the combination of a certain number of gates.

Next to this, we have briefly seen how current hardware implementations of quantum devices could be improved by using photons instead of electrons to carry information: this leads to the definition of photonic quantum computers, which can work at room temperatures, since photons are less affected by noise with respect to electrons. Quantum optical computers are very well suited to work adopting a continuous variable (CV) formalism, which opposes to the already described one, usually addressed as the discrete variable (DV) formalism. The main differences are that the involved Hilbert spaces are now infinite dimensional and the operators have continuous spectra: the continuous component is provided by this aspect. We have explained the elements at the basis of this formalism, namely qumodes (which are the continuous counterpart of qubits), states, operators, transformations and the phase space description. Furthermore, we have remarked how the continuous variable formalism, though less present in literature, can turn to be convenient in certain situations in which continuous quantities are involved: using the DV formalism would require to discretize the involved variables, thus leading to information loss and worse performances. The goal of this thesis was indeed to analyze some situations in which this kind of formalism is more convenient with respect to the discrete one.

After this theoretical introduction, we have explained in a more detailed way quantum machine learning and remarked the fact that it is not so clear which benefits could emerge from the introduction of a quantum component into machine learning field. We then introduced the concept of quantum parametrized or variational circuit, which are currently the preferred way to solve machine learning tasks. Such circuits are composed of the initial preparation of the state, an encoder part to insert input data into the circuit and a model component constituting the core of the circuit and formed by gates whose arguments are the trainable parameters. Finally, a measurement is performed, thus returning a classical quantity, which will be used by a classical machine to update the trainable weights with the goal of minimizing a given objective function. This helps us to understand how quantum and classical computers can work in synergy in a hybrid quantum-classical approach. We then presented the most common information encoding techniques, together with their pros and cons. Furthermore, we have seen how quantum variational circuits - and in particular their encoding component - have a strong bond with kernel methods, of which we presented the mathematical theory in a formal way. Next to this, we began describing PennyLane, a library for automatic differentiation, quantum machine learning and optimization over hybrid quantum-classical computations. In particular, we

have analyzed how PennyLane computes gradients and how it handles quantum circuits.

In the final chapter, we presented some examples in which quantum machine learning is exploited together with the continuous variable formalism to solve different kind of problems. We start with two examples taken from the literature, Monte Carlo Integration and Gaussian process regression: these two have only been inserted and briefly described, but we did no attempt in reproducing the results in the papers. They are followed by other two examples taken from literature, Variational classifiers and Neural networks: this time we tried to implement the algorithms described in the reference articles and we presented and commented the obtained results. The last part of this section is devoted to an application of continuous variable quantum Neural nets, employed for time series forecasting. This application was developed in a more detailed way with respect to the previous ones and it constitutes the core of its chapter. Two different kinds of quantum neural networks were employed, feedforward and recurrent, while a classical parallel was done in both cases. We gathered and commented all the results and, moreover, we provided some starting points for further researches.

The obtained results encourage us to keep exploring the field of quantum machine learning, since a lot is already to be discovered. They also highlight some limitations of current quantum devices and simulators, especially the reduced number of qumodes it is possible to simulate, but we can be confident we will see a lot of improvements in the very next years.

Appendix A

Proof of minimum uncertainty for coherent states

Let us use this space to prove that a coherent state has minimum uncertainty, as promised in Section 2.3.3. The following computations are inspired by some remarks taken from [Anlage, 2016]. For convenience, we remind the definition of quadrature operators

$$\hat{x} = \sqrt{\frac{\hbar}{2}}(\hat{a} + \hat{a}^\dagger), \quad \hat{p} = -i\sqrt{\frac{\hbar}{2}}(\hat{a} - \hat{a}^\dagger)$$

and we remark that for a coherent state it holds

$$\hat{a}\,|\alpha\rangle = \alpha\,|\alpha\rangle\,.$$

We want to prove that

$$\langle\Delta\hat{x}\rangle_\alpha \langle\Delta\hat{p}\rangle_\alpha = \frac{\hbar}{2}, \quad \text{or equivalently} \quad \left\langle\Delta\hat{x}^2\right\rangle_\alpha \left\langle\Delta\hat{p}^2\right\rangle_\alpha = \frac{\hbar^2}{4},$$

where $\left\langle\Delta\hat{x}^2\right\rangle_\alpha = \left\langle\hat{x}^2\right\rangle_\alpha - \langle\hat{x}\rangle_\alpha^2$ and $\left\langle\Delta\hat{p}^2\right\rangle_\alpha = \left\langle\hat{p}^2\right\rangle_\alpha - \langle\hat{p}\rangle_\alpha^2$.

We can start computing each necessary quantity:

$$\langle\hat{x}\rangle_\alpha = \sqrt{\frac{\hbar}{2}}\left\langle\alpha|(\hat{a} + \hat{a}^\dagger)\middle|\alpha\right\rangle = \sqrt{\frac{\hbar}{2}}(\langle\alpha|\hat{a}|\alpha\rangle) + \left\langle\alpha|\hat{a}^\dagger\middle|\alpha\right\rangle) = \sqrt{\frac{\hbar}{2}}(\alpha + \alpha^*),$$

$$\langle\hat{p}\rangle_\alpha = -i\sqrt{\frac{\hbar}{2}}\left\langle\alpha|(\hat{a} - \hat{a}^\dagger)\middle|\alpha\right\rangle = -i\sqrt{\frac{\hbar}{2}}(\langle\alpha|\hat{a}|\alpha\rangle) - \left\langle\alpha|\hat{a}^\dagger\middle|\alpha\right\rangle) = -i\sqrt{\frac{\hbar}{2}}(\alpha - \alpha^*),$$

$$\left\langle\hat{x}^2\right\rangle_\alpha = \frac{\hbar}{2}\left\langle\alpha|(\hat{a} + \hat{a}^\dagger)^2\middle|\alpha\right\rangle = \frac{\hbar}{2}\left\langle\alpha|(\hat{a}\hat{a} + \hat{a}\hat{a}^\dagger + \hat{a}^\dagger\hat{a} + \hat{a}^\dagger\hat{a}^\dagger)\middle|\alpha\right\rangle = \frac{\hbar}{2}\left[(\alpha + \alpha^*)^2 + 1\right],$$

$$\left\langle \hat{p}^2 \right\rangle_\alpha = -\frac{\hbar}{2}\left\langle \alpha|(\hat{a}-\hat{a}^\dagger)^2\Big|\alpha\right\rangle = -\frac{\hbar}{2}\left\langle \alpha|(\hat{a}\hat{a}-\hat{a}\hat{a}^\dagger-\hat{a}^\dagger\hat{a}+\hat{a}^\dagger\hat{a}^\dagger)\Big|\alpha\right\rangle = -\frac{\hbar}{2}\left[(\alpha-\alpha^*)^2-1\right].$$

Basically, in the previous computations we have used the fact that if

$$\hat{a}\,|\alpha\rangle = \alpha\,|\alpha\rangle \quad \text{then} \quad \langle\alpha|\hat{a}|\alpha\rangle = \alpha\,\langle\alpha|\alpha\rangle = \alpha.$$

Putting all of these together:

$$\left\langle \Delta\hat{x}^2 \right\rangle_\alpha = \left\langle \hat{x}^2 \right\rangle_\alpha - \langle\hat{x}\rangle_\alpha^2 = \frac{\hbar}{2}\left[(\alpha+\alpha^*)^2+1\right] - \frac{\hbar}{2}(\alpha+\alpha^*)^2 = \frac{\hbar}{2},$$

$$\left\langle \Delta\hat{p}^2 \right\rangle_\alpha = \left\langle \hat{p}^2 \right\rangle_\alpha - \langle\hat{p}\rangle_\alpha^2 = -\frac{\hbar}{2}\left[(\alpha-\alpha^*)^2-1\right] + \frac{\hbar}{2}(\alpha-\alpha^*)^2 = \frac{\hbar}{2}.$$

Therefore,

$$\left\langle \Delta\hat{x}^2 \right\rangle_\alpha \left\langle \Delta\hat{p}^2 \right\rangle_\alpha = \frac{\hbar^2}{4}.$$

■

Appendix B

Adam optimizer

Adam optimization is an effective algorithm which can be used instead of Stochastic Gradient Descent to update the weights of a neural network. In order to have a better insight of this, it is convenient to have a look to the Gradient Descent algorithm, its Stochastic version and to Adagrad algorithm. The main references for this topic are [Kingma and Ba, 2014] and [Xanadu, 2019].

B.0.1 Gradient Descent optimization

Given the following optimization problem, in which we aim to minimize a function in the $n-$ dimensional space $\mathbb{R}^n$,

$$\min f(x), \qquad x \in \mathbb{R}^n,$$

the Gradient Descent algorithm starts from a random solution x_0 and updates it at every step t as

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x^{(t)}),$$

where η is a suitable step-size. This method relies on the fact that for a given function $f(x)$ the maximum descent direction from a certain point $\tilde{x}$ is provided by the opposite of the gradient in that point, $-\nabla f(\tilde{x})$.

The Gradient descent algorithm, though simple and intuitive, presents some drawbacks:

- Computing the gradient can be expensive and inefficient in some situations especially when dealing with large datasets: the stochastic version of this procedure helps to overcome this issue.
- The choice of a suitable step-size is not so obvious: this problem should be tackled using a line search method.
- When the problem is not well-conditioned, the level curves can turn to be too elliptical in shape, thus resulting in a "zig-zag" path towards the solution, which interferes with convergence.

B.0.2 Stochastic Gradient Descent optimization

A valid alternative to the previous method is provided by its stochastic variant. In fact, the Stochastic Gradient Descent algorithm consists in replacing the descent direction $\nabla f(\tilde{x})$ with $g^{(t)}(x^{(t)})$, where $\{g^{(t)}(x)\}$ is a sequence of random variables such that

$$\mathbb{E}(g^{(t)}(x)) = \nabla f(\tilde{x}).$$

Stochastic Gradient Descent algorithm is usually preferred to the vanilla version for a lot of reasons; to mention only some of them, $g^{(t)}(x)$ is usually easier to compute than the actual gradient and randomness helps to avoid being stuck in local optima. Numerical experiments show that this algorithm performs better than Gradient Descent and for this reason it is often used.

B.0.3 Adagrad optimization

Adagrad stands for ADAptive GRADient algorithm and it is characterized by the choice of adjusting the learning rate for each individual parameter based on past gradients. Parameters update is therefore done componentwise:

$$x_i^{(t+1)} = x_i^{(t)} - \eta_i^{(t+1)} \partial_{x_i} f(x^{(t)}),$$

where we have obviously replaced the gradient with a partial derivative. The learning rate at iteration t is given by

$$\eta_i^{(t+1)} = \frac{\eta_{\text{init}}}{\sqrt{a_i^{(t+1)} + \epsilon}}, \qquad a_i^{(t+1)} = \sum_{k=1}^{t} (\partial_{x_i} f(x^{(k)}))^2.$$

One of the main drawbacks of this algorithm is the squared derivative accumulation at denominator: each addend is positive, so the denominator keeps growing at each step t. As a result, the learning rate becomes increasingly small and there is the possibility that the algorithm stagnates with no possibility of improving.

B.0.4 Adam optimization

One of the limits of Stochastic Gradient Descent is that the step-size (or learning rate) is maintained constant during training. A valid and more efficient alternative is Adam optimization, which takes its name from ADAptive Moment estimation. Numerical experiments have shown that Adam is often more efficient than other stochastic methods ([Kingma and Ba, 2014]).

In contrast with Stochastic Gradient Descent version, Adam uses a step-dependent learning rate, a first moment a and a second moment b, thus resulting in the following update rule:

$$x^{(t+1)} = x^{(t)} - \eta^{(t+1)} \frac{a^{(t+1)}}{\sqrt{b^{(t+1)}} + \epsilon}.$$

Here ϵ is used to avoid division by 0 and the step-dependent parameters are updated as:

$$a^{(t+1)} = \frac{\beta_1 a^{(t)} + (1-\beta_1)\nabla f(x^{(t)})}{1-\beta_1},$$

$$b^{(t+1)} = \frac{\beta_2 b^{(t)} + (1-\beta_2)(\nabla f(x^{(t)}))^{\odot 2}}{1-\beta_2},$$

$$\eta^{(t+1)} = \eta^{(t)} \frac{\sqrt{1-\beta_2}}{1-\beta_1}.$$

$(\nabla f(x^{(t)}))^{\odot 2}$ denotes the element-wise square operation, meaning that each element in the gradient is multiplied by itself. Initially the first and second moments are 0.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Abraham Asfaw, 2020] Abraham Asfaw, Luciano Bello, Y. B.-H. S. B. L. C. A. C. V. J. C. R. C. A. F. J. G. S. G. L. G. S. D. L. P. G. F. H. T. I. D. M. A. M. Z. M. R. M. G. N. P. N. A. P. M. P. A. R. J. S. J. S. J. S. K. T. M. T. S. W. J. W. (2020). Learn quantum computation using qiskit.

[Adesso et al., 2014] Adesso, G., Ragy, S., and Lee, A. R. (2014). Continuous variable quantum information: Gaussian states and beyond. *Open Systems & Information Dynamics*, 21(01n02):1440001.

[Andersen et al., 2015] Andersen, U. L., Neergaard-Nielsen, J. S., Van Loock, P., and Furusawa, A. (2015). Hybrid discrete-and continuous-variable quantum information. *Nature Physics*, 11(9):713–719.

[Anlage, 2016] Anlage, S. (2016). Lecture notes: Coherent states of the harmonic oscillator.

[Arbel, 2018] Arbel, N. (2018). How lstm networks solve the problem of vanishing gradients.

[Arute et al., 2019] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A., et al. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

[Ball, 2020] Ball, P. (2020). The best quantum computers money can buy: Underpowered, unreliable but amazing.

[Benedetti et al., 2019] Benedetti, M., Lloyd, E., Sack, S., and Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001.

[Bergholm et al., 2018] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Blank, C., McKiernan, K., and Killoran, N. (2018). Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*.

[Bromley et al., 2020] Bromley, T. R., Arrazola, J. M., Jahangiri, S., Izaac, J., Quesada, N., Gran, A. D., Schuld, M., Swinarton, J., Zabaneh, Z., and Killoran, N. (2020). Applications of near-term photonic quantum computers: Software and algorithms. *Quantum Science and Technology*.

[Calude and Calude, 2017] Calude, C. S. and Calude, E. (2017). The road to quantum computational supremacy. In *Jonathan M. Borwein Commemorative Conference*, pages 349–367. Springer.

[Das et al., 2018] Das, S., Siopsis, G., and Weedbrook, C. (2018). Continuous-variable quantum gaussian process regression and quantum singular value decomposition of nonsparse low-rank matrices. *Physical Review A*, 97(2):022315.

[Grimmer et al., 2018] Grimmer, D., Brown, E., Kempf, A., Mann, R. B., and Martín-Martínez, E. (2018). A classification of open gaussian dynamics. *Journal of Physics A: Mathematical and Theoretical*, 51(24):245301.

[Hubregtsen et al., 2020] Hubregtsen, T., Pichlmeier, J., and Bertels, K. (2020). Evaluation of parameterized quantum circuits: on the design, and the relation between classification accuracy, expressibility and entangling capability. *arXiv preprint arXiv:2003.09887*.

[Killoran et al., 2019a] Killoran, N., Bromley, T. R., Arrazola, J. M., Schuld, M., Quesada, N., and Lloyd, S. (2019a). Continuous-variable quantum neural networks. *Physical Review Research*, 1(3):033063.

[Killoran et al., 2019b] Killoran, N., Izaac, J., Quesada, N., Bergholm, V., Amy, M., and Weedbrook, C. (2019b). Strawberry fields: A software platform for photonic quantum computing. *Quantum*, 3:129.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Klipfel, 2009] Klipfel, J. (2009). A brief introduction to hilbert space and quantum logic.

[Kopczyk, 2018] Kopczyk, D. (2018). Quantum machine learning for data scientists. *arXiv preprint arXiv:1804.10068*.

[National Academies of Sciences et al., 2019] National Academies of Sciences, E., Medicine, et al. (2019). *Quantum computing: progress and prospects*. National Academies Press.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Preskill, 2018] Preskill, J. (2018). Quantum computing in the nisq era and beyond. *Quantum*, 2:79.

[Rasmussen and Williams, 2016] Rasmussen, C. and Williams, C. (2016). *Gaussian processes for machine learning*. The MIT Press.

[Rebentrost et al., 2018] Rebentrost, P., Gupt, B., and Bromley, T. R. (2018). Photonic quantum algorithm for monte carlo integration. *arXiv preprint arXiv:1809.02579*.

[Schuld and Killoran, 2019] Schuld, M. and Killoran, N. (2019). Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504.

[Schuld et al., 2015] Schuld, M., Sinayskiy, I., and Petruccione, F. (2015). An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185.

[Serafini, 2017] Serafini, A. (2017). *Quantum continuous variables: a primer of theoretical methods*. CRC press.

[Takeda and Furusawa, 2019] Takeda, S. and Furusawa, A. (2019). Perspective: Toward large-scale fault-tolerant universal photonic quantum computing. *arXiv preprint arXiv:1904.07390*.

[Torre, 2017] Torre, C. (2017). Lecture notes: The formalism of quantum mechanics.

[Tyc and Sanders, 2004] Tyc, T. and Sanders, B. C. (2004). Operational formulation of homodyne detection. *Journal of Physics A: Mathematical and General*, 37(29):7341.

[Weather Underground API, 2019] Weather Underground API (2019). Daily climate time series data. data retrieved from Kaggle, `https://www.kaggle.com/sumanthvrao/daily-climate-time-series-data`,.

[Wittek, 2014] Wittek, P. (2014). *Quantum machine learning: what quantum computing means to data mining*. Academic Press.

[Xanadu, 2019] Xanadu (2019). Pennylane website.

[Xanadu, 2020] Xanadu (2020). Xanadu website.