

SCUOLA
NORMALE
SUPERIORE

Classe di Scienze

Corso di perfezionamento in
Fisica

XXXVII ciclo

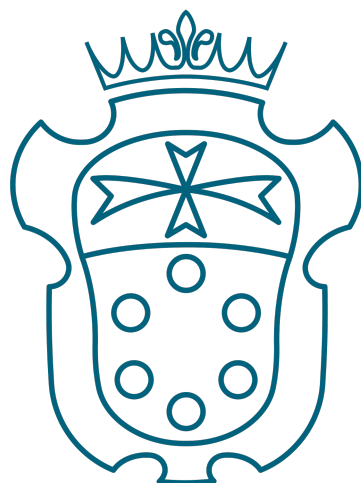
**Quantum algorithms for image classification
and nonlinear differential equations**

Settore Scientifico Disciplinare FIS/02

Candidato

dr. Paolo Alessandro Xavier Tognini

Anno accademico 2024/2025



SCUOLA NORMALE SUPERIORE

Faculty of Sciences
Ph.D. in Physics

Quantum algorithms for image classification
and nonlinear differential equations

Supervisor:

Prof. Giacomo De Palma

Candidate:

Paolo Alessandro Xavier
Tognini

Academic Year 2024-2025

Abstract

This thesis presents two works related to quantum algorithms.

In the first part, we propose a new quantum neural network for image classification, which is able to classify the parity of the MNIST dataset with full resolution with a test accuracy of up to 97.5% without any classical pre-processing or post-processing. Our architecture is based on a mixture of experts whose model function is the sum of the model functions of each expert. We encode the input with amplitude encoding, which allows us to encode full-resolution MNIST images with 10 qubits and to implement a convolution on the whole image with just a single one-qubit gate. Our training algorithm is based on training all the experts together, which significantly improves trainability with respect to training each expert independently. In fact, in the limit of infinitely many experts, our training algorithm can perfectly fit the training data. Our results demonstrate the potential of our quantum neural network to achieve high-accuracy image classification with minimal quantum resources, paving the way for more scalable and efficient quantum machine learning models.

In the second part, we study with classical simulations and analytical calculations the quantum algorithm of [Lloyd *et al.*, arXiv:2011.06571] for solving nonlinear differential equations, showing its strong limitations.

Qual è 'l geomètra che tutto s'affige
per misurar lo cerchio, e non ritrova,
pensando, quel principio ond'elli indige,

tal era io a quella vista nova:
veder voleva come si convenne
l'imago al cerchio e come vi s'indova;

ma non eran da ciò le proprie penne:
se non che la mia mente fu percossa
da un fulgore in che sua voglia venne.

A l'alta fantasia qui mancò possa;
ma già volgeva il mio disio e 'l *velle*,
sì come rota ch'igualmente è mossa,

l'amor che move il sole e l'altre stelle.

Acknowledgements

I want to take a moment to recall everyone I have to thank after such a long journey. I have been in Pisa, and in Scuola Normale Superiore (SNS), for nine years of my life. In these last three or four years I casted a brief glance towards the intersection between Artificial Intelligence and Quantum Physics, between the depths of what it *means* for humans to learn, to understand, to reason and why simple nonlinear models seems to do it too, and the abyss of the ontological pillars upon which the material World seems to stand, a quantum wilderness in which numbers seems to decompose into prime factors a little faster. The two human enterprises of understanding Reality and of understanding Understanding itself are truly part of a single quest that goes to the depth of the ultimate mysteries, and that finds the adventurer nearer and nearer the mind and heart of God.

I thank my parents, Leda and Pietro, for they supported me all along these last few years, and in particular my father who helped me in programming and who has much greater patience than is humanly possible.

I thank my ex colleagues of my first years in physics in Scuola Normale, with which I kept in touch even though they're scattered throughout strange and remote lands. I thank the colleagues and friends that stayed in Pisa who always cheered me up.

I thank Scuola Normale Superiore (SNS), a miraculous institution that puts together such bright and curious young people so they can pursue their interests in knowledge with a passionate company, extremely competent professors and plentiful resources.

I thank my friends at the university parish of San Frediano, who brightened these last three years for me with kindness and joy.

I thank my roommates and friends that lived with me in a beautiful house near the Arno river, in which magical things happen. They are family.

I give a special thanks to my supervisor Prof. Giacomo De Palma for his

availability, his insights, his patience with me.

I was not the best programmer at the beginning, I am not it now, but in the end I arrived at the end of a journey. I have fought the good fight, I have finished the race, I have kept the faith. But my journey is not at its end. But at its beginning.

May I say these words at the end of my whole journey...!

This thesis was realized during the Artificial Intelligence revolution, in which I wish to contribute in the near future; in an age of confusion and of doubt in who are we and what do we want; and the same is true for me. I guess I have to thank Francesco Mele for his suggestion of pursuing a Ph.D. in Quantum Machine Learning. I thank my theater group here in Pisa for the friendships and the insights: because theater is life. In these last few weeks in which all the magic of all these years came together, I want to thank Marco Eterno, Francesco Zigliotto for the encouragements, Mauro for his praise and good feelings, Giuseppe Ielo and Lavinia Ambrosi for being family and all my present and past roommates, I want to thank Bruno for his technical help, Giacomo for his theoretical help, Edo for his practical insights on life, Gabriele for his spiritual ones. I want to thank all the extraordinary people I met in these last years and that sometimes are lost in the mists of memories, in particular the one that changed me so much that it is better to leave her in those mists. I thank that still small voice that promised good to me, and that laid an amazing Grace upon me in these last few months, through a discussion with a girl under a starry sky in a medieval town, through the Sun whose mirror image shines and flicker over the Tyrrhenian Sea, through the pages of the books and works of art I learned again to appreciate. I thank again my supervisor for his extraordinary availability on short notice to correct and review these pages, and for his patience. I thank my good friend Nunzia for the last three years and for the last few days, and hopefully for the days ahead, which I hope will always be bright for her.

The chance we were promised is now.

Contents

I	Introduction	11
1	Preface	13
1.1	Our work	13
1.2	Outline of the thesis	16
2	Quantum computing	19
2.1	Quantum mechanics	19
2.2	Quantum computing	29
2.3	Quantum gates	31
2.4	Quantum algorithms	34
2.5	Quantum resources	41
2.6	Quantum hardware progress	45
2.7	Applications of quantum computing	48
II	A quantum neural network for image classification	51
3	Machine learning	53
3.1	Feed Forward Neural Networks	56
3.2	The unreasonable effectiveness of deep learning	61
3.3	Convolutional Neural Networks	68
3.4	Architectures for Classical Neural Networks	71
3.5	Machine learning and physics	82
4	Quantum Neural Networks	87
4.1	Two approaches for quantum machine learning	89
4.2	The theory of Quantum Neural Networks	90

4.3	Classical vs. Quantum machine learning	94
4.4	Why Quantum Vision?	99
4.5	Why not Quantum Transformers?	101
4.6	Heuristics for Quantum Neural Networks	103
5	Our <i>Mixture of Quantum Experts (MoQE)</i>	107
5.1	Our architecture	109
5.1.1	Mixture of (Quantum) Experts	111
5.1.2	The limit of infinitely many experts	112
5.2	Experiments	112
6	Conclusions and perspectives	115
III	Quantum algorithms for nonlinear differential equations	117
7	Quantum algorithms for linear differential equations	119
7.1	Introduction to the HHL algorithm	119
7.2	A sketch of the HHL algorithm	121
7.3	Applications of the HHL algorithm	123
7.4	The quantum algorithm to solve linear differential equations .	126
8	Quantum algorithms for nonlinear differential equations	129
8.1	The quest for an algorithm	129
8.2	A quantum algorithm to solve nonlinear differential equations	132
9	Our study of the quantum algorithm for nonlinear differential equations	135
9.1	Introduction to our study	136
9.1.1	Toy models for embedding x	139
9.2	The simulations	144
9.2.1	Code 1	145
9.2.2	Code 2	157
9.3	Analytical Calculation	164
9.4	Possible extensions of the model	171
10	Conclusions and perspectives	173

IV Appendices 175**A Appendices 177**

A.1 The MNIST dataset	177
A.2 Convolutional Neural Networks	177
A.3 Convolutions with a single 1-qubit gate	179
A.4 Extended numerical results	181
A.5 Classical Quadratic classifier	184

Part I

Introduction

Chapter 1

Preface

1.1 Our work

This doctoral thesis is divided in two independent parts, Part II and Part III, which present two different projects both related to quantum computing.

After a general introduction to quantum computing (Part I) necessary to both parts of the thesis, and on machine learning (chapter 3), we are ready to introduce the recent field of quantum machine learning (QML) (chapter 4), a relatively recent branch of quantum computing which explores the use of quantum computation to enhance the capabilities of machine learning algorithms. There are different approaches in the literature, but we will focus on quantum neural networks (QNN), *i.e.*, quantum circuits which depend on some parameters which are trained through gradient descent, so that the QNN learns to perform the desired task [1]. In the literature there are many attempts to train QNNs ([2, 3] are just two examples), and we discuss both them and their limitations in general. The main obstacle is the problem of the so-called barren plateaus [4]: the magnitude of the gradient of the cost function decreases exponentially in the number of layers of the network, so that the QNN is not trainable. There are many features of a QNN which bring about barren plateaus [4–7] and we are careful to avoid them. Another problem is the reasonable doubt that many of the hybrid classical-quantum neural networks present in literature work due to their classical pre-processing or post-processing which does the heavy lifting, while the quantum part is almost superfluous. To avoid this, in our opinion a purely quantum NN is

needed.

We are thus ready to present the first project of this Ph.D. thesis: the Mixture of Quantum Experts (MoQE) [8], a new quantum neural network architecture which we call for image classification, able to classify the parity of the MNIST dataset with full resolution with a test accuracy of up to 97.5% without any classical pre-processing or post-processing.

We chose the task of parity classification because it is a binary task easy to implement, but also because it is a non-trivial task with respect to simply distinguishing two digits.

The architecture is inspired by the structure of convolutional neural networks (CNNs) and by the mixture of experts (MoE) where the cost function is the sum of the cost function of each expert, and where all experts are trained together. The input is encoded with amplitude encoding [9], which as we will see will allow us to use only 10 qubits, and to implement a convolution using a single one-qubit gate.

To our knowledge our work is the first instance of a purely-quantum neural network that reaches an accuracy of up to 97.5%, without classical pre- or post-processing, to a non-trivial task such as classifying the parity of the digits of the MNIST dataset. It also shows the potential to achieve high-accuracy image classification with minimal quantum resources, as we use a very small number of qubits. As the architecture was not optimized for the MNIST dataset and is general for all vision tasks, our MoQE may pave the way for more scalable and efficient quantum machine learning models.

We presented the MoQE in our paper [8] and here we reproduced its content in chapter 5. Conclusions and perspectives are presented in chapter 6.

Part III presents our study, through classical simulations and analytical calculations, of the quantum algorithm presented in [10] to solve nonlinear differential equations.

As quantum neural networks are the first main branch of Quantum Machine Learning, quantum speedups based on the HHL algorithms or on its variants are the second.

Before delving into the quantum algorithm for *nonlinear* differential equations, we present the quantum algorithm for *linear* ones (chapter 7). The algorithm is the HHL algorithm [11], an ingenious way to solve a linear system and encode its solution in a quantum state in logarithmic time on the input. The HHL algorithm is explained at length. Then it is used to build the quantum algorithm to solve linear differential equations. In the following

chapter (chapter 8), we briefly overview the literature on quantum algorithms to solve nonlinear differential equations, and then we explain the algorithm developed in [10]. This is the one we study in the second project of this Ph.D. thesis.

The main idea of the quantum algorithm is to make many different copies of the quantum state that encodes the dependent variable. Then, the interactions between the copies changes the state so that a single copy, after tracing out all other copies, behaves as if the variable changed with a nonlinear differential equation.

This is the main idea, but there are some caveats. One problem is that this interaction is obtained through a non-hermitian hamiltonian, thus one must take care of normalization. A second problem is that the interaction between the copies generates entanglement, so that the composite state approximation (*i.e.* our approximation of the complete quantum state of all copies as a tensor product of separate copies) - which was used to calculate which non-hermitian hamiltonian acting on all copies corresponds to which nonlinear differential equation for the variable encoded in a single copy - becomes more and more untenable.

These problems are discussed directly in our study of the quantum algorithm applied to a simple nonlinear differential equation particularly suitable to be treated by it, which is done in chapter 9.

We start by applying the quantum algorithm to solve the desired equation. We show that the normalization procedure presented in the Supplementary material of [10] cannot be performed, thus we find an alternative and simple way to perform it, which has different variants called *modes* which however give very similar predictions. We thus decided to use a *qubit* model, in which the dependent variable is encoded into a single qubit, and then multiple copies of that qubit are taken.

Then, we present the results of our experiments: we wrote two codes that performed classical simulations of the *qubit* model for the desired equation, and then we solved the model analytically through the use of a saddle point calculation, which is exact in the limit of infinitely many copies of the qubit. All results of the two simulations and of the analytical calculation agree among themselves, but they do not agree with the actual solution of the original equation to be solved, thus showing that the algorithm does not succeed in finding the desired solution. We discuss this in chapter 10.

1.2 Outline of the thesis

The thesis is organized as follows.

- **Part I**

- **chapter 2** is an introduction to quantum mechanics, quantum computing and quantum algorithms. We review the progress of the field in the light of the Quantum Revolution of one century ago, and the different directions of recent research such as the quest for quantum advantage and of applications of quantum technology, with a particular focus for the subjects of our two projects: quantum machine learning and quantum algorithms for nonlinear differential equations.

Then the thesis is divided in two main parts to introduce and present our two projects.

- **Part II**

We present our first project, the Mixture of Quantum Experts (MoQE).

- **chapter 3** is an introduction to classical machine learning, the field also known as Artificial Intelligence.
- **chapter 4** is an introduction to quantum neural networks, with their difficulties.
- **chapter 5** presents the MoQE, the quantum neural network we invented to solve classification tasks for images. We present the architecture of the circuit explaining the reasoning behind its structure, and we present the results of our simulations of the quantum circuit.
- **chapter 6** Conclusions and perspectives related to our work on the MoQE.

- **Part III** We present our second project, our study of the quantum algorithm presented in [10] to solve nonlinear differential equation.
 - **chapter 7** We present the HHL quantum algorithm to solve linear systems, and how to use it to solve linear differential equations.
 - **chapter 8** We present the quantum algorithm of [10] to solve nonlinear differential equations that we intend to study.
 - **chapter 9** We present classical simulations and analytical calculations to study the quantum algorithm previously discussed and its limitations.
 - **chapter 10** Conclusions and perspectives related to our work on the quantum algorithm for nonlinear differential equations.

Chapter 2

Quantum computing

2.1 Quantum mechanics

This thesis is built upon a conceptual foundation which is nothing less than the most philosophically radical and yet mathematically simple theory of fundamental physics, *i.e.*, Quantum Mechanics. Until now only five truly independent frameworks or paradigms of physics have been discovered: classical mechanics, quantum mechanics, special relativity, general relativity and thermodynamics. Theoretical physics at its highest points have tried to unify these different domains. To give just some examples: Quantum Field Theory is the framework that unifies quantum mechanics with special relativity through the concept of quantum field; Quantum Gravity is the quest to describe the quantum effects of gravity, usually by treating general relativity as a quantum field of some sort, as is attempted in string theory; Black Hole Thermodynamics studies the thermodynamics of black holes which involve quantum effects of gravity at the lowest order; and Quantum Thermodynamics tries to apply and extend the laws of thermodynamics to quantum systems.

Even in of itself, Quantum Mechanics was a conceptual revolution for our view of the world. It emerged about one hundred years ago after thirty years of surprising and confounding experiments that challenged the established theories [12]. Even today, in stark contrast with the other theories, it is difficult to define its main principle: as the famous physicist Richard Feynman remarked, “If you think you understand quantum mechanics, you don’t understand quantum mechanics”.

We will however attempt to explain its main ideas.

Quantum Mechanics, as many of the other fundamental theories of physics, can be seen in many different ways. Feynman himself pointed out [13] the interesting fact that the whole mystery of quantum mechanics is contained in the double-slit experiment: if we have a device that generates quantum particles, e.g. electrons, one at a time and let them pass through a wall pierced with two parallel slits towards some detector beyond the two slits, we have that each quantum particle behaves like a wave during the travel through the slits, making interference with itself (more precisely, with its possible different travel routes), but it behaves like a particle when it arrives to the detector, which always detects full particles with the same intensity, unlike the fluctuating intensity of waves. Deepening the mystery, there is the fact that keeping the detector fixed, the frequency of detection (which is proportional to the probability that the detector arrives at destination) is directly proportional to the square of the intensity at the point of the detection of a wave that traveled through the double slit.

The Schrödinger picture of Quantum Mechanics is that a single particle after it is emitted and before it is detected behaves like a wave, called a wave function $\Psi(\mathbf{x}, t)$, which evolves according to the Schrödinger equation; and that the probability that it is detected at coordinates $(\mathbf{x}, t)$ is simply $|\Psi(\mathbf{x}, t)|^2$. In simpler words: the value $\Psi(\mathbf{x}, t)$ is a complex number called the **amplitude** of the wave function, and it evolves in a deterministic way with a differential equation. The absolute square value of the amplitude at a given point gives the probability of finding the single particle at that point (technically, the probability density). The fact that the theory gives only a probability means that there is some intrinsic indeterminacy which escapes the theory and in which no regularity has been found until now: this is what is meant when it is said that Quantum Mechanics is a nondeterministic theory. The output of the experiment cannot be fully predicted from its input: only probabilities can. This is what Einstein could not accept and drove him to protest that God does not play dice. However, for what we know now, it seems He does. This simple analysis already reveals the most surprising features of quantum mechanics: the intuitive notion of causality has to be reviewed as the result of the measurement at least in part *does not have a cause* and is nondeterministic; the intuitive notion of trajectory has to be reviewed as between emission and absorption a particle does not have a single trajectory but behaves like a wave passing through both slits and the trajectories interfere between themselves.

The picture may already be complicated, but it is still easily visualizable. However, this is true only in the case of a single particle. When one has two particles at positions $\mathbf{x}$ and $\mathbf{x}'$ respectively, the two particles are not represented at each time by two three-dimensional wave functions, but by a single six-dimensional wave function $\Psi(\mathbf{x}, \mathbf{x}', t)$. In general, if we have N particles the wave function lives in a $3N$ -dimensional space and is not easy to visualize. This is very important: the state of the world is described together as a single vector in a big vector space.

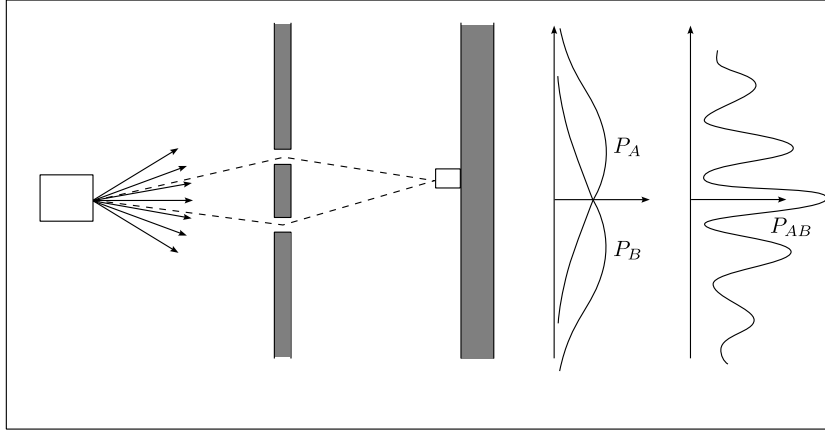


Figure 2.1: The two possible trajectories of the particle through the two slits are indistinguishable, the amplitudes are added together (sometimes with constructive, sometimes with destructive interference) to get the total amplitude. By taking the square norm, one gets the total probability of finding the particle, which shows the interference pattern. The image is an original reproduction from Figure 1.3 of [13].

Another interesting aspect of the two-slit experiment: if one illuminates one of the slits, so that the light interacts with the electrons, then one can measure the light and know the slit through which the electron passed. Thus the two trajectories become distinguishable, as the electrons are emitted one by one, and by measuring the light one knows precisely which electron goes through which slit. In this case, the electrons which interacted with the light are distributed as if they went through the first slit, and the others are distributed as if they went through the second slit: the interference pattern vanished! The two slit experiment is shown in Figure 2.1 for the case without light source, which shows the interference pattern; and in Figure 2.2 for the

case with the light source, which destroys the interference pattern.

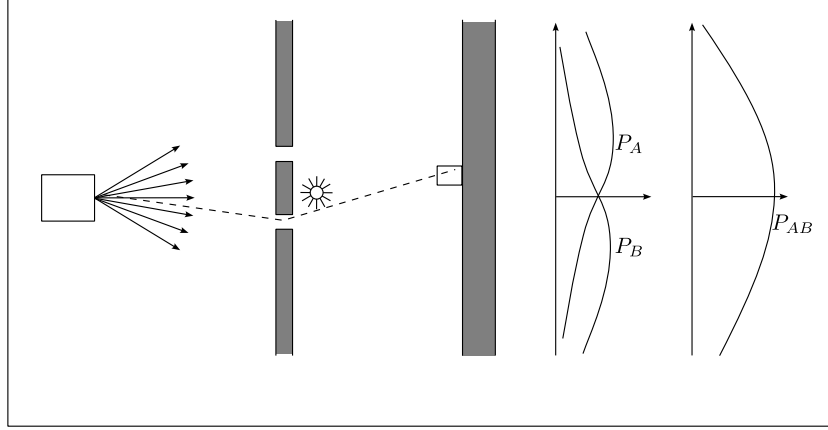


Figure 2.2: When a light source allows to determine the slit through which the particle passed through, the two possible trajectories of the particle become distinguishable, the probabilities are calculated separately with no interference and then are added together to get the probability of finding the particle, which shows no interference pattern. The image is loosely based on Figure 1.4 of [13].

This shows another crucial ingredient: different trajectories, *i.e.*, possible evolutions of the system, interfere if they are indistinguishable: their amplitudes are summed (like wave intensities) and the squared amplitude is the probability of the event; otherwise they do not interfere and are considered separate events: each of them is an amplitude, each of them is separately squared to give a probability, and then the two events have two probabilities which may be summed.

If we call Ψ_A the amplitude of the wave function in the case where only slit A is open, and Ψ_B the amplitude in the case where only slit B is open, we can clearly distinguish the two cases. If both slits are open and no light source reveals though which slit the electron passed through, then the two possible trajectories are indistinguishable (one cannot know through which slit the electron passed). In this first case, *the amplitudes add*:

$$\Psi_{AB} = \Psi_A + \Psi_B.$$

Squaring the amplitude to obtain the probability, we have

$$P_{AB} = \Psi_{AB}^2 = |\Psi_A + \Psi_B|^2.$$

In the second case, where the light source reveals the trajectory of the electron, the two possible trajectories are two distinguishable events, and thus one has to first calculate the two probabilities through squaring, and then the total probability is given as the sum of probabilities: as it would happen with classical events, if either of two or more distinguishable quantum events may happen, *the probabilities add*:

$$P_A = |\Psi_A|^2, \quad P_B = |\Psi_B|^2,$$

$$P_{AB} = |\Psi_A|^2 + |\Psi_B|^2.$$

The main idea to combine amplitudes is that for processes that happen in parallel *which are indistinguishable*, amplitudes are added (or subtracted if the particles are fermions, but we will ignore this fact in this presentation); while for processes that happen in series, amplitudes are multiplied. For distinguishable processes, amplitudes are not related.

This must be confronted to the usual rules to combine probabilities, where probabilities for processes that happen in series are multiplied, and for processes that happen in parallel are summed. In quantum mechanics one does these operations to amplitudes.

This view gives rise to what is usually called the path integral formulation of quantum mechanics. In classical mechanics, we describe the state of the world as a set of fields. Of the possible trajectories from state of the world A to state of the world B , the one that happens is the one that has a stationary (*i.e.*, maximum or minimum or inflection point with derivative equal to zero) value for the action S , which is a quantity that depends on the trajectory, *i.e.*, on the whole history of the fields from time t_A to time t_B .

In quantum mechanics, there is no single trajectory that happens: all of them contribute with an amplitude e^{iS} , and those that get to the same final state, and thus are indistinguishable *a posteriori*, are summed. Then, the modulus of the total amplitude is squared to give the probability. Only then, probabilities for distinguishable final states can be summed (e.g. I sum the probabilities of all distinguishable configurations in which an electron arrived at detector one, regardless of what happens to the other detectors).

As we defined it here, in the limit $S \gg 1$ we have that by changing trajectory the action is rapidly changing, thus the amplitudes rotates very fast in the complex plane and by summing them they cancel among themselves. Only near a stationary trajectory there is a bunch of trajectories which do not

cancel themselves: that is the reason why in classical mechanics it seems that only one trajectory happens, and it a stationary point of the action.

This path integral formulation of quantum mechanics is already sufficient to formulate the whole quantum field theory and is the basis of all theoretical physics, as it can easily incorporate the space-time structure of special relativity. The Standard Model of particles is a quantum field theory.

This interpretation gives the intuitive gist of quantum theory, but we will leave it as it stands. For our purposes, we concern ourselves to deepen the explanation of the world.

We thus present the **Dirac notation** for quantum mechanics, which will be used throughout the thesis. The idea is that the possible states of the world (*i.e.*, positions of particles or values of the electromagnetic field in each point, etc.) are described by N parameters and thus live in a Hilbert space $\mathcal{H}$. The current state of the world is one unit-norm vector in this space, and it is the ket $|\Psi\rangle$. In the ket notation, each column vector or ket $|\Psi\rangle$ has a respective hermitian conjugate, *i.e.*, a transpose conjugate, the row vector or bra $\langle\Psi|$. The norm $\langle\Psi|\Psi\rangle$ is one to ensure that the sum of probabilities of all possible outcomes is one, as it should.

We have now a deterministic evolution, given by

$$|\Psi(t)\rangle = U |\Psi\rangle$$

where $U = e^{-iHt}$ is a Unitary matrix obtained from the (possibly time-dependent, but here considered time-independent) Hamiltonian H . The matrix is Unitary so that the norm of the amplitude vector is preserved, and thus all probabilities keep adding to one.

To calculate the probability that an initial state of the world $|\Psi_A\rangle$ evolves to a final state of the world $|\Psi_B\rangle$, we first calculate the amplitude

$$\mathcal{A} = \langle\Psi_B| U |\Psi_A\rangle$$

and then we square it to get the probability:

$$P = |\langle\Psi_B| U |\Psi_A\rangle|^2.$$

If this eventuality is realized, *i.e.*, if a detector reveals that the final state of the world is indeed Ψ_B , then the state vector changed non-deterministically from $U |\Psi_A\rangle$ to Ψ_B . This gives rise to the famous measurement problem of

quantum mechanics, which asks about the reasons and dynamics behind this feature of the theory. No solution is possible without asking deep philosophical questions about what one means by a measurement or by observer or by classical world, so we will not venture to treat this problem here.

This formulation however constraints us to choose a single possible final state, while of course when we measure the position of a particle it can give different possible answers. The solution is that the possible results of a measurement are orthogonal and can form a basis of the Hilbert space (non-orthogonal measurements are called weak measurements and are a topic of recent research). For each measurement we can then define an operator Θ so that the expected value of the measurement is defined as

$$\hat{\Theta} := \langle \Psi | \Theta | \Psi \rangle ,$$

which for the case $\Theta = |\Psi_B\rangle \langle \Psi_B|$ reproduces the previous rule.

There is a final extension. One can create a **mixed state** which cannot be described as element of the Hilbert space $\mathcal{H}$. If we can produce the states $|\Psi_k\rangle$ for $k = 1, \dots, n$, we can easily arrange a state that with probability p_j corresponds to $|\Psi_k\rangle$. To express this state, we notice that by rearranging the previous equation we get the following formula for the expected value of a generic operator:

$$\hat{\Theta} = \text{tr} (\Theta |\Psi\rangle \langle \Psi|) = \text{tr} (\Theta \rho) ,$$

where we defined the density matrix ρ of a quantum system. As now the formula is linear in ρ , which is the variable that identifies the state, we can now write for the mixed state defined above

$$\hat{\Theta} = \sum_{k=1}^n \text{tr} (\Theta \rho_k) = \text{tr} \left(\Theta \sum_{k=1}^n \rho_k \right) = \text{tr} (\Theta \rho) .$$

Thus we see that the states should be represented by a density matrix which satisfies

$$\rho \in \mathcal{L}(\mathcal{H}) \quad \rho \geq 0 \quad \text{tr}(\rho) = 1.$$

This formulation is the most general.

Quantum mechanics was formulated for the first time one hundred years ago, in 1925, as the matrix mechanics by Werner Heisenberg, but it was a collective process. Its impact in our notions of theoretical physics has

obviously been enormous, and probably philosophically more challenging than the revolutions of special or general relativity, or even chaos theory. It also changed the lives of many of us giving rise to the development of transistors and electronic components, lasers, LEDs, magnetic resonance imaging, fiber-optic communications, a better understanding of chemistry and of materials, atomic clocks, sensors.

However, in the last few decades new far reaching applications are appearing. Some are refinements of old technologies, like quantum sensors or semiconductor materials. Others are new technologies, mostly related to the thermodynamic properties of small open systems, to their emission and absorption properties (like quantum batteries [14]); some are more dubious but they are pursued nonetheless (like quantum radar [15] and many others which will be briefly touched later); or to quantum computing. It is in particular to this last topic that researches and governments turned their attention. This will be the subject of the next section.

Before going on to quantum computing, we want to introduce a crucial idea of quantum mechanics which is absent in classical mechanics: **Entanglement**. This is a property where the quantum state of the parts of a physical system cannot be described independently by the other parts. This idea was introduced by Einstein, Podolsky and Rosen in the famous EPR paper of 1935 [16], where they described pejoratively as a *spooky action at a distance* (as some parts of a system *seem* to influence other parts even if they are at a considerable distance: this is however only apparent, as sending information instantaneously using Entanglement is impossible), and saw it as a sign that quantum mechanics was not the definitive theory of Reality. The effect was however observed, and quantum mechanics after its first hundred years still seems to be the framework in which all *known* laws of fundamental physics are written.

To explain Entanglement, let us get back to the double slit experiment. When the quantum particle is emitted, it is at a precise location: after it passed through the slits and is about to reach the detector in the final wall, it is in a so-called **quantum superposition**, or just *superposition*, of states. This means that it is not true that it either passed through the first slit or through the second one: its state can only be explained by its amplitude, which is the amplitude of a particle that passes through the first slit, and the amplitude of a particle that passes through the second one. The mathematical framework of quantum mechanics, which will be explained in the next section section 2.2,

is based on this idea: the state of a quantum physical system, also called the quantum state, is described by its amplitude $|\Psi\rangle$. In the case of the double slit experiment, the amplitude is a superposition of two states:

$$|\Psi_{AB}\rangle = |\Psi_A\rangle + |\Psi_B\rangle$$

One of the most famous popularization of the concept of superposition is Schrödinger's cat, a thought experiment in which a cat is in a superposition of being alive and being dead, as shown in Figure 2.3. In practice, only small physical systems can exhibit quantum superposition.

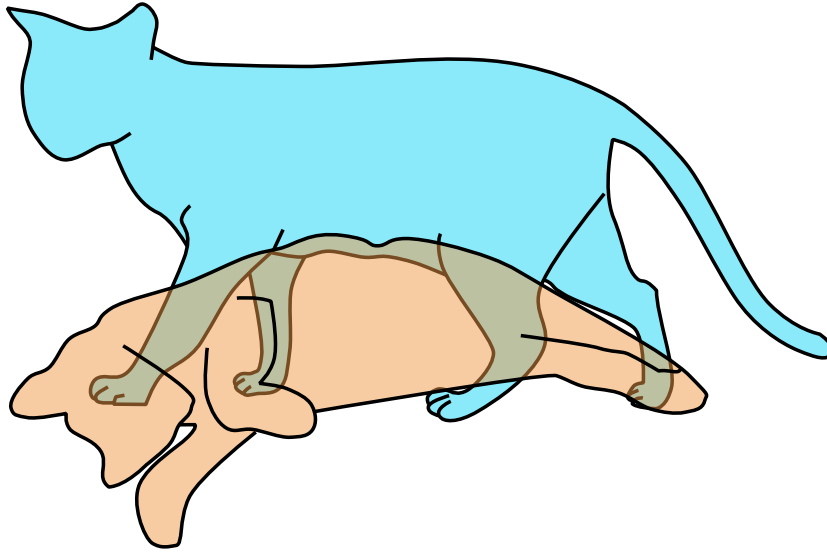


Figure 2.3: Schrödinger's cat is a thought experiment: if everything is made of quantum matter which can be in a superposition of different states, then also macroscopic objects can. Thus a cat can be in a superposition of being alive and being dead. In reality, interaction with the outside environment destroys superposition (through a process called decoherence), making the qubits of quantum computers very fragile, and Schrödinger's cat practically impossible. The image is loosely based on image https://commons.wikimedia.org/wiki/File:Schrodingers_cat.svg by Smite-Meister, released under the Creative Commons Attribution-Share Alike 3.0 Unported licence.

Now, let us imagine another situation: a system with spin zero (i.e. zero internal angular momentum) emits two photons, the particles of light. As

a photon can have only spin plus one or minus one (called spin up or spin down respectively), and as the sum must be zero to preserve the angular momentum, the possibilities are

$$|\uparrow\downarrow\rangle, |\downarrow\uparrow\rangle.$$

If our light source is symmetric, the emission should be equiprobable. Thus the two photons emitted are in the state

$$|\Psi\rangle = \frac{|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle}{\sqrt{2}}$$

where the multiplicative constant is added to normalize the state, as explained in the next section section 2.2.

Now, the quantum state of the two photons has a fascinating property: after the two photons are separated by a certain distance, even of many meters, one can measure the first photon and learn, for example, that it is in the state $|\uparrow\rangle$; as a consequence, *we instantaneously know that the second photon is in the state $|\downarrow\rangle$* (and viceversa: if the state of the first is spin down, the state of the second is spin up).

One possible attempt to explain this nonintuitive phenomenon was that the two photons, when they are created, have already decided which has spin up and which has spin down, just like if I hold a coin in one of my hands, each hand already knows if it has a coin or not, before being measured. In technical language, this means that there are *local hidden variables*: the particles already decided their spin, but this information is hidden, but is localized in each of the two particles. This idea was ruled out by Bell's theorem in 1964 [17]. There may be nonlocal hidden variables: somewhere, the particles have already decided which has which spin. However, those who may believe that the particles have already decided the outcomes for all possible measurements (like spin left or spin right, and all other possible directions), *i.e.* that they already decided their state independently of the measurement, which means *non-contextuality*, they are wrong: this is ruled out by the Kochen-Specker theorem, proved by Bell in 1966 [18].

In brief, Entanglement has no intuitive explanation up to now, and it is the reason why Feynman said that no one really understands quantum mechanics. It must be taken as a primitive phenomena, which itself will help understand other more complex phenomena.

2.2 Quantum computing

Quantum computing was advocated by Feynman [19, 20] while talking about the possibility to simulate Nature by doing the calculations on a computer, an idea which comes from Laplace. Feynman remarked:

I'm not happy with all the analyses that go with just classical theory, because Nature isn't classical, dammit, and if you want to make a simulation of Nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem!

Feynman did not develop the theory very far. The main idea is that, like computers have a universal framework that uses bits, zeroes and ones, and that can allow to compute everything that a classical system may compute; so a universal framework for quantum computers was developed that uses qubits.

A **qubit** is simply the quantum analogue of a bit. A bit is a unit of measure of information, and is the maximum quantity of information that can be stored in the simplest non-trivial system, a two-state system, with states 0 and 1. Analogously, a qubit is a unit of quantum information. This is a primitive concept, that is to be understood as the kind of stuff that a quantum computer can memorize and work with. A qubit is the maximum quantity of quantum information that can be stored in the simplest non-trivial system, a two-state quantum system, *i.e.*, a quantum system with a basis of two states: $|0\rangle$ and $|1\rangle$. This is called the computational basis: when we have n qubits, the basis starts with $|00\dots 0\rangle$ and goes through all 2^n possible strings of length n made of zeroes and ones.

The most general qubit is thus

$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle,$$

where α and β are complex numbers. We also recall that if we measure the qubit state in the computational basis, we either get $|0\rangle$ or $|1\rangle$, and the respective probabilities are

$$|\langle 0|\Psi\rangle|^2 = \alpha^2, \quad |\langle 1|\Psi\rangle|^2 = \beta^2,$$

where we used the fact that the computational basis is orthonormal, *i.e.*

$$\langle 0|0\rangle = \langle 1|1\rangle = 1, \quad \langle 0|1\rangle = \langle 1|0\rangle = 0.$$

$$|\Psi|^2 = \langle \Psi | \Psi \rangle = 1 \rightarrow \alpha^2 + \beta^2 = 1,$$

Figure 2.4: The Bloch Sphere. The computational basis $|0\rangle$ and $|1\rangle$ is shown at the poles. In the equator the state $|\alpha\rangle$ is defined as the state $\frac{1}{\sqrt{2}}(|0\rangle + e^{i\alpha}|1\rangle)$. A general pure state is on the surface, is identified by the latitude θ (calculated from the north pole) and the longitude ϕ , and it represents the quantum state $|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$. The points in the interior of the Bloch sphere do not represent pure states but mixed quantum states: the average point among two pure states is simply prepared as the first or second pure state with 50% probability for each case.

These two degrees of freedom can be represented as the surface of the Bloch sphere (see Figure 2.4). The state which forms an angle θ with $|0\rangle$ and angle ϕ from an arbitrary meridian, corresponds to

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

Each point in the surface of the Bloch sphere thus represents a pure state Ψ which is a complex combination of $|0\rangle$ and $|1\rangle$ with norm (*i.e.*, $|\Psi|^2$) equal to one. When one measures a state in a certain orthonormal basis, which corresponds to two opposite points of the Bloch sphere, with which our state has an angle $\tilde{\theta}$, the probabilities of the two possible measurements are

$$\cos^2\left(\frac{\theta}{2}\right), \quad \sin^2\left(\frac{\theta}{2}\right),$$

respectively.

We briefly mention the fact that when we treat not only pure states but also mixed states, *i.e.*, states represented with density matrices, also the interior of the Bloch sphere becomes accessible. The classical mixture of two pure states Ψ_1 and Ψ_2 with probabilities p and $1 - p$ respectively, is the point in the segment that connects the two pure states that divides the segments in two parts of lengths proportional to p and $1 - p$. These states will not be present in an ideal quantum circuit, but they emerge when the qubits interact with the environment, and they lose coherence (a process called decoherence) and usually collapse to the maximally mixed state $\rho = \frac{1}{2}$, which is at the center of the Bloch sphere Figure 2.4.

2.3 Quantum gates

Now let us introduce the **one-qubit gates**. We can act upon our qubit state with a 2×2 Unitary matrix given by exponentiation of a Hamiltonian (*i.e.*, a hermitian matrix that appears in the Schrödinger equation and that gives time evolution) in the following way:

$$U(t) = e^{-iHT}.$$

The most general two-dimensional Hamiltonian is

$$H = h_0 \mathbb{1} + h_1 \sigma_x + h_2 \sigma_y + h_3 \sigma_z$$

where we introduced the Pauli matrices

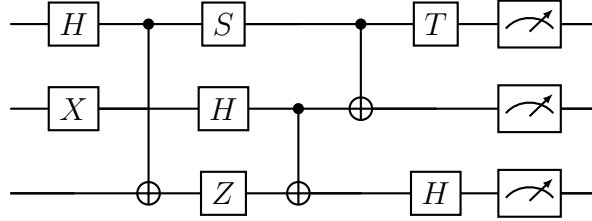
$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Through these evolutions we obtain the more general one-qubit gate: a gate is simply a Unitary operator (a Unitary matrix) that acts on a single qubit. It is represented as the following figure:

$$|\psi\rangle \text{ --- } \boxed{U} \text{ --- } U|\psi\rangle$$

in which the wire represent the qubit that goes left to right, and in the middle we act on the qubit with the most general 2×2 unitary matrix $U(t)$.

A **quantum circuit** is a specific arrangement of qubits and of gates which from an input state calculate an output state which can be measured. An example is the following circuit with many gates and entanglement:



where the qubits are initialized as some input state $|\Psi\rangle$, which encodes the data to be elaborated. Then the gates are applied: they are unitary matrices that act on the qubits. In the end the result is measured, for example with single-qubit measurements. One could also implement a two-qubit measurement or other things, but they are more complex to implement.

Let us give a quick review of some famous one-qubit gates and two-qubit gates. We have the **Pauli gates**, which can be used directly as unitary matrices. The σ_x gate is called the *bit flip* gate, or the *not* gate, as it exchanges the states in the computational basis, thus generalizing the classical NOT gate:

$$\sigma_x(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle.$$

The σ_z gate is called the *phase flip*, because it multiplies the state $|1\rangle$ by a phase $e^{i\pi} = -1$. These multiplicative phases in the amplitude are of course

not measurable, as only the square of the amplitude gives a probability, but the physical effect is the relative phase between the coefficients of $|0\rangle$ and $|1\rangle$:

$$\sigma_z(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle.$$

The σ_y is simply a combination of the previous two:

$$\sigma_y = i\sigma_z\sigma_x.$$

Then we have the **Phase gate**, which gives a continuous relative phase between $|0\rangle$ and $|1\rangle$:

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix},$$

so that

$$P(\phi)(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle + e^{i\phi}\beta|1\rangle.$$

For $\phi = \pi$ we get back the *phase flip* gate.

We can also obtain the **Pauli rotations**

$$R_X(\alpha) = e^{-i\alpha\sigma_x}, \quad R_Y(\alpha) = e^{-i\alpha\sigma_y}, \quad R_Z(\alpha) = e^{-i\alpha\sigma_z}.$$

We also have the **Hadamard gate**. By defining

$$|+\rangle := \frac{|0\rangle + |1\rangle}{2}, \quad |-\rangle := \frac{|0\rangle - |1\rangle}{2}$$

we have that the Hadamard gate

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

maps the computational basis to the one we just defined:

$$H|0\rangle = |+\rangle, \quad H|1\rangle = |-\rangle,$$

and vice-versa:

$$H|+\rangle = |0\rangle, \quad H|-\rangle = |1\rangle.$$

We end the treatment of one-qubit gates by recalling the theorem stating that the set of gates $\{P(\phi), H\}$ is a universal set for one-qubit gates. More precisely, for each one-qubit gate U we can write

$$U = P(\alpha)HP(\beta)HP(\gamma).$$

We can also write an analogous statement for rotations, *i.e.*, for the set of gates $\{R_X(\alpha), R_Y(\beta)\}$:

$$U = R_y(\alpha)R_x(\beta)R_y(\gamma),$$

where instead of x, y we could have used any two axes, by symmetry.

We now introduce the two-qubit gates. The most important are just two of them.

The first is the **Controlled NOT**, which when defined in the computational basis of the first qubit, it applies a σ_x to the second qubit only if the first qubit is $|1\rangle$, and does nothing if the first qubit is $|0\rangle$:

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

The second is the **Controlled U**, or *Controlled unitary*: it applies a U matrix to the second qubit only if the first qubit is $|1\rangle$, and does nothing if the first qubit is $|0\rangle$:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} \\ 0 & 0 & U_{21} & U_{22} \end{pmatrix}.$$

We end this brief overview by presenting the theorem that the set of quantum gates $\{P(\phi), H, \text{CNOT}\}$ is a universal set for quantum computing, provided that we can use a CNOT gate among any chosen pair of qubits. The same is clearly true for the set $\{R_X(\alpha), R_Y(\beta), \text{CNOT}\}$.

2.4 Quantum algorithms

The paradigm of quantum computing we have seen has been explored and developed in the last fifty years. It provided a generalization of classical computing, and the hope was to find some problem for which there was a provable **quantum advantage**, *i.e.*, the problem is hard to solve on classical computers and it requires a long time, while it is easy to solve on quantum computers.

If then a quantum computer can solve in practice a problem that no classical computer could solve in reasonable time, the achievement is called **quantum supremacy** [21].

The idea of a quantum advantage seemed reasonable as a quantum computer can do all operations a classical computer can: all classical gates can be reproduced by quantum ones, and the computational basis mimics exactly classical bits.

But the main idea is that quantum mechanics allows more operations, for example by allowing the use of *Entanglement*. In other fields than computation, the Entanglement property of quantum states already allowed to surpass classical results. One example is the use of Entanglement to show that quantum mechanics can violate Bell's inequalities, while classical mechanics cannot [17].

The main intuition is to use entanglement as a quantum resource (see section 2.5). This form of reasoning allows one to use it to do quantum teleportation, *i.e.*, teleport a quantum state from a place to another by simply exchanging an entangled pair and sending two classical bits to the destination. It also allows other interesting results for quantum communication, but they will not be treated.

Now, after fifty years, it can be said that the most general algorithms that show some form of quantum advantage are mainly four. We will briefly review them.

The first one is **Shor's algorithm** for factorization. It was developed by Peter Shor in 1994 [22, 23]. The problem is the following: we have a number of the form pq where p, q are two prime numbers that we are meant to find. This problem is believed to be unfeasible in polynomial time on classical computers; that is the reason for the RSA algorithm for public-key cryptography. This is a widely used cryptographic system that guarantees the security of information, and it does so by having each agent have a private key composed of two prime numbers p, q , and a public key $n = pq$ which anyone can have access to, and another integer number e (with an additional requirement we will explain soon). Now if we want to send a message m to that agent, we encode it by calculating $m^e \bmod n$, which can be calculated just with the public key variables n, e .

Now, the agent that want to decode the message can use the formula

$$a^{(p-1)(q-1)} = 1 \bmod n,$$

which applies for each a coprime with n . This can be improved to

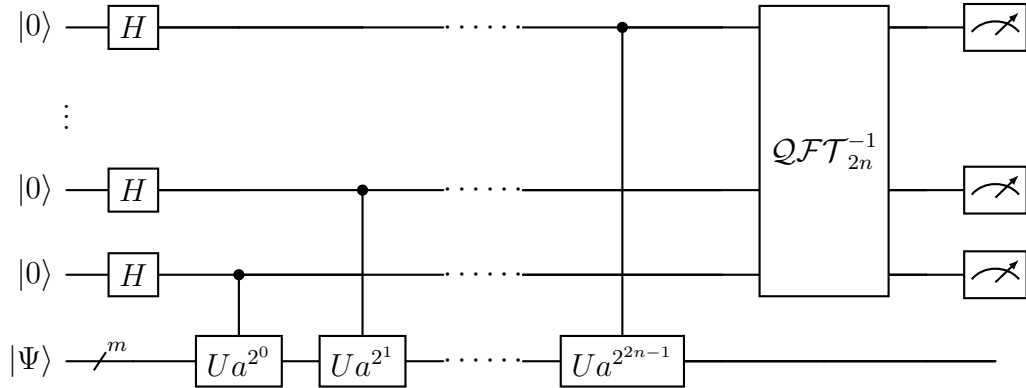
$$a^{\text{lcm}((p-1)(q-1))} = 1 \pmod{n}.$$

In this way, the agent can easily decode the message by finding the d such that

$$(m^e)^d = m \pmod{n}.$$

It is clear that this cryptographic procedure can be broken if one can easily factorize the public key pq into p and q . Shor's algorithm does precisely that, making use of the quantum phase estimation algorithm, which makes use of an operator called the Quantum Fourier Transform (QFT , not to be confused with QFT or Quantum Field Theory).

Here we show the quantum circuit for **Quantum Phase Estimation**, to give a sense of the complexity of the algorithm:



where the last wire represents a bundle of m qubits, initialized in the quantum state $|\Psi\rangle$.

Recently (2023) an enhancement to Shor's algorithm appeared in the form of the Regev algorithm, which achieves a complexity of $O(n^{3/2} \log n)$ quantum gates compared to Shor's $O(n^2 \log n)$, but it requires more qubits [24]. However, the main result is that factorization can achieve an exponential speed up over the known classical solution (we notice that it is not known if the classical solution is optimal, even though it is believed so; its runtime is $\exp \tilde{\Omega}(\sqrt[3]{n})$ and the problem is not NP complete).

Shor's algorithm is already having a strong impact in cryptography, as it will allow quantum computers to crack both RSA and elliptic curve cryptography [25], which is the other widely used cryptographic system. The number of qubits necessary are on the order of thousands of logical qubits, which due to

decoherence and interactions with the environment will need about a million of physical qubits [26] (we will briefly delve into quantum hardware later). However, the prospective for current cryptography is dim. The solutions proposed is to shift towards **post-quantum cryptography**, *i.e.*, cryptographic systems specifically studied to be resistant to quantum algorithms; or more radically to quantum cryptography, *i.e.*, exchanging entangled quantum states to guarantee that private keys cannot be copied by an eavesdropping third party which is proved thanks to a property called monogamy of entanglement (maximal entanglement can be shared between two qubits but no more than two).

The second algorithm is **Grover's algorithm**, also known as the *quantum search algorithm*. It was discovered by Lov Grover in 1996 [27]. The task is an unstructured search problem: we have a black box function that for a unique specific input produces a particular output value. Classically, to find the input we should evaluate the function $O(n)$ in most cases (where n is the number of possible inputs of the function). Instead, there is a quantum algorithm that with high probability (which can be brought to $1 - \epsilon$) gives the solution with just $O(\sqrt{n})$ evaluations. This algorithm is known to be asymptotically optimal [28], *i.e.*, there will not come a solution with a different asymptotic. In 2017 Grover's algorithm was implemented in a scalable quantum computer [29].

Grover's algorithm can in theory speed up searches of unstructured classical databases, and people is looking for specific practical applications [30].

The third algorithm is the **HHL** algorithm. It was introduced in 2008 by Aram Harrow, Avinatan Hassidim and Seth Lloyd: hence the name. The HHL algorithm is capable of solving a linear system of n equations $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ in $O(\log(n))$ time! This is extraordinary, as simply to read the solution the time required is $O(n)$. This in fact has made it difficult to find a practical application, as suggested by Aaronson in [31]. However this algorithm will be explained in chapter 7.

The fourth algorithm is simply **quantum simulation**: the task is to approximate the time evolution operator e^{-iHt} of a given Hamiltonian H . For a longer discussion on why it counts as a true algorithm at the same level of the others, one can see the discussion of [32].

The four quantum algorithms we described are shown in Figure 2.5 Now that we examined the main results of the quest for quantum advantages of the last fifty years, it is time to ask some important questions. Are these the only possible quantum algorithms? Are they related somehow, or is there any

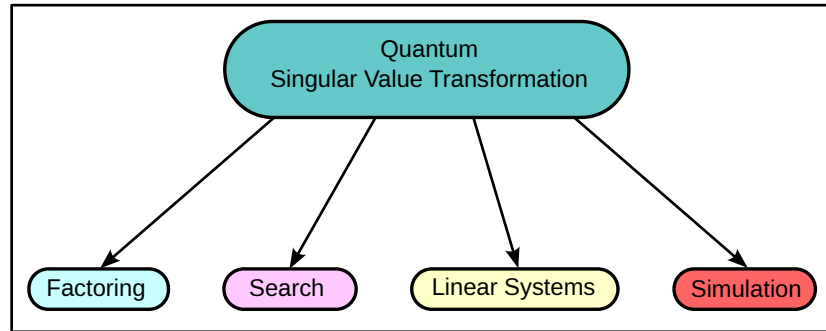


Figure 2.5: The four quantum algorithm we described in this section, seen in a unified framework [32].

other possible solution? This questions are related to deeper questions about what is computable or not in quantum mechanics and in classical mechanics, and how the computation scales with the input.

The calculability question is related to the general theory of computation. The Church-Turing thesis already hypothesized that the intuitive notion of computable quantity is captured by the universal Turing machine (a Turing machine that can simulate all other Turing machines), *i.e.*, any computable quantity in the intuitive meaning of the term, is computable by a universal Turing machine. By switching from classical to quantum computation, the set of computable functions does not change. The great question is whether functions that are hard to compute become easy to compute in the new setting.

The question of how difficult is a computational task, and how much resources like time and memory are needed to solve it, is the topic of **Computational Complexity theory**, or simply *Complexity theory*. To formalize the question in a way that can be tackled by mathematical methods, the idea is to study a problem for different dimension n of the input, and we study the time or memory required by a proposed algorithm to solve it, which is indicated by the big O notation. The main difference in complexity is thus between problems which can be solved in polynomial or exponential time.

There are however different complexity classes: P , NP , NP complete, $PSPACE$, EXP and so on. However, it is still unknown whether $P \neq NP$, and is one of the Millennium Prize problems, even though it is the consensus that P and NP are different. The introduction of quantum computers allows one to define new complexity classes, such as BQP and QMA . We will

not delve ourselves into the details (see Figure 2.6 for the basic inclusions between the most used complexity classes), but it is a fascinating subject that dives deep in the profound meaning of quantum theory and of its form of computation.

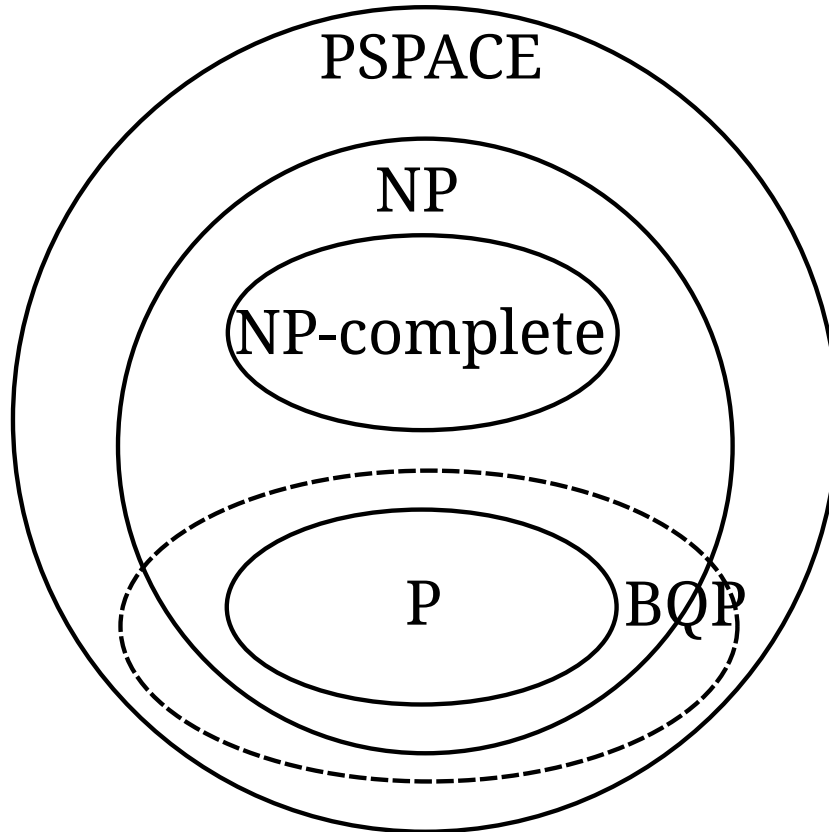


Figure 2.6: A basic scheme of the inclusions among the most used Complexity classes. It is widely believed that $P \neq NP$, even though it has never been proven.

The fact that the classical complexity theory is far from accurate, and is still based on consensus even on questions as in the $P \neq NP$ problem, has certainly made the certainty of quantum advantage more difficult. Often, like in the factorization problem, we can only say that a quantum algorithm is better than the *known* best classical solution. Which sometimes changes, as we will see with the quantum algorithm for recommendation systems in section 4.3.

Another interesting discussion on the possibility of quantum speedups, in particular to huge (exponential) ones, is [33].

The idea is that the lesson from Shor's algorithm, Grover's algorithm and the HHL algorithm is that some sort of abelian group structure is needed to have these exponential advantages. It is less clear if nonabelian group structure is sufficient, and with even less structured problems it seems that only polynomial speedups are possible (see Figure 2.7).

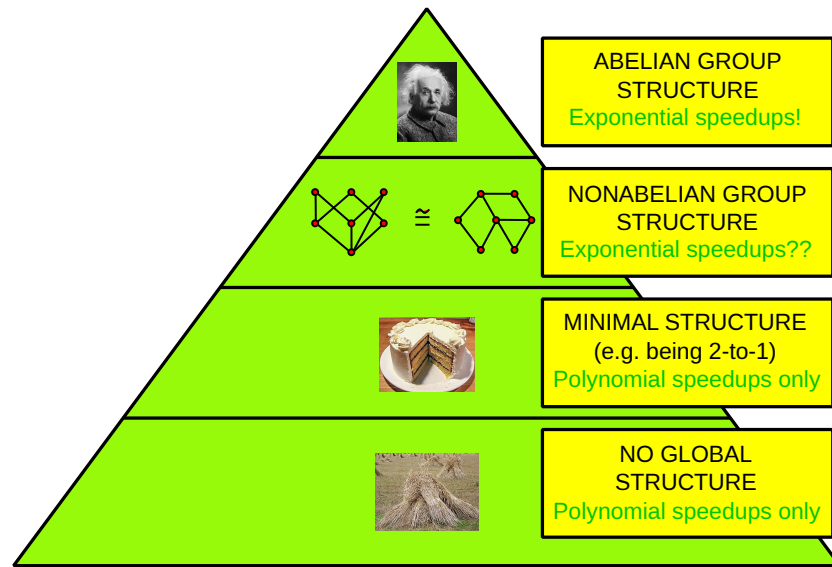


Figure 2.7: This image is based on figure 6 of [33]. It shows how the possibility of a quantum speedup, and how great the speedup is, may depend on the presence of structure in the input data of the task, which has to be encoded in the quantum circuit.

We will see in later chapters that the goal of this thesis is to tackle such less-structured problems. Thus a true exponential speedup is not probable; even though one can have an exponential advantage in some aspects (like in the number of qubits for our QNN in chapter 5, but with some catch in other parts of the algorithm).

We will stop the discussion for now. The quest for quantum advantage is still one of the greatest theoretical questions on the nature of quantum mechanics: is it a computationally different theory from classical mechanics, or not?

A natural question is to ask whether the change from Quantum Mechanics to

more profound theories like Quantum Field Theory, which is both quantum and relativistic, or the still speculative String Theory, would entail a change in the hardness of computation. The complexity of these theories and their variety makes reasoning hard, but the impression is that usually QFT computation gives no more results than quantum computation, but that string theory is so various that it is still not known. We recall that in a sense QFT and string theory are descriptions in different regimes of the same theory, as shown in weak-strong dualities such as the AdS/CFT correspondence.

2.5 Quantum resources

We want to explore briefly about the ways to think about quantum computation as a theory of resources. The question is: is there a simple way to think of quantum computation as classical computation plus a single and easily understandable extra ingredient X? Then, it would be easier to study the contribution the ingredient X can make to different problems.

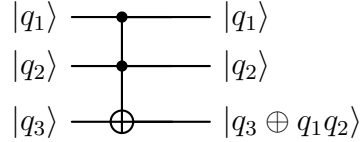
The early idea was that the quantum resource was simply **Entanglement**. It is certainly true that it is a resource for processes like quantum teleportation, and other results in quantum communication. However, there are circuits with a lot of entanglement that do not do anything more efficiently than a classical computer.

To understand this, let us think about an *efficiently classically simulable* quantum computer. By efficiently classically simulable, we mean that if to solve a problem requires a certain time $O(f(n))$ in it, it requires $O(\text{poly}(f(n)))$ time in a classical computer.

An easy example is to do exactly the usual classical computations in a quantum computer. The idea is this: we know that the **Toffoli gate**, or *Control-Control-NOT gate*, or *CCNOT gate* for short (as it is simply a controlled Control-NOT: the third bit is changed only if *both* the first and second bit are equal to one) is universal for classical reversible computation, which is classical computation that cannot destroy information. We notice that if we initialize the input qubits of a quantum computer in the computational basis, then by acting on them many times with a CCNOT quantum gate, *i.e.*, a three-qubit gate that does a bit flip in the third qubit only if the first two are in the state $|11\rangle$, and not if they are in one of the other three possible states $|00\rangle, |01\rangle, |10\rangle$ (the CCNOT gate can be built using many CNOT gates), then we can do all classical reversible computations, but we cannot do anything more. Thus

our quantum circuit will be easily classically simulable. The quantum circuit indeed has no entanglement. Thus entanglement seems necessary to have any sort of quantum advantage. This idea was explored more deeply in Jozsa *et al.* [?]: they find, with many important caveats, that if one builds a quantum circuit with no quantum entanglement is used, then no exponential quantum speedup is possible. The caveats include restricting the calculations to pure states. The paper itself states that entanglement should not be viewed as a key resource for quantum computational power.

We show the Toffoli gate and its action in the following quantum circuit. The $\oplus$ is the symbol for bitwise addition *i.e.* addition without reminders, between the third qubit and the product (logical AND) of the first two.



Another interesting fact is that there are quantum circuits which use and create a lot of entanglement, which are still classically simulable: a standard example are Clifford gates, which we will introduce in a moment. This does not mean that entanglement cannot be seen as a quantum resource: only that it is not a guarantee of quantum non-triviality. Indeed, the Clifford gates themselves offer us another candidate as a resource for quantum computation: it is the so-called **quantum magic**. Briefly, the idea is that there is a set of quantum gates called Clifford gates that is efficiently classically simulable: they are the Hadamard gate, the newly defined S gate or phase gate, and the CNOT gate:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \quad \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The Gottesman-Knill theorem assures us that if a quantum circuit initialized on a stabilizer state, *i.e.*, $|00 \dots 0\rangle$ with some Clifford gates applied, and then is acted upon again only with Clifford gates, and then measurements on the computational basis happen, then the computation is efficiently classically simulable [34].

The Clifford gates set is clearly non universal. Maybe surprisingly, it becomes universal [35] if one adds a single extra one-qubit gate, called the T gate:

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

The idea now is that a quantum circuit with Clifford gates can be doped with T gates and the more it is doped, the more the quantum circuit is not efficiently classically simulable. The quantity of T gates is then called *quantum magic*, and one can study different ways to measure it. A state has a certain amount of magic dependent on the number of T gates used, along with Clifford gates, to produce it from the $|00 \dots 0\rangle$ state. For low-magic quantum circuits, the simulation time increases exponentially with the number of added T gates [36]. This depends however on how the T gates are introduced, and maybe at least two layers are necessary to insure non-classical simulability [37]. Quantum magic is also related to other measures such as Wigner negativity and mana, which may be used to calculate it [38].

Quantum Magic has found applications in many implementations of quantum theory. For example, just as there are quantum phases of matter which can be classified by their patterns of long-range entanglement which can be absent, or present in different arrangements [39]; so there are phases of matter with a macroscopic quantity of magic [40]. One can study not only how entanglement increases and spreads in random quantum circuits, but also how quantum magic does [41]. The quantum magic of a Black Hole may also make it harder to reconstruct information from its evaporation [42]. It is also present at critical point in some Conformal Field Theories [43], while it is absent in almost all process of Quantum Electrodynamics [44]. Other research suggests that quantum magic and quantum entanglement are in competition and there are states dominated by one of the two [45] and that somehow both entanglement and quantum magic are needed for a non-efficiently classically simulable quantum circuit [46], so that a new measure called non-local quantum magic is introduced. In the context of AdS/CFT duality, some research suggests this non-local quantum magic is dual to gravitational backreaction [47], exactly as entanglement is dual to the space-time geometry (so that when entanglement disappears, lengths increase until space-time becomes disconnected, see [48]). Quantum magic is also related to the emergence of space-time and to chaotic regimes [49]. Finally, quantum magic allowed to prove that Quantum Chaos, a feature of chaotic

quantum system, is not efficiently classically simulable [50].

All these examples are to show that quantum magic is a very recent new

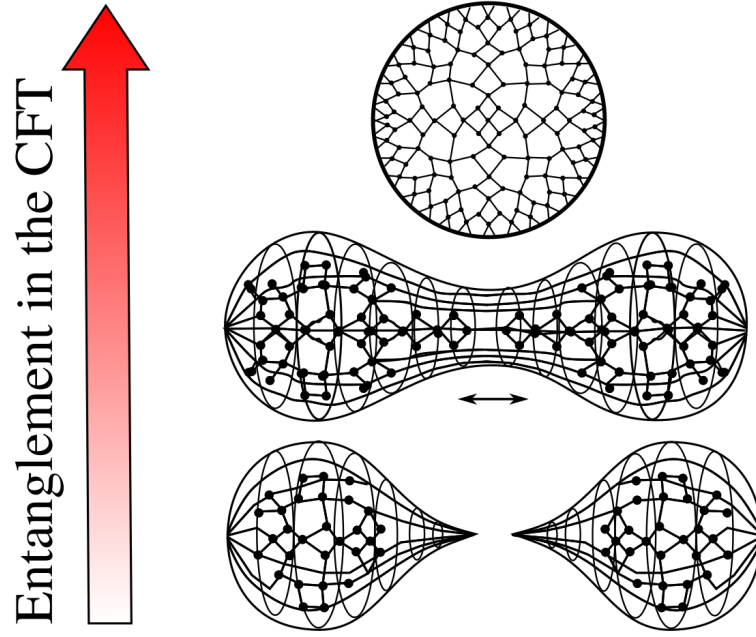


Figure 2.8: Entanglement and quantum error correction are not concepts confined to quantum computation or quantum mechanics, but are used throughout physics and have universal character: as an example, in AdS/CFT duality the entanglement of a Conformal Field Theory is dual to the space-time of a quantum gravity theory in Anti-de Sitter space; while quantum error correction codes give robustness to the space-time of the dual theory [48]

paradigm to understand what is strange, what is quantum, in quantum theory, just as entanglement does. It is surprising that the matter can be dealt with such a different approach, which was unveiled to human understanding almost a century after the discovery of quantum theory. The spontaneous question is whether there are other ways to conceptualize the strangeness of quantum theory, or if they are inescapable. In fact, there is at least another way, which we briefly mention: it is by thinking about a quantum resource dubbed **fermionic magic**.

Even this third example starts with a new class of gates which are efficiently

classically simulable [51], called *matchgates*. Just by adding an extra two-qubit gate, the SWAP gate (which trivially exchanges the two qubits), the set of gates becomes universal and hard to simulate classically (!). It was later found by Valiant *et al.* in 2001 that this surprising behaviour has a simple explanation: matchgates simulate a model of noninteracting fermions in one dimension, while the SWAP gates allow for interaction between fermions: while the noninteracting theory is not universal for quantum computing, the interacting theory is [52]. The same paper shows that there is no such thing as a bosonic magic, as for bosons the noninteracting theory is already universal. This leaves interesting questions for anyonic magic, which we will leave unanswered.

The quantity of SWAP gates is called *fermionic magic* or *interactivity*. Here we show an example of a SWAP gate:



This review of possible quantum resources should make us pause and think that it is not so obvious what makes quantum circuits difficult to simulate by classical computers. Thus, any quest for quantum advantage does not have a predefined framework that clearly explains where to look for possible quantum algorithms and where not to. As some of the progresses in efficient simulations of quantum circuits with fermionic magic is very recent [53–55], it is somehow expected that new paradigms or at least new breakthroughs will appear in the following decades.

2.6 Quantum hardware progress

We want now to talk about the progress in the realization of quantum hardware.

The theory of classical computation was developed by Turing in the 1950s, which was preceded by various insights by Charles Babbage and Ada Lovelace in the late 1800s, well before the first scalable computers were built: they required the transistor, the physical implementation of bits and logical gates, the computer architecture, the development of algorithms to solve problems. The same is true for quantum computation: after the insights of Richard Feynman, the development of quantum algorithms is ongoing since the 1980s. Quantum hardware is still difficult to obtain, and in this year (2025) the

biggest quantum computers have of the order of a thousand qubits. These are physical qubits, *i.e.*, physically implemented qubits. The greatest difference between qubits and bits is that qubits are intrinsically fragile: any uncontrolled interaction with the environment disturbs their state, which brings to decoherence and errors in the calculations. This happens to a very lower rate with classical bits. The solution is believed to be some sort of quantum error-correction: algorithms that encode a single logical qubit into several physical qubits, so that even if physical qubits are subject to errors, the logical qubit can correct its error faster than they appear, thus ensuring the correctness of the calculation. Quantum error correction constructions are very geometric, and can be related to tilings of the plane such as Penrose Tiling [56] and there is the suspicion that it is dual to space-time geometry in the AdS/CFT correspondence [57], showing that it is not just a practical necessity but a deep and profound structure, related to the universally meaningful task of insuring the stability of quantum information against environment interactions.

An impressive milestone was the recent implementation (December 2024) of scalable quantum error correction up to 100 physical qubits [58], as we showed in Figure 2.9. Without such a provable mechanism, the possibility of scalable quantum computer would be in jeopardy.

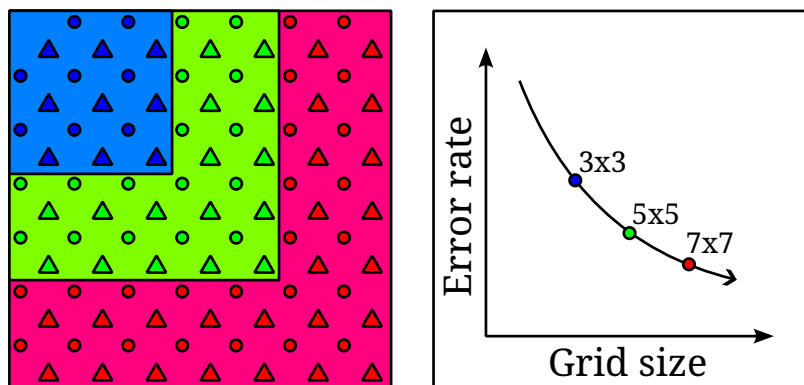


Figure 2.9: Recent progress in scalable quantum error correction such as the one in [58] gives hope that the mere possibility of scalable quantum computers will one day become a reality. Data qubits are shown as triangles and measurement qubits as circles. This image is loosely based on a figure by Marc Belan (Quanta Magazine) found in [59].

There is no consensus, but it is thought that to simulate a single logical qubit, about 1000 physical qubits are needed. Thus showing that a credibly stable logical qubit is now theoretically within reach.

Many private companies are investing in the realization of bigger and bigger quantum computers. As to this date (October 2025), IBM Quantum plans to deliver a system with 100 million quantum gates on 200 *logical* qubits in 2029.

An interesting question is when will RSA cryptography be cracked by quantum computers? It seems that about 1000 qubits are needed for 2048-bit RSA (a commonly used version), but far more quantum gates [26]. It is to be understood that by using 4096-bit RSA or higher, quantum computers can still win after a few years, as they scale better. Also, the intrinsic error of quantum gates forces us to require more physical qubits for a single logical qubit, and to use more quantum gates. An interesting discussion can be found here: [60]. There is no conclusion, but maybe millions or billions of physical qubits with some trillions of physical gates can do the job. IBM Quantum says that it will double the number of qubits every year, thus adding three orders of magnitude each decade, thus cracking 2048-bit RSA between 2045 and 2055. If this requires two years instead of one (as these predictions seem too optimistic), the result is that 2048-bit RSA will be cracked between 2065 and 2085. Time will tell.

For now, we are still in an era of non-scalable quantum computers. This has been called the Noisy Intermediate-Scale Quantum (NISQ) era by John Preskill in 2018 [61], as quantum computers are still noisy and with around 100 – 1000 qubits. Even though incapable of solving full-scale RSA, they should already be able to solve some tasks faster than classical computers. The first claim of quantum supremacy was made by Google in 2019 [62], but it was later found that classical algorithms could indeed easily solve the same task. Nevertheless, the time is ripe for the first concrete applications of quantum computers. The foundations of the hope that quantum computers have capabilities surpassing those of classical computers is best presented by Preskill again in the same paper [61], and is based on the following three arguments.

1. Quantum algorithms for classically intractable problems. We have seen some instances of problems which are thought to be classically hard to solve, like factoring, and we have an exponentially faster quantum algorithm. Maybe there is a faster classical algorithms, but none has

been found for centuries.

2. Complexity theory arguments. There are some results in complexity theory, based on some reasonable assumptions, that quantum states which are easy to prepare have superclassical properties *i.e.*, measuring them we can sample from probability distribution which are classically hard to sample from [63], [64].
3. No known classical algorithm can efficiently simulate a quantum computer. Despite many decades of effort, no efficient algorithm has been found to simulate quantum systems yet.

This is of course no guarantee of any quantum algorithm applicable to any task of human interest, or anything besides the quantum algorithms we already saw, but it is evidence that points in a certain research direction. Only time will tell.

2.7 Applications of quantum computing

The last topic of this chapter is the long term applications of quantum computers.

We briefly touched on the subject, but there are many reasons to think that quantum computer will not easily solve NP complete problems, and certainly they will not substitute classical computers in most tasks. The possibility of quantum advantage lies in carefully studied tasks, and the quantum algorithms rely on the four previously mentioned (see section 2.4).

The most certain application seems now to be the application of Shor's algorithm to break RSA and elliptic curve cryptography.

Other commonly cited applications are in quantum simulations of molecules in chemistry and biology, even though only a throughout study of the matter would allow to precisely say where a quantum advantage could lie. Another factor is that some previously intractable problems in this sector, like the folding problem of proteins, was recently solved at the lowest order by Artificial Intelligence, *i.e.*, by some form of *classical* computation (AlphaFold, 2020, presented in [65]). The problem consists in finding the structure of a protein given the amminoacidic sequence; a higher order problem would be to find the change in structure given a change in the amminoacidic sequence, and this problem is still to be solved. However, progresses in Machine Learning

may happen before quantum computers become viable in practice.

As we have already said, quantum computation is related to many other fields which are developing, like quantum cryptography, quantum communication, the attempt to build a quantum internet *i.e.*, a way to exchange qubits among the world; quantum sensors; quantum batteries and many more. A treatment of all these applications is beyond the scope of this review chapter, which is more focused to direct quantum computation applications.

Such direct application must rely on some form of quantum advantage which may be alternative to the four quantum algorithms reviewed before. There is hope as there are a lot of such algorithms which are occasionally found, even though not immediately applicable to a task of practical interest. Research is hard and the pay-off may seem distant at first, but it is necessary to discover new approaches and new territories of the imagination, which may lead to unforeseen solutions.

This thesis will focus on two attempts. The first is related to quantum machine learning, a relatively new branch that tries to combine the recent success of machine learning to the new setting of qubits and quantum computation. The second is a way to somehow extend the HHL algorithm from linear equations to nonlinear ones. Two parts of the thesis will be devoted to study each of these problems, with a couple of introductory chapters for each, in which we will discuss not only the prerequisites for our original research but also we will try to sketch somehow a comprehensive view of these two fields by exposing their hopes, their results, their difficulties and the failed and successful attempts at understanding how quantum computing can benefit Mankind as a whole.

Part II

A quantum neural network for image classification

Chapter 3

Machine learning

Machine Learning, also known to the public with the popular expression Artificial Intelligence, is a field which is exploding in the last decade both in performance and in investments [66] [67] and which promises to revolutionize both intellectual and material production on Earth and may well do so [68], and to change the conditions of Mankind for the better [69] or, hopefully not, for the worse [70].

The goal of this chapter is to briefly present its scope, its history, its main concepts and the lessons one can take from the field. As the first project presented on this thesis is on the topic of quantum machine learning, we presented before the general field of quantum computing and we now venture to explore the subject of machine learning. Only with such an understanding it will be possible to compare quantum machine learning with its classical equivalent, to understand the advantages and the greater difficulties, and to have a glimpse of some insights which are necessary to navigate the field.

That being said, we begin our exploration. It was said by Richard Feynman that one truly understand only what he can build. If that is true, then the field of Artificial Intelligence is not just the attempt to build some human analogue of Intelligence and of Understanding by scratch for economic gain, or even for humanitarian goals: it is the first serious attempt by Human Beings to understand their own Minds, or at least that universal and general rational faculty of their own minds that sets them apart from all other animals and which, for what we know now, may be truly unique in the whole observable universe with its hundred billions galaxies each with their hundred billions stars.

Science-fiction writer Greg Egan once was asked why he wrote stories which

delve into theoretical physics on a quest to the true nature of Reality in the Universe, and of Human Consciousness. He answered: *because if one gets rid of what exists, and of Consciousness, there's not much left.*

It is in this spirit that we, having already explored the ontological questions about the deep structure of the laws of theoretical physics that sustain the Universe, we embark on the remaining task of reviewing the surprising discoveries Mankind made in the last decade on the nature of what is Intelligence by trying to build it from linear algebra up to a general tool that can understand natural language and situations and solve complex problems, truly one of the most exciting human Adventures of all of Time.

We start by defining the terms.

The expression Artificial Intelligence is usually criticized by those who say that AI models are not truly intelligent, but simply mimic intelligence, and are in fact stochastic parrots and the like. In fact, Artificial Intelligence is a name that does not reflect everything that it is done in the field, but its goal: we will thus say that Artificial Intelligence is the field which has the goal to gradually reproduce all that Intelligence can do through artificial means, or more specifically classical computation. In particular, if this intelligence were truly general and can successfully approach all kinds of tasks which Humans deal with, it is called **Artificial General Intelligence**, or *AGI* for short, obtaining which is the goal of companies such as OpenAI and others.

In this vast field, there are many possible approaches (see Figure 3.1). The one that in the last decade has truly shown to work, and to be able to achieve extraordinary results and which may be able to deliver AGI in the not-so-distant future, is Machine Learning.

The main idea of Machine Learning, which means rote learning, or mechanical learning. is a simple yet extraordinary one, and is best explained through an example.

Let us imagine that we want to classify some images: we want to distinguish images that contain a cat from those which do not; or we have some images of handwritten digits (for example from the widely-used MNIST dataset) and we want to tell which digits are they.

The main paradigm of computer science until the advent of Machine Learning was to write software to define some function f , such that given a certain input it would give the desired output. For the task of image classification, this would entail to conceive a definition of cat which takes the pixels and tells if those pixels correspond to a cat or not, and then write it down in a

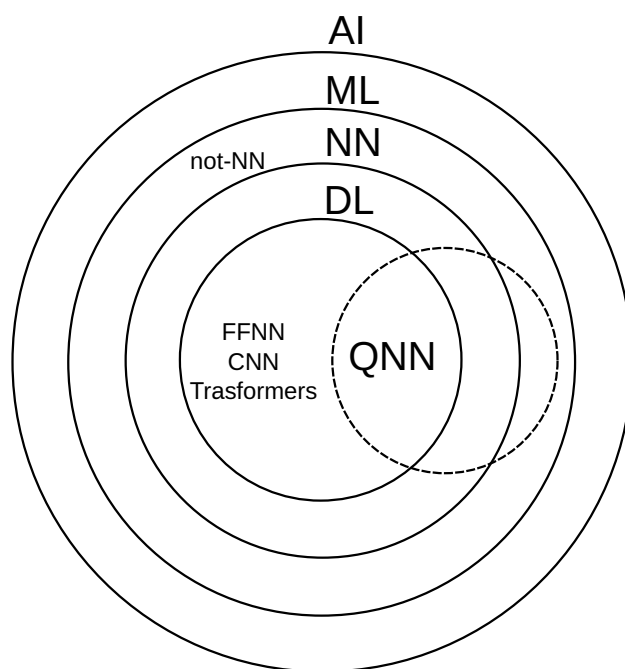


Figure 3.1: The subdivision of fields inside Artificial Intelligence: Machine Learning; which includes both non-Neural Network methods and Neural Networks; which includes Deep Learning, which includes FFNN, CNN and Trasformers. We also showed Quantum Neural Networks, which will be the topic of chapter chapter 4

program. This has always been unfeasible. An anecdote from the early days in AI, when it was not clear the difficulty of the problem, is that professor Seymour Papert and Marvin Minsky at MIT tasked an undergrad student, Gerald Sussman, to solve computer vision over the summer. Needless to say, the problem revealed to be much more complicated than imagined, or more precisely: it is simple, but only if done in a very different way, which was impossible with the technology and datasets of the time.

The core idea of machine learning is to renounce to write f yourself, and to create a system that allows the calculator to obtain f itself without human intervention! The way to do this is to give the calculator a lot of examples of what the still-not-existent function f should do, *i.e.*, thousands of already-classified images: each image must be associated to a human-assigned label that tells if there is a cat or not, or which digit it does represent.

Maybe surprisingly, a few thousand of examples is enough to allow a computer to calculate such a thing.

3.1 Feed Forward Neural Networks

To understand how machine learning works, we will go through a concrete example. We will solve a simple vision task, and in doing so we will introduce neural networks and the simplest neural network (NN) possible: the feed forward neural network (FFNN), shown in Figure 3.2.

Let us imagine that the images we want to classify are the handwritten digits of the MNIST dataset. It is a set of 60,000 images, all made of 28×28 pixels. We will arrange them in a line, made of 784 real numbers that will be called neurons (or perceptrons), for analogy with the human brain, and will be drawn as circles. A neuron is simply a memory place where a real number is written: in concrete implementation, the neuron contains a float with the desired precision. The whole machine learning system, or model, will be made of this neurons: that is why the model is called a neural network. The number stored inside the neuron is called the **activation**. The first layer of neurons is made of 784 neurons, and they contain the input: the 784 intensities of the pixels of the input image, which for simplicity we imagine being black-and-white (for RGB images we would need 3 neurons for each pixel).

Then we add two other intermediate layers, each made of 16 neurons, and a final layer with 10 neurons. Each neuron will contain a float that can be calculated from the neurons of the previous layer, as we will shortly see: that is the network aspect of the neural network. The last layer is the final layer, on which we read the answer: the neurons correspond to the ten possible digits from 0 to 9, and the neuron that is most activated, *i.e.*, which will contain the greatest float, will correspond to the predicted digit. The machine learning system, the neural network, will thus take as input the image, made by the intensity of all its pixels, and return its prediction for the digit represented in the image.

We did not explain how to calculate the float of each neuron: it is done by a linear combination of the neurons of the previous layer, followed by a nonlinear operation so that the new neuron is activated only when the linear combination goes over a certain cutoff, similarly to biological neurons. If we call $x_i^{(L)}$ the i -th neuron of layer L , then we have

$$x_i^{(L+1)} = \sigma\left(A_{ij}^{(L)} x_j^{(L)} + b_i^{(L+1)}\right).$$

Here, matrix $A_{ij}^{(L)}$ connects the i -th neuron of layer $L + 1$ with the j -th neuron of layer L , and it can be thought as the strength of a synaptic connection. It can be in fact drawn as a line that connects the two neurons, with its thickness proportional to its numerical value, and the color dependent on its sign (it can be positive or negative).

The function σ is a nonlinear function, called the *nonlinear activation function*. In its absence, the whole neural network would be linear, which is not what one seeks (a superposition of a 6 and a 9 should probably give an 8 as a prediction, thus the model cannot be linear). For simplicity, a single function σ is used throughout the model. A function like the hyperbolic tangent can be used, or even the Rectified Linear Unit, or ReLU. The use of a certain function instead of others is sometimes a matter of the easiness to calculate its derivatives for backpropagation (which we will explain in a moment), but sometimes it seems almost a matter of fashion. In the present time (2025) the most advanced AI model usually use the SwiGLU as the nonlinear function. As we see, it is like the ReLU but smoother. Usually, all these activation functions change their behaviour around 0, and behave very differently for negative and for positive numbers.

The values $b_i^{(L)}$ can be thought then as the activation energy of the neuron: they shift the value of the input $A_{ij}x_j$ for which the activation function changes its behaviour, from 0 to $-b$. It is truly the analogous of the activation energy of biological neurons.

We have visualized the calculations neuron per neuron. If we treat each layer of neurons as a vector, and also the activation layers, and if we treat the synapses as a matrix, and if we recall that the nonlinear activation function must be applied component-wise *i.e.*, separately to each component of the vector, then we can write

$$\mathbf{x}^{(L+1)} = \sigma\left(A^{(L)}\mathbf{x}^{(L)} + b_i^{(L+1)}\right).$$

We introduce for completeness the **preactivation**

$$\mathbf{h}^{(L+1)} = A^{(L)}\mathbf{x}^{(L)} + b_i^{(L+1)}$$

which is simply the value that is acted upon with the nonlinear activation function to get the value of the neurons in layer $L + 1$. Taking everything together, we have for the whole neural network

$$\mathbf{y}_{pred} = \sigma\left(A^{(3)}\sigma\left(A^{(2)}\sigma\left(A^{(1)}\mathbf{x} + b_i^{(2)}\right) + b_i^{(3)}\right) + b_i^{(4)}\right). \quad (3.1)$$

Our neural network is simply this function, that takes as input the 784-dimensional vector $\mathbf{x}$ with the intensities of the pixels of the image to be classified, and which gives as output the 10-dimensional vector $\mathbf{y}_{pred}$, of which the highest component indicates the predicted digit (if the k -th component is the highest, the prediction is the digit k).

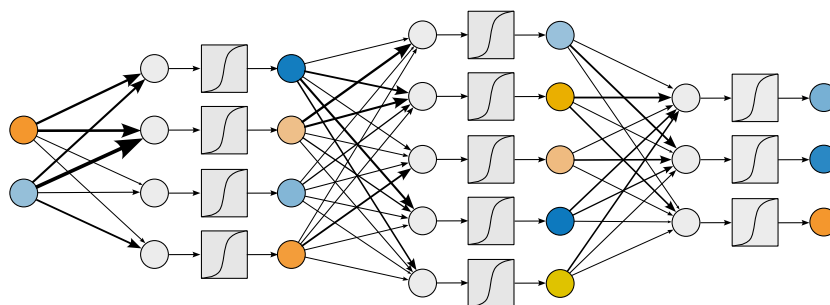


Figure 3.2: A Feed Forward Neural Network, or FFNN, with two hidden layers. The two input neurons have positive (warm colors) or negative (cold colors) values. The values of the neurons are linearly combined with coefficients that can be drawn as synapses to give a value which, after subtracting the activation energy, is called the pre-activation (the grey neuron). After passing through a nonlinear function, we obtain the activations of the neurons of the following layer. These calculations continue from layer to layer up to the (in our case three) output neurons. The image is loosely based on a figure of Deep neural networks from Lamarr Institute which can be found at <https://lamarr-institute.org/blog/deep-neural-networks/>.

Now, this function has the right dimensions for input and output, but is it the right function? It depends on the value of the **parameters**, also called **weights**, which are usually indicated by θ_i , or θ in vector notation:

$$\theta = \{A^{(1)}, A^{(2)}, A^{(3)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \mathbf{b}^{(3)}\}.$$

These values are at first initialized randomly (maybe they are extracted from a Gaussian distribution with mean and variance so that the result y_{pred} has norm one in average). The idea is that, by seeing many examples of images and seeing the right label for them, the parameters are slowly adjusted to have better predictions. This procedure is called gradient descent and, maybe surprisingly, it allows one to obtain a function with a percentage of success exceeding 99%, and which can be brought to even higher values with other tricks. Let us explain the idea of gradient descent and of backpropagation.

The idea is that for each image, our randomly-initialized neural network gives a prediction $\mathbf{y}_{pred}$, which is almost surely not equal to the desired result $\mathbf{y}_{label}$, a *one-hot encoding vector*, *i.e.*, a vector with all zeroes except a one in the right position, the k -th component, where k is the label of the image, *i.e.*, the human-compiled digit that corresponds to the digit handwritten in the image. The idea is to use a **cost function**, also called a *loss function*, that quantifies the error:

$$C = C(\mathbf{y}_{pred} - \mathbf{y}_{label}).$$

For example, we can choose a quadratic cost function

$$C = |\mathbf{y}_{pred} - \mathbf{y}_{label}|^2$$

which has zero as its minimum value.

Now, the idea is that by moving each parameter θ_i , we change the cost function. We thus would like to change each of them in such a way to reduce the cost function, and we change it in a way proportional to the slope (as in a parabola the slope is directly proportional to the distance to the minimum, it seems to make sense) and proportional to a coefficient λ called the learning rate. The update rule for all parameters is then

$$\theta_i \Rightarrow \theta_i - \lambda \frac{\partial C}{\partial \theta_i}.$$

These calculations are then done by repeated use of the chain rule. For example, the derivative with respect to an element of $A^{(1)}$, using (3.1), is

$$\frac{\partial C}{\partial A_{ij}^{(1)}} = \sigma'(h^{(4)}) A^{(3)} \sigma'(h^{(3)}) A^{(2)} \sigma'(h^{(2)}),$$

where σ' is the derivative of the nonlinear activation function σ , and where the values $h^{(j)}$ of the preactivations are those of the feed-forward calculation which led to the calculation of C : they can be saved in the memory so that they do not need to be recalculated for each derivative. A framework to avoid calculating anything twice is implicit in the widely used Pytorch library, which at the time of writing is the standard in both academy and industry for machine learning models. Machine Learning is in fact just an example of the new paradigm of **differentiable programming**, in which a computer program that depends on parameters can be differentiated via automatic differentiation.

We gave the main idea of the **training phase** of the model: the model, randomly initialized, is given as input, one by one, some of the images of the dataset, let's say 50,000 from the original 60,000: they are the training set. For each of them, we calculate the cost function C and we update the values of all parameters θ_i , using a small enough learning rate, e.g. $\lambda = 0.01$. We define one **epoch** as the process of giving each image of the training dataset to the model exactly once. We can train for more than one epoch.

After some epochs, let us say ten, the cost function is usually gone down so much that the number of accurate predictions is over 96%. The results can be improved and we briefly summarize some tricks. For example, one can update the parameters θ_i only after having seen a full batch of different images, let us say a thousand each time:

$$\theta_i \Rightarrow \theta_i - \lambda \sum_{\ell=1}^B \frac{\partial C_{\ell}}{\partial \theta_i}.$$

where ℓ is the index of the image of the batch, and $B = 1,000$ is the batch size.

At each epoch it may be a good idea to reshuffle the order in which the images are presented to the neural network, to avoid oscillatory behaviours in the training.

Another technique called dropout purposefully excludes a certain percentage of neurons (let's say 20%), chosen randomly, from the calculations of the neurons of the next layer: in this way, the neural network learns in a robust way, and its pattern recognition is not dependent on any single neuron, but it is distributed in the whole network, like in humans (if a neuron cell die, we usually do not forget forever a specific detail of our memory). This makes the resulting neural network more independent of contingent aspects of the original images, and thus more accurate.

The accuracy of the network can be measured in different ways. Certainly, during a training epoch, one classifies all images, but this is done with a changing network. We have to fix the parameters of the network and do an epoch in which we calculate $\mathbf{y}_{pred}$ and use it to extract its maximum component, associated with the prediction, which in an abuse of notation can be called y_{pred} , which can be confronted with the expected label y_{label} . The percentage of correct answers is called **training accuracy**.

3.2 The unreasonable effectiveness of deep learning

It may appear not very surprising that the model can recognize digits when it was repeatedly told their labels. That is the reason to define test accuracy. The dataset images not included in the training set, in our case 10,000 of them, constitute the test set. We use the neural network with fixed parameters and do an epoch with the test images, to measure the accuracy for images it has not seen before. This is the most surprising aspects of machine learning: it is able to recognize even unseen features. Is this *generalization* capacity of machine learning algorithms that is the key of their success: a large language model such as ChatGPT may be trained on trillions of words, but it can generalize and answer to words it has never seen because it has seen so much text and understands so much the correlations among its different parts that it gets to the gist of meaning buried beneath the symbols, and can answer in a reasonable way to questions it has not explicitly seen before.

In the same way, if we trained our image classifier to recognize cats by training it on images with a cat and without it, it would be able to recognize cats that it has not seen before. Clearly there are some limits: if he sees an image of a drawing of a cat, it may or may not recognize it; if it is shown a lion, it may or may not recognize it. It can only generalize some concept of which it has seen a sufficiently representative number of instances: in this, it is not dissimilar to a baby human being, which may fail to generalize the concept of cat to a lion if he never saw a lion before.

Generalization clearly can happen because there is something that all pictures of cats have in common, as each handwritten version of a digit has something in common which the human mind can recognize, even though it is too difficult to formalize it in a computer program. As an interesting experiment, one can randomize the labels of the handwritten digits and try to train the neural network nevertheless. The model learns to assign the correct random labels to train images, but it fails to do so to test images, as expected, as they were in no way correlated with what the experiment has done!

In a way, machine learning sheds light on one of the most profound philosophical problems of philosophy, the problem of Universals: how it is possible that we somehow know that all cats have something in common (which we will call the Universal of a cat, or cat-ness), even though all example of cats are different?

This problem was tackled by Plato, Aristotle, and then by medieval scholars such as Thomas Aquinas, and finally by Kant. The great question is whether these Universals, these common properties which are too difficult to express in words or in computer programs, but which we seem to know, exist in the thing themselves or if they are just a fiction invented by our human minds. Plato would have said that the Idea (Universal) of cat-ness existed before the Universe, and was used by the Demiurge to build cats. Aristotle would have said that a cat, as all things, has an essence of cat-ness in it, which can be known by our senses. Kant would instead have said that we can never know what exists, but that our senses and our mind arrange our experience by using categories like space and time and movement, and it is by means of these categories that we see that there is something in common to all cats, cat-ness, which is a construct of our mind. Later darwinist philosophers would say that the concept of animal, from which that of cat-ness, evolved in our mind through evolution: it is helpful to understand that there is a lion, so that we can run, even though each lion is different. We recognize what all lions have in common, as it is helpful in survival. This however do not explain how our mind is able to see something in common in different lions.

Somehow the best answer is provided by machine learning, and it somehow agrees with Aristotle and Thomas Aquinas, even though it is now on a solid mathematical and empirical foundation: they exist before our human act of comprehension. The idea is that cats exist in the world and they are certainly different, but clearly they have something in common. This property is not a linear property like being of colour red, which is easy to define by referring to the light wavelength: it is a nonlinear property, which can be discovered by taking a picture and applying a nonlinear function, the neural network function f , which tells us whether there is a cat or not. This function f is today present in the human mind, obtained through our long evolution history of billions of years, as the animals who failed to develop useful features of the outside environment had a disadvantage in survival. In our machine learning model, the feedback is not given by death (even though there are genetic models which learn like darwinian evolution: they have slightly-different offsprings and they reproduce less if they are not able to accomplish the desired tasks): the feedback is given by backpropagation, and the labels are already given. This is the difference between **supervised learning**, in which we give to the machine learning model the correct labels for the task it wants to perform, and **unsupervised learning**, where we give a bunch of data and the model has to find regularities in the data by itself, like seeing many

images and understanding that they come in ten different kinds, which then a human recognizes to be the ten possible digits of our base 10 digit system. The conclusion is that these Universals exist, they are expressed by nonlinear functions too complex to be explained by words, but they are codified somewhere in our minds, and in the machine learning models which we can now train in our computers (as shown in Figure 3.3). These general properties like cat-ness are represented by complex functions dependent on thousands or maybe millions of parameters, but nevertheless they already exist before the experiment.

Another interesting and confusing aspect is that the parameters that allows one to recognize cats are not unique. The optimized parameters may live in a space of millions of parameters, but it is a space with many local minima, all of which have an accuracy greater of 99%, and in practice they can easily be reached from a not-too-unreasonable random starting point and by setting a good learning rate. This is the so-called **unreasonable effectiveness of machine learning**, of which a nice treatment is in [71].

For one of the best discussions over the deep reasons why deep learning works so well, one can read the lucid paper by Max Tegmark *et al.* [72], which suggests that all data we see in Nature was generated in a somehow hierarchical structure at each level of which nonlinear operations were performed, and thus deep learning does nothing but undo the process of generations of data. It also asks questions about whether nonlinearity are so necessary to make deep learning work (not always, in fact) and how many neurons are needed if we use neural network with a single hidden layer. An example of how subsequent nonlinear operations generate the observation data we find in Nature is given in Figure 3.4.

We however only gave empirical demonstration and some speculations about the potential of machine learning: we could ask whether there are some more strict results. Let us go back to the dimensions of our FFNN, *i.e.*, (784, 16, 16, 10), which were imposed by already knowing they work. The input and output dimensions were fixed by the nature of the problem: e.g. to classify images with cats we would have needed just two neurons, or maybe just one by watching the sign of its activation. The two intermediate layers are called **hidden layers**, and if they are at least two we are doing **deep learning**. Large language models can use hundreds of hidden layers.

We could ask now ourselves why is it different to put at least two hidden layers, and whether there are theorems which guarantee us about the existence of a solution. Clearly there is no theorem that guarantees us that the property of

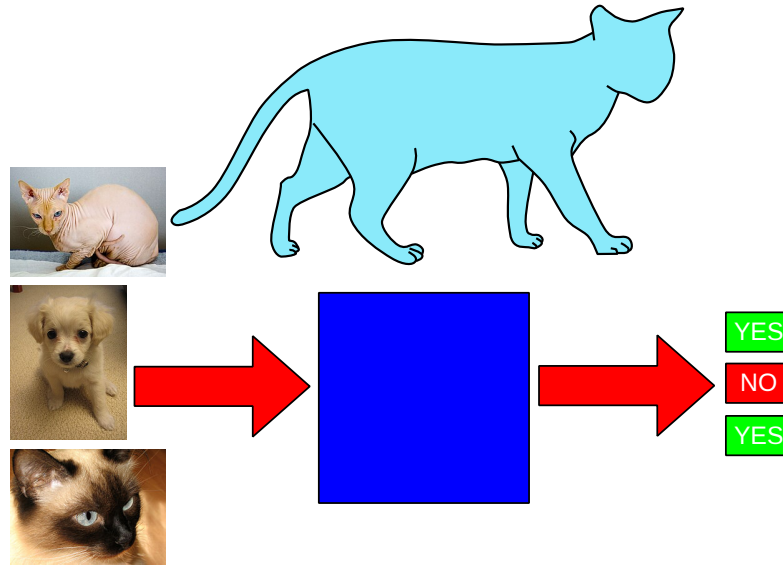


Figure 3.3: A machine learning model is a black box (the blue square) that learns the difficult-to-define nonlinear property that all cats have in common, which philosophers called the *Universal*, or *Platonic form*, or *Essence* of a cat, or also *cat-ness* (the blue ideal Cat). Even though this property is difficult to define with language, it is a mathematical function that can be expressed by neural networks and which can be found with supervised learning with images of cats labelled by humans, or also by unsupervised learning (in a much more long and complex way). The image of the ideal cat is loosely based on image https://commons.wikimedia.org/wiki/File:Schrodingers_cat.svg by Smite-Meister, released under the Creative Commons Attribution-Share Alike 3.0 Unported licence; the image of real cats and real non-cats are from Wikipedia.

cat-ness can be achieved with no more than such-and-such neurons per layer, as cat-ness cannot be defined independently of these already-trained machine learning models. The main theorems are the following.

Universal Approximation Theorem. A neural network with a single hidden layer of big enough dimension can approximate arbitrarily well any function with the same number of input or output.

The proof is simple: depending on the function σ , one can build some bump-functions which have area one and are non-zero only in the regions between

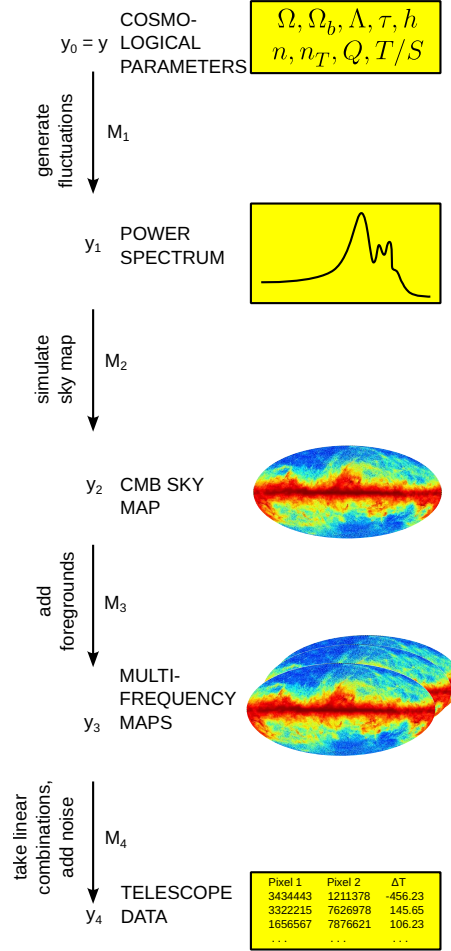


Figure 3.4: In Nature we find data which is the result of many usually-nonlinear operations performed in series: Deep Learning, with its nonlinear layers arranged in series, is thus a natural way to disentangle the complexities generated by Nature. The image is loosely inspired by a part of figure 3 of [72]

a and b , which can be changed. This can be used as an approximation of a delta function to build up any desired function with a small error.

The problem with this theorem is that is useless in practice, as the single hidden layer that can approximate a given function has an exponentially larger number of neurons than those required by a deep learning model, *i.e.*, a

model with at least two hidden layers. It is this empirical, qualitative property that distinguish deep learning models from single-hidden-layer models.

Besides, the theorem does not say anything about the difficulty of training such a single-hidden-layer model, which is usually greater than for a deep learning one.

No Free Lunch Theorem. It says that, among different machine learning architectures (*i.e.*, different high-parameter functions such as the one in (3.1) and different algorithms to train them), none of them is better than another in all possible tasks.

The concept of **architecture** is to be understood: it is the form of the function whose parameters, whose weights, we are optimizing; but it includes the strategy chosen to optimize those weights (usually it is gradient descent with backpropagation). Until now we saw just a single example of an architecture, the simplest one: the Feed Forward Neural Network or FFNN. There are now about fifteen different main architectures in machine learning, plus a plethora of small variants. This theorem says that, in theory, by doing an average on all possible tasks, no architecture is *absolutely* better of another. It may be better of another in a certain task, but it cannot be better of another *in all possible tasks*.

However, this theorem too is in fact useless. The reason is that we are not interested in all possible tasks, but only in tasks which are somehow useful to humans, and this is already a limitation. For image classification, we do not want to operate with all possible images, but only of images of things which can be useful to humans: images of the real world, like images of nature, of cities, of human or animal tissues, and the like. These are not random combination of pixels: they display certain regularities, certain continuities, and usually can be conceptualized easily, because they have certain features which are implicit. Soon we will talk about how, by expliciting what we know about the task, we usually are able to conceive a better ML architecture for that task. This is not a theorem, but a very useful insight, very reminiscent of Noether's Theorem in physics. In physics, by knowing a symmetry of a dynamic problem we can reduce the dimensions of the problem: in the same way in machine learning, by knowing some symmetry or any feature of the data on which we are to perform a certain task, we can embed this symmetry in an architecture which takes automatically that symmetry into account, and thus can achieve better results with less calculations. This insight can informally be dubbed the **Noether's "theorem" for machine learning**,

and is shown in Figure 3.5.

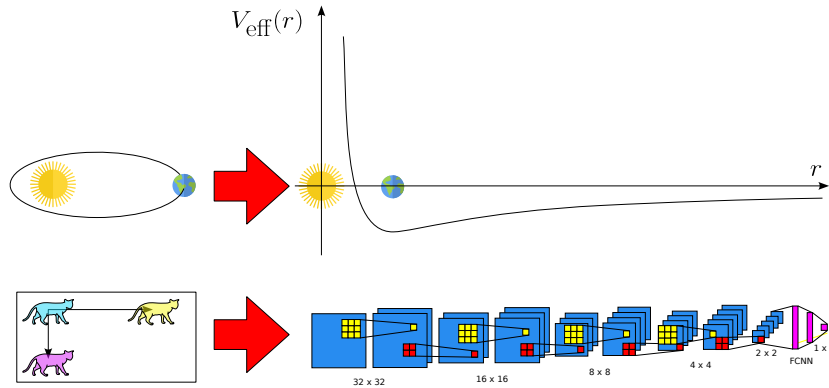


Figure 3.5: Noether’s theorem for physics and its informal application to machine learning. Physicists apply Noether’s theorem when they use their knowledge of the symmetries of a physical problem, e.g. the rotational symmetry of the gravitational force of the Sun upon the Earth, to simplify the problem from 2D to 1D, using the effective potential $V_{\text{eff}}(r)$. In the same way, designers of new machine learning architectures can use their knowledge of the symmetries of a machine learning task, e.g. translational symmetries that act on a cat in an image, to simplify the neural network to solve the task: the Convolutional Neural Network can use less weights than typical Feed-Forward Neural Networks to solve computer vision tasks (even though the typical CNN found in the literature uses more weights than the FFNN). The image of the Sun and of the Earth are taken from <https://www.svgrepo.com>. The images of the cats are loosely based on https://commons.wikimedia.org/wiki/File:Schrodingers_cat.svg by Smite-Meister, released under the Creative Commons Attribution-Share Alike 3.0 Unported licence.

We repeat this fact: we are not interested in all possible tasks, but in specific tasks; and even the set of all tasks in which we are interested is the kind of tasks in which humans in this world may be interested, and not a most general task. Nothing thus precludes a single architecture from being able to outperform all others in most if not all possible tasks. We will briefly explain that the Transformer architecture, which is behind almost all large language model such as ChatGPT, is in fact state-of-the-art (SOTA) in natural language processing (NLP) *i.e.*, in manipulating language and answer to questions in natural language; in computer vision and image classification; and in many others. The same thing is true for our human brain, which uses

the same architecture of biological neurons, the neocortex microcolumns, to treat diverse tasks such as language processing and vision.

We are now ready to present briefly how to build new neural network architectures by applying the insight of paying attention to the symmetries of the task: this will be crucial in our work presented in chapter 5.

3.3 Convolutional Neural Networks

A slightly more complex architecture is the **Convolutional Neural Network**, or *CNN*, from which we got the main inspiration for our quantum model in chapter 5.

It is clear that the FFNN architecture we presented is so general that its input data does not need to be made by images: pixel intensities are codified in neurons, and then all neurons can be permuted with all other neurons, ignoring notions of proximity and distance between pixels. Instead, CNNs are specifically thought for computer vision, *i.e.*, for the task of analyzing image. The main insight is to think about the structure of the task, and the symmetries of images. For example, let us imagine that we want to know if there is a cat or not somewhere in a picture. The fact that a cat is present or not in a certain corner of the image should be done by a certain function f , realized by a trained neural network. The idea is that there is a spatial translational symmetry: the same function f that looks for a cat, or a certain feature, in a part of the image, should be the same that looks for it in another part of the image!

This is the central insight of CNNs. Even though the translational symmetry is in fact quantized, as the image is divided in pixels, it can be implemented in this way: we apply a convolution (see Figure 3.6), for example with a 3×3 matrix, called the **kernel** or the **filter**, *i.e.*, we look for a certain 3×3 -pixel pattern among the image, and we activate the neurons of the following layer depending on the presence or not of that pattern. The exact calculation of the convolution is to multiply each element of the 3×3 kernel matrix with the corresponding 3×3 sub-matrix of the original image, and summing the results we can fill an image with more or less the same dimensions of the original one. The exact dimensions depends on parameters like stride and padding: the stride is the steps we move the kernel from two times we apply it (one could also have a horizontal stride different from the vertical one, if one wished so), and the padding is the number of lines and columns of extra

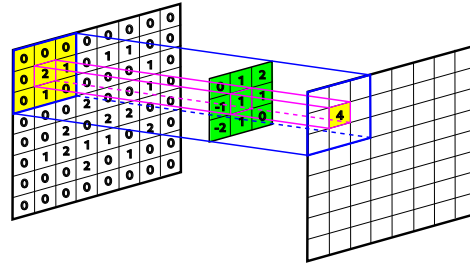


Figure 3.6: A convolutional layer with a 3×3 kernel (shown in green). The calculation that gives the result is to multiply each term in the 3×3 kernel (shown in green) with the corresponding term in a 3×3 subset of the original image (the subset is shown in yellow to the left), and then to sum the results and write it in the pixel corresponding to the central pixel of the original subset (the output pixel is shown in yellow to the right).

zeroes we add around the original image to allow the kernel to be applied at least partially to the outside of the original image. If one uses a 3×3 kernel, a stride of 1 and a padding of 1, one gets a final image with the same dimensions of the original one: for simplicity we will describe this specific version of the CNN.

What we described is a simplified version of a **convolutional layer**. The most general version of the convolutional layer starts from an image with c **channels**, which at the beginning can be thought of the three colors of an RGB image ($c = 3$): the input image is thus of dimension $n \times n \times c$. Then, the kernel can have dimension $3 \times 3 \times c \times c'$, where the index of dimension c is contracted with the channel index of the original image. In the end, we get an image of dimension $n \times n \times c'$.

Now, the idea is the following: to understand whether there is a cat in the figure, we may start to look for the presence of cat ears, of cat eyes, of fur and other smaller features, and after having found them through some kernels, we can see if they are all present (if not covered by another object) and if they are arranged among themselves in the right configuration: this is done by a kernel at a higher level. This leads us to think about applying more than one convolutional layer. But more: the idea is that while former kernels should look for smaller and local features, later kernels should look for bigger and global features that span the whole image. A way to realize this is to alternate convolutional layers with some layers that reduce by half the resolution of the image, so that later 3×3 kernels can span greater distance in the original

image, finding bigger features.

These new layers are called **pooling layers**. Pooling can be done in different ways, but the simplest one is to simply divide an image in blocks of 2×2 pixels, and to average the pixels within each block. There are some variants in which one takes the maximum, or some other weighted combinations, but they are more complex.

By doing some alterations of convolutional layers and pooling layers, one can stop when reaching an image of dimension $(\tilde{n}, \tilde{n}, \tilde{c})$. If one wants to minimize the number of parameters (as we will be trying to do in the classical analogue of our quantum model in chapter 5), we could proceed until we have dimensions $(1, 1, \tilde{c})$, but usually one stops earlier and adds a FFNN in the end. The reason is that when the convolutions already act on information that comes from the whole image, all global features should already have been found, and all values of the image of dimension $(\tilde{n}, \tilde{n}, \tilde{c})$ are pieces of global information: they are equally important and there is no sense of proximity nor distance between them, as they could be permuted in any way. Thus the right way to act on them is through a FFNN, which does not presume any sense of proximity in its input data.

After a hidden layer (usually of dimension $\approx 4\tilde{n}^2\tilde{c}$), the output number of neurons depends on the classification: it could be one or two for the presence or absence of cats, ten for handwritten digits, thousands for more complex classification tasks.

An example of a CNN optimized by us to have a minimal number of parameters is shown in Figure 3.7. This model is described at length in appendix section A.2, and was used as a classical comparison to our quantum neural network, treated in chapter chapter 5.

The CNN is the best application of the insight of looking for symmetries in the task: as we are looking for a small-scale feature in the image, we use exactly the same weights to look for it in all parts of the image, through a convolution, instead of training different weights to look for the same things. This gives the CNN the potential to use less than a thousand weights (as we did in chapter 5) where the FFNN would use about ten thousands of them. In practice, CNNs use a lot of images.

The CNN is the main inspiration for our quantum neural network which we present in chapter 5, but in general we were inspired by the insight of understanding the symmetries of the task to build a suited architecture for that task.

We briefly give some other examples of this line of reasoning, by giving a

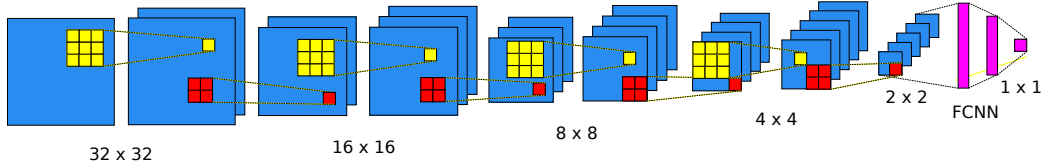


Figure 3.7: General structure of our classical Convolutional Neural Network. The image undergoes a series of convolutional layers with a 3×3 kernel matrix (yellow), and pooling layers (red). When a small resolution of 2×2 is achieved, the pixels are flattened and given to a fully connected neural network (purple). To compare the classical CNN with our quantum architecture, the number of parameters of the fully connected NN is kept small.

review of the most important neural network architecture used in AI up to this date.

3.4 Architectures for Classical Neural Networks

While the FFNN is the simplest neural network which in principle can be applied to any task, and the CNN uses spatial translational symmetry for the task of computer vision (image classification and the like), we have that **Recurrent Neural Networks** or *RNN* use temporal translational symmetry to deal with Natural Language Processing, or NLP (answering users question in natural language, or reading reports, or translations, and the like).

Let us treat, for example, the problem of language translation: we have some English sentences and we want to translate them into Italian. The idea is to have somehow a black box model that takes input words one by one in time, and outputs words one by one. As in the English language (and in the Italian language and in most languages) there are about 50,000 words or word roots which we will call **tokens** (like play and -ing are considered two different word roots; there is a whole subject called tokenization that studies which is the best way to divide words into tokens), these tokens can be encoded as one-hot vectors. The black box clearly has an internal memory, a hidden state of dimension h . The hidden state changes at each time step, and its dimension must be big enough to allow the model to keep the context in mind when doing the translation.

This model is called a Recurrent NN, because the weights do not change at

each time step, but they are kept fixed. This is a form of time invariance. The reason for remarking it is that, expanding what the network does, is receiving at different steps the various words, at periodic times, and outputting the words of the translation: in this context, the fact that the weights repeat periodically is nontrivial, and is in fact the key idea of the RNN. This is showed in Figure 3.8.

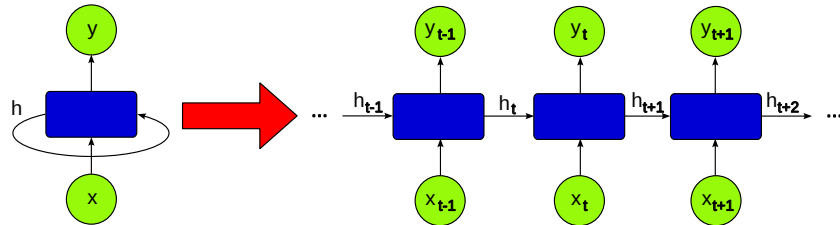


Figure 3.8: A Recurrent Neural Network, or RNN: it can be seen both as a neural network of L layers in which at each time step a new input goes in, a new output goes out, and the hidden states change; and as a very long neural network which is periodic and repeats itself after L layers, where the inputs go in and the outputs go out at a distance of L layers, and the hidden state is simply the neuron state of the network after $L, 2L, 3L, \dots$ layers.

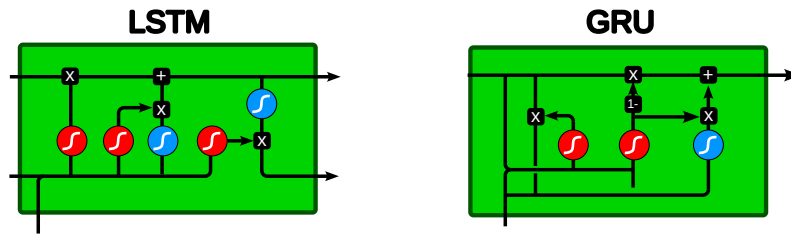


Figure 3.9: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are examples of improvements on the RNN basic structure. They became obsolete after the invention of the Attention mechanism and of the Transformer [73]. The image is loosely inspired by a figure of [74]

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), shown in Figure 3.9, are just variants of RNN with a specific form of the black box model, and are both used for NLP tasks. The research in this topic

went on for quite a while, but a problem was found, which has a surprising analogy to statistical mechanics. In fact, the structure of RNN is similar to a 1D chain of systems, with h as the dimension of the interaction between nearby system. It is to be expected that, under reasonable assumption, the correlation between two output words of a RNN decays exponentially on their distance. It has a typical length λ .

The question is: does this reflect what we do know about natural language? Do the correlation between words in a book, or in all words uttered by a human being throughout his or her life, have a typical scale?

The fact that a book can reference at the end details only spoken of at the beginning, and the fact that I can recall and say words that I have spoken in the first five years of my life, show that natural language seems not to have a typical scale: it seems not to decay at all! The solution is then that, to accurately understand how to place a certain word or sentence in its context, or to translate it, or to answer a question after a long report, it is better to have in mind the whole context, the whole text, the whole report, and to see if any part of it is related to the immediate task of processing a single word. In simple words, the idea is to put each word in relation to all previous words, thus doing $O(n^2)$ calculations where n is the number of input words. This leads us to the concept of *Attention* [73] and to the most widely used neural network architecture that impacted so much the whole world in the last eight years: the **Transformer**.

A Transformer is simply a neural network architecture that implements in the simplest possible way the fact that we have to do $O(n^2)$ calculations. Let us imagine that each of the n input tokens is encoded in a vector x_i^α , where the latin letter indexes such as $i = 1, \dots, n$ go through the number of tokens of the input text, while the greek letter indexes $\alpha = 1, \dots, N$ go through the word of the vocabulary of tokens, which to give the order of magnitude can be taken as $N = 50,000$.

Now, to perform $O(n^2)$ calculations, the easiest way is to make each token interact with all other tokens. This interaction can be seen in physical terms as an interaction among 50,000 different possible particles, and as such it defines something analogous to a Field Theory where Feynman diagrams describe interactions between particles. The difference is that QFTs are linear, while in machine learning we allow ourselves to perform elementwise a nonlinear operation σ .

We will thus try to define this nonlinear interaction theory in the easiest possible way. The easiest interacting scalar theory is a ϕ^3 theory, thus meaning

that the interaction would take two tokens and give a single token. One of the many problems with this scheme (which reminds us of pooling layers) is that the number of tokens is not necessarily conserved, and we would like it to be conserved in an interaction layer so that we can stack different layers on top of one another. The solution is to use four-vertex interactions, so that we have a theory that conserves the number of particles (a non-relativistic theory, if one wills). The most general vertex is thus expressed by a vertex $A^{\alpha\beta\gamma\delta}$ that tells how the token vectors x^γ and x^δ (associated to the γ -th and δ -th tokens of the vocabulary respectively), once transformed, will have a component parallel to token vectors x^α and x^β . The complexity of the matrix seems to be N^4 , but in fact we can think of a better theory.

A simple change is to introduce a one-way interaction, so that we have $A^{\alpha\beta\gamma}$ with vector x^β becoming x^α while interacting with vector x_γ which in turn does not change. By doing the interaction with all pairs of tokens in both possible orientations, we have again all interactions. This brings down the number of degrees of freedom to N^3 .

We can also make so that x^β changes in a way linearly proportional to x^γ , let us say Bx^γ . This brings down again the number of degrees of freedom to $O(N^2)$ and, after changing the names of the indexes, gives the equation

$$x'_i{}^\alpha = A^{\alpha\beta} Bx^\beta$$

where x' is the state of the token vector after the interaction.

If we interpret the action of the interaction on a token vector as a small correction to its value, and including all interactions of a token x^α (as in a single layer every token should interact with each other token) we obtain the formula

$$x'_i{}^\alpha = \sum_j A^{\alpha\beta} Bx_j^\beta.$$

Now, the problem is how to calculate $A^{\alpha\beta}_{ij}$. In some way, $A^{\alpha\beta}_{ij}$ is a measure of how much the meaning of token x_i^α is related to the meaning of token x_j^β . We thus expect to obtain it as some sort of scalar product, thus containing a term of the form $x \cdot x$, which is the bare minimum to rightly contract the indices and reproduce a scalar product. Also, to avoid that the vector norms increase without bounds in these interactions, we would like that

$$\sum_j A^{\alpha\beta}_{ij} = 1$$

so that we can interpret them as percentages of relation of meaning. To satisfy these requirements, we would want the matrix A to be some sort of scalar product between the token vectors, and to satisfy the normalization condition we apply the operation called softmax:

$$A_{ij}^{\alpha\beta} := \text{softmax}_j \left(x_i^\alpha M^{\alpha\beta} x_j^\beta \right),$$

where the softmax is simply the exponential renormalized so that the sum adds up to one:

$$\text{softmax}_j \left(c_j \right) := \frac{e^{c_j}}{\sum_j e^{c_j}}.$$

Thus the complete formula becomes

$$x_i'^\alpha = \sum_j \text{softmax}_j \left(x_i^\alpha M^{\alpha\beta} x_j^\beta \right) B x_j^\beta$$

where, by decomposing $M^{\alpha\gamma} = (Q^T)^{\alpha\beta} K^{\beta\gamma}$ and taking $B = V$, and defining the query, key and value vectors as

$$q_i^\alpha := Q^{\alpha\beta} x_i^\beta, \quad k_i^\alpha := K^{\alpha\beta} x_i^\beta, \quad v_i^\alpha := V^{\alpha\beta} x_i^\beta,$$

we can write

$$x_i'^\alpha = \sum_j \text{softmax}_j \left(q_i^\alpha k_j^\alpha \right) v_j^\beta$$

which is how the calculation is usually presented in the Artificial Intelligence literature. This calculation takes the name of **Attention mechanism**, as every token pays attention to each other token.

One should notice that our treatment involved the core idea of the Transformer, which is however a bit more complicated than that. As we previously said, a Tokenizator must be chosen to parse the input text into tokens. They have to be encoded in some vector space as vectors x_i^α . We also ignored the fact that somehow each of these vectors should depend on the position of the word, otherwise the Transformer knows nothing of the order of the words it gets as an input! There are many ways to do this positional encoding, like adding a vector whose components are the values of a sinusoid whose frequency depends on the position of the word. Also, in each layer there is an Attention mechanism with query, key and value, but also a FFNN. There are also residual connections where the input is added to the output to help gradient flow better when doing the chain rule in gradient descent

calculations, which helps training (as the output can be seen as a perturbation of the input, like in weak-interaction physical theories). At the end of the Transformer layers, a FFNN allows the system to choose which neuron of 50,000 (associated to the tokens of the vocabulary) is the most active: that indicates the next token of the answer. This is appended in the index and the system does again its nonlinear calculation to predict another token: this is how present Transformer works, as auto-regressive models *i.e.*, models that predict a token at a time.

The Attention mechanism was discovered by researchers at Google in 2017 while trying to solve the issues with the decay of the inner memory of Recurrent Neural Networks. They developed a variant that implemented some form of the Attention mechanism, with its $O(n^2)$ calculations, and they discovered that it worked so well that they decided to scrap the rest of the RNN and built a new architecture from scratch that used only the Attention mechanism. That is the reason behind the title of their revolutionary article: *Attention is all you need* [73].

This is not only insightful about the way of thinking about architectures, *i.e.*, be ready to scrap most things when it is clear they are no longer relevant, but it is also an occasion to review very briefly the whole story of the field of Artificial Intelligence. The reason is that the whole story is full of fundamental lessons that are necessary to understand how the future story of quantum machine learning, *i.e.*, machine learning done by quantum circuits with a large number of trainable parameters instead that by classical ones, will be. The idea of some sort of artificial neural network is already found in 1943 in a work of McCulloch and Pitts [75], but the perceptron (*i.e.*, the artificial neuron that is activated with a nonlinear function applied to a linear combination of inputs) was first conceived by Frank Rosenblatt at Cornell Aeronautical Laboratory in 1957 [76], and was soon implemented into an IBM704 machine. However, the technology at the time was not able to implement the idea at the scale at which the idea would have worked.

After about sixty years, Moore's Law allowed the computational power accessible to mankind to increase exponentially, as the number of transistors per integrated circuit doubled about every two years. There have been periods of hype like the 1980s where it seemed that expert systems could be the right way to artificially reproduce intelligence for practical purposes, but that lead to nowhere, thus giving rise to the AI winter of low investments in research: this happened in 1974 - 1980 and also in 1987 - 2000. In that period, some algorithms surprisingly beated humans in some apparently difficult tasks like

a game of chess (Deep Blue vs Kasparov, 1997). At that time, there was still no algorithm that could recognize whether there was a cat in an image or not! The complexity of a task highly depends on the intrinsic nature of the subject that has to do the task, be it a human or a classical computer. We finally get to 2012, where for the first time a neural network solved a non-trivial task in computer vision, the ImageNet competition, which started in 2010. The ImageNet project contains more than 20,000 categories of objects, and a total of 14 million images. The existence of these datasets, pioneered by AI researcher Fei-Fei Li since 2006, was one of the three necessary ingredients for modern deep learning. The other aspects are computational power, which was allowed by the steady progress of Moore's Law, and the third was the slow but decisive progress in architecture design: in this case, from FFNNs to CNNs.

This three necessary elements were already present in 2012, and allowed Convolutional Neural Network to basically solve the computer vision problem, which was unsolved since the summer an undergraduate student failed to solve it in the Sixties, as already told. Since then, neural networks gained prominence among other machine learning techniques.

At the time, the great problem was the absence of a good architecture for natural language processing. The Transformer was developed in 2017, and provided just what was needed. At the time, about 70% of AI research was done by Google, but shortly afterwards OpenAI was founded to combat this monopoly, but the private sector with its vast resources could deploy bigger and more powerful models. In the meantime, both the academy and the private sectors looked for new and better architectures for different tasks, and some examples in the private sector are Ian Goodfellow (Google, OpenAI, Apple) and Geoffrey Hinton (Google), the so-called Godfather of AI recently awarded the Nobel prize in physics in 2024 for its work on neural networks. In 2017, the Transformer completely changed the picture, as there were many competing ideas of architectures each devoted and optimized to a specific different tasks. The Transformer instead is universal: is the easiest possible architecture which does $O(n^2)$ calculations on its input. The fact that the Attention mechanism should be of the form

$$x'_i \approx (x_i \cdot x_j)x_j$$

where we purposefully ignored all linear operators to leave only the fundamental structure, is something that can be recovered from scratch by looking at the possibilities of index contraction. It is so universal that, even though it

was conceived for natural language processing, it was understood that it was able to deal with computer vision tasks (the **Vision Transformer**, or *ViT*, which has performances comparable to CNNs), to protein folding prediction (in 2020 AlphaFold, a 100-layer transformer, basically solved the problem at the lowest order, as we already mentioned) to many more.

A spontaneous objection is that by moving from the $O(n)$ calculations of an RNN to the $O(n^2)$ calculations of a Transformer, the new model may be better but would be impossible to train. This is wrong due to the fact that the RNN, which we explained, does the processing of each token *in series*, while in the Transformer each Attention layer allows for the interactions between tokens to be computed *in parallel*.

Crucial was the fact that in the 1990s the videogame industry (and in particular the company NVIDIA) has brought into existence Graphic Processing Units (GPUs) which are optimized to do linear algebra computations in parallel (which were needed to calculate the screen image of the game in parallel). These GPUs have now found large scale applications in machine learning in general, but especially in implementations of the Transformer, allowing it to be competitive. The evaluation of the best model sometimes does not only depend on mathematics, but also in the contingencies of the technological progress of our society.

Since 2017, there have been about two phases. In the first, up to 2022-2023 with the release and large-scale diffusion of the use of ChatGPT by OpenAI and of other **Large Language Models** or *LLMs* for short (*i.e.*, transformers of billions of parameters and tens if not hundreds of layers that can answer questions by users in natural language) the Transformer worked so well that all attempts to work on a better and more sophisticated architectures were frustrated by Richard Sutton's so-called Bitter Lesson [77], *i.e.*, the (maybe for someone) unpleasant fact that the ideas that win in AI are those that are simple and can scale better, rather than those that are too sophisticated and are better at a specific scale, but must be rethought from the start every time the scale increases.

The Bitter Lesson was valid for the first five years of the Transformer, as its size increased by a factor of 10 every year: to get a better model, money and resources were better spent on increasing scale than in researching more complex architectures. In the meantime, the size of the models increased by 10x each year, much more than Moore's law, thus they reached saturation around 2023 where 100 million dollars were needed to train a one-trillion parameter Large Language Model such as GPT-4.

Since then, AI research made by the Big Tech companies of the US and China and by smaller AI startups, and also by the Open Source communities, worked on finding ways to make the transformer more efficient, bringing about different small improvements which are sometimes covered by the term Transformer++. As the architecture was conceived with some twenty fast and almost arbitrary convention decisions in 2017, many times $10\times$ factors of improvement have been found.

This brings an end to the review of AI history and of its insights. The hope of building AGI to benefit Mankind and revolutionize Education and Healthcare, to substitute most human work, and to make intelligence a low-price commodity to be used by humans, is still ongoing. It is not clear if the Transformer will be the architecture that brings us there. The idea of a vertical asymptote in the increase of intelligence with time, also called the Technological Singularity, is not very clear, even though it seems very probable that, if one concedes that intelligence can be reproduced by neural networks (that one can easily concede), then in the remote future it seems inevitable that resources of astronomical if not galactical scale may be used to power such a system, which will be incomparably more intelligent than Mankind. This raises the question, which is in fact by far more urgent, of **Alignment** *i.e.*, of making sure that the behavior of these AIs that we are developing, and of the intelligences of the future, will be aligned with human ethics, human values and human interests: as these latter concept are not well defined, and as a failure could entail unforeseen catastrophic consequences, this necessary task faces Mankind as a whole and needs the most careful consideration. Even though the transition to a world of AI superintelligences is well governed, the questions remain about the role of human beings in society. As computers became better than humans in chess, humans tried to define themselves by those abilities which computers could not reproduce, like art or understanding, contemplation and ethics. However, today a model is able to generate works of art and be creative in many different ways, and write essays of philosophy, and the like: if their abilities continue to improve, a good question is what the human being will think about themselves and their role in the world. But even though these speculations help us remember the supreme importance of this field, they are not the subject of our investigation.

Other architectures that we will not treat are: Graph Neural Network (GNN) in which neurons in each layer are arranged as a graph (associated to a molecule or a road network that should be studied) and a neuron is con-

nected to the neurons associated to the nearby places in the graph in the preceding layer; Reinforcement Learning (RL) that trains a neural network to play a video-game without supervision, by reinforcing the parameters that contributed more to decisions that gave points in the game; the Autoencoder (AE) which is a neural network which is trained to reproduce the original image, but with the complication that layer size decreases and then increases to the original size, so that the network is forced to learn a smarter and smaller representation for the original data (an image, a text, a video or whatever data desired); the Variational AutoEncoder (VAE) which is a variant of the AE in which one finds a smaller representation that allows to easily generate new instantiation of the kind of data (for example, after having learned to compress images of cats, the VAE can produce images of new cats); the Generative Adversarial Network (GAN) that is made of two NNs, the Generator which tries to generate new instances of a kind of data (for example images of cats) and the Discriminator which tries to find which images of cats are generated and which are original, so that training the two NNs together one obtains a good Generator; the Diffusion model which uses statistical mechanic ideas to generate new images after dividing the process of blurring an image in little steps and learning how to reverse each of those steps; the Echo State Network which is a random but fixed arrangement of neurons with forward and backward connections plus a trainable FFNN in the end, which has found applications in time series analysis (for finance, for example [78], in which RNNs are also used); State-space models like Mamba and others which try to improve RNNs to reach the performances of Transformers; and some other variants of the above. Some less practical but more artistic variants are Neural Style Transfer, that allows one to reproduce an image with the artistic style of another, showing that AI has somehow understood the qualitative common element of all impressionist paintings or of all Picasso's cubist paintings and can reproduce it at will; and Deep Dream which generates the image that maximizes the detection of some class of objects, e.g. cats, through gradient ascent, and thus generates onirical landscapes where e.g. cats can be found anywhere at all scales.

Probably one of the interesting variants is the U-Net, which is a variant of the CNN for computer vision. This gives us an excuse to delve into a deeper discussion in how to use symmetry to build an architecture to solve a specific task.

The idea is that in the task of vision (the task we deal with in our work presented in chapter 5) strictly speaking the spatial translational symmetries

are not the only ones when one tries to find a cat in an image. The other symmetries are the rotational symmetries. If the cat is small enough, also a dilation symmetry will be needed.

Rotational symmetries are clearly broken by the divisions of the image in pixels, just as the translation symmetries: but even though translation symmetries are preserved in their discretized versions, (almost) no rotational symmetry remains (except the $\pi/2$ one).

However, the fact that a cat remains the same even though it is rotated is important, and should be used to improve the performance of our model. As the symmetry cannot be incorporated in the neural network architecture, it should be incorporated in the training data. The idea is to add to the training data some rotated version of the same images by a range of different angles (usually between $-\pi/2$ and $\pi/2$) to help the model understand that they all count as instances of the same object, for example a cat. The images clearly were approximate rotations, and some algorithm has to be devised to rotate an image already divided into an horizontal and vertical grid of pixels.

To implement the dilation symmetry, which is useful in the task of finding objects in an image, and in **image segmentation** (*i.e.*, the task of dividing the image into different objects such as people, cars, roads, traffic lights for computer vision applied to automatic guided vehicles), a simple framework is the following: a Convolutional Neural Network which uses alternatively convolutional layers and pooling layers which we recall change the scale at which we are looking at the image, and at each scale it looks for certain patterns. In contrast to Convolutional Neural Networks, it uses the same kernel to recognize the same feature at all scales.

This main idea can be developed into the **U-net**, which is a CNN used for image segmentation. In a U-net, the scale at which we look at the image is increased by each pooling layer, followed by a reverse process in which the scale is increased again. This figure can be drawn to have the shape of a letter U: the forward process is the descending part of the letter, while the reverse process is the ascending part. In addition, from each scale there comes out some data that skip the higher-resolution part of the U-net and go to the part with equal resolution in the second part of the network.

The main idea of the U-net is to find features at all scales of a CNN, and then use them to do an inverse CNN which from the highest scale and lowest resolution incorporates the information of found features at each scale up until the lowest scale and higher resolution.

3.5 Machine learning and physics

Artificial Intelligence is a fascinating subject, which is already having and will have a deep impact on human society in the few decades to come. The AI revolution will change the human condition, for the better or for the worse. It is also a very interesting topic in itself, as one is trying to understand the implicit mathematical structure behind tasks associated with common human activities such as vision or natural language, and thus it is a way to improve our understanding of ourselves. The connections with Biology have been profound since the beginning: the name neural network itself suggests the imitation of biological neurons, even though the artificial ones are simpler. In the search of new architectures, inspiration from biology was always strong: Convolutional Neural Networks were influenced by biological ideas on how the visual cortex analyzed the data from the human eye (for a comparison between CNNs and Human Eye vision see [79]) and also Recurrent Neural Network can be improved through inspiration by biology ([80]). Also, the human neocortex, *i.e.* the outer layer of our cerebral cortex involved in higher-order brain functions such as sensory perception, spatial reasoning, cognition, vision and language, is composed of about 10 billions of the human brain's 80 billion neurons, all arranged in many copies of a small structure called the **microcolumn**: a six-layer architecture of a few hundred biological neurons, arranged as an FFNN which goes towards the center of the brain. The existence of a single biological architecture in our brains for a great variety of tasks, such as vision and language, mirrors the universality of the Transformer as an artificial neural network architecture. The fruitful interactions between machine learning and biology have given contributions to both fields; but the same can be said about the connections between machine learning and physics.

The connections are various. We have already seen that deep learning is essentially linear algebra with a single nonlinear element. An interesting aspect is that machine learning somehow works with just linear operations, if one discards some data. The intuitive idea is that linear operations on a high-enough dimensional space can simulate nonlinear operations. This does not always work, but it will be useful for Quantum Machine Learning as Quantum Mechanics is in fact linear.

Another connection is with statistical mechanics. When we make a huge number of copies of a classical system, like some billions of copies, statistics begins to apply and the behavior of some macroscopic variables is sometimes

simple enough that it can be treated by simple equations, and the response of the system to variations of some macroscopic parameter is sometimes abrupt and discontinuous at some derivative: this is the case of phase transitions. As neural network architectures are made of a large number of copies of an artificial neuron, these reasoning applies, and AI is full of phase transitions. An interesting result is the following: in the large width limit, *i.e.*, in the limit of a layer with a number of neurons $N \rightarrow +\infty$, we have that the neural network becomes indistinguishable from a simpler machine learning system called a **Gaussian process** (GP). In particular, the random initialization of the network induces a probability distribution for the activation of the output neurons; in the large width limit, that distribution of the output converge to a Gaussian [81] (see Figure 3.10).

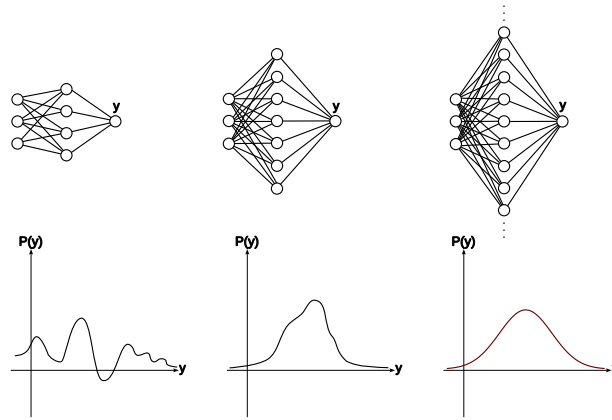


Figure 3.10: By increasing the number of neurons N in the hidden layer of a randomly-initialized neural network, the probability distribution of the output neurons (induced by the random initialization of the parameters) converges to a gaussian distribution. We show this in the simple case of a single output neuron.

This may appear natural as the weights of the NN are chosen from the same probability distribution and thus for the Central Limit Theorem, the final distribution should be a Gaussian. It is maybe less trivial to see that also the training process of the NN has a correspondence in the training process of the GP. Even less clear is to see that, by considering the analogy between the Gaussian process and a propagator of a free particle in QFT, one can try to reduce the width of the layer to a finite N and do an expansion in $\frac{1}{N}$ (as

is done sometimes in QCD in expanding over the number of quark colors) obtaining a Feynman diagram expansion for the behavior of the network [82]. This gives the first insights about some sort of **NN/QFT Correspondence** [83], [84], [85], which as the AdS/CFT duality would be a weak-strong duality, between large width neural networks and calculations that corresponds to interaction between particles: when the neural network is bigger and requires more calculation (strong), the dual particles interact less (weak).

We have to remember that Feynman Diagrams can be seen as simply a way to solve differential equations with repeated approximations: they can be made to appear also in finance and other fields unrelated to physics. Here however there seems to be a true duality where the calculation becomes simpler (a Gaussian Process) where it should become more complex, and we can expand around that GP.

Another interesting theoretical development around the Large Width limit of neural networks is the **Tensor Program** framework: a set of tools that allow to extend that results to many widely-used architectures such as CNNs and Transformers, and that allows one to easily recover standard results in Random Matrix Theory and also unifies ideas in Mean Field Theory which is usually linear to Neural Networks where there is a controlled elementwise nonlinearity. The Tensor Program papers were written by Greg Yang since 2019 [86–94] and his theoretical analysis allowed some practical results such as μ -transfer [93]): a set of results that allows one to do training experiments on small network and that guarantee that, by training a bigger network with its width, length and dimensions scaled in a certain way, then trainability will not get worse. This is a result widely used in the AI private sector today. With this, we finish our journey. Artificial Intelligence is a theory ultimately founded on linear algebra, just as Quantum Mechanics; it has many complex relations to physics which we are just beginning to unravel; it allows us to go into the depths of the nature or Intelligence and the process of understanding of data, of finding meaningful and useful representation of any kind of data, and thus the field is simply one of the most profound undertaking Mankind has ever faced.

We will now face an interesting crossover: as machine learning is ultimately rooted in the foundations of classical computation, and as the recent decades have seen us explore the possibilities springing from a shift from classical computation to quantum computation, we will ask ourselves the question of whether quantum computing can help in improving the execution of machine learning algorithms. This question is not new, it has been asked for more

than a decade, and there have been different approaches to it. We will review the state of the art in answering that question, and then we will be able to present our contribution to it.

Chapter 4

Quantum Neural Networks

We are now ready to treat the recent field that lies at the intersection of the two extraordinary topics of the last two chapters, Quantum computing and classical machine learning, which is **quantum machine learning**: the field that tries to use the greater power of quantum computing to improve machine learning algorithms [95], [96].

Before delving into the subjects and its definition, we may start by taking notice that there are many different ways in which quantum physics (and more specifically quantum algorithms) and machine learning interact and combine themselves, and not all of them are quantum machine learning. An example would be the use of machine learning algorithms to better understand quantum physics and quantum technologies [97], like quantum sensing and others, or even to have a way to express many-body quantum states. This gives rise to a series of fruitful ideas of which we briefly mention two. The first are **Tensor Networks** or *TNs*, which are graphs where each node represents a tensor and an edge represents the summation over an index, and unattached edges (the legs) represent free indices. They are a useful ansatz for the wave function that solves a many-body systems, as the form of the graph is arranged to reflect the physical structure of the system: to study a $1D$ many-body quantum system with long scale entanglement one can use a tensor network called Matrix Product State (MPS), while in the $2D$ case one can use a Projected Entangled-Pair State (PEPS). In each case the connections between edges may correspond to more than one index, being proportional to the expected interaction between nearby systems. They do not directly use machine learning, are just inspired by it, but they can be applied to machine learning in many ways, for example as in [98].

The second and more direct application are **Neural Network Quantum States** or **Neural Quantum States** (*NNQs* or *NQs*) in which the many-body quantum state, which lives in an exponentially big Hilbert Space, has as components the output of an artificial neural network which is trained through gradient descent to reproduce the many-body system of interest: a review can be found in [99]. It can be understood as an extension of the variational method in elementary quantum mechanics, where to find the ground state wave function we use an ansatz, *i.e.*, a parametric form of the solution, and try to find the best form of the parameter: this is the same, but with more parameters, and the ansatz is a specific neural network architecture, dependent on the problem at hand. In some cases, even the transformer has been proposed as an architecture for NQs [100, 101].

Among many interesting crossovers maybe too long to list, we briefly cite the idea to use machine learning to help a quantum computer simulate a quantum theory (the SYK model) that is dual to a theory of quantum gravity using AdS/CFT, and in particular to perform a calculation on the quantum hardware that is dual to a particle passing through a wormhole, from a side to the other! The clearly fascinating idea was implemented in [102] by making use of machine learning to simplify the quantum theory: the SYK hamiltonian for $N = 10$ fermions, the bare minimum to have a traversable wormhole behaviour, has 210 terms; machine learning helped in finding a sparsification of the SYK system (*i.e.*, setting to zero many fermionic interactions) that preserves gravitational-like phenomena in the dual, bringing down the theory to $N = 7$ fermions and just 5 terms (!). The wormhole teleportation protocol, which required simulating negative energy density in the gravitational side, required 9 qubits and 164 two-qubit gates.

The idea was later criticized in [103] because the small-size hamiltonian does not exhibit thermalization as claimed, because some operators were used in the training so their behaviour was not independent, and mainly because the property of exhibiting a gravitational-like behaviour in the supposed dual theory, is something that is common to all such small-size fully-commuting models, and is not preserved in scaling up the dimension. The debate is ongoing, but the attempt was fascinating by itself. We are however interested in the direct application of quantum computing to machine learning, and this will be the subject of our review of the field.

The chapter is organized in the following way. First we show how quantum machine learning is divided into two main approaches: the one that uses Variational Quantum Algorithms (VQAs), also called Quantum Neural Net-

works (QNNs); and the one that tries to find a quantum advantage (usually with the HHL algorithm) in linear algebra operation to be used for machine learning algorithms. The discussion on HHL is postponed to chapter 7. So we explain in detail the idea of Quantum Neural Networks. We also make a comparison with Classical Machine Learning, showing that even though there may be some advantages in shifting to the quantum counterpart, there are also many difficulties one has to struggle with. We briefly summarize some recent discoveries in quantum machine learning to give a gist of the state of the art. Then we explain why we chose the task of computer vision for our proposed new quantum architecture instead of other possible machine learning tasks. We very briefly touch the interesting topic of the Quantum Transformer, which cannot be treated at length for matters of time. Finally, we review the different heuristics that drove us in conceiving our quantum architecture, which will be the topic of chapter 5.

4.1 Two approaches for quantum machine learning

As we have said, the goal of quantum machine learning is to improve on machine learning algorithms by shifting from classical to quantum computation. There are two main approaches.

The first one relies on using complexity theory trying to prove rigorous quantum advantages in certain settings, which typically require a rather unconventional access to the data. An idea is to leverage the HHL algorithm to solve linear differential equations: as linear systems are ubiquitous in machine learning algorithms especially for non-neural-network ones, any ways to solve them in less time can improve the efficacy of all machine learning methods that requires them. This application for HHL, and the deep difficulties in applying it which are lucidly discussed in [31], will be better discussed in chapter 7. However, the HHL is not the only algorithm that can be used. Many quantum algorithms have been proposed to surpass a *classical machine learning* one (*i.e.*, a machine learning algorithm for a classical computer). A sketch table can be found in [104, Table 1]: it includes many non-neural-network machine learning algorithms, but also some neural-network ones. Most of the presented classical algorithms require $O(n)$ calculations, in particular almost all the neural network ones (even if it is not indicated, we recall that the Trans-

former requires $O(n^2)$ calculations); the quantum algorithms proposed give sometimes a polynomial advantage as $O(\sqrt{n})$ and sometimes an exponential one up to $O(\log n)$. The problem of most of these is maybe the same as for the HHL algorithm: it is not clear how to extract the useful information once it is calculated. For example, in the HHL algorithm a linear system is solved in $O(\log n)$ time, but it is still necessary a time of order $O(n)$ to read the solution. One should find some circumstance in which the information necessary is only a part of the solution, and that the piece of information required is easily accessible in time $O(\log n)$. This is highly nontrivial and for the HHL algorithm it is discussed in [31] and in chapter 7, as already said. For other linear algebra quantum speedup algorithms there are similar problems.

Let us now shift to the second approach which will be the main topic of this section.

4.2 The theory of Quantum Neural Networks

The main approach is to use a **Variational Quantum Algorithm** or *VQA* (sometimes also called **Quantum Neural Network** or *QNN*) which is a quantum circuit whose gates depend on trainable parameters also called weights, and whose output is obtained through some measurements along some basis. The idea is the same as in classical machine learning: we have a black box, to which we feed the data as input, and it gives an output which depends on both the input and on the weights. At the start, the black box is initialized with random weights, and thus it gives random outputs. We then slowly change the weights through a training procedure, until the trained black box is able to perform the desired task. As we will see, the training procedure is different than in classical machine learning, but the result is similar.

Let us formalize the above general definition. The quantum state produced by the circuit is

$$|\Psi(\Theta, x)\rangle = U(\Theta, x) |\Psi_0\rangle$$

where Θ contains all weights, $U(\Theta)$ is a unitary matrix, and x is the input data, and the starting state can be chosen to be e.g. the tensor product of n copies of the $|0\rangle$ state, but not necessarily. Now, in our research we treat VQAs where the input is encoded in the input quantum state of the circuit (there are also cases in which it can be encoded both as the input quantum

state and in some of the following parametric quantum gates). Thus we can write

$$|\Psi(\Theta, x)\rangle = U(\Theta) |\Psi(x)\rangle = U(\Theta)V(x) |\Psi_0\rangle$$

where U and V are both unitary matrices, while $|\Psi(x)\rangle$ is the state. Here the encoding happens at the beginning, and then only gates are used.

By measuring the quantum circuit we get the black box function we are training, from input to output:

$$f(\Theta, x) := \langle \Psi(\Theta, x) | \hat{O} | \Psi(\Theta, x) \rangle.$$

To evaluate this function, many measurements are necessary. Here lies one of the first difficulties of quantum machine learning: while to double the precision of the outputs of a neural network it is sufficient to do the same calculations with floats with a double precision than before, in quantum machine learning the error of the measurements decreases as $\propto \frac{1}{\sqrt{N}}$ where N is the number of measurements. Thus the calculations needed to double the precision is exponentially greater than in classical machine learning. This problem is related to the barren plateau problem which will be explained later. For now, we will say that we will not need an arbitrary great precision for the quantum measurements in our model, thus avoiding the problem. Now, one can define a cost function, as in classical machine learning, which will be in general some function of

$$C = C(\mathbf{y}_{pred} - \mathbf{y}_{label})$$

as for example the quadratic cost function

$$C = |\mathbf{y}_{pred} - \mathbf{y}_{label}|^2$$

or some of its variants, as we will see in chapter 5. The idea now is the same as in classical machine learning: we would like to use gradient descent to change each weight proportionally to the derivative of the cost function with respect to the weight itself:

$$\theta_i \Rightarrow \theta_i - \lambda \frac{\partial C}{\partial \theta_i}. \quad (4.1)$$

Now, a problem arises. How is it possible to calculate this derivative? It depends on

$$\frac{\partial f(\Theta, x)}{\partial \theta_i}$$

where f is the function that gives the output of the quantum measurements on a quantum function. How is it possible to calculate it? One idea is to use a finite-difference method, but it would be only an approximation, and the optimization landscape of the cost function may give rise to errors that would accumulate and render training impossible. This would make quantum machine learning completely unpractical at once. The solution is given by the surprising existence of the so-called **parameter-shift rules**, found by Mitarai *et al.* [105] and then expanded by Schuld *et al.* in [106]. They are defined as *formulae* of the form

$$\frac{\partial}{\partial \theta_i} f(\Theta, x) = \sum_j a_j f\left(\Theta + \Delta(j), x\right),$$

which allow to calculate a derivative *exactly* as a linear combination of evaluations of the function on different values of the parameters. The intuitive reason for the existence of these rules is that the weights parametrize unitary gates, usually Pauli rotations, which are akin to usual rotations and sinusoids and trigonometric functions. For them, we have

$$\frac{\partial}{\partial x} \sin(x) = \frac{1}{2} \sin\left(x + \frac{\pi}{2}\right) - \frac{1}{2} \sin\left(x - \frac{\pi}{2}\right),$$

which comes from the trigonometric identity for angle addition. This is a parameter-shift rule. Thus, if it works for trigonometric functions, it is not so strange that it works also for general rotation gates.

The complete proof is straightforward (it is reproduced from [107]). If we are evaluating the derivative with respect to a parameter θ_i which appears in U_i (we assume the simple case in which each parameter appears exactly once, which is enough for our work in the next section), we find that, if U_i is indeed a unitary rotation of the form

$$U_i(\theta_i) = e^{-i\frac{\theta_i}{2}\sigma_i}$$

where $\sigma_i = \sigma_i^\dagger$ is a Pauli operator, then we have

$$\nabla_{\theta_i} U_i(\theta_i) = -\frac{i}{2} \sigma_i U_i(\theta_i) = -\frac{i}{2} U_i(\theta_i) \sigma_i.$$

Before evaluating the whole gradient $\nabla_{\theta_i} f(\Theta, x)$ we have to introduce a new notation. Instead of

$$f(\Theta, x) := \langle \Psi(\Theta, x) | \hat{O} | \Psi(\Theta, x) \rangle$$

where

$$\Psi(\Theta, x) = U_N(\theta_N) \dots U_1(\theta_1) U_0(x) |\Psi_0\rangle,$$

(with N the number of unitary gates) we define

$$\Psi_{i-1}(\Theta, x) = U_{i-1}(\theta_{i-1}) \dots U_1(\theta_1) U_0(x) |\Psi_0\rangle,$$

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

so that we have

$$f(\Theta, x) = \langle \Psi_{i-1} | U_i^\dagger(\theta_i) \hat{O}_{i+1} U_i(\theta_i) | \Psi_{i-1} \rangle$$

where the dependence on θ_i is explicit. Thus, the gradient is

$$\nabla_{\theta_i} f(\Theta, x) = \frac{i}{2} \langle \Psi_{i-1} | U_i^\dagger(\theta_i) \left(\sigma_i \hat{O}_{i+1} - \hat{O}_{i+1} \sigma_i \right) U_i(\theta_i) | \Psi_{i-1} \rangle.$$

Now one makes use of the mathematical identity

$$\sigma_i \hat{O} - \hat{O} \sigma_i = -i \left(U_i^\dagger \left(\frac{\pi}{2} \right) \hat{O} U_i \left(\frac{\pi}{2} \right) - U_i^\dagger \left(-\frac{\pi}{2} \right) \hat{O} U_i \left(-\frac{\pi}{2} \right) \right)$$

which is valid for all observables $\hat{O}$, and which was proven by Mitarai *et al.* in [105]. Applying it for $\hat{O}_{i+1}$, one gets

$$\begin{aligned} \nabla_{\theta_i} f(\Theta, x) &= \\ &= \frac{1}{2} \langle \Psi_{i-1} | \left(U_i^\dagger \left(\theta_i + \frac{\pi}{2} \right) \hat{O}_{i+1} U_i \left(\theta_i + \frac{\pi}{2} \right) - U_i^\dagger \left(\theta_i - \frac{\pi}{2} \right) \hat{O}_{i+1} U_i \left(\theta_i - \frac{\pi}{2} \right) \right) | \Psi_{i-1} \rangle \end{aligned}$$

which is equivalent to

$$\nabla_{\theta_i} f(\Theta, x) = \frac{1}{2} \left(f \left(\Theta + \frac{\pi}{2} \delta_i, x \right) - f \left(\Theta - \frac{\pi}{2} \delta_i, x \right) \right), \quad (4.2)$$

where δ_i is a set of parameters that is equal to 1 when the parameter index is i , and is 0 otherwise. We have now proved our first parameter-shift rule, which is sufficient for our work in the next section.

It is also possible to prove more general parameter shift rules, e.g.:

$$\nabla_{\theta_i} f(\Theta, x) = \frac{1}{2 \sin(\alpha)} \left(f \left(\Theta + \alpha \delta_i, x \right) - f \left(\Theta - \alpha \delta_i, x \right) \right), \quad (4.3)$$

where α is any angle (this is [108, equation 9]). It is clear that for $\alpha = \pi/2$ equation (4.3) implies equation (4.2).

The parameter shift rule (4.2) allows to exactly (!) calculate gradients of our quantum circuits in a relatively efficient way. For each parameter, just two evaluations of the quantum circuit with different values of the parameters are needed. In practice, we change the quantum gate $U_i(\theta_i)$ to $U_i\left(\theta_i + \frac{\pi}{2}\right)$ and $U_i\left(\theta_i - \frac{\pi}{2}\right)$, and we run the quantum circuit in each of the two cases with the two altered U_i gates: we have to run them enough times to get quantum measurements precise enough (the precision ϵ may be arbitrarily chosen beforehand). The main thing is that the number of quantum runs to perform a single gradient step (assuming a batch size of one) is $O(N)$, where N is the number of weights. The calculations of the gradient are clearly performed by a classical computer.

4.3 Classical vs. Quantum machine learning

This is maybe a key issue of this procedure: some may criticize it by saying that it is still not a truly *quantum* machine learning, as the gradient descent is not done in a quantum way, but still in a classical way, by computing the derivative to calculate the next steps according to (4.1). Ways to have a truly *quantum* training are for example thermal relaxation or thermal annealing procedures, that uses the quantum nature of the circuit to explore faster the landscape of states. We will discuss these objections and these ideas later in this subsection, and briefly in the conclusions (chapter 6). but for now this is to be understood as a pure quantum circuit with no classical parts: classical calculations are needed just to update the weights during training.

We have seen the key definitions of VQAs, the quantum analogue to classical neural networks, and how to work with them. We will now do a brief comparison between classical neural networks and quantum neural networks. The ideas that led to the development of the idea of quantum machine learning is the hope of better performances in some tasks. We recall that the word quantum in quantum machine learning stands for the type of algorithm, not for the type of input data. It can be used to analyze classical or quantum data indifferently; in the next chapter chapter 4 we will choose to use our quantum algorithm to analyze classical data. With this in mind, we can say that the idea to obtain some quantum advantage was paramount at the

start: we have already saw the slow but decisive discovery of some quantum algorithms that allow to solve problems faster than the best known classical algorithms. For quantum neural networks, the intuitive guess is that by optimizing the weights in a quantum landscape, there are somehow more trajectories that connect the states than in a classical landscape: quantum gates can generate entanglements while classical gates can only perform a subset of the quantum-allowed operations.

Another advantage, which is not always used but it is heavily used in the quantum neural network we present in the next section, is that while in classical machine learning one has the limitation that a minimum number of bits is required to encode the input data, e.g. the intensities of all the pixels of an image up to a certain precision, in quantum machine learning (and in quantum information in general) one can encode all human knowledge in the exact phase of a single qubit, with the catch that one can extract from it a single bit and that it is difficult to manipulate. Without resorting to such extreme possibilities, in our work we make use of **Amplitude Encoding**, *i.e.*, we encode the input as the amplitudes of the input quantum state, where we recall that the amplitudes are exponential in the number of qubits. This will be better explained in the next section.

There are however some deep downsides, which are not immediately fatal to quantum machine learning, but which must be understood.

The first is that the **no-cloning theorem** asserts that one cannot build a gate that for all possible quantum states $|\Psi\rangle$ can take $|\Psi\rangle \otimes |0\rangle$ as input and give $|\Psi\rangle \otimes |\Psi\rangle$ as output. One can however do this for classical bits, and obviously with quantum states in the computational basis (*i.e.*, with $|0\rangle$ and $|1\rangle$ and their tensor products) which are just classical bits in disguise.

This means that one cannot fully copy a quantum state with the easiness with which a classical neural network copies the value of a neuron and uses it to calculate the activation of all neurons of the next layer, which may be thousands. The only possibility is to make copies at the start: for example, the input state could be made of different copies of a single state:

$$|\Psi_0\rangle = |\chi_0\rangle^{\otimes n},$$

and then use the n different copies of $|\chi_0\rangle$, thus effectively acting as if one had just copied them, even though they are present from the beginning. But all quantum states that arise subsequently cannot be copied.

Another partial problem is that quantum mechanics is linear, while machine learning sometimes needs a nonlinearity. This is easily solvable in different

ways: one can perform the same calculations on the $|\chi_0\rangle$ defined above to obtain $|\chi\rangle$, and then do a linear operation on two copies:

$$|\tilde{\Psi}\rangle = U |\chi\chi\rangle.$$

The result will be quadratic in $|\chi\rangle$, which is a nonlinear operation: if for example we parametrize the state with

$$|\chi\rangle = \begin{pmatrix} 1 \\ x \end{pmatrix}$$

(where we ignore the normalization factor), then by applying the infinitesimal SWAP gate

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 - \epsilon^2 & \epsilon & 0 \\ 0 & \epsilon & 1 - \epsilon^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(where it is to be understood that the basis is $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ respectively), we have at first order

$$|\tilde{\Psi}\rangle = \begin{pmatrix} 1 \\ x + x\epsilon \\ x + x\epsilon \\ x^2 \end{pmatrix}$$

from which we get that, tracing out the second qubit, we have that the first qubit is in the state

$$|\phi\rangle = \begin{pmatrix} 1 \\ \tilde{x} \end{pmatrix}$$

with

$$\tilde{x} = x \left(1 + \frac{1 - x^2}{1 + x^2} \epsilon + O(\epsilon^2) \right)$$

which is indeed a nonlinear operation.

In the end, the difficulty of obtaining nonlinear operations can be circumvented by taking copies at the beginning, but this requires to double the number of qubits for each layer of nonlinear operations that acts on the whole circuit, and it is somehow intuitively understood that it is not the best use of

precious resources like qubits.

The last problem of quantum machine learning is the necessity of exponential resources to increase output precision of a quantum measurement: we will see the relations with barren plateaus in a moment.

In the end, it is not clear if the upsides surpass the numerous downsides: indeed, no example of quantum advantage of a quantum neural network has been found, and is also still difficult to find quantum neural networks that can successfully perform a nontrivial task such as those of classical neural networks!

There are many reasons for this. One of them is that classical neural networks usually work with at least thousands of parameters and with datasets of at least thousands of examples. Current quantum circuits have less than a thousand qubits, but in practice we can use tens of qubits and hundreds of gates (each of which correspond to about one parameter), and errors in those gates are still non-negligible. Our solution will consist in trying to drastically reduce the number of quantum resources needed, both in qubits and in gates terms. Usually quantum neural networks are classically simulated, due to the absence of large-scale quantum computers: as a classical simulation is inefficient, this severely limits the scale that can be reached with such an approach; it also somehow makes it difficult but not impossible to have a quantum advantage (we are using a classical computer to simulate a quantum algorithm to beat a classical computer: but the comparison of performances must be done taking into account that the quantum algorithm would take less time than its simulation).

Another important reason is that *classical machine learning is already unreasonably effective* [71] and, as we have seen, most neural networks have a complexity of $O(n)$ which is difficult to beat (as $O(\log n)$ algorithms have the problems of the HHL algorithm, better explained in chapter 7) except for transformers, which have complexity $O(n^2)$. The possibility of realizing a quantum transformer are indeed extraordinary and very interesting, but they will be treated separately (see section 4.5).

In a few words, classical machine learning is already effective by itself. Usually one does the quantum version of X to improve X, but if X is not broken at least a little (and machine learning certainly is not!), it is not clear why a quantum version should be sought in the first place. The main problem is that the room for improvement seems so small (except for transformers). In the view of these ideas, it is not surprising that some papers question that quantum advantage is even a goal of the study of quantum neural network

[108], which can be understood as a temporary strategy (as asking different questions in a field can lead to more progress in the original quest for quantum advantage; for example the question: which is the right quantum analogue of the perceptron, *i.e.*, the right quantum perceptron?) or as a definitive pivot of scope, as quantum neural networks could be just another mathematical structure with few or none practical applications, as they will never beat neural networks. It is still too early to make a definitive statement, but certainly there is no clear reason for which quantum machine learning will beat classical machine learning in any practical task.

This perspective is probably realistic but a bit too harsh: in practice, in the last decades researchers found some quantum algorithms of quantum machine learning for some specific tasks, and there are still many ideas still to be tried. We give here a very brief summary of some of them. Even though our presentation will give only some examples and be incomplete, it will give at least the gist of both the way research proceeds in quantum machine learning, and an approximate idea of the state of the art of the field.

We have already talked about quantum vision, where many algorithms make heavy use of classical pre-processing or post-processing, and those which do not, like the Google Quantum MNIST architecture [2], have an accuracy which do not reach the level of the simplest and most natural Feed Forward Neural Networks (96 – 98%); even though the Google Quantum MNIST has the same accuracy of FFNNs with a comparable number of parameters. However, an accuracy of 90% in recognizing digits is still not quite impressive for practical applications, and it does not count as having *solved* the task at a level comparable to human capabilities.

As we have said, besides quantum neural networks there is also the search for quantum algorithms for specific tasks, usually using the HHL algorithm to solve linear systems. It is not known which application may work of the many proposed: we already gave a table of proposals for quantum speedups of known machine learning algorithms [104], both with and without neural networks. However, there are also attempts to use quantum speedups for specific problems in specific fields, instead of general many-purpose machine learning algorithms: for example, quantum finance [78], [109], where even a small advantage in time efficiency is heavily rewarded. No practical application is however in use today. Research happens in disparate topics such as quantum algorithms for plasma physics [110], [111], simulation of nonlinear differential equations in general (as we will see in chapter 8) or speculations related to how to build a Quantum Internet [30]. An exhaustive list would

take too much.

A critical view may be given by the fact that a quadratic advantage may not be enough [112]. At least in one instance it was found a polynomial advantage of order 5, in high-energy physics, in an algorithm that calculates scattering amplitudes in Quantum Field Theory [113]. We recall that this only shows that a classical algorithm has not been found.

A cautionary tale is provided by the quantum algorithm by Kerenidis and Prakash [114] to speedup recommendation systems, *i.e.*, those algorithms used by social networks or subscription on-demand streaming services. A couple years after their discovery, Ewin Tang used their quantum algorithm as a basis to build a new classical algorithm that achieves the same performance without the use of quantum computers! Replicas of this procedure of **ewinization**, as has been dubbed by Scott Aaronson, for other quantum algorithms, if possible, would still count as progress, found through quantum computing, but would at the same time impair quantum advantage in other fields besides recommendation systems.

Other more wild ideas are still to be pursued: for example to have physics-informed Quantum Machine Learning algorithms such as the one in [115]. Preskill suggested to look for quantum advantage in finding ground states of quantum many-body systems faster than classical computers [116] (also [117] to better understand the problem); others have suggested a change of paradigm such as finding a way to have a model in which the training is performed using quantum computers but inference is performed on normal classical ones, so that there is no need to give a quantum computer to each user [118]; and finally, the suggestion by some that the difficulties in present-day-version of Quantum Machine Learning are a consequence of the fact that the field is *still too classical* because gradient descent is performed by a classical computer: this should be solved by some form of quantum improvement of gradient descent, maybe some thermalization procedure that automatically finds local minima, similar to the suggestion of Preskill *et al.* in [116].

4.4 Why Quantum Vision?

We choose the task of vision, and in particular of (parity) classification of the handwritten digits of the MNIST dataset, as it has many different advantages.

The first question was whether to use a classical dataset or a quantum dataset. The problem with the latter is that they are not standardized, and thus we would have to build it ourselves (which would be laborious), also leaving door to the possibility that some simple classical algorithm would be able to efficiently operate on that new dataset. Thus we chose to study the action of a quantum neural network on a somehow-standardized classical dataset.

The Quantum Neural Network presented in the tutorial of TensorFlow Quantum [119] (which reproduced the results of the Google Quantum MNIST classifier paper [2]) immediately caught our attention: it dealt with a fairly-well standard database, the MNIST dataset of handwritten digits, and dealt with a very simple task: discriminating between a pair of different digits. This task seemed too trivial: to distinguish a one from an eight probably the quantity of ink (*i.e.*, the average intensity of the image) is enough, without the need of quantum effects at all! The advantage was that it was a binary task, thus we could use as predictor the sign of the sum of Z measurements of the result (at the end of the heuristic section 4.6 we explain why we chose this observable instead of others). Besides, one could conceive a nontrivial binary task such as classifying *the parity* of the digits, *i.e.*, whether they are odd or even: this is still binary, thus requiring our measurement to produce a single output bit, while being nontrivial, as it requires seeing that there is something in common among all odd digits even if they are written differently, and thus it probably requires understanding the form of the digits at some level.

The MNIST dataset is well known and has already been used in many quantum neural network papers, and it is also widely used with many different kinds of classical machine learning algorithms, even non-neural-network ones. Here [120] is a table that shows the MNIST classification accuracies for different methods.

The task of vision is indeed a very suited one, as it could be done in few qubits: as we intended to do a classical simulation of the quantum algorithm (for the scarce availability of quantum computers at our disposal) quantum. In the Google Quantum MNIST classifier, the original 28×28 images are first reduced in dimension to 4×4 images (!) and then are encoded into 16 qubits (plus one extra control qubit, this gives 17 qubits for their quantum circuit). Such a low resolution will make our own goal of parity classification almost impossible. We instead will use amplitude encoding, codifying the intensities of the pixels into the amplitudes, *i.e.*, the coefficients of the quantum state expressed in the computational basis. So we are able to use few qubits as

well.

In general, the possibility of dealing with the classification among two digits first and then on parity classification, with the same quantum circuit, gives us the possibility of using the first trivial task as a milestone and the second task as the real goal.

It is true that computer vision is already solved by classical machine learning since 2012 and that it requires $O(n)$ computations, so quantum advantage is not really expected. It is however important that no purely quantum algorithm has been found that achieves an accuracy above 97% on a nontrivial vision task.

We have already argued why digit classification is trivial, as it may depend just on the average intensity. That is why it is not so surprising that the Google Quantum MNIST classifier is able to distinguish pairs of digits.

When nontrivial task are dealt with in the literature, usually the quantum neural network uses a classical pre-processing or post-processing of the data, *i.e.*, the input data is first passed through some classical calculations (which may be a Fourier transform or others), and after the quantum measurement is performed, the results are further elaborated (sometimes even through a neural network layer!). These procedures, applied to not-so-difficult tasks, may give rise to the reasonable suspect that it is the classical part of the computation that does all the heavy lifting in solving the task, while the quantum part is not determinant and could in fact be easily replaced with another classical part. It is to avoid these doubts that we perform no classical pre-processing nor post-processing.

In simple words, the reason for us focusing on the task on quantum vision is not that we hope for a quantum advantage with respect to classical vision, but that *to our knowledge no one has done quantum vision with an accuracy higher than 97% with a purely quantum circuit and a nontrivial classification task.*

This are the main reasoning that drove us towards the task of vision and more precisely of classifying the parity of the handwritten digits of the MNIST dataset.

4.5 Why not Quantum Transformers?

The reasoning we provided for choosing Quantum Vision as the task for our quantum neural network leaves open the choice of the architecture. The

process of validation of the architecture was long and many attempts failed; and we arrived at a solution (which will be presented in chapter 5) that is mainly inspired by the Convolutional Neural Network.

One question one could ask is why didn't we pursue the avenue of Quantum Transformer? Here we will give some reasons, taking the opportunity to briefly touch upon a promising and interesting topic.

The main naive reason for working on quantum transformers is that is the only neural network architecture which can be truly improved by quantum algorithms, as its runtime is $O(n^2)$. Any quantum advantage in the transformer architecture would be deeply impactful, as since its discovery in 2017 this architecture has dominated natural language processing giving rise to the Large Language Models that are being widely used in multiple fields, and also in computer vision (which is indeed the task we chose to work on!), protein structure prediction, reinforcement learning, multimodal learning, robotics and probably many others. Even though Artificial Intelligence is a field in rapid development, the transformer architecture or at least its attention mechanism seems to have been resilient in the last eight years, and the attempts to reduce its computational complexity to $O(n \log n)$ or even just to $O(n^{\frac{3}{2}})$ were not completely successful and usually bring drastic decreases in performance.

If a quantum transformer could display a quadratic advantage (nothing more can be easily attempted for natural language, as $O(n)$ steps are required just to type in the question or read the answer), this would be enough to be impactful, as can be verified by some back-of-the-envelope calculations [121], but only for very big Large Language Models. Otherwise, like the rest of quantum computing, a quadratic advantage is not enough in practice, as argued in [112] (a higher-order polynomial advantage would be needed, but in the quantum transformer it would be probably impractical for the aforementioned reasons).

The reasons not to work on a quantum transformer is that, after having decided to work in quantum vision, here most tasks can usually be solved with a number of steps linear in the input dimension, as Convolutional Neural Networks ($O(n)$ complexity) are competitive against Vision Transformers (ViT) ($O(n^2)$ complexity, which is thus not necessary).

The quantum advantage would be recognizable only for inputs of big dimensions, but a small model that solved the parity classification task would have shown a research direction. The concrete problem was that it was not immediately clear in which way to implement a Quantum Attention mechanism, especially when the input was encoded through Amplitude Encoding. As

Amplitude Encoding allows naturally to perform a convolution in a single 2-qubit gate (!), as we will see in section A.3, we pursued the direction of building a QNN inspired by the Convolutional Neural Network instead.

The quest for a quantum transformer is already on the way in the research community: from more naive approaches [122] to the more general Quixer architecture [123] that uses the Quantum Singular Value Transform that unified the most important known quantum algorithms.

One of the first proposals in this field was the **Quantum Vision Transformer** or *QViT* by Kerenidis *et al.* presented in [3], where many models are proposed but the one simulated in the end was in fact a quantum analogue not of the transformer but of the MLP-Mixer [124], another classical architecture for vision. Besides, classical pre-processing was used, in the form of Principal Component Analysis of patches of the original image, each patch of dimension 7×7 .

A completely different approach to Quantum Transformers, starting from the attempt to optimize linear algebra operations, is presented in [125].

The field clearly arises great interest: even though the scale of quantum computer probably prevents any application for Quantum Transformers, research goes on in the quest to give a quantum advantage to the best architecture known up to now that is capable to reproduce, at least at some level, the extraordinary human capabilities of understanding language and of reasoning. A good review up to at least June 2025 can be found here: [126].

4.6 Heuristics for Quantum Neural Networks

Quantum Machine Learning is a recent field, thus it is just natural that research starts by groping around: especially in a field like quantum neural networks, in which it is not clear where to begin, which are the right tasks to tackle, which quantum architectures certainly will not work. However, in the last few years, some preliminary conjectures arose. We used most of these results of the literature as heuristics to avoid trying architectures doomed to fail. All of them are implemented in the quantum architecture we present in chapter 5.

The main problem in training quantum neural networks is certainly the **barren plateau problem** [4]: the magnitude of the gradients decreases exponentially in the number of qubits, which means that the landscape of the loss function is almost flat (a plateau) and it is extremely difficult to find

a minimum. The measurements needed to estimate the direction increase exponentially.

Many can be the sources of barren plateaus or similar problems, and certainly not all of them have been uncovered. However, many results show how they arise in certain conditions, which we were careful to avoid.

For example, we know that **2-designs**, *i.e.*, parametrized unitaries which appear equal to the Haar measure up to second moment of the distribution (k -th moment for k -designs), cause barren plateaus [4]. As random quantum circuit of even just polynomial length are approximate 2-design [127], we were careful to limit the length of the quantum circuit.

Another factor is the use of **global measurements**. Performing global measurements on the output qubits causes barren plateaus, while performing many local measurements in shallow circuits can avoid them [128].

Maybe not surprisingly, **Problem-inspired architectures** perform generally better [129, 130] than hardware-efficient architectures: we were completely inspired by the task of computer vision, and our quantum neural network is not optimized for a specific quantum computer.

Also, Entanglement between visible qubits and non-visible ones impair learning [131], as for example in quantum analogues to Hopfield machines, so it is better not to perform measurements along the circuit but to measure all qubits in the end instead.

Somehow analogous to the classical bias-variance trade-off, *i.e.*, the fact that a classical machine learning model which for different values of the parameters has a higher variance it also has a smaller bias when used to optimize a certain distribution, there is a quantum counterpart: an **expressibility-trainability trade-off** [7]. In practice, a model which has too much expressibility, *i.e.*, which has so many parameters that could fit a large number of functions, is more difficult to train. This is the reason why, when we had to choose a 2-qubit gate as a building block to be used all over our quantum neural network, we did not use the most general 2-qubit gate which should have 10 parameters: we used a number small enough to allow the whole QNN to be trainable, and high enough to be able to perform the task. As we will see, our proposed QNN uses a 2-qubit gate with 4 parameters.

A similar problem arises with **bad initialization**, which can make the model untrainable even in the absence of barren plateaus [132]. We did not find any problem with initialization.

In short, these heuristics helped us to devise the main characteristics of our quantum architecture: we use 2-qubit gates that balance trainability and

expressibility by having 4 parameters (expressing the most general 2-qubit gate would require 10 parameters); we measure all output qubits with local measures; we are not able to use shallow networks but their length is still linear in the number of qubits; we use a problem-inspired architecture; we are careful to initialize the circuit in a way that does not invalidate training.

Chapter 5

Our *Mixture of Quantum Experts* (*MoQE*)

In this chapter, we present the results of the paper *Solving MNIST with a globally trained Mixture of Quantum Experts*, which presents the first project of this thesis: a new quantum architecture, namely the **Mixture of Quantum Experts** or *MoQE* for short. It is a quantum neural network built to solve quantum vision in a relatively nontrivial task such as parity classification of the handwritten digits of the MNIST database, with an accuracy greater than 97%, using a purely quantum architecture *i.e.*, without classical pre-processing or post-processing. It is to our knowledge the first architecture to do so.

The specific architecture was found with trial and error of different models, always keeping in mind the heuristics to avoid barren plateaus discussed in section 4.6. The final architecture is inspired to both Convolutional Neural Networks and to Mixture of Experts models; it succeeds in its task without having been specifically fine-tuned for the MNIST classification problem: in principle, it can do a binary classification of any image, even of higher resolution, with minimal changes (to increase resolution by a factor of 2 one must add a pair of new qubits).

The paper includes a brief review of the field of Quantum Machine Learning and of the methods to avoid barren plateaus; a description of our goals in building the architecture and the quantum architecture itself, a brief discussion of the limit of infinitely many experts (which is analogous to the Large Width limit of both classical and quantum neural networks). Then we describe the experiments, *i.e.*, classical simulation of the quantum neural network, and we show the results and briefly discuss the conclusions. A more complete

discussion may be found in chapter 6.

Then we appended the Supplemental material of the paper, which contains some precisations on the MNIST dataset; on the classical neural networks that were compared to the quantum ones; the reason why a single 2-qubit gate can perform a convolution; a complete presentation of all the data obtained in the simulations; and finally a brief discussion on the limitations of our algorithm due to its nature as a quadratic classifier.

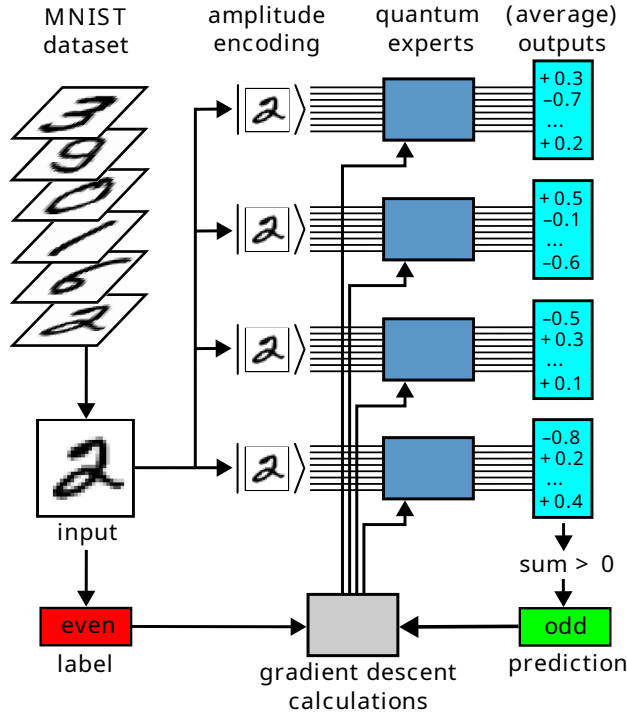


Figure 5.1: Scheme of our Mixture of Quantum Experts (MoQE) architecture. Each expert encodes an input MNIST image in 10 qubits through amplitude encoding, applies a parametric quantum circuit (blue box) and measures each qubit in the Z basis; their average values are thus numbers in the interval $[-1, +1]$. The whole procedure is repeated n times, where n is the number of experts ($n = 4$ in the figure), with different parameters. The sign of the sum of the outputs of all experts gives the binary prediction, which is confronted with the true label. Through (classical) gradient descent calculations, the parameters of the quantum experts are updated together.

5.1 Our architecture

In classical ML, a standard architecture that solves Vision is the Convolutional Neural Network (CNN) (see Appendix B of the Supplemental Material [133] and [134, 135]), which looks for features at increasingly larger scales via a combination of convolutional and pooling layers. Our quantum circuit architecture is based on the same idea: to look for features on small scales first and then to use the results to look for features on larger scales. A quantum generalization of CNNs has been proposed in [136, 137]. The main difference between such quantum CNNs and our architecture is that, thanks to amplitude encoding, in our architecture each convolution operation requires only a single quantum gate. In fact, with amplitude encoding, a single 1-qubit gate performs a convolution, as is shown in Appendix C of the Supplemental Material [133].

Our scalable Mixture of Quantum Experts (MoQE) architecture, which is schematically shown in Figure 5.1, differs from others in the literature [2, 138–140] in some key aspects:

1. the classification is done purely at the quantum level, without any classical neural network pre-processing or post-processing;
2. we leverage amplitude encoding to process full-resolution images without downsampling;
3. inspired by CNNs, the variational circuit is defined assuming correlations between neighboring pixels;
4. we use a mixture of experts, where each expert is a quantum neural network and, crucially, all experts are trained together.

As a relevant problem, we focus on the classification of the parity of MNIST digits, which has been widely used in the Quantum Vision literature [2, 138–140]. MNIST images have a resolution of 28×28 pixels. We encode each full-resolution image in the amplitudes of the wavefunction of 10 qubits as follows. The 28×28 image is embedded in a 32×32 image by padding white pixels at the borders. For any $\mathbf{x}, \mathbf{y} = 0, \dots, 31$, let $c_{\mathbf{x}, \mathbf{y}}$ be the intensity of the pixel in the range $[0, 1]$. Let us express $\mathbf{x} = (x_0, x_1, \dots, x_4)$ and $\mathbf{y} = (y_0, y_1, \dots, y_4)$ in binary using 5 bits each, where the first bit encodes the most significant digit of the coordinate, and so on. Then, by mapping

each bit into a qubit, the image is encoded in the 10-qubit wavefunction

$$|\psi\rangle = \frac{1}{\mathcal{N}} \sum_{\mathbf{x}, \mathbf{y} \in [0,1]^5} c_{\mathbf{x}, \mathbf{y}} |x_0, y_0, \dots, x_4, y_4\rangle, \quad (5.1)$$

where $\mathcal{N}$ is the factor that normalizes the wavefunction, which is different for each image. Amplitude encoding can be compiled in a quantum circuit in several ways: for instance, by first finding the corresponding matrix product state representation [141] and then mapping that into a quantum circuit [142].

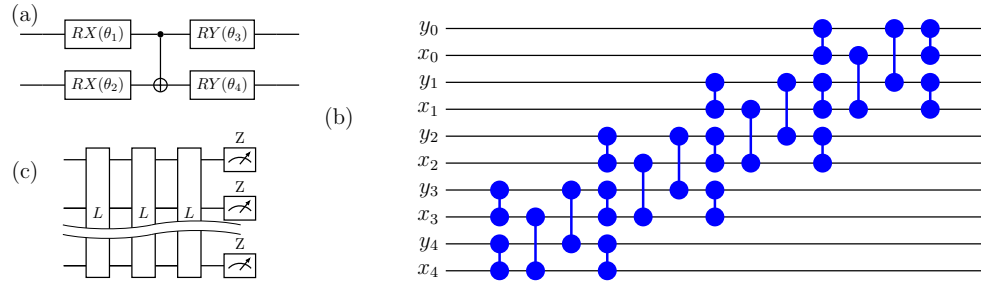


Figure 5.2: (a) Our elementary 2-qubit gate with 4 parameters. This will be the building block of the layers of our quantum circuit. (b) The architecture of each layer: information goes from lower resolution pixel (below) to higher resolution ones (above), moving upward like climbing a ladder. Each blue gate is the 2-qubit gate presented in (a). (c) The final 10-qubit quantum circuit contains three repetitions of the above layer (denoted with L), with the Z measurements at the end – for clarity, only three qubits are shown.

We are now ready to build our circuit from the bottom up. The building block of our architecture is the 2-qubit gate with four parameters of Figure 5.2(a). Such gates are used to build the layer in Figure 5.2(b) in which we look for patterns at small scales and then use the output to look for patterns at higher and higher scales. In a layer, the 2-qubit gates iteratively act on couples of qubits in (5.1) of the form (x_i, y_i) or (x_i, x_{i+1}) or (y_i, y_{i+1}) . We employ 3 layers, for a total of 252 parameters, and then each qubit is measured in the Pauli-Z basis – see Figure 5.2(c). The output is given by the normalized sum of the outcomes. The normalization is such that the empirical variance of the result on the MNIST images is equal to one.

Our architecture enjoys a very useful feature: if the qubits are arranged on a line as in Figure 5.2, the 2-qubit gates of the main circuit only act on

qubits with distance not greater than two. Moreover, if we arrange the x_i and y_i qubits on two adjacent lines forming a 2×5 grid, then all 2-qubit gates only act on neighboring qubits. Therefore, a small quantum computer with a 2×5 grid of qubits capable of single-qubit Pauli rotations and CNOT gates between neighboring qubits can efficiently implement our architecture (except for amplitude encoding).

5.1.1 Mixture of (Quantum) Experts

To increase the number of parameters without increasing either the number of qubits or the number of layers, we opted for a variant of a technique called Mixture of Experts (MoE), widely used in classical AI [143], even in large language models such as DeepSeek-V3 [144]. Our quantum neural network architecture, schematically shown in Figure 5.1, is composed of n experts, where each expert consists of the quantum circuit described in Figure 5.2 and has the same input image, but different independent parameters. The output of the whole network is the sum of the outputs of each expert divided by a normalization constant proportional to $\sqrt{n}$, and the binary classification prediction is given by the sign of the total output. The normalization factor is irrelevant for the classification task, but affects the cost.

Training consists of minimizing the square loss cost function $C = (f - \ell)^2$, where f is the output of the circuit for a given image and $\ell \in \{\pm 1\}$ is the true label, *i.e.*, the parity of the digit represented in the image. We train our network with gradient descent, where we employ the parameter-shift rule [107] to compute the cost gradient. We stress that we do not train each expert independently, but rather train the global output of the network as a whole. Therefore, the gradient descent update of each parameter will generally also depend on the parameters of the other experts.

The mixture of experts will be hard to simulate classically whenever the single experts are hard to simulate. The number of parameters increases linearly with the number of experts and so does the computational cost. Nevertheless, we will empirically show that, over a certain regime, the overall computational cost to achieve a certain accuracy is lower than that of training fewer experts for more epochs.

5.1.2 The limit of infinitely many experts

The idea of an architecture made by a mixture of quantum experts has been inspired by recent results on the infinite-width limit of quantum neural networks [145–147]. Such results state that in the limit of infinite width (*i.e.*, of infinitely many qubits), the probability distribution of the model function of a quantum neural network trained with gradient descent on a supervised learning problem converges to a Gaussian process, provided that the past light cones of the measured qubits are small enough and that the network is not affected by barren plateaus. The mean and the covariance of the limit Gaussian process can be computed analytically, and reveal that the cost function decays exponentially in the training time and therefore the training is always able to perfectly fit the training examples in a time that grows at most polynomially in the number of qubits [145, Theorems 5.4 and 5.9]. In particular, if barren plateaus are not present, the issue of bad local minima disappears in the limit of infinite width. Furthermore, the model function remains smooth in the limit despite the divergence in the number of parameters.

These results can be directly applied to our quantum neural network, since our mixture of experts can be considered as a single network whose width is the sum of the widths of each expert. The condition on small light cones is automatically satisfied as the experts are disconnected. For our network, the limit of infinite width becomes the limit of infinitely many experts. In such a limit, the results above guarantee that the network is trainable and that the model function remains smooth.

5.2 Experiments

We used all images from the MNIST dataset and performed simulations with $n = 1, 2, 4, 8, 16, 32$ experts, respectively. The code, available both at [this github repository](#) and also at [this one](#), is written with PennyLane [148], using the Pytorch backend. We employed the widely used Adam algorithm [149] for gradient descent. The cost function for each gradient descent step was computed on a batch of size 4.

A summary of the results of our simulations is given in Figure 5.3 and Table 5.1, while extended numerical results are presented in Appendix D of the Supplemental Material [133]. The horizontal axis in Figure 5.3 shows the

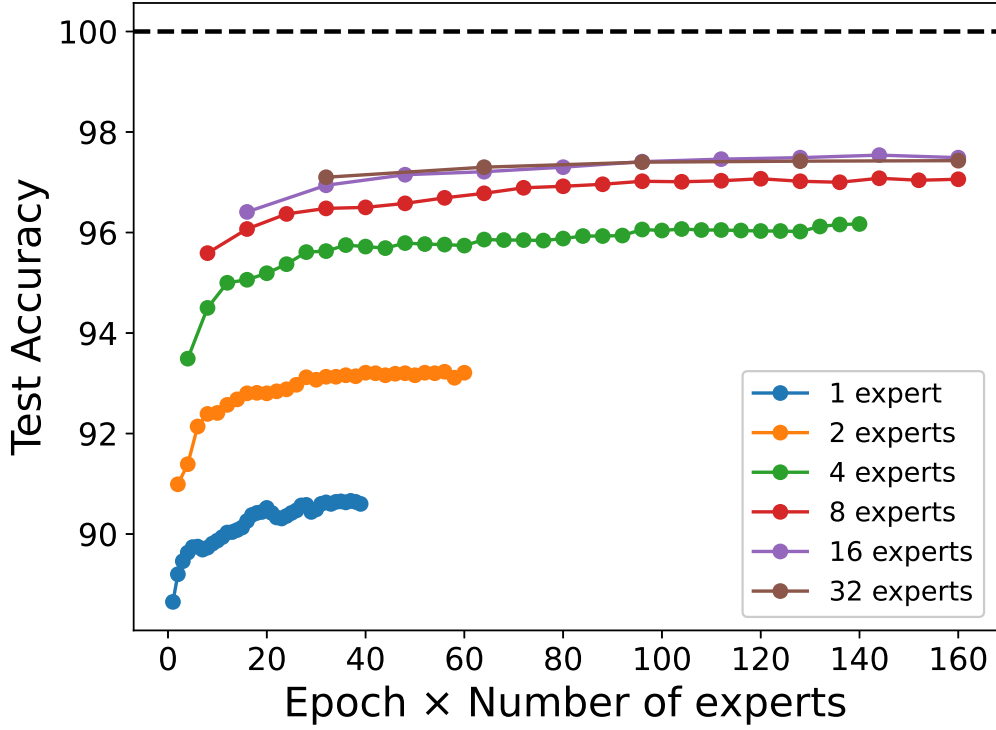


Figure 5.3: Test accuracy in function of the computation, which in our model is proportional to the number of training epochs multiplied by the number of experts. This plot allows us to see what is the best way to allocate the compute to reach the maximum accuracy. We have cut some of the data to ease the comparison between different numbers of experts.

total computational cost, which is proportional to the number of epochs times the number of experts, since each expert requires evaluation and measurement of the same quantum circuit, but with different parameters. In spite of the (linearly) higher computational cost required to train more experts, we show that the test accuracy for the same computational cost is actually higher when the number of experts trained together is increased, until saturation at about 16 to 32 experts. This is the central result of our paper and shows the remarkable advantage of globally training a mixture of quantum experts.

The maximum training accuracy saturates at 97.5% for about 16 experts. While the obtained accuracy is still below what is possible to achieve with classical neural networks, even when using a comparable number of parameters

#experts	train acc.	test acc.	#epochs
1	90.66%	90.60%	39
2	93.37%	93.21%	30
4	96.18%	96.17%	35
8	96.86%	97.06%	20
16	97.38%	97.54%	12
32	97.55%	97.43%	5

Table 5.1: Results for the simulation of our model with different number of experts. We tabulate the training and test accuracy at the end of the training. We highlighted in bold the best result of each column.

(see Appendix B of the Supplemental Material [133]), we believe that the obtained accuracy is close to the optimal one allowed by our model class and that the better performances of other classical models are due to their higher or different non-linearity. In fact, due to amplitude encoding, ignoring the normalization factor, the function class defined by our current MoQE is quadratic in the input, since the expectation value of the output of each expert has the form $\langle \psi | \hat{O} | \psi \rangle$ for some observable $\hat{O}$, and since $|\psi\rangle$ is linear in the input when using amplitude encoding – we can indeed ignore the normalization constant since it is the same for all experts and only provides a global factor that does not affect the sign and hence the model prediction. In Appendix E of the Supplemental Material [133] we show the results obtained by training a general classical quadratic model, which basically reaches 98% test accuracy, only slightly better than our best performing MoQE model. However, such a general quadratic classifier requires $\approx 3 \cdot 10^5$ parameters, significantly more than those used in our largest MoQE. Therefore, we believe that our architecture approaches the best performing classical model within its function class, yet with fewer parameters, and that higher accuracy can only be obtained by changing the function class. One possible way to do so in future studies is to mix the amplitude encoding with data re-uploading layers [150], which are capable of enhancing the model’s non-linearity.

Chapter 6

Conclusions and perspectives

We have proposed a scalable Mixture of Quantum Experts (MoQE) architecture for quantum Vision inspired by the CNN and MoE architectures of classical Machine Learning. Our MoQE architecture has been validated by 97.5% test accuracy in parity prediction in the MNIST dataset. Thanks to amplitude encoding, our quantum neural network needs few qubits (10 for the MNIST images) and can perform the equivalent of an entire classical convolutional layer with a single 1-qubit quantum gate. Our architecture is purely quantum and does not require any classical pre-processing or post-processing. The mixture of experts allows our architecture to increase the number of parameters without increasing the number of required qubits. The model function is not bounded to be quadratic, since the input wavefunction of each expert can consist of multiple copies of the image. Although state-of-the-art classical CNNs and humans can reach an accuracy of 99.8% [151], this is the first time that a purely quantum architecture is successfully applied to the non-coarse grained MNIST database.

Part III

Quantum algorithms for nonlinear differential equations

Chapter 7

Quantum algorithms for linear differential equations

This chapter begins the second part of our thesis, devoted to the study of the quantum algorithm of [10] and of its possibilities and limitations. The work presented a quantum algorithm for nonlinear differential equations. To better understand the particularity of that reasoning, it is better to start with the quantum algorithm for *linear* differential equations [152], which is itself based on the quantum algorithm to solve linear systems *i.e.*, the HHL algorithm, named after Harrow, Hassidim and Lloyd who discovered it in 2008 [11], and later improved by Childs, Kothari and Somma in 2017 [153]. This chapter will be devoted to a discussion over the HHL algorithm in general and how to apply it to solve linear differential equations, with in mind the generalization to nonlinear ones, which will be the topic of the next chapter. First, we will introduce the HHL algorithm with its caveats and give a sketch of the algorithm itself. We then will briefly comment on the difficulties on finding concrete applications, talking about how the algorithm was used in the literature as a basis for other algorithms for quantum machine learning. In the end, we will present the way to solve linear differential equations.

7.1 Introduction to the HHL algorithm

The HHL algorithm is one of the four main quantum algorithms we commented on in section 2.4. The problem it tackles is to solve for $\mathbf{x}$ the

linear system

$$A\mathbf{x} = \mathbf{b},$$

where $\mathbf{x}, \mathbf{b}$ are real vectors of dimension N , and A is a $N \times N$ real symmetric matrix. The HHL algorithm promises to do so in time $O(\log(N))$ (!), which should indeed appear surprising: a time $O(N^2)$ is needed just to read the entries of matrix A , and even without reading them, a time $O(N)$ is needed to read the solution itself x .

In fact, HHL does not solve any possible linear system. The extraordinary claim that it solves $A\mathbf{x} = \mathbf{b}$ in logarithmic time is subject to four important conditions (this discussion comes from the concise and clear paper by Scott Aaronson [31]):

1. The state $|b\rangle = \sum_{i=1}^N b_i |i\rangle$, needed by the algorithm, can be prepared efficiently (for simplicity we assumed $\mathbf{b}$ is a unit vector). This request is crucial: if loading $\mathbf{b}$ requires N^c computations for some constant c , then the exponential speedup vanishes at the very first step. In theory one can use a quantum RAM to store the classical values of the b_i s, but this requires either that the vector is relatively uniform *i.e.*, no component is vastly larger than the others or that the quantum RAM contains not just the components of $\mathbf{b}$ but also the partial sums $\sum_{i=1}^j b_i^2$, or pointers to the largest components of $\mathbf{b}$. Another workaround is that $\mathbf{b}$ is given by a simple explicit formula, so that $|b\rangle$ could be efficiently calculated by the quantum computer itself, without consulting a quantum RAM. One of these very special conditions must happen to avoid irreparably slowing down the whole HHL algorithm.
2. The quantum unitary transformation e^{-iAt} (unitary because A is hermitian) can be applied efficiently for various values of t : if it instead it takes N^c time for some constant c , the exponential speedup vanishes. If the matrix A is sparse (*i.e.*, if it has $O(N)$ nonzero entries; e.g. let us say that it has no more than s nonzero entries per row for some $s \ll N$) and if for each i the quantum RAM stores locations and values of the nonzero entries of the i -th row, then e^{-iAt} can be applied in nearly linear time [154]. There are other special cases in which a quantum computer can efficiently apply e^{-iAt} .
3. The matrix A needs not only to be invertible but *robustly* invertible, which is formalized by asking it to be well-conditioned, *i.e.*, to have

a not-too-big ratio between the magnitudes of its largest and smallest eigenvalue, called the condition number: $\kappa = |\lambda_{max}/\lambda_{min}|$. The HHL algorithm requires a time nearly linear in κ : thus, if $\kappa \propto N^c$, the exponential speedup vanishes.

4. Finally, the output of the HHL algorithm is not the solution $\mathbf{x}$, but the state $|x\rangle = \sum_{i=1}^N x_i |i\rangle$ of $\log_2 N$ qubits, which encodes the entries of x in its amplitudes (through amplitude encoding, in practice). If one wants to calculate the scalar product $\langle x|z\rangle$ with some other fixed vector $\mathbf{z}$, this can be done efficiently; but to recover the whole vector x one has to perform $O(N)$ scalar product operations, and thus the exponential speedup vanishes.

The presence of the condition number κ is very natural as it appears also in classical analysis of linear systems: for example, a proposal to develop a thermodynamic analogical solver of linear system found out that to perform a matrix inverse, where the classical algorithms can do it in $O(d^\omega)$ time (where $\omega \approx 2.3$ is the matrix multiplication constant, associated to the still unknown best possible classical algorithm) while the newly proposed thermodynamic algorithm requires $O(d^2 \kappa \epsilon^2)$ time, where ϵ is the desired precision and κ is indeed the condition number.

The last point about the output of the HHL algorithm, instead, clearly shows that, strictly speaking, the HHL algorithm does *not* solve a system of linear equations in logarithmic time: if given a linear system $A\mathbf{x} = \mathbf{b}$, and assuming the ability to efficiently prepare the state $|b\rangle$ and the unitary transformation e^{-iAt} , it can prepare a superposition of the form of $|x\rangle$ in a time

$$O\left(\log N \cdot \frac{\kappa s}{\epsilon}\right)$$

where κ is the linear system's condition number, s the sparsity, ϵ the desired error of the solution $\mathbf{x}$. Clearly such an algorithm comes with many caveats and limitations, but before delving in them we will now present a sketch of the algorithm itself.

7.2 A sketch of the HHL algorithm

First, we need to fix our notation. We have

$$A|x\rangle = |b\rangle$$

by definition, where $\mathbf{x}$ and $\mathbf{b}$ are mapped into $|x\rangle$ and $|b\rangle$ through amplitude encoding. Now, since A is hermitian (and real), one can use its spectral decomposition to write

$$A = \sum_{j=1}^N \lambda_j |u_j\rangle \langle u_j|$$

and assuming A is invertible we can write also

$$A^{-1} = \sum_{j=1}^N \lambda_j^{-1} |u_j\rangle \langle u_j|.$$

We now express $|b\rangle$ in the eigenbasis of A :

$$|b\rangle = \sum_{j=1}^N b_j |u_j\rangle,$$

so that we can write

$$|x\rangle = A^{-1} |b\rangle = \sum_{j=1}^N \lambda_j^{-1} b_j |u_j\rangle, \quad (7.1)$$

which the HHL algorithm can efficiently prepare if one assumes to be able to efficiently prepare $|b\rangle$ and efficiently implement the unitary

$$U = e^{-iAt} = \sum_{j=1}^N e^{-i\lambda_j t} |u_j\rangle \langle u_j|.$$

The algorithm involves four main steps: quantum phase estimation (QPE), eigenvalue inversion, uncomputation (*i.e.*, we invert the QPE) and rejection sampling. Let us review them (we follow the presentation by Lee *et al.* in [155], reproduced also in [156]):

Quantum Phase Estimation. The QPE allows us to use an efficient implementation of U to implement

$$|0\rangle_N \otimes |b\rangle_V \otimes |0\rangle \quad \Rightarrow \quad \sum_{j=1}^V b_j |\lambda_j\rangle_N \otimes |u_j\rangle_V \otimes |0\rangle,$$

where the N at the pedix of a ket shows the dimensionality of its Hilbert space, and $|\lambda_j\rangle_N$ is the binary representation of λ_j .

Eigenvalue Inversion. By applying a rotation to the last auxiliary qubit conditioned on $|\lambda_j\rangle$, we can get

$$\sum_{j=1}^V b_j |\lambda_j\rangle_N \otimes |u_j\rangle_V \otimes |0\rangle \Rightarrow \sum_{j=1}^V b_j |\lambda_j\rangle_N \otimes |u_j\rangle_V \otimes \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)$$

where $C > \lambda_{max}$ is a normalization constant.

Uncomputation. To perform the last step we will need to ignore the first part of the system, which is now not possible as it is entangled with the rest. So we disentangle it by performing the QPE operation in reverse:

$$\begin{aligned} & \sum_{j=1}^V b_j |\lambda_j\rangle_N \otimes |u_j\rangle_V \otimes \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) \Rightarrow \\ & \Rightarrow \sum_{j=1}^V b_j |0\rangle_N \otimes |u_j\rangle_V \otimes \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right). \end{aligned}$$

Now we can simply ignore the first part of the system *i.e.*, $|0\rangle_N$, as it is completely unentangled with the rest.

Rejection Sampling. It means we measure the ancilla qubit in the computational basis and we discard all results where the state is $|0\rangle$. We get

$$C \sum_{j=1}^V \frac{b_j}{\lambda_j} |0\rangle_N \otimes |u_j\rangle_V \otimes |1\rangle.$$

By ignoring the first part of the system and the auxiliary qubit, and ignoring the normalization constant, we get

$$\sum_{j=1}^V \frac{b_j}{\lambda_j} |u_j\rangle_V$$

which is indeed $|x\rangle$ according to equation (7.1).

7.3 Applications of the HHL algorithm

We briefly have to talk about the possibilities and the difficulties of applying the HHL algorithm to useful tasks. We follow the main insights

given by Scott Aaronson in [31].

An interesting result that HHL is *universal for quantum computation* [11] which means that we can encode any quantum algorithm, like Shor's, and write it as a system of 2^N linear equations to be solved by HHL in polynomial time in N . We have already seen that all quantum algorithms are unified in a single framework given by the Quantum Singular Value Transform [32], but this is another way to get such a Grand Unification using the HHL algorithm itself.

However, the HHL should maybe be seen as a *template* for building quantum algorithms for specific purposes, and not as a quantum algorithm by itself, due to the caveat of the difficulty of reading the data from the output $|x\rangle$: one has to start with a task, and see if one can use the HHL algorithm for that task by addressing all the caveats, and see whether one can find a quantum advantage for that specific task. At the time of [31] *i.e.*, in 2015, the only attempt to apply HHL from start to finish was the work by Clader, Jacobs and Sprouse [157], which applied it to calculate electromagnetic scattering cross-section for systems involving smooth geometric *qutrit* figures: one had to reduce Maxwell's equations to a linear system, show that A is sparse, that κ is upper-bounded at least numerically and with a suitable preconditioning, provide that A and $|b\rangle$ are explicitly calculable, and being interested in calculating only some observables of x .

All this work clearly does not assure us that there is no classical algorithm to solve this cross-section task in logarithmic time.

Other applications where for k -means clustering, support vector machines, data fitting, and computing some properties of Google PageRank vectors: they are all examples of classical non-neural-network algorithms firmly rooted in linear algebra operations. The problem is always that, even if one addresses all the caveats, one cannot be sure that there is no classical algorithm that can achieve the same performance. This is not impossible, but one has to be mindful of the problem. We will consider just two application of the HHL algorithm to get the gist of the difficulties of applying the HHL algorithm to perform some useful task and of the possible ways to solve them.

The first is a 2013 algorithm by Lloyd, Mohseni and Rebentrost [158]: we are given a state $|u\rangle$ and two clusters $|v_1\rangle \dots |v_m\rangle$ and $|w_1\rangle \dots |w_m\rangle$, each of them of $\log_2 N$ qubits. One wants to know if $|u\rangle$ is nearer to the center of the first cluster or to the center of the second one, which are respectively

$$|v\rangle := \frac{1}{m}(|v_1\rangle + \dots + |v_m\rangle), \quad |w\rangle := \frac{1}{m}(|w_1\rangle + \dots + |w_m\rangle).$$

The algorithm by [158] solves the problem by using HHL to estimate the inner products $\langle u|v\rangle$ and $\langle u|w\rangle$ to accuracy ϵ in time $O(\log(mn)/\epsilon)$.

It would seem an exponential speedup, but the caveats of the HHL intervene: can the states $|u\rangle, |v_j\rangle, |w_j\rangle$ be efficiently prepared? If they can be efficiently prepared e.g. because all amplitudes are nearly uniform (*i.e.*, no one has a magnitude far greater than the others), then this fact would allow us to estimate the inner products $\langle u|v\rangle$ and $\langle u|w\rangle$ with a *classical* random sampling algorithm in $O(\log(mn)/\epsilon^2)$ steps! In this specific case, thus, there is a classical algorithm that is at most quadratically slower in the desired precision ϵ .

If the quantum states can be efficiently prepared for some other reason, for example because we have the Fourier transform of $|v\rangle$ *i.e.*, $|\tilde{v}\rangle$, then we can estimate $\langle u|F|\tilde{v}\rangle$ in logarithmic time [159, 160], while classical computation requires longer time. But there is still the same problem that $|u\rangle$ and $|\tilde{v}\rangle$ need to be prepared efficiently. In general: in all applications of HHL there must be a fast way to create certain superpositions of data, and also a reason for it there *not* to be a fast way to generate the same probability distribution with a classical computer. The Fourier transform is a nice trick that obtains this, but there are others.

The second and final example is the Lloyd, Garnerone and Zanardi paper [161] where they proposed a quantum algorithm based on HHL to calculate Betti numbers (a topological feature of the data) in polynomial time. The inputs consist on only $n(n-1)/2$ real numbers (the distances between each pair of points out of n points) and from that they build a quantum superposition of 2^n simplices (*i.e.*, the higher dimensional extensions of the triangle and of the tetrahedron): the takeaway is that most of the needed data is computed from a much smaller input. The main drawback is that preparing this superposition takes a time that grows like $\sqrt{2^n/S}$ where S is the total number of simplices in the complex. Usually $S \ll 2^n$, and in this case the algorithm requires exponential time thus giving no advantage; but even the ability to efficiently estimate Betti numbers for those cases where $S \approx 2^n$ is an example of how the limitations of the HHL algorithm can be overcome in specific applications. To wrap up the discussion, we say that the HHL algorithm may not be a straightforward way to solve systems of linear equations in logarithmic time, as sometimes it is mistakenly oversimplified, but it may be the case that it will give an exponential speedup in some practical task, where all caveats of the algorithm are satisfied by the kind of data and of the quantities that need to be calculated, and where there is no comparable classical algorithm that does the same. Many attempts to achieve a quantum speedup through the

HHL algorithm have been in non-neural-network machine learning algorithms and in particular in the topology sector, as we have seen: more recently, Lloyd *et al.* have found a quantum algorithm to calculate Khovanov polynomials (a topological quantity which is the categorization of the Jones polynomials, used in topological QFTs) [162]. Research continues in this and other topics, which is one of the way to obtain a quantum speedup in machine learning by applying quantum computing to speed up linear algebra operations directly. This spirit could be applied also to produce quantum analogues of neural network algorithms such as the transformer, as in [125]. But all these applications are not directly related to our application of the algorithm.

7.4 The quantum algorithm to solve linear differential equations

We now present how to apply the HHL algorithm to a task strongly related to the second project of this thesis: the task of solving linear differential equations. For simplicity we review the original method of [152] (just as done in [10], where it is shown how to extend it to nonlinear ones): to obtain optimal scaling and computational complexity, more sophisticated techniques are required [163].

Let us consider a linear differential equation of the form

$$\frac{dx}{dt} + Ax = b(t) \quad (7.2)$$

where $x, b \in \mathbb{C}^D$ (to simplify the notation, in this and in the following sections we do not write vectors in bold) and A is a $D \times D$ matrix. We now discretize time with intervals of size Δt and we take $k = 0, \dots, T$ as the index of discretized time in x_k, b_k . We now will integrate the equation numerically. We start with

$$x_0 = \Delta t b_0$$

where Δt is added to ease the notation. This will be our initial condition. We then add the update rule

$$x_{k+1} = x_k - \Delta t A x_k + \Delta t b_k$$

which is simply the Euler forward method for numerical integration. More complex methods are available to have greater numerical stability [164] such

as Euler backward, Crank-Nicholson, Runge-Kutta and others higher order methods, but we will not use them in our work.

We obtain the linear system of equations

$$-\begin{pmatrix} -\mathbb{1} & 0 & 0 & \dots & 0 & 0 \\ \mathbb{1} - \Delta t A & -\mathbb{1} & 0 & \dots & 0 & 0 \\ 0 & \mathbb{1} - \Delta t A & -\mathbb{1} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -\mathbb{1} & 0 \\ 0 & 0 & 0 & \dots & \mathbb{1} - \Delta t A & -\mathbb{1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_{T-1} \\ x_T \end{pmatrix} = \begin{pmatrix} \Delta t b_0 \\ \Delta t b_1 \\ \Delta t b_2 \\ \dots \\ \Delta t b_{T-1} \\ \Delta t b_T \end{pmatrix}. \quad (7.3)$$

We can now understand how to solve the linear system using the HHL algorithm. The idea is to add a time-step quantum register $|k\rangle$ and create the quantum history states that contain the whole evolution of x and b :

$$|X\rangle := \sum_{k=0}^T |x_k\rangle |k\rangle$$

$$|B\rangle := \Delta t \sum_{k=0}^T |b_k\rangle |k\rangle$$

One can also define

$$\mathcal{M} := \sum_{k=0}^T \mathbb{1} \otimes |k\rangle \langle k| - \sum_{k=0}^{T-1} (\mathbb{1} - \Delta t A) \otimes |k+1\rangle \langle k|$$

so that the linear system becomes

$$\mathcal{M} |X\rangle = |B\rangle$$

which can easily be inverted using the HHL algorithm. One gets the output state $|X\rangle$ which can undergo post-processing to reveal features of the solution of the linear differential equation (7.2), with the caveats explained in the previous discussions about the HHL algorithm.

We are now ready to extend this algorithm to nonlinear differential equations.

Chapter 8

Quantum algorithms for nonlinear differential equations

This section deals with the quest for quantum algorithms for nonlinear differential equations, which would extend the treatment of chapter 7 for linear differential equations.

We start by giving a review of the literature in this field, centered in fluid dynamics applications and in Carleman linearization. In the second part, we describe instead the quantum algorithm of [10], which is similar in spirit to the one for linear differential equations previously described. It is this algorithm which we studied in the second project of this thesis, which is treated in chapter 9.

8.1 The quest for an algorithm

Research on quantum algorithms for nonlinear differential equations has gained momentum in recent years, motivated by the development of analogous quantum algorithms for linear systems we showed in chapter 7. An example of application of quantum algorithms for nonlinear differential equations is fluid dynamics: a pedagogical introduction by Succi *et al.* can be found in [165], thus many of the following papers are in this field.

Among the most influential approaches, **Carleman linearization** has emerged as a central idea in the field.

Carleman linearization embeds a nonlinear system into an infinite-dimensional linear one by introducing monomials of the original variables as new variables,

which can then be truncated and treated with quantum linear-system algorithms. The pioneering work of Liu *et al.* [166] established that dissipative quadratic Ordinary Differential Equations (ODEs) can be simulated with polynomial complexity in favorable regimes, proving convergence of the Carleman truncation and identifying dissipation-dominance conditions under which the method remains efficient. Subsequent studies, such as the Lattice-Boltzmann Carleman quantum algorithm [167], applied Carleman linearization to fluid dynamics and lattice Boltzmann models, showing convergence at low and moderate Reynolds numbers and analysing circuit depth and nonlocality issues that arise from the growth of the Carleman hierarchy. The *Three Carleman routes* study [168] examines how the linearization behaves when applied to different formulations of fluid dynamics, *i.e.*, Lattice Boltzmann, Navier–Stokes, or Grad: it was found that a Carleman–Grad formulation offers a balanced compromise between convergence and computational overhead. The Carleman–Grad approach is also treated in [169]

Alongside these developments, advances in quantum linear algebra subroutines have provided key improvements. A paper by Krovi [170] improved on quantum algorithms for linear and nonlinear ODEs and refined the runtime analysis in terms of matrix exponential norms, extended applicability to non-diagonalizable matrices, and also suggested that the Carleman-linearized systems may benefit from such techniques. More recently, new studies have introduced three major refinements to Carleman-based algorithms: high-precision solvers with logarithmic dependence on the target error, rescaling strategies to mitigate the exponential growth with truncation order, and tighter error bounds for truncated systems, as shown in [171]. These advances demonstrate that Carleman-based quantum approaches, once considered impractical, can achieve polynomial scaling in physically relevant regimes.

A complementary line of work explores bosonic-mode embeddings and compact mappings. Using Kowalski-type bosonizations, researchers have represented nonlinear terms as interactions among bosonic modes, enabling unitary implementations of nonlinear updates. The Quantum lattice Boltzmann via bosonic mappings proposal [172] illustrates how to translate streaming and collision operators into unitaries and how to map truncated Fock spaces compactly onto qubits, achieving logarithmic qubit scaling with truncation size.

In parallel, variational and hybrid methods have offered NISQ-compatible alternatives. The paper on differentiable quantum circuits for nonlinear PDEs [173] and related works apply parametrized quantum circuits to approximate solutions of equations such as Navier–Stokes or nozzle flows. These hybrid

strategies use automatic differentiation and are truly in the spirit of quantum neural networks: while they lack rigorous error guarantees and face the usual issues of variational optimization, they provide a flexible framework for small-scale nonlinear simulations.

Other studies such as [174] have approached nonlinearity indirectly by reformulating problems in terms of Liouville or ensemble equations, where nonlinear dynamics of a state are replaced by linear evolution of a distribution function. For instance, the Liouville quantum formulation of fluid dynamics line of research interprets turbulence and fluid flow as the evolution of probability densities, which are linear but high-dimensional, requiring hundreds of qubits even for coarse discretizations.

Across these efforts, several themes emerge. The most recurring challenge is the exponential growth of variables with Carleman truncation order (which can be an issue also in the method we will explore in chapter 9), which demands clever rescaling or hybridization to maintain efficiency. Another major issue is non-unitarity: the linearized operators are generally non-unitary, requiring block-encoding and post-selection with low success probabilities, although oracle-based constructions proposed in works like [175] have improved these metrics. Advances in high-precision quantum linear solvers have further changed asymptotic dependencies, and the selection of problem classes (such as dissipative quadratic systems, reaction–diffusion equations, or low-Reynolds-number flows) has been essential to demonstrate plausible speedups.

Finally, perspective papers such as [176] by Lloyd *et al.* emphasize that algorithmic progress must go hand in hand with quantum hardware considerations.

In summary, the field has progressed from conceptual mappings of nonlinear problems to structured algorithmic frameworks capable of handling specific nonlinear models with polynomial resources. Carleman linearization remains the leading formal route, especially when combined with modern quantum linear algebra techniques and rescaling schemes, while bosonic-mode embeddings, Liouville formulations, and variational circuits provide alternative or complementary perspectives.

8.2 A quantum algorithm to solve nonlinear differential equations

Here we reproduce the quantum algorithm to solve nonlinear differential equations presented in [10].

We want to solve a nonlinear differential equation of the form

$$\frac{dx}{dt} + f(x)x = b(t), \quad (8.1)$$

which is simply a nonlinear analogue of equation (7.2) where A has been substituted with $f(x)$, which is still a $D \times D$ matrix that is a polynomial of order p of vectors x and $x^\dagger$, each with components $x^1 \dots x^D$. If the polynomial is homogeneous and p is even (with $p = 2m$) one can write

$$f(x) = x^{\dagger \otimes m} F x^{\otimes m} \quad (8.2)$$

for a suitable fully symmetric tensor F . A nice trick is to increase the dimensionality of x by adding a component x^0 always equal to a constant value smaller than one, so that one can always write equation (8.2), where $f(x)$ is a polynomial of degree p , by using $m = \lceil p/2 \rceil$.

The main idea to solve equation (8.1) is to take $n \gg m$ copies of the initial state of our unknown variable x_0 , so that we start with $x_0^{\otimes n}$. Then, we apply the Hamiltonian

$$H = -i \frac{1}{\binom{n-1}{m-1}} \sum_{j_1 \dots j_m} F_{j_1 \dots j_m}$$

which acts on $x_0^{\otimes n}$ (this is [10, equation (3)]). $F_{j_1 \dots j_m}$ is simply the tensor F applied to the m subsystems labelled by the j indices: they are m copies of the n copies of the system. The sum runs over each possible set of m copies taken from the original n copies. The binomial coefficient is thus necessary to renormalize the action of the Hamiltonian.

Intuitively, the idea is that for a short time Δt the evolution of the whole system of n copies is given by

$$x^{\otimes n} \Rightarrow e^{-iH\Delta t} x^{\otimes n} = \left(\mathbb{1} - iH\Delta t - \frac{1}{2}H^2\Delta t^2 + O(\Delta t^3) \right) x^{\otimes n}$$

but if we restrict to any one of the copies by tracing out all the others, we should get

$$x \Rightarrow \left(\mathbb{1} - \Delta t f(x) \right) x + O(E^2 \Delta t^2) \quad (8.3)$$

which is indeed a nonlinear evolution! We recall that the idea of making many copies of a system at the beginning is one of the ways to introduce nonlinearities in encoded variables inside quantum circuits.

The quantity E is the average value of $|f(x)|$ over the time evolution. We can easily see that equation (8.3) can be integrated until t only if the number of steps $T = t/\Delta t$ is sufficiently large so that the total error $E^2 T \Delta t^2 \ll \epsilon$ i.e., it is smaller than the desired accuracy of the calculation, which gives the condition

$$E^2 t \Delta t \ll \epsilon,$$

which can also be rewritten as

$$T \gg \frac{E^2 t^2}{\epsilon}.$$

Thus, the number of discretized steps must grow quadratically with the time over which the equation needs to be integrated.

Other conditions that must be satisfied are that the number of copies must grow quadratically with the evolution time too:

$$n > \frac{E^2 t^2 m^2}{T \epsilon},$$

where m depends on the nonlinear differential equation (from the supplemental material of [10]).

The quantum algorithm to solve a nonlinear differential equation is in fact completely analogous to the quantum algorithm for linear ones. One takes now n copies of the input, and defines

$$|B\rangle^{(n)} := \Delta t \sum_{k=0}^T |b_k\rangle^{\otimes n} |k\rangle, \\ \mathcal{M}^{(n)} := \sum_{k=0}^T \mathbb{1} \otimes |k\rangle \langle k| - \sum_{k=0}^{T-1} \left(\mathbb{1} - \Delta t \frac{1}{\binom{n-1}{m-1}} \sum_{j_1 \dots j_m} F_{j_1 \dots j_m} \right) \otimes |k+1\rangle \langle k|. \quad (8.4)$$

One can now solve equation

$$\mathcal{M}^{(n)} |X\rangle^{(n)} = |B\rangle^{(n)}$$

finding the history state

$$|X\rangle^{(n)} \approx \sum_{k=1}^T \left(|x_k^{(1)}\rangle \otimes \dots \otimes |x_k^{(n)}\rangle \right) \otimes |k\rangle,$$

where

$$|x_k\rangle = \left(1 - \Delta t f(x_{k-1})\right) |x_{k-1}\rangle + \Delta t |b_k\rangle$$

is the Euler forward solution to equation (8.1). We recall that each copy of the system $|x_k^{(j)}\rangle$ evolves in the same way, by symmetry in both the initial conditions and in the interactions: $|x_k^{(j)}\rangle = |x_k\rangle$ for all k .

There is another subtlety, which is normalization. In the hermitian version of the nonlinear Schrödinger equation, the states $|x_k\rangle$ remain normalized to one, but under the action of the non-Hermitian operator $\mathcal{M}^{(n)}$ defined in equation (8.4), one has to find a way to impose normalization.

The Supplemental condition paper [10] suggested to solve the problem by adding a new variable x^{D+1} , and rescaling by a constant all previous variables $x^0, \dots, x^D$ so that the normalization is assured at the start. To preserve normalization through the evolution, the idea is to impose

$$\frac{d}{dt}x^{D+1} = \frac{d}{dt}(1 - \tilde{x}^\dagger \tilde{x})^{\frac{1}{2}},$$

where we define $\tilde{x}$ as the vector composed of all other components rescaled. So we get

$$\frac{d}{dt}x^{D+1} = -\frac{1}{2}(1 - \tilde{x}^\dagger \tilde{x})^{-\frac{1}{2}} \left(\frac{d\tilde{x}^\dagger}{dt} \tilde{x} + \tilde{x}^\dagger \frac{d\tilde{x}}{dt} \right),$$

where now one can use the Taylor expansion of $(1 - y)^{-\frac{1}{2}}$ around $y = 0$ to build a Hamiltonian that gives exactly the interaction that preserves normalization. However, in the next chapter we will explain how this procedure does not work in practice, thus forcing us to consider alternative ways to normalize the state.

Chapter 9

Our study of the quantum algorithm for nonlinear differential equations

This chapter contains our study of the potentialities and difficulties of applying the quantum algorithm for nonlinear differential equations presented in [10]. To better understand it, we focused mainly on a specific nonlinear differential equation:

$$\frac{dx}{dt} = -x^3. \quad (9.1)$$

As it will soon become clear, equation (9.1) is very suitable to be treated by the quantum algorithm we intend to use. However, our study nevertheless found strong obstacles and ultimately the quantum algorithm failed in finding the solution even for this simple equation.

The section is organized in the following way. First we explain how the quantum algorithm can be applied to solve equation (9.1). We show that the normalization procedure presented in the Supplementary material of [10] cannot be performed, thus we find an alternative and simple way to perform it. We find that here too we can use a *qubit* model. Then, we present the results of our experiments: we wrote two codes that performed classical simulations of the *qubit* model for equation (9.1), and then we solved the classical simulation calculations analytically. All results of the two simulations and of the analytical calculation agree among themselves, but they do not agree with the original equation to be solved (9.1).

9.1 Introduction to our study

To solve equation (9.1) by following the quantum algorithm of [10] which we presented in chapter 8, the idea is to find the right Hamiltonian to implement our desired equation. We have to make n copies of the initial state $|\phi\rangle$, in each of which we embed the variable x , to get $|\Psi\rangle := |\phi\rangle^{\otimes n}$. The state will be chosen to be *real*: this fact will be crucial in the following calculations to guarantee that the normalized state is orthogonal to its time derivative (see equation (9.6)). Then we normalize the state, and calculate how the normalized single copy evolves when an arbitrary Hamiltonian acts on all copies: only then we can choose the Hamiltonian that induces the desired evolution law for x .

We will now explain the procedure step by step.

We start by taking a quantum state $|\phi\rangle$ and making n copies of it. According to [10], the number of copies will be taken to infinity, but clearly not in our classical or quantum simulations. The complete quantum state, *i.e.* the quantum state of the n copies is:

$$|\Psi\rangle := |\phi\rangle^{\otimes n}.$$

Then we call A the matrix that gives the evolution:

$$\frac{d}{dt} |\Psi\rangle = A |\Psi\rangle.$$

We note that the i of the Schrödinger equation is absent. In the following, the matrix A will usually be hermitian (in fact, it will be chosen to be real), but we will deal with the general case. The matrix A will be thus called a *fake Hamiltonian*. We will see later on how an accurate choice of the fake Hamiltonian can bring a dynamics that implicitly solves a nonlinear differential equation.

As the matrix A in general does not preserve the norm, we have to normalize the states separately. We define the normalized single copy and the normalized complete state:

$$|\hat{\phi}\rangle := \frac{|\phi\rangle}{\sqrt{\langle\phi|\phi\rangle}} \quad |\hat{\Psi}\rangle := \frac{|\Psi\rangle}{\sqrt{\langle\Psi|\Psi\rangle}} = |\hat{\phi}\rangle^{\otimes n}.$$

Now, as A gives the evolution of the complete state $|\Psi\rangle$, we calculate the evolution of the normalized complete state $|\hat{\Psi}\rangle$:

$$\begin{aligned} \frac{d}{dt} |\hat{\Psi}\rangle &= A \frac{|\Psi\rangle}{\sqrt{\langle\Psi|\Psi\rangle}} - \frac{|\Psi\rangle}{\langle\Psi|\Psi\rangle} \frac{d}{dt} \sqrt{\langle\Psi|\Psi\rangle} = \\ &= A \frac{|\Psi\rangle}{\sqrt{\langle\Psi|\Psi\rangle}} - \frac{|\Psi\rangle}{\langle\Psi|\Psi\rangle} \frac{1}{2} \frac{\langle\Psi| (A + A^\dagger) |\Psi\rangle}{\sqrt{\langle\Psi|\Psi\rangle}}. \end{aligned}$$

Now we notice that if we divide matrix A into a hermitian part and an anti-hermitian part ($A = A_H + A_A$), only the former contributes. By using the definition of the normalized complete state, we obtain:

$$\frac{d}{dt} |\hat{\Psi}\rangle = \left(A - \langle\hat{\Psi}| A_H |\hat{\Psi}\rangle \right) |\hat{\Psi}\rangle.$$

If we use $\Re, \Im$ to indicate the Real part and the Imaginary part respectively, one can see that the previous equation can be rewritten as

$$\frac{d}{dt} |\hat{\Psi}\rangle = \left(A - \Re \langle\hat{\Psi}| A |\hat{\Psi}\rangle \right) |\hat{\Psi}\rangle. \quad (9.2)$$

In our attempt to solve equation (9.1) we will make sure of choosing both a real matrix A and a real $|\phi\rangle$, so we will drop the Real part.

To write the time evolution of the normalized single copy as a function of the fake Hamiltonian, we have to make some assumptions on the form of the fake Hamiltonian. We recall that in the quantum algorithm we want to implement, the interactions between copies are symmetric [10, equation (3)]. Thus we can write:

$$A = \sum_i A^{(1)} + \sum_{i < j} \frac{A^{(2)}}{n-1} + \dots + \sum_{i_1 < i_2 < \dots < i_p} \frac{A^{(p)}}{\binom{n-1}{p-1}} + \dots, \quad (9.3)$$

where the summation indices run between 1 and n , except when otherwise stated. However, for simplicity, we will reduce ourselves to study the algorithm just for the most simple cases, *i.e.* $p = 1, 2$. We will see how this is sufficient to deal with equation (9.1), which is one of the reason why we chose such a simple equation in the first place. Thus the form we will use is

$$A = \sum_i A^{(1)} + \sum_{i < j} \frac{A^{(2)}}{n-1}, \quad (9.4)$$

and we will make use of equation (9.3) just to treat the cases $p = 1, 2$ at once. As the differential equation 9.2 for Ψ is linear in A , we calculate the contribution of each $A^{(p)}$ separately, and then we put the contributions together. Afterwards, we will keep only the terms with $p = 1, 2$.

We start from equation (9.2), and we apply to the left $\langle \hat{\phi}_{[1]} |$, which is our notation for the bra of all normalized single copies except the first one $|\hat{\phi}_1\rangle$. Recalling that the states are normalized, we get at $t = 0$:

$$\begin{aligned} & \frac{d}{dt} |\hat{\phi}_1\rangle + (n-1) \langle \hat{\phi} | \frac{d}{dt} |\hat{\phi}\rangle |\hat{\phi}_1\rangle = \\ & = \langle \hat{\phi}_{[1]} | \sum_{i_1 < i_2 < \dots < i_p} \frac{A^{(p)}}{\binom{n-1}{p-1}} |\hat{\phi}\rangle^{\otimes n} - \sum_{i_1 < i_2 < \dots < i_p} \langle \hat{\phi}^p | \frac{A^{(p)}}{\binom{n-1}{p-1}} |\hat{\phi}^p\rangle |\hat{\phi}\rangle \end{aligned} \quad (9.5)$$

The second term of the left hand side of the equation (LHS from now on) is equal to zero because the time derivative of a *normalized, real* state, is orthogonal to the state itself: the reason is that for a real state we have

$$\langle \hat{\phi} | \frac{d}{dt} |\hat{\phi}\rangle = \frac{1}{2} \frac{d}{dt} \langle \hat{\phi} | \hat{\phi} \rangle \quad (9.6)$$

which, for a normalized state which remains normalized at all times, it is equal to zero. The reality of the quantum state is crucial to the whole reasoning: this is the reason we required it at the beginning.

Going back to equation (9.5), we see that the first term in the right hand side (RHS from now on) divides itself in two cases: in the first case, the indices include 1 ($i_1 = 1$, as it is the smallest index); in the second case, they do not include it. The RHS becomes:

$$\begin{aligned} & \sum_{1=i_1 < i_2 < \dots < i_p} \langle \hat{\phi}^{(p-1)} | \frac{A^{(p)}}{\binom{n-1}{p-1}} |\hat{\phi}^{(p-1)}\rangle |\hat{\phi}_1\rangle + \sum_{1 < i_1 < i_2 < \dots < i_p} \langle \hat{\phi}^{(p)} | \frac{A^{(p)}}{\binom{n-1}{p-1}} |\hat{\phi}\rangle^{(p)} |\hat{\phi}_1\rangle + \\ & - \sum_{i_1 < i_2 < \dots < i_p} \langle \hat{\phi}^p | \frac{A^{(p)}}{\binom{n-1}{p-1}} |\hat{\phi}^p\rangle |\hat{\phi}_1\rangle = \\ & = \frac{\binom{n-1}{p-1}}{\binom{n-1}{p-1}} \langle \hat{\phi}^{(p-1)} | A^{(p)} |\hat{\phi}^{(p-1)}\rangle |\hat{\phi}_1\rangle - \frac{\binom{n}{p} - \binom{n-1}{p}}{\binom{n-1}{p-1}} \langle \hat{\phi}^p | A^{(p)} |\hat{\phi}^p\rangle |\hat{\phi}_1\rangle. \end{aligned}$$

We finally get the time evolution of the normalized single copy under the interaction between p copies:

$$\frac{d}{dt} |\hat{\phi}_1\rangle = \langle \hat{\phi}^{(p-1)} | A^{(p)} |\hat{\phi}^{(p-1)}\rangle |\hat{\phi}_1\rangle - \langle \hat{\phi}^p | A^{(p)} |\hat{\phi}^p\rangle |\hat{\phi}_1\rangle.$$

Combining all the interactions, and dropping the label of the first copy, we get the time evolution of the normalized single copy under the fake Hamiltonian A :

$$\frac{d}{dt} |\hat{\phi}\rangle = \sum_{p=1}^{+\infty} \left(\langle \hat{\phi}^{(p-1)} | A^{(p)} | \hat{\phi}^{(p)} \rangle - \langle \hat{\phi}^p | A^{(p)} | \hat{\phi}^p \rangle |\hat{\phi}\rangle \right), \quad (9.7)$$

which for $p = 2$ (which is the case we will be interred in) gives

$$\frac{d}{dt} |\hat{\phi}\rangle = \langle \hat{\phi} | A^{(2)} | \hat{\phi}^{(2)} \rangle - \langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle |\hat{\phi}\rangle. \quad (9.8)$$

Before moving on, we have to note an important point. The complete quantum state is initialized at $t = 0$ as a composite state $|\hat{\Psi}\rangle = |\hat{\phi}\rangle^{\otimes n}$. However, as time evolution goes on, the fake Hamiltonian mixes the copies so that the state at time $t \neq 0$ is no longer a composite state of separate single copies. In general we have

$$|\hat{\Psi}(t)\rangle \neq |\hat{\phi}(t)\rangle^{\otimes n}.$$

This is true only at $t = 0$ by construction. At later times, the approximation of considering the complete state a composite state, which we call the **composite state approximation**, is violated more and more. These violations are non-negligible, and are fatal to many computations, as we will see.

9.1.1 Toy models for embedding x

Equation (9.7) will allow us to look for a fake Hamiltonian A that gives the desired evolution for x ; but first we have to embed x into the single copy $|\hat{\phi}\rangle$.

Many possible choices are allowed. We will try the most natural one, suggested in the paper [10] and we will explain its problems: the normalization procedure does not work, and another one must be found. We give our proposal. At the end of this subsection, we will be ready to do numerical experiments and analytical calculations.

We recall the embedding described in the quantum algorithm of [10], which will henceforth be called the *original paper*. The main idea (which we explained at length in chapter 9) is that when we have the differential equation

$$\frac{dx}{dt} + f(x)x = b(t)$$

we can write $f(x) = (x^\dagger)^{\otimes m} F x^{\otimes m}$ for a suitable tensor F : these are equations (1) and (2) of the original paper. From these one writes the Hamiltonian

$$H = -i \frac{1}{\binom{n-1}{m-1}} \sum_{j_1 \dots j_m} F_{j_1 \dots j_m},$$

which is equation (3) of the original paper.

Our equation is of the desired form, with $b(t) = 0$, $f(x) = x^2$ and thus we have $m = 1$ and $F = \delta_{11,11}$ (which is our notation for the operator $|11\rangle\langle 11|$) and where x is embedded as the coefficient of $|1\rangle$. To be complete, we follow the Supplementary material of the original paper in choosing the following embedding: the coefficient of $|0\rangle$ is a constant necessary to produce non-homogeneous polynomials as functions $f(x)$ for the differential equation (our equation does not need it, but for completeness we add it now). As $m = 1$ we need only one direction, *i.e.* $|1\rangle$, to embed x (by using x as its coefficient, maybe multiplied by a constant). Finally, we add $|2\rangle$ with a coefficient to normalize the whole state. We have:

$$|\hat{\phi}\rangle = \begin{pmatrix} \hat{\phi}_0 \\ \hat{\phi}_1 \\ \hat{\phi}_2 \end{pmatrix} := \begin{pmatrix} \sqrt{1-c^2} \\ cx \\ c\sqrt{1-x^2} \end{pmatrix}. \quad (9.9)$$

We take as fake Hamiltonian

$$A = \alpha \delta_{11,11} + B$$

as suggested before. Here α is a constant to be determined to get the slope of the solution right, and B is a term to be determined. B will implement the normalization of the state for now, let us assume $B = 0$. The evolution equations of ϕ are given by (9.8)

$$\frac{d}{dt} |\hat{\phi}\rangle = \langle \hat{\phi} | A^{(2)} | \hat{\phi}^{(2)} \rangle - \langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle |\hat{\phi}\rangle. \quad (9.10)$$

From this we obtain:

$$\begin{cases} -\frac{c\dot{c}}{\sqrt{1-c^2}} = \langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle \\ \dot{c}x + c\dot{x} = \alpha c^3 x^3 + \langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle \\ \dot{c}\sqrt{1-x^2} - \frac{cx\dot{x}}{\sqrt{1-x^2}} = \langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle \end{cases}.$$

Now, we will normalize, thus $B \neq 0$. If normalization is preserved, we clearly have $\langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle = \langle \hat{\phi}^2 | \frac{d}{dt} | \hat{\phi}^2 \rangle = 0$ (we recall that $|\hat{\phi}\rangle$ is real). Now $\dot{c} = 0$ as desired, thus the equations simplify. The equation for x is already the desired one. However, the third equation must evolve in such a way to preserve the normalization of the state (otherwise the previous simplification would not work!). Thus we have:

$$\begin{cases} \dot{c} = 0 \\ \dot{x} = \alpha c^2 x^3 \\ -\frac{cx\dot{x}}{\sqrt{1-x^2}} \stackrel{?}{=} -\alpha c^3 x^4 (1-x^2)^{-\frac{1}{2}} \end{cases} \quad (9.11)$$

where the symbol $\stackrel{?}{=}$ indicates that it is the value required to satisfy normalization, but it is not immediately clear how to recover it from a fake Hamiltonian B . Probably there is no way to obtain it in a finite number of terms.

Before continuing, we note that the choice of the parametrization of the constant c is clearly important, but simple variants do not change the overall arguments. For example, by choosing the parametrization

$$|\phi\rangle = \begin{pmatrix} c \\ cx \\ \sqrt{1-c^2-c^2x^2} \end{pmatrix} \quad (9.12)$$

instead, we would have got

$$\begin{cases} \dot{c} = 0 \\ \dot{x} = \alpha c^2 x^3 \\ -\frac{c^2 x \dot{x}}{\sqrt{1-c^2-c^2x^2}} \stackrel{?}{=} -\alpha c^5 x^4 (1-c^2-c^2x^2)^{-\frac{1}{2}} \end{cases} \quad (9.13)$$

This parametrization was the one suggested in the supplementary material of the original paper.

Now, there are two roads.

The first one is the one proposed by the supplementary material of the original paper in equation (S5). The idea is to use the Taylor expansion of $(1-y)^{-\frac{1}{2}}$ to get polynomial terms in x which can be obtained by interactions with more and more copies of the state. Using this expansion, the third equation, using the second parametrization above (9.12), (9.13), becomes:

$$-\frac{c^2 x \dot{x}}{\sqrt{1-c^2-c^2x^2}} \stackrel{?}{=} -\alpha c^5 x^4 \left(1 + \frac{1}{2}(c^2 + c^2x^2) + \frac{3}{8}(c^2 + c^2x^2)^2 + \dots \right).$$

Now we encounter a problem even with the first term. The first term is clearly a combination of ϕ_0 and ϕ_1 :

$$\frac{d}{dt}\phi_2 = -\alpha\phi_0\phi_1^3\left(1 + \frac{1}{2}(\phi_0^2 + \phi_1^2) + \dots\right)$$

and to influence the time evolution of the normalization component ϕ_2 it must appear in the fake Hamiltonian either of the form

$$B = -\beta\delta_{201,111} + \text{other terms}$$

or maybe

$$B = -\beta\delta_{211,011} + \text{other terms}$$

or others. The problem of these terms is that they do not only give the desired evolution for ϕ_2 , but also they alter the evolution of ϕ_1 (and in the first case also of ϕ_0)! It is clear that those contributions cannot be cancelled. Thus the procedure suggested in equation (S5) of the original paper, which is essential for normalization, cannot be performed in general. We will look for another way to normalize the state.

The second road will be presented using the first parametrization above (9.9), (9.11). Here the third equation reads

$$-\frac{cx\dot{x}}{\sqrt{1-x^2}} = -\alpha c^3 x^4 (1-x^2)^{-\frac{1}{2}}.$$

We simply solve it by making a time-dependent fake Hamiltonian

$$B = \beta(t)\delta_{22,22}$$

from which we get as a third equation

$$-\frac{cx\dot{x}}{\sqrt{1-x^2}} = \beta(t)c^3(1-x^2)^{\frac{3}{2}}$$

which, by using the second equation $\dot{x} = \alpha c^2 x^3$, gives the condition that $\beta(t)$ must satisfy:

$$\beta(t) = -\frac{\alpha x^4}{(1-x^2)^2} \tag{9.14}$$

(we will soon discuss how one can know $x(t)$ to obtain the right value of $\beta(t)$ at all time). Now the 2-copies interaction is complete:

$$A^{(2)} = \alpha\delta_{11,11} + \beta(t)\delta_{22,22}.$$

We can verify that the term that we put equal to zero before, is in fact zero:

$$\langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle = \alpha c^4 x^4 + \beta(t) c^4 (1 - x^2)^2 = 0,$$

due to the definition of β in equation (9.14).

As we want the slope of the equation to be right, we get

$$\alpha = -\frac{1}{c^2}; \quad \beta(t) = -\frac{\alpha x^4}{(1 - x^2)^2}. \quad (9.15)$$

The fake Hamiltonian is

$$\begin{aligned} A &= \sum_{i < j} \frac{A_{ij}^{(2)}}{n-1} = \sum_{i \neq j} \frac{A_{ij}^{(2)}}{2(n-1)} = \sum_{i \neq j} \frac{\alpha \delta_{11,11} + \beta(t) \delta_{22,22}}{2(n-1)} = \\ &= \sum_{i \neq j} \left(|A| \delta_{11,11} + |B(t)| \delta_{22,22} \right) \end{aligned}$$

where it is clear that $\delta_{11,11}$ and $\delta_{22,22}$ act on the Hilbert spaces of the two copies of indices i, j . We have also defined

$$|A| = -\frac{1}{2(n-1)c^2}; \quad |B(t)| = -\frac{1}{2(n-1)} \frac{|A|x^4}{(1-x^2)^2}.$$

If the composite state approximation holds (which will not be the case, as we will see later), our model perfectly simulates the differential equation

$$\dot{x} = -x^3$$

as desired.

Before continuing, we recall that we have not described how to evaluate $x(t)$ at all times to calculate $\beta(t)$. This problem is crucial in our proposed normalization procedure. We have to admit that realizing equation (9.14) is unfeasible on quantum computers: to estimate $x(t)$ at time t , we would need to perform a quantum measurement on the quantum system, which would destroy the quantum state itself, thus impeding our ability to continue evolution. We would need to recreate many times the quantum state just to perform measurements at time t that would allow us to continue the time evolution for a time step more until $t + dt$: this approach is impractical. Nevertheless we will implement this idea in classical simulations, where

measurements are not destructive. We do this for different reasons: the first is that *a posteriori* we found that the solution is insensitive to a small change in the method to approximate $\beta(t)$ as we will briefly discuss later; the second reason is that we are no longer certain that the method will solve the original equation (9.1), as the calculations have been done in the composite state approximation, which may not hold.

Indeed, in the empirical and analytical study of this model, we will see that the evolution is quite different from that of the solution to equation (9.1). The only possible reason is the failure of the composite state approximation, which is shown to be fatal for this and other easier or more complex models.

9.2 The simulations

In order to study our model which tries to solve equation (9.1), we understand that there are different possible choices to calculate $\beta(t)$ (but as we found, the result is not too sensible on small errors on $\beta(t)$). Besides, even after having chosen the technique to choose $\beta(t)$, there are different methods to perform the calculations to obtain $\tilde{x}(t)$, which is our estimate of $x(t)$.

The methods we tested in the experiments are the following:

1. **Code 1.** The code can use a qutrit or a qubit single copy state $|\phi\rangle$, it solves the differential equation step by step, by writing the complete quantum state $|\Psi\rangle$ using second quantization, so that the memory requirement is $O(n^D)$ with $D = 2, 3$, instead of $O(2^n)$.
2. **Code 2.** For a given time t , it calculates directly the value of $\tilde{x}(t)$ (which is our estimate of $x(t)$ assuming the validity of the composite state approximation) without the need of writing in memory all the components of the complete quantum state $|\Psi\rangle$, and without the need to perform calculation of quantities at previous times.
3. **Analytical Calculations** We noticed that the calculations of Code 2 in the limit of infinite copies $n \rightarrow +\infty$ require to evaluate integrals that can be solved with saddle point expansion, which in the limit is exact.

The codes for the two numerical methods are available in [this github repository](#).

We present the theoretical underpinnings of the two Codes and of the Analytical Calculations in their respective sections. As we will see, each one builds up on calculations of the previous one. Before doing that, we briefly recall the discussion on the *modes*, which we postponed to know. We recall that to evolve the system we need to know $|B(t)|$, which can be calculated from Equation 9.16: one needs to use some estimate of $x(t)$. The most simple one is the **Analytical mode**: we simply use $\tilde{x}(t)$. This is used in Code 2.

Another possibility is the **Calculation mode**: we use the value of B that preserves normalization of the state, calculated by solving the quadratic equation

$$|\Psi(t + dt)|^2 = |\Psi(t)|^2$$

which can be expanded to

$$|\Psi(t) + d\Psi_A(t) + d\Psi_B(t)|^2 = 1$$

where $d\Psi_B$ is the part of the infinitesimal change that is proportional to $|B(t)|$. The equation is a second degree equation in B , and one has to find the right solution, which is the lowest in absolute value. This was used in Code 1.

9.2.1 Code 1

Qubit variant

As we have already discussed, the general approach suggested by the original paper [10], when applied to our specific Equation 9.1, implies the need to use a qutrit as the single copy state $|\phi\rangle$ which will encode the variable x : one component to encode x (component 1), one constant component to allow the creation of more general polynomials (component 0) and one component to normalize the state (component 2). The single copy could be

$$|\phi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{x}{\sqrt{2}} \\ \gamma \end{pmatrix}$$

where γ is such to have $\langle\phi|\phi\rangle = 1$:

$$\gamma = \frac{\sqrt{1 - x^2}}{\sqrt{2}}.$$

This is simply what we called the first parametrization (9.9), (9.11) where we ignore component 0 and we choose $c = 1$.

We noticed however that the 2-particle interaction given by (9.1.1) does not act on component 0: thus we reduce the qutrit to a qubit, by dropping component 0 and renaming the other two components. We also have to renormalize the state, *i.e.* we multiply each component by $\sqrt{2}$, to keep the norm equal to one. The single state is now

$$|\phi\rangle = \begin{pmatrix} \phi_0 \\ \phi_1 \end{pmatrix} = \begin{pmatrix} x \\ \sqrt{1-x^2} \end{pmatrix},$$

where we notice that we changed the name of the two components: now x is encoded in component 0, while the component 1 normalizes the state.

Then we take n copies:

$$|\Psi\rangle = |\phi\rangle^{\otimes n}.$$

We define the density matrix $\rho = |\Psi\rangle\langle\Psi|$ and then we define the reduced density matrix $\rho_{red} = \text{tr}_{2\dots n}(\rho)$, which should be equal to $|\phi\rangle_1\langle\phi|$. The subscript 1 indicates that this is the first copy of $|\phi\rangle$. The vector Ψ evolves according to

$$\frac{d}{dt} |\Psi\rangle = A |\Psi\rangle.$$

We choose to use a matrix A made only of 2-copies interactions: we thus have:

$$A = \sum_{i < j}^n \frac{A_{(ij)}}{n-1} = \sum_{i \neq j}^n \frac{A_{(ij)}}{2(n-1)}.$$

We recall (9.8):

$$\frac{d}{dt} |\hat{\phi}\rangle = \langle\hat{\phi}| A^{(2)} |\hat{\phi}^2\rangle - \langle\hat{\phi}^2| A^{(2)} |\hat{\phi}^2\rangle |\hat{\phi}\rangle,$$

where $|\hat{\phi}\rangle$ is the normalized vector of the single copy. Using the explicit form of the 2-copies interaction

$$A^{(2)} = \alpha\delta_{00,00} + \beta(t)\delta_{11,11}.$$

we easily get

$$\frac{d}{dt} |\hat{\phi}\rangle = \alpha x^3 |1\rangle + \beta(t)(1-x^2)^{\frac{3}{2}} |2\rangle - \left(\alpha x^4 + \beta(t)(1-x^2)^2 \right) |\hat{\phi}\rangle.$$

As the state has to be normalized, we have to impose that $\langle \hat{\phi}^2 | A^{(2)} | \hat{\phi}^2 \rangle = 0$ as before, so we get

$$\alpha x^4 + \beta(t)(1 - x^2)^2 = 0 \quad \Rightarrow \quad \beta(t) = -\frac{\alpha x^4}{(1 - x^2)^2}.$$

We thus find

$$\alpha = -1, \quad \beta(t) = +\frac{x^4}{(1 - x^2)^2},$$

which is simply equation (9.15) with $c = 1$ (instead of using $c = \frac{1}{\sqrt{2}}$ in the *qutrit* model we sketched at the beginning of this subsection), as the value of c has changed with the new *qubit* encoding.

We recall that the fake Hamiltonian is again

$$\begin{aligned} A &= \sum_{i < j} \frac{A_{ij}^{(2)}}{n-1} = \sum_{i \neq j} \frac{A_{ij}^{(2)}}{2(n-1)} = \sum_{i \neq j} \frac{\alpha \delta_{11,11} + \beta(t) \delta_{22,22}}{2(n-1)} = \\ &= \sum_{i \neq j} \left(|A| \delta_{11,11} + |B(t)| \delta_{22,22} \right), \end{aligned}$$

with

$$|A| = -\frac{1}{2(n-1)}; \quad |B(t)| = -\frac{1}{2(n-1)} \frac{|A|x^4}{(1-x^2)^2}. \quad (9.16)$$

Empirically, in the *qubit* variant of the code the results are slightly better than in the *qutrit* variant, *i.e.* more similar to the desired solution, but still very far from it. Trying different values of c in the *qutrit* variant, we found that the greater the better. Obviously $0 < c \leq 1$, where for $c = 1$ one of the three component is not used and the code can be run without it, which gives an obvious increase in performance in term of time and memory requirements. These are the reason we plotted the *qubit* solution simulation only, and we do not show the results of the *qutrit* one.

Second quantization of the qubit model

We recall that, using the qubit parametrization of the single copy given by (9.2.1), we have that each single copy state is given by

$$|\phi\rangle = \begin{pmatrix} \phi_0 \\ \phi_1 \end{pmatrix} = \begin{pmatrix} x \\ \sqrt{1-x^2} \end{pmatrix}.$$

The complete state

$$|\Psi\rangle = |\phi\rangle^{\otimes n}$$

has the complete basis $\{|i\rangle\}_{i=0,1}^{\otimes n}$, which can be rearranged using **second quantization** notation into the complete basis $\{|k, n-k; s\rangle\}_{k=0}^n$, where k indicates the number of 0s in the tensor product, $(n-k)$ is the number of 1s, and the different possible orderings of the 0s and the 1s can be characterized by the nontrivial permutation $\sigma_{s;k,n-k}$ parametrized by the index s :

$$|k; s\rangle = \sigma_{s;k,n-k} \left(|0\rangle^{\otimes k} |1\rangle^{\otimes (n-k)} \right).$$

Using this basis, we have that the coefficient Ψ_k at time $t = 0$ is:

$$\Psi_k(t = 0) = \phi_0^n \phi_1^{(n-k)},$$

where we note that it is independent of s . The multiplicity of basis states for a given k is clearly

$$\mu_k = \binom{n}{k},$$

so that the index $s = 1, \dots, \mu_k$.

Time evolution is given by the fake Hamiltonian of equation (9.1.1):

$$A = \sum_{i \neq j} \left(|A| \delta_{00,00} + |B| \delta_{11,11} \right),$$

where the names of the basis vector is changed as each particle is a qubit rather than a qutrit.

The first term acts on all coefficients of $|k, n-k; s\rangle$ in a way that preserves the symmetry between the copies, *i.e.* independently of s . This implies that $\Psi_k(t)$ remains independent of s , by symmetry, not only at $t = 0$ but at all times. This is the rationale that allows us to use second quantization: instead of the 2^n components of the complete state, we keep track of only $n+1$ components due to the fact that, given a certain k in the range $0 \leq k \leq n$, all μ_k basis vectors with k zeroes have the same time evolution, thus we can keep track of just one of them. This gives obviously a computational and a memory advantage to our code, and is clearly an exact method.

Let us explore the details of time evolution. It acts on coefficients Ψ_k of basis vectors with at least two 0s (*i.e.* $k \geq 2$), it makes the coefficient evolve in a way proportional to itself, and the contribution is proportional to the number

of ways one can find two 0s among k of them, times 2 (as the two 1s can be selected in two different orders). We thus have:

$$\frac{d}{dt}\Psi_k(t) = 2\binom{k}{2}|A|\Psi_k(t) + 2\binom{n-k}{2}|B(t)|\Psi_k(t) \quad (9.17)$$

where we added the contribution of the second term of the fake Hamiltonian, which comes from an analogous reasoning. This equation allows us to calculate the evolution of $\Psi_k(t)$ step by step, using each value to calculate the next value $\Psi_k(t + dt)$, where the time step dt is chosen to be finite but small (the use of a differential as a finite time step is an abuse of notation, as it stands for a Δt , but dt is the notation used in the python implementation of Code 1).

We recall that $|B(t)|$ depends on $x(t)$ and thus must be estimated in some way (while in quantum computers it is impossible to evaluate it without destroying the quantum state entirely). We call the possible choices on how to estimate it, with the term **modes**. We will discuss them in section 9.2

Now, we calculate $\tilde{x}(t)$, *i.e.* the estimate of x obtained by assuming the **composite state approximation**. The idea is to calculate the **reduced density matrix**

$$\rho_{red} := \text{tr}_{2,3,\dots,n}(\rho),$$

where the trace is over all indices except index 1. If the composite state approximation were true, then we could write this reduced density matrix *exactly* as the density matrix associated to a pure state

$$|\tilde{\phi}\rangle = \begin{pmatrix} \tilde{\phi}_0 \\ \tilde{\phi}_1 \end{pmatrix}$$

but given the fact that it is only an approximation, we have

$$\rho_{red} \approx |\tilde{\phi}\rangle \langle \tilde{\phi}| = \begin{pmatrix} \tilde{\phi}_0\tilde{\phi}_0 & \tilde{\phi}_0\tilde{\phi}_1 \\ \tilde{\phi}_1\tilde{\phi}_0 & \tilde{\phi}_1\tilde{\phi}_1 \end{pmatrix}.$$

Now we recall that the fake Hamiltonian was specifically chosen so that, if the approximation were exact, the single copy would indeed evolve as

$$\begin{pmatrix} x(t) \\ \sqrt{1-x^2(t)} \end{pmatrix}$$

where $x(t)$ follows equation (9.1).

Thus, if the approximation were exact, we would have that the components of $|\tilde{\phi}\rangle$, *i.e.* $\tilde{\phi}_0, \tilde{\phi}_1$, would be equal to $x(t), \sqrt{1-x^2(t)}$ respectively. However the approximation cannot be exact, thus we say that

$$\tilde{\phi}_0 = \tilde{x}; \quad \tilde{\phi}_1 = \sqrt{1 - \tilde{x}^2},$$

where we define $\tilde{x}(t)$ as our *estimate* of $x(t)$, which assumes an approximation we know to be, strictly speaking, false.

As one can write

$$\rho_{red} \approx |\tilde{\phi}\rangle \langle \tilde{\phi}|$$

it follows that the matrix is near to a matrix of rank one, with eigenvalues near 1 and 0 respectively, and eigenvectors near $|\tilde{\phi}\rangle$ and the vector orthogonal to it with the hermitian product. To extract the value of $\tilde{x}$ the easiest way is to find the eigenvector with maximum eigenvalue, eigenvector that we will call v , which we identify with

$$|\tilde{\phi}\rangle = \begin{pmatrix} \tilde{\phi}_0 \\ \tilde{\phi}_1 \end{pmatrix},$$

The calculations are thus the following. We start with

$$|\Psi\rangle = \sum_{k=0}^n \sum_{s=1}^{\binom{n}{k}} \Psi_k(t) |k, n-k; s\rangle.$$

We separate the first copy from the others:

$$|\Psi\rangle = \sum_{k=0}^{n-1} \sum_{s=1}^{\binom{n-1}{k}} \begin{pmatrix} \Psi_{k+1} \\ \Psi_k \end{pmatrix} \otimes |k, n-k-1; s\rangle.$$

We rewrite it as

$$|\Psi\rangle = \sum_e |c\rangle |e\rangle$$

where we adopted a more brief notation for the quantum state of the first copy

$$|c\rangle := \begin{pmatrix} \Psi_{k+1} \\ \Psi_k \end{pmatrix}$$

and for the quantum state of the last $n - 1$ copies

$$|e\rangle := |k, n - k - 1; s\rangle.$$

We notice that the set of $|e\rangle$ vectors is by definition an orthonormal base for the Hilbert space of the last $n - 1$ copies of our quantum system.

We can now calculate the density matrix using its definition:

$$\rho := |\Psi\rangle\langle\Psi| = \sum_e \sum_{e'} |e\rangle\langle e'| |c\rangle\langle c'|.$$

From this we can easily calculate the reduced density matrix ρ_{red} . From its definition we have

$$\rho_{red} := \sum_{e''} \langle e'' | \rho | e'' \rangle,$$

where the $|e''\rangle$ vectors live in the Hilbert space of the last $n - 1$ copies. Concluding the calculations by using the definition of the density matrix and then using the fact that the set of $|e\rangle$ vectors is, as we already said, an orthonormal base for the Hilbert space of the last $n - 1$ copies, we have

$$\rho_{red} := \sum_{e''} \langle e'' | \rho | e'' \rangle = \sum_{e''} \sum_e \sum_{e'} \langle e'' | e \rangle |c\rangle\langle c'| \langle e' | e'' \rangle = \sum_e |c\rangle\langle c|$$

from which we get

$$\rho_{red} = \sum_{k=0}^{n-1} \sum_{s=1}^{\binom{n-1}{k}} M_k = \sum_{k=0}^{n-1} \binom{n-1}{k} M_k$$

where we used the fact that the matrix M_k does not depend on s . We defined the matrix M_k to be

$$M_k := \begin{pmatrix} \Psi_{k+1}\Psi_{k+1} & \Psi_{k+1}\Psi_k \\ \Psi_k\Psi_{k+1} & \Psi_k\Psi_k \end{pmatrix}.$$

Thus as a conclusive formula we have

$$\rho_{red} = \sum_{k=0}^{n-1} \binom{n-1}{k} \begin{pmatrix} \Psi_{k+1}\Psi_{k+1} & \Psi_{k+1}\Psi_k \\ \Psi_k\Psi_{k+1} & \Psi_k\Psi_k \end{pmatrix}.$$

From here we extract the eigenvector v associated with the greatest eigenvalue, λ . As the eigenvector is defined up to a multiplicative constant, it should be taken with norm one and, if possible, with both components real and positive: in the composite state approximation this should be true. From this vector v we can extract $\tilde{x}$ with

$$\tilde{x} = v_0,$$

which concludes our calculations.

This qubit model is used both in Code 1 and Code 2. In Code 1 the $n + 1$ components $\Psi_k(t)$ are indeed all saved and evolved according to (9.17) through finite time steps of dt , and all the calculations above are implemented, up to $\tilde{x}$. We will see later how one can calculate them on the spot with Code 2, without the need to calculate their value at previous times.

Algorithm and implementation of Code 1

Code 1 can be found and run in [this github repository](#).

It is written in Python and uses the python library `python-flint` to treat numbers with arbitrary numerical precision.

We present only the *qubit* variant of the code, and not the *qutrit* variant, which is always worse or exactly equal in results (but always worse in memory). We chose to use $x_0 = 0.6$ as a starting point. We evolved the system up to $T = 1.0$, as it is enough to observe the disagreement between the simulation and the desired solution of Equation 9.1. We tried different values for dt for our simulations: $dt = 0.01$ and $dt = 0.001$. The number of copies for our simulations are chosen to be the powers of two up to 256: we explain why a higher number of copies, though possible (up to a certain point, around 1024), is not really useful. The parameter *dps*, required to fix the number precision of the floats, was chosen to be 16384. Other simulations have been done for different values of these parameters, and we discuss the effect of doing so, but the results shown are representative of the general behavior of our model.

Simulation results of Code 1

The results are shown below. We did two sets of simulations for different values of the time step dt , and for each of them we simulated Code 1 with different numbers of copies n we plotted our estimate $\tilde{x}(t)$ in a graph, together

with the analytical solution of equation (9.1):

$$x = \frac{1}{\sqrt{2t + \frac{1}{x_0^2}}},$$

and together to our **fit of empirical solution**, which is simply a reference curve that we reproduce in all our graphs of Code 1 and Code 2, so that they can be compared and it is easy to see that the results are similar.

Our simulations with time step $dt = 0.01$ are shown in Figure 9.1, Figure 9.2, Figure 9.3, Figure 9.14 where we paid attention to different parts of the graph; while our simulations with a time step $dt = 0.001$ are shown in Figure 9.5, Figure 9.6, Figure 9.7, Figure 9.8.

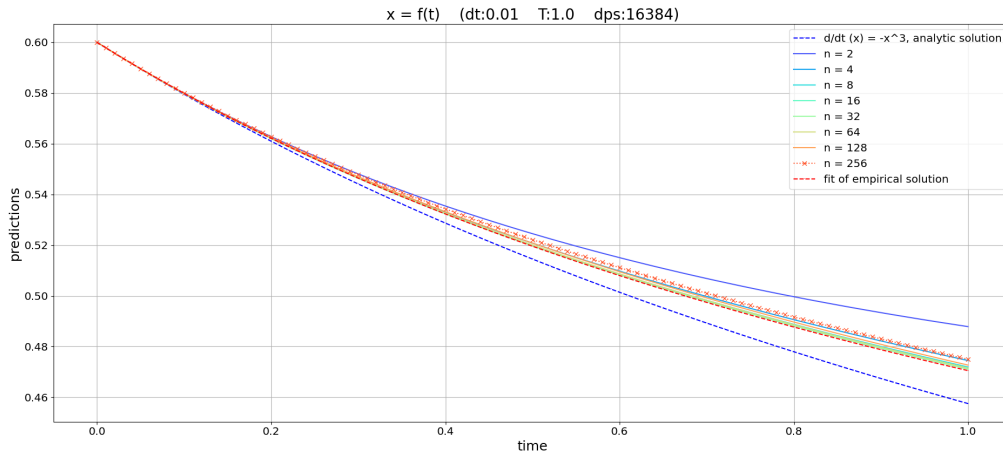


Figure 9.1: Code 1 predictions until $T = 1$ with a time step of $dt = 0.01$.

The main result is that the simulation predictions for higher and higher values of n do indeed seem to converge, but they do **not** converge to the analytical solution of equation (9.1) which is plotted in the graphs as the dashed blue line.

Another important detail is what we called the **bouncing effect**: increasing the number n of copies, the prediction seems to converge until a specific $\bar{n}$, but then for $n > \bar{n}$ it starts again to diverge. Also, for $n < \bar{n}$ the prediction decreases but with smaller and smaller distances. For $n > \bar{n}$ instead, the

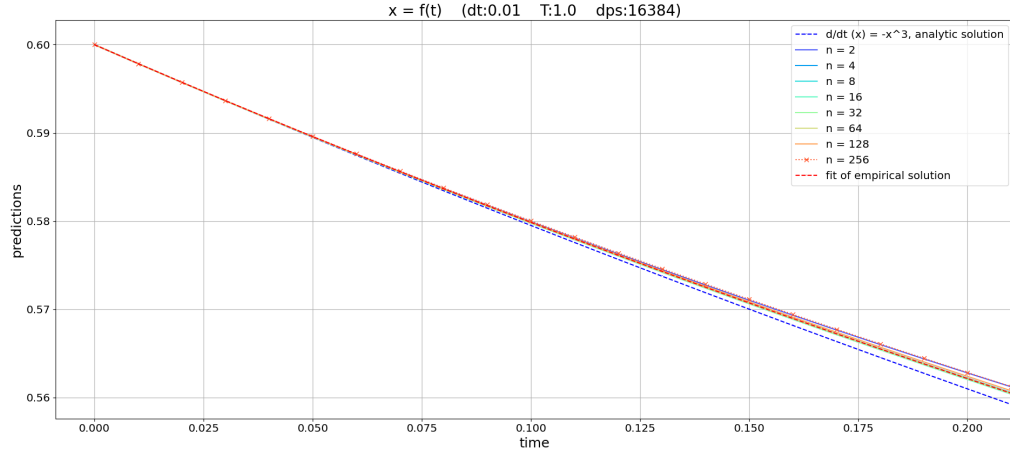


Figure 9.2: Code 1 predictions until $T = 0.2$ with a time step of $dt = 0.01$.

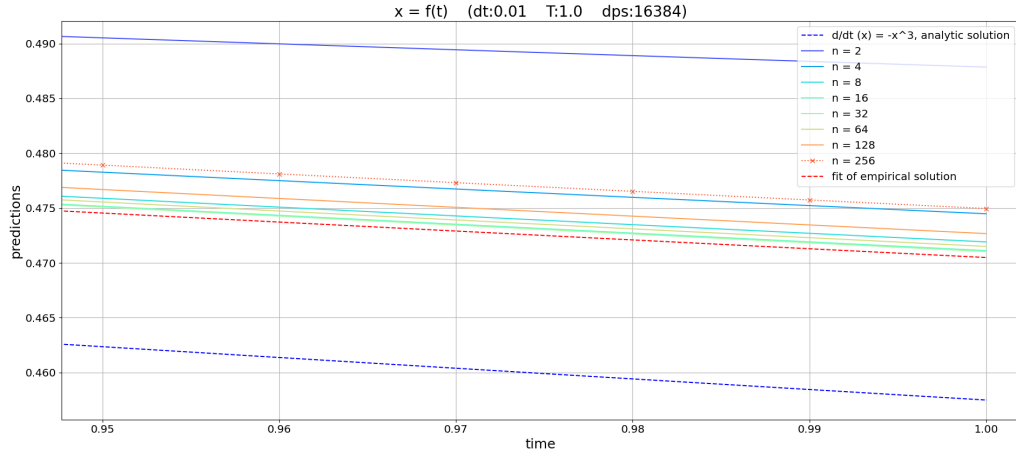


Figure 9.3: Code 1 predictions from $T = 0.95$ to $T = 1.0$ with a time step of $dt = 0.01$.

prediction begins to increase (as if it bounced) with greater and greater distances.

This effect has been found only in the simulations done with Code 1, and is absent in Code 2. The analytical calculations are valid only in the limit

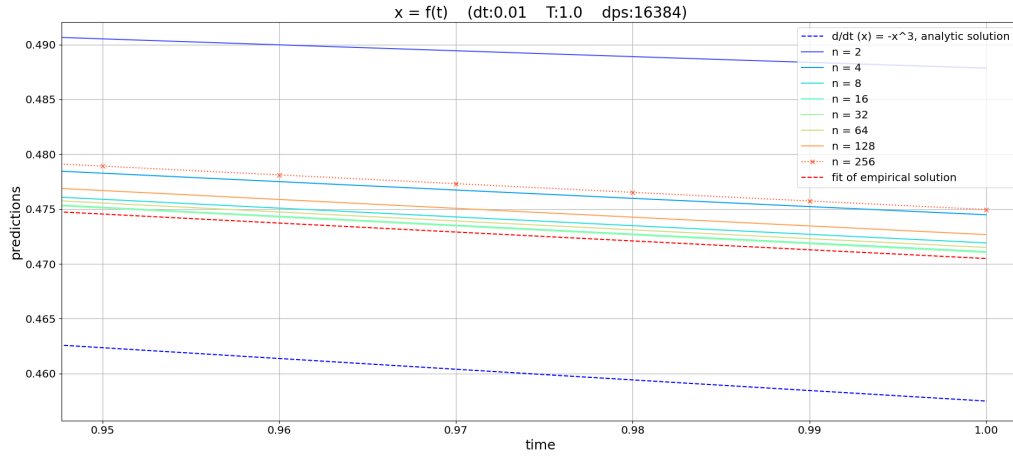


Figure 9.4: Code 1 predictions from $T = 0.99$ to $T = 1.0$ with a time step of $dt = 0.01$. We zoomed in on the high- n simulations to see the bouncing effect.

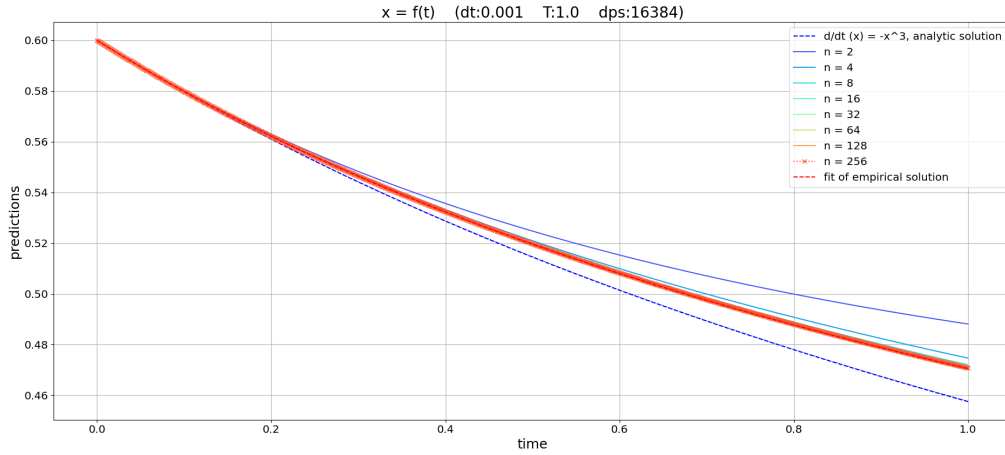


Figure 9.5: Code 1 predictions until $T = 1$ with a smaller time step of $dt = 0.001$.

$n \rightarrow +\infty$, thus the limit is already done. The number $\bar{n}$ is dependent on the dt and the dps variables (we recall that the dps variable controls the numerical precision of the code numbers): $\bar{n}$ can be increased by diminishing dt , which requires however to increase dps also, to allow the code to find the

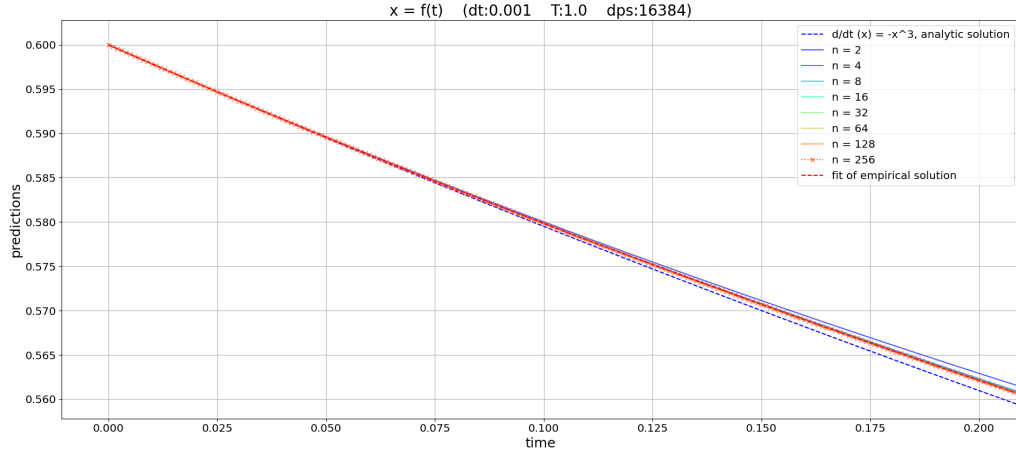


Figure 9.6: Code 1 predictions until $T = 0.2$ with a smaller time step of $dt = 0.001$.

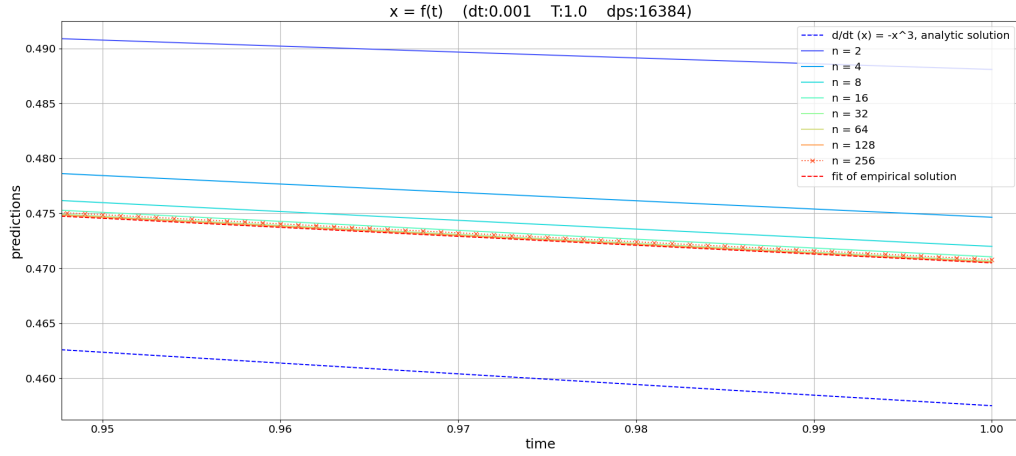


Figure 9.7: Code 1 predictions from $T = 0.95$ to $T = 1$ with a smaller time step of $dt = 0.001$.

correct eigenvalues. Thus the computational cost of doing simulations for high n becomes prohibitive: they are useless if $n > \bar{n}$ (due to the bouncing effect), and to increase $\bar{n}$ it is necessary to increase both the number of steps T/dt and the number precision of all calculations. All simulations done with Code

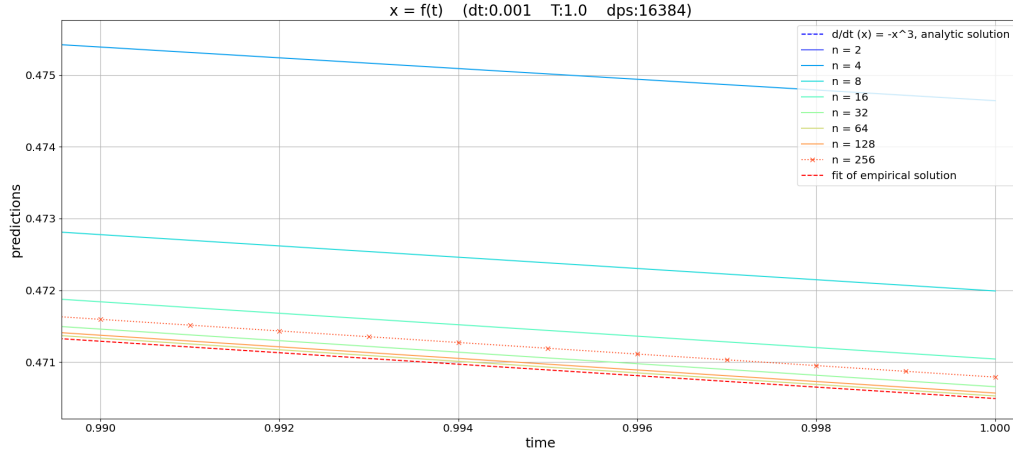


Figure 9.8: Code 1 predictions from $T = 0.99$ to $T = 1.0$ with a smaller time step of $dt = 0.001$. We zoomed in on the high- n simulations to see the bouncing effect, reduced by the choice of a smaller time step.

1 had $n \leq 4096$, and the bouncing effect was reduced to at most $\bar{n} \leq 512$. The error at $t = 1$ between the estimate $\tilde{x}(t)$ and the desired solution x_{sol} is of about 4%.

9.2.2 Code 2

The second code does not simulate the complete quantum state, but it calculates $\tilde{x}(t)$, and possibly other quantities of interest, using the fact that the desired differential equation $\dot{x} = -x^3$ has a solution in which the components of the complete state evolve independently (there are more complex cases of equations we could have chosen in which the components mix, and we briefly touch on the subject in section 9.4).

We start from Equation 9.17:

$$\frac{d}{dt} \Psi_k(t) = k(k-1)|A|\Psi_k(t) + (n-k)(n-k-1)|B(t)|\Psi_k(t),$$

which can be solved by separation of variables (as $\Psi_k(t)$ is just a scalar) and has solution

$$\Psi_k(t) = e^{k(k-1)|A|t + (n-k)(n-k-1) \int^t dt |B(t)|} \Psi_k(0). \quad (9.18)$$

It is crucial for this step that $\Psi_k(t)$ be a scalar, which is caused by the fact that the components do not mix (otherwise we should have written a vectorial differential equations in which the $\Psi_k(t)$ were just entries of the vector).

This solution reveals the first idea of Code 2: to bypass the step by step calculations and directly calculate the result at any desired time t , which is possible thanks to (9.18). To plot $\tilde{x}(t)$ however we will still need to do the calculations for the whole time interval we are interested in. We will also see the second idea of Code 2: it is not needed to save in memory all $n + 1$ components $\Psi_k(t)$ of the second quantization notation, because they are not needed to calculate those at the subsequent time step. It is also possible to calculate them on the fly using (9.18), and use them to calculate quantities of interest such as $\tilde{x}(t)$.

To see how this is possible, we continue with the calculation.

As we have already seen in Code 1, the idea is to calculate the reduced density matrix

$$\rho_{red} = \sum_{k=0}^{n-1} \binom{n-1}{k} \begin{pmatrix} \Psi_{k+1}\Psi_{k+1} & \Psi_{k+1}\Psi_k \\ \Psi_k\Psi_{k+1} & \Psi_k\Psi_k \end{pmatrix}$$

and to calculate the eigenvector v with greatest eigenvalue, whose component with index 0 will give $\tilde{x}$, in the composite state approximation.

Now we do this calculation explicitly. Only for this calculation, we define α, β, γ such that

$$\rho_{red} = \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}.$$

We of course know that the trace is equal to one:

$$1 = \text{tr}(\rho_{red}) = \alpha + \gamma.$$

Even if we didn't show the plots for lack of time, the classical simulations of Code 1 show that for high enough n the purity of ρ_{red} go to 1. This gives the condition

$$1 = \text{tr}(\rho_{red}^2) = \text{tr} \left(\begin{pmatrix} \alpha^2 + \beta^2 & \beta(\alpha + \gamma) \\ \beta(\alpha + \gamma) & \beta^2 + \gamma^2 \end{pmatrix} \right) = \alpha^2 + 2\beta^2 + \gamma^2.$$

We can write now

$$0 = 1 - 1 = \text{tr}(\rho_{red}^2) - \text{tr}^2(\rho_{red}) = 2(\beta^2 - \alpha\gamma).$$

Now that we know that $\gamma/\beta = \beta/\alpha$, is it easy to see the form of v that satisfy the eigenvector equation with eigenvalue 1:

$$\begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix} \begin{pmatrix} 1 \\ \gamma/\beta \end{pmatrix} = \begin{pmatrix} \alpha + \gamma \\ \gamma + \gamma^2/\beta \end{pmatrix} = \begin{pmatrix} 1 \\ \gamma/\beta \end{pmatrix},$$

where in the last step we used $\alpha + \gamma = 1$ and $\beta^2 = \alpha\gamma$. As we know that the eigenvector should be of the form

$$v = \begin{pmatrix} \tilde{x} \\ \sqrt{1 - \tilde{x}^2} \end{pmatrix} \propto \begin{pmatrix} 1 \\ \gamma/\beta \end{pmatrix}.$$

From the previous relations among α, β, γ it follows that $\gamma = 1 - \alpha$ and $\beta = \sqrt{\alpha(1 - \alpha)}$. We thus have

$$\frac{\sqrt{1 - \tilde{x}^2}}{\tilde{x}^2} = \frac{\gamma}{\beta} = \frac{1 - \alpha}{\sqrt{\alpha(1 - \alpha)}} = \frac{\sqrt{1 - \alpha}}{\sqrt{\alpha}}.$$

We thus have

$$\tilde{x} = \sqrt{\alpha}.$$

Sadly, we empirically see that the relationship $\alpha + \gamma = 1$ is not satisfied: the trace of $\rho_{red}(t)$ does not remain equal to 1 in our simulations in general. The reason is that evolution is in general non-unitary for different choices of $\delta_2(t)$, and even though we use different approximations for it, the trace is very sensible to even small errors in our estimate of $\delta_2(t)$. The value of $\text{tr}(\rho_{red})$ is subject to a decay, or to an increase, depending on the choice of $\delta_2(t)$. This will be shown in the simulation results for Θ in section 9.2.2.

Luckily, we empirically see that a great error in this factor does not give a great error in $\tilde{x}$, if we first normalize the reduced density matrix so that it has trace 1. The procedure is the following:

$$\hat{\rho}_{red} = \frac{\rho_{red}(\alpha, \beta, \gamma)}{\alpha + \gamma} = \rho_{red}(\hat{\alpha}, \hat{\beta}, \hat{\gamma}),$$

with

$$\hat{\alpha} := \frac{\alpha}{\alpha + \gamma}; \quad \hat{\beta} := \frac{\beta}{\alpha + \gamma}; \quad \hat{\gamma} := \frac{\gamma}{\alpha + \gamma}.$$

Redoing the calculations above, we obtain

$$\tilde{x} = \sqrt{\hat{\alpha}} = \sqrt{\frac{\alpha}{\alpha + \gamma}}.$$

This is the estimate of $x(t)$ according to Code 2, and as we will see it remarkably agrees with the results from Code 1.

For completeness we recall the definitions of α, β, γ :

$$\begin{cases} \alpha = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_{k+1} \Psi_{k+1} \\ \beta = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_{k+1} \Psi_k \\ \gamma = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_k \Psi_k \\ \Theta := \alpha + \gamma = \sum_{k=0}^n \binom{n}{k} \Psi_k \Psi_k \end{cases} \quad (9.19)$$

where for the last equation we used the the *formulae* for α and γ , we changed the variable k and we used the recursive definition of the binomial coefficients. We also have

$$\tilde{x}^2 = \hat{\alpha} = \frac{\alpha}{\Theta}.$$

Algorithm and implementation of Code 2

Code 2 can be found and run in [this github repository](#).

It is written in Python and uses the python library `python-flint` to treat numbers with arbitrary numerical precision.

We used again a *qubit* code, clearly. The following graphs are for $x_0 = 0.6$, $T = 1.0$, $dt = 0.001$. The parameter *dps*, required to fix the number precision of the floats, was chosen to be 1024: it is enough to perform all calculations correctly. The simulations have been done for different values of these parameters, and we discuss the effect of doing so, but the results shown are representative.

Simulation results of Θ in Code 2

In the previous calculations related to Code 2, we claimed in equation. We will show the result for the **Analytic** mode, for which we have $\delta_2 = \delta_2(x)$, *i.e.* we use the value of x calculated from time to time during the simulation (ignoring the fact that doing so in a quantum computer is impractical for the reasons already discussed).

The results are shown in Figure 9.9 and Figure 9.10, which show the decay of Θ .

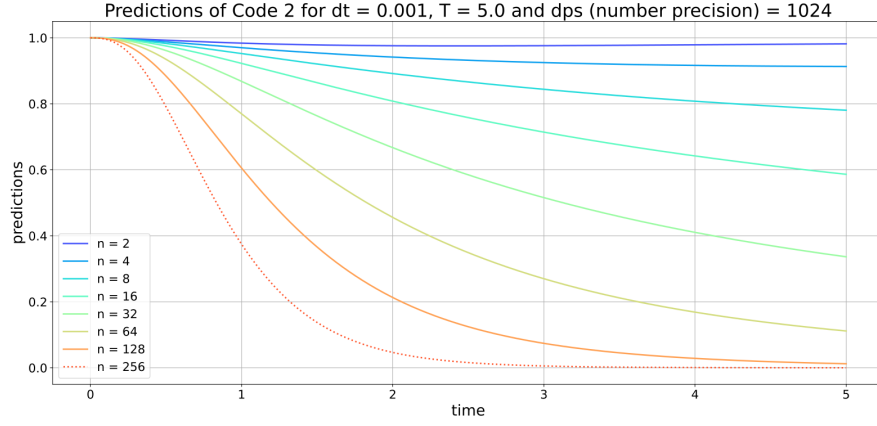


Figure 9.9: Code 2 predictions of Θ until $T = 5$.

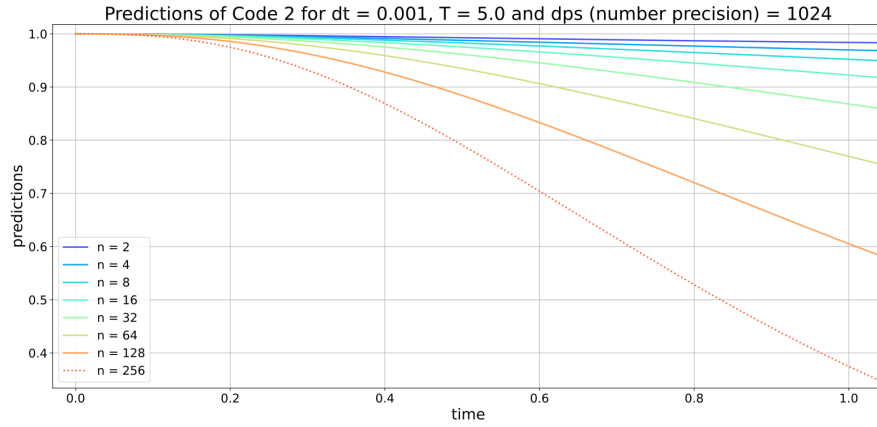


Figure 9.10: Code 2 predictions of Θ until $T = 1$.

Even if it is not plotted, it was found that Θ is very sensible to small errors in the approximation of δ_2 , which may even change the decay to an

increase. This is compatible to the fact that the correct behavior of Θ should be that of a constant equal to 1. We will however notice that the prediction x is not very sensible to variations of δ_2 . Thus the fact that $\Theta \neq 1$ is not fatal to our goal of finding the dynamics of x .

Simulation results of Code 2

The results of the simulations for different number of copies n for the chosen value of the parameters x_0, T, dt, dps , are shown in Figure 9.11, Figure 9.12, Figure 9.13, Figure 9.14.

The results agree with those of Code 1. As a consequence, they do *not* agree with the expected solution y_{sol} . The error at $t = 1$ between the estimate $\tilde{x}(t)$ and the desired solution x_{sol} is still of about 4%, just as in Code 1.

We can see that Code 2 simulations do not present what we called the bouncing effect: they seem to converge nearby the fit of empirical solution, which was nearby the lowest point reached by Code 1 simulations. This is compatible with the fact that both simulations converge to the same function, *different* from x_{sol} .

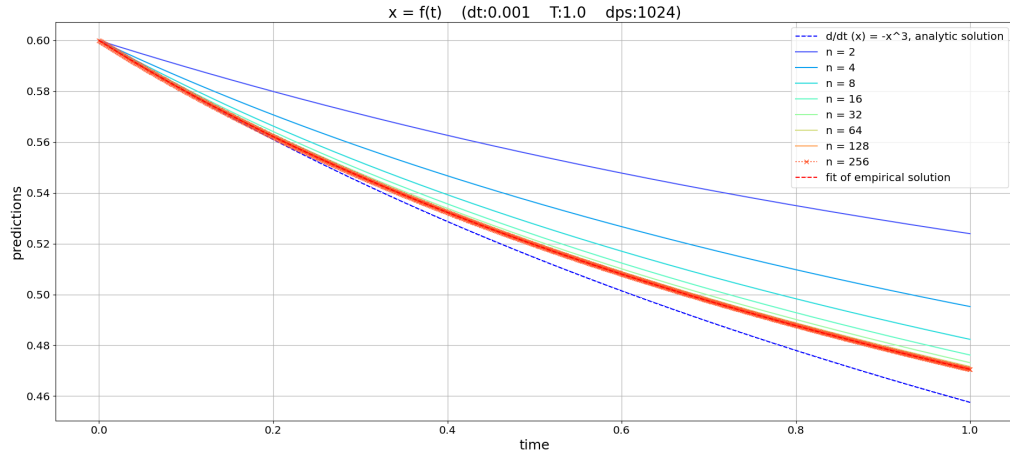


Figure 9.11: Code 2 predictions until $T = 1$ with a time step of $dt = 0.001$.

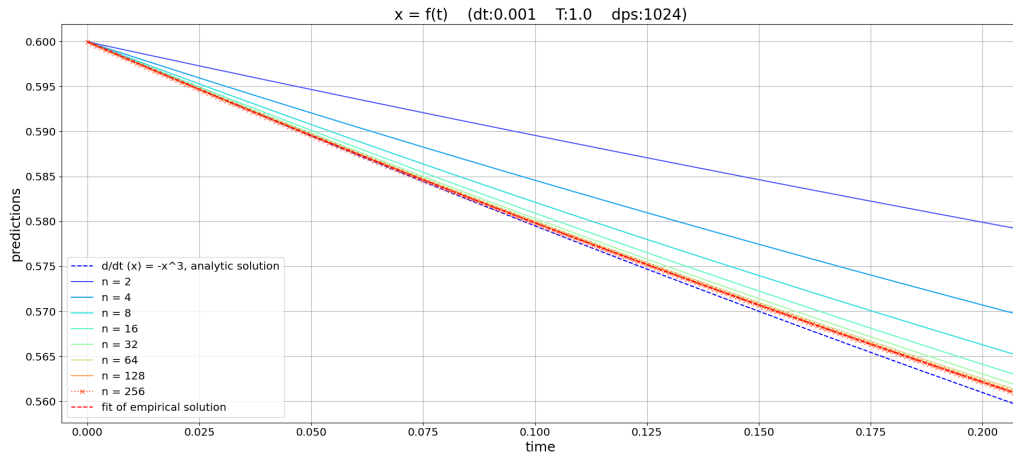


Figure 9.12: Code 2 predictions until $T = 0.2$ with a time step of $dt = 0.001$.

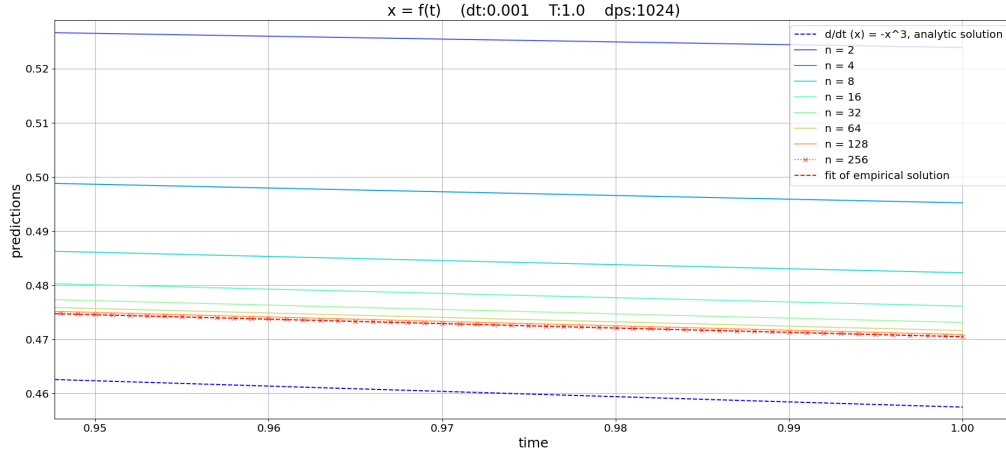


Figure 9.13: Code 2 predictions from $T = 0.95$ to $T = 1.0$ with a time step of $dt = 0.001$.

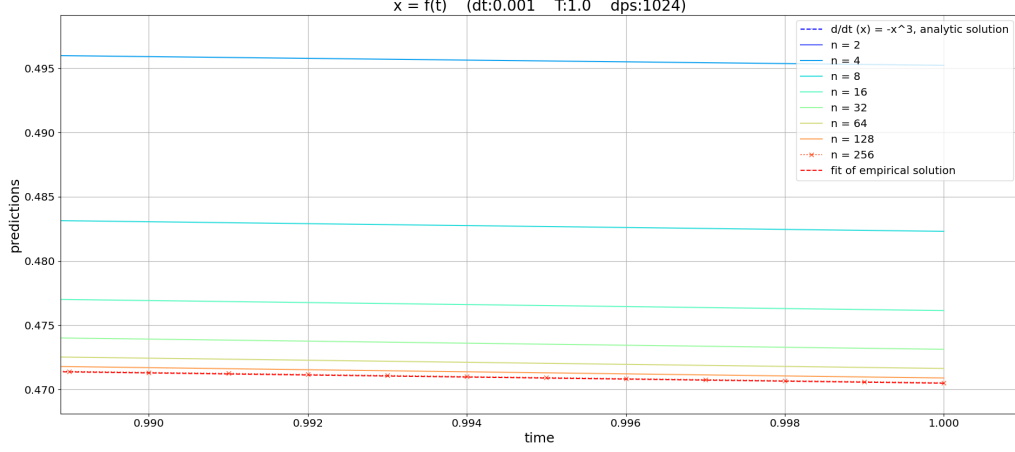


Figure 9.14: Code 2 predictions from $T = 0.99$ to $T = 1.0$ with a time step of $dt = 0.001$. We zoomed in on the high- n simulations to see the convergence.

9.3 Analytical Calculation

We are now ready for the analytical calculation: it starts where the calculations of Code 2 ended. We introduce the change of variables:

$$\delta_1 := n|A|t; \quad \delta_2(t) := n|B(t)|t; \quad (9.20)$$

from which we have

$$\begin{aligned} \Psi_k(t) &= + \left(k(k-1)\delta_1 + (n-k)(n-k-1)\delta_2(t) \right) \Psi_k(0) = \\ &= x_0^k \left(\sqrt{1-x_0^2} \right)^{n-k} \left(k(k-1)\delta_1 + (n-k)(n-k-1)\delta_2(t) \right); \end{aligned} \quad (9.21)$$

where we recall that

$$O(\delta_1) = O(\delta_2) = O(n^0 t^1),$$

and that $\delta(t)$ inherits from $B(t)$ the property of not being directly simulable by quantum computers, and the intrinsic dilemma of how to estimate $x(t)$ at each time to evaluate it correctly.

We also recall the *formulae* for calculating $\tilde{x}$ which we obtained before in equations (9.19), (9.2.2):

$$\tilde{x}^2 = \frac{\alpha}{\Theta},$$

where

$$\begin{cases} \alpha = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_{k+1} \Psi_{k+1} \\ \beta = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_{k+1} \Psi_k \\ \gamma = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_k \Psi_k \\ \Theta := \alpha + \gamma = \sum_{k=0}^n \binom{n}{k} \Psi_k \Psi_k \end{cases} \quad (9.22)$$

Now we are interested in evaluating these sums. In the limit of $n \rightarrow +\infty$, one can approximate the sum with an integral

$$\sum_{k \sim 0}^{\sim n} = n \int_0^1 dy$$

where we changed variables according to

$$k = ny.$$

To obtain $\tilde{x}$ we just need to evaluate α and Θ , even though these techniques can be used also for the other two integrands. We first evaluate Θ , and is to be understood that we will work in the limit $n \rightarrow +\infty$ from now on:

$$\Theta = n \int_0^1 dy \binom{n}{k} \Psi_k(t) \Psi_k(t) = n \int_0^1 dy e^{ng(y)+h(y)},$$

where the logarithm of the integrand has been divided according to the dependence on n . We will look for g and h such that

$$O(g(y)) = O(n^0); \quad O(h(y)) = O(n^0).$$

The idea is to perform a saddle point expansion: the quantity to be integrated is a very peaked function in the limit $n \rightarrow +\infty$, thus it can be expanded around its peak and there it will appear to be a gaussian. The integral of a gaussian can be easily performed, thus giving a value for the integrand which will be exact in the limit of infinite copies.

The calculation is straightforward:

$$\begin{aligned} ng(y)+h(y) &= \log n! - \log k! - \log(n-k)! + \frac{1}{2}k \log(x_0^2) + \frac{1}{2}(n-k) \log(1-x_0^2) \\ &\quad + 2k(k-1)\delta_1 + 2(n-k)(n-k-1)\delta_2. \end{aligned}$$

We use the new variable $y = n/k$ defined above, and we define

$$y_0 = x_0^2; \quad \tilde{y} = \tilde{x}^2;$$

and in general we define the change of variables

$$y = x^2$$

with which the original differential equation $\dot{x} = -x^3$ becomes

$$\dot{y} = -2y^2$$

which has solution

$$y_{sol} = \frac{y_0}{1 + 2y_0 t}.$$

Getting back to our calculations, Stirling's approximation allows us to take all terms proportional to n together to find

$$g(y) = -y \log \frac{y}{y_0} - (1-y) \log \frac{1-y}{1-y_0} + 2y^2 \delta_1 + 2(1-y)^2 \delta_2$$

while the others give

$$h(y) = -\frac{1}{2} \log \left(2\pi n y (1-y) \right) - 2y \delta_1 - 2(1-y) \delta_2 + O\left(\frac{1}{n}\right).$$

The idea is now that, as $n \rightarrow +\infty$, the integrand is a constant multiplied by $e^{-ng(y)}$, which becomes a delta function centered around its maximum $y = y_*$.

The integral can thus be approximated as a gaussian integral:

$$\Theta = n \int_0^1 dy e^{ng(y)+h(y)} = n \int_0^1 dy e^{ng(y_*) + \frac{ng''(y_*)}{2}(y-y_*)^2} e^{h(y)}.$$

The function $e^h(y)$ can be evaluated on $y = y_*$ as it is constant under the action of the rest of the function. Recalling the gaussian integral formula

$$\int_{-\infty}^{+\infty} dx e^{-a(x-x_0)^2} = \sqrt{\frac{\pi}{a}}$$

and extending the bounds of the integral to $+\infty$ and $-\infty$ (as these tails do not contribute, as the argument is concentrated around $y = y_*$), we finally get

$$\Theta = n e^{ng(y_*)+h(y_*)} \sqrt{\frac{2\pi}{-ng''(y_*)}}.$$

Calculating the derivatives of $g(y)$ gives

$$\begin{cases} g(y) = -y \log \frac{y}{y_0} - (1-y) \log \frac{1-y}{1-y_0} + 2y^2\delta_1 + 2(1-y)^2\delta_2 \\ g'(y) = -\log \frac{y}{y_0} + \log \frac{1-y}{1-y_0} + 4y\delta_1 - 4(1-y)\delta_2 \\ g''(y) = -\frac{1}{y} - \frac{1}{1-y} + 4\delta_1 + 4\delta_2 \end{cases} \quad (9.23)$$

from which we easily have

$$\Theta = \frac{e^{ng(y_\star) - 2y\delta_1 - 2(1-y)\delta_2}}{\sqrt{-g''(y_\star)}} \frac{1}{\sqrt{y(1-y)}} e^{O\left(\frac{1}{n}\right)}.$$

The last exponential disappears as $n \rightarrow +\infty$, thus we have

$$\Theta = \frac{e^{ng(y_\star) - 2y\delta_1 - 2(1-y)\delta_2}}{\sqrt{-g''(y_\star)}} \frac{1}{\sqrt{y(1-y)}}.$$

To calculate α there is no need to do more calculations:

$$\alpha = \sum_{k=0}^{n-1} \binom{n-1}{k} \Psi_{k+1} \Psi_{k+1} = \sum_{k=1}^n \binom{n-1}{k-1} \Psi_k \Psi_k = \sum_{k=1}^n \binom{n}{k} \Psi_k \Psi_k \frac{k}{n},$$

where we see that when the sum becomes an integral, the integrand is equal to the one before except for a factor of y , which gives a $y_\star$ in the result. We thus get

$$\alpha = \Theta y_\star.$$

We thus get

$$\tilde{y} := \tilde{x}^2 = \frac{\alpha}{\Theta} = y_\star$$

which means that the expected value of $y = x^2$ such that x satisfies our desired differential equation (9.1), *i.e.* $\tilde{y}$, is simply equal to the value $y_\star$ that maximizes $g(y)$, and which can be found by imposing $g'(y) = 0$. Thus $\tilde{y}$ can be found by imposing

$$g'(y) = 0$$

where for simplicity we write $y = \tilde{y} = y_\star$, now that we found that they coincide. We also have to ask $\Theta = 1$, which, by taking the logarithm, gives two conditions for the part proportional to n and for the other. These

equations are not independent: the error of order $O(\frac{1}{n})$ of the first, appears in the second:

$$\begin{cases} g(y) = \frac{1}{n}r(y) + O(\frac{1}{n^2}) \\ -2y\delta_1 - 2(1-y)\delta_2 - \frac{1}{2}\log(-g''(y) \cdot y(1-y)) + r(y) = 0 \end{cases}$$

where $r(y) = O(\frac{1}{n})$.

Using equation (9.23) for $g''(y)$, the second condition above becomes

$$-2y\delta_1 - 2(1-y)\delta_2 - \frac{1}{2}\log(1 - 4y(1-y)(\delta_1 + \delta_2)) + r(y) = 0$$

which combined with $g'(y) = 0$ gives $r(y) \neq 0$. However it is not necessary for us to solve this condition: we can simply solve

$$g(y) = 0$$

by remembering that it has an error of order $O(\frac{1}{n})$ that allows us to get also

$$\Theta = 1.$$

Unfortunately, as we have already seen in Code 2, the quantity Θ is too sensible to small errors in our estimates of $\delta_2(t)$, thus the condition $\Theta = 1$ will usually not be imposed.

The different possible modes of the calculation, *i.e.* the possible choices of equations to be used, are the following:

$$\begin{cases} g(\tilde{y}) = 0 \\ g'(\tilde{y}) = 0 \end{cases},$$

where it is understood that $\Theta = 1$. The second mode is the *Analytic mode*

$$\begin{cases} \delta_2 = \delta_2(y(t)) \\ g'(y) = 0 \end{cases}$$

which uses the $y(t) = x^2(t)$ where $x(t)$ is the value of $\tilde{x}(t)$ calculated at each time.

We solved with Sage math the first case, which was the easiest to solve.

As we know $\delta_1 = -t$, we obtained two equations for two unknowns: y and δ_2 . The results are:

$$\begin{cases} \delta_2 = \frac{y_0^2}{(y_0-1)^2}t + \frac{4y_0^3}{(y_0-1)^3}t^2 + \frac{8y_0^3(4y_0-3)}{y_0-1} + \frac{16}{3} \frac{y_0^5(14y_0^2-34y_0+25)}{(y_0-1)^2}t^4 + O(t^5) \\ y = y_0 - 2y_0^2t + \frac{2y_0^3(4y_0-3)}{y_0-1}t^2 - \frac{8}{3} \frac{y_0^4(16y_0^2-23y_0+8)}{(y_0-1)^2}t^3 + \\ + \frac{2}{3} \frac{y_0^5(396y_0^3-832y_0^2+568y_0-125)}{(y_0-1)^3}t^4 + O(t^5) \end{cases}$$

The convergence of the Taylor series expansion polynomials of $x = \sqrt{y}$ to all orders up to order 9 is shown in Figure 9.15, Figure 9.16, Figure 9.17. It can be clearly seen that it converges to the fit of empirical data (the red dashed line) instead of converging to the analytical solution x_{sol} . This is compatible to the proposition that the Analytical Calculation is in agreement with Code 1 and Code 2.

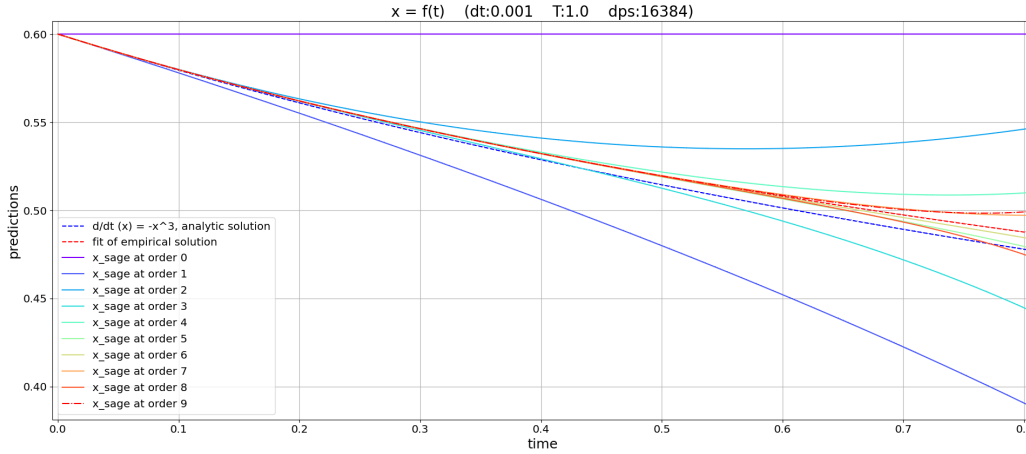


Figure 9.15: Analytical calculation predictions performed with SageMath until $T = 0.8$.

The difference with y_{sol} is in fact already apparent in the Taylor series e at order 2! As we can see

$$y_{sol} = \frac{y_0}{1 + 2y_0t} = y_0 - 2y_0^2t + 4y_0^3t^2 + \dots$$

It can be seen that the quadratic term is already wrong. This shows why it is a problem for $y_0 \rightarrow 1$, while for $y_0 \rightarrow 0$ the error is small; but the series

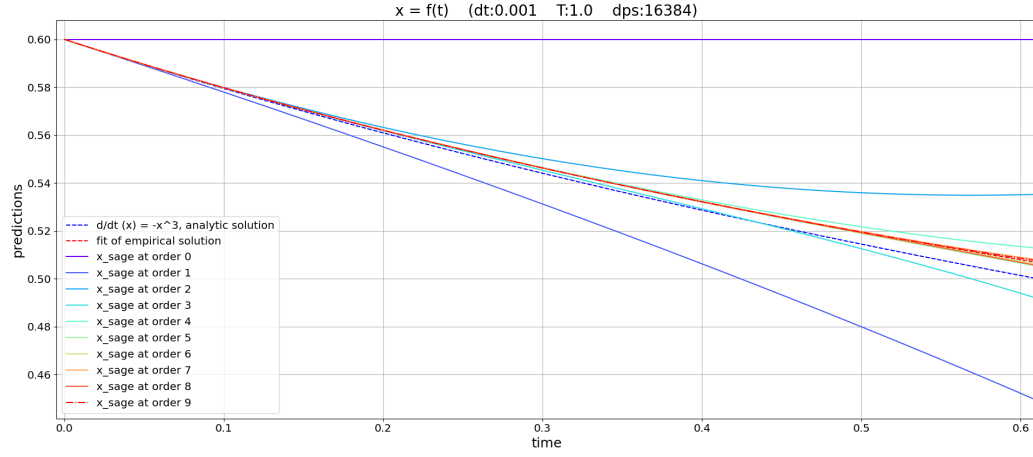


Figure 9.16: Analytical calculation predictions performed with SageMath until $T = 0.6$

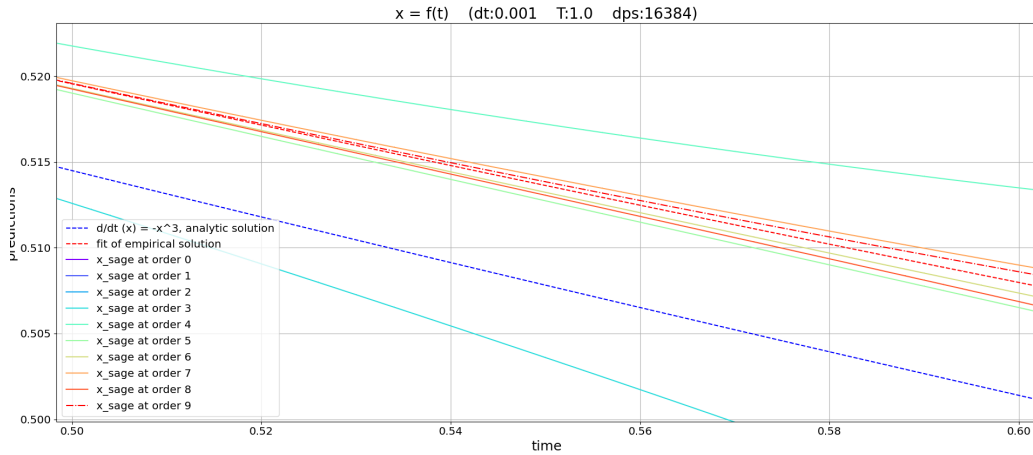


Figure 9.17: Analytical calculation predictions performed with SageMath from $T = 0.5$ to $T = 0.6$.

becomes linear in that limit, so we are just fitting a straight line! This is certainly not a noteworthy accomplishment. Some workarounds have been tried, but with no effect. We also explored

some more general techniques for more general differential equations than the original one (9.1). We very briefly summarize all of these possible extensions in the next section section 9.4.

9.4 Possible extensions of the model

Possible extensions of the model presented above are:

1. using a single copy $|\phi\rangle$ with p components with $p > 2$: this case can be solved quite similarly to the previous ones;
2. using interacting Hamiltonians with higher order, which do not mix different components of the complete state $|\Psi\rangle$;
3. using interacting Hamiltonians which mix different components of the complete state $|\Psi\rangle$: in some particular cases one can recover some integral-differential equations, which are however very complicated to solve;
4. we noticed that if $g(y)$ were different and had the form $g(y) = f(y, y_0) + y\delta_1 + 2(1 - y)\delta_2$, the solution would agree with the expected solution $y_{sol} = y_0 e^{-\alpha t}$. This is however a trivial linear ODE. Also, we could not find a fake Hamiltonian that gives this equation for $g(y)$.

None of these ideas shed light on how to solve a desired nonlinear differential equation with a new normalization.

Chapter 10

Conclusions and perspectives

The quantum algorithm presented in [10] promises to solve *nonlinear* differential equations by encoding the initial variables on multiple copies of a quantum state. We decided to study the quantum algorithm to understand its potentialities and its limitations in a practical setting, understanding what computational resources are necessary to achieve a certain accuracy.

We chose a specific nonlinear equation, $\dot{x} = -x^3$, particularly suitable to be studied with the quantum algorithm. We noticed at the start that it is impossible to follow the normalization procedure suggested in the Supplementary material of the original article of the quantum algorithm. Thus we tried a specific parametrization (a way to encode the variable x in each of the multiple copies) and a specific way to normalize the state, which is impractical on quantum computers, for absence of a better alternative. The specific normalization is realized by adding a time dependent term to the time evolution of the multiple copies: the time dependence has been encoded on a variable called $\delta_2(x(t))$, which depends on the way we chose to estimate $x(t)$: this gives rise to different *modes*.

Then we studied the dynamics with two different numerical codes, available at [this github repository](#), and with an Analytic Calculation done with the saddle point approximation, which is exact in the limit of a large number of copies. For each of these methods, different *modes* were possible.

The results all agree among themselves, and are mostly insensitive to the exact choice of the *mode*; but all these results are very different from the solution x_{sol} of the equation we were trying to solve. They seem to converge in the limit of many copies, *i.e.* $n \rightarrow +\infty$, to some other function, and the difference at time $t = 1$ between this function and x_{sol} is already of 4%. The

second time derivatives of the solution of the actual differential equation and of the output of the algorithm do not agree already at $t = 0$, as can be explicitly seen in the Analytical Calculation.

Further research into possible parametrizations and normalization techniques may allow to find a way to simulate the dynamics of a target nonlinear differential equation known in advance, thus rescuing the main intuition of the quantum algorithm we studied.

Part IV

Appendices

Appendix A

Appendices

A.1 The MNIST dataset

The MNIST [177] is a dataset of labeled images of handwritten digits, widely used to benchmark vision models. Each image has 28×28 pixels. There are 60,000 images in the training set and 10,000 in the test set. We used 50,000 images from the training set, selected with a different seed in each iteration, and we kept the remaining 10,000 images available for potential statistical controls that eventually proved unnecessary. The 10,000 test images are all used and are the same in each iteration.

A.2 Convolutional Neural Networks

We have compared our quantum neural network with a classical CNN with roughly the same number of parameters. This is unusual, as the number of parameters of our quantum neural networks goes from 252 for one expert to 8064 for thirty-two experts, which is at least an order of magnitude smaller than the number of parameters of the typical CNNs employed for MNIST, which is more than 100,000 and usually of the order of millions [134, 135]. In this appendix we will describe how CNNs work in general, and then we will explain how we kept the number of parameters so small in our specific instance.

An image of dimension $n \times n$ with c channels, *i.e.*, c numbers for each pixel ($c = 3$ for RGB images, $c = 1$ for BW images) is embedded into a tensor of size (n, n, c) . Since the MNIST images are black and white, our inputs have

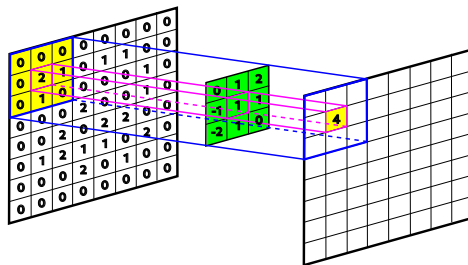


Figure A.1: A convolutional layer with a 3×3 kernel (shown in green). The calculation that gives the result is to multiply each term in the 3×3 kernel (shown in green) with the corresponding term in a 3×3 subset of the original image (the subset is shown in yellow to the left), and then to sum the results and write it in the pixel corresponding to the central pixel of the original subset (the output pixel is shown in yellow to the right).

$c = 1$. In each convolutional layer, the network computes the convolution between a kernel matrix and the image (see Figure A.1): using a 3×3 kernel, a stride of 1 and a padding of 1, the output of the convolutional layer will have the same dimension as the input. If instead of a single kernel we have c kernels (or analogously if the kernel is a tensor of dimension $(3, 3, c)$), we will have an output image of size (n, n, c) . If the starting image has already a nontrivial number of channels, e.g. dimension (n, n, c) , a kernel of dimension $(3, 3, c, c')$ will give, after a contraction in the indices of dimension c , a resulting image of dimension (n, n, c') . This is the most general convolutional layer that we will use. The usual strategy is to alternate convolutional layers with pooling layers, which divide the image into blocks of size 2×2 and replace each block with a single pixel whose intensity is the average intensity of its original components. The pooling layers decrease the resolution of the image, preserving the information at a larger scale. If the input image has size (n, n, c) with n even, the output image has size $(n/2, n/2, c)$. We embed the 28×28 MNIST images in the 32×32 images by adding zeroes at the boundary. Our CNN applies five pooling layers alternated with convolutional layers. Usually after some number of convolutional layers, the $(\bar{n}, \bar{n}, \bar{c})$ output image is flattened to a vector of $\bar{n}^2 \bar{c}$ components, and then a fully connected neural network is used to calculate the output.

Now we focus on our version of the CNN, which we show in Figure A.2. We wanted a small number of parameters. As they are mostly concentrated in the final fully connected NN, we tried to use convolutions to reduce the

#q par.s	#cl par.s	c_1	c_2	c_3	c_4	h	train acc.	test acc.
252	175	2	2	2	4	4	94.74 ± 1.14	95.07 ± 1.72
504	465	2	4	4	4	4	96.91 ± 0.17	97.11 ± 0.83
1008	905	4	4	4	8	8	97.53 ± 0.38	97.72 ± 0.29
2016	1969	8	8	8	8	4	98.85 ± 0.05	98.74 ± 0.05
4032	3969	16	16	8	4	2	99.000 ± 0.004	98.99 ± 0.01
8064	7829	32	16	12	8	8	99.262 ± 0.003	99.04 ± 0.02

Table A.1: Parameters of the classical CNNs: c_1 , c_2 , c_3 , c_4 are the number of channels of the output of the four convolutional layers respectively, while h is the number of neurons of the hidden layer of the fully connected NN at the last step of the CNN. We present train and test accuracy, and the number of parameters next to that of the quantum circuit we want to compare it to.

dimensions of the input of the fully connected network as far as we could. For that reason, we applied four convolutions, interleaved with pooling layers, until our image was of dimension 2×2 with c_4 channels. Then we flattened this tensor to a $4c_4$ vector, and this is the input of the final fully connected NN with hidden layer h and output layer 1. The output is a single number, the sign of which gives the prediction of our CNN binary classifier.

As classical CNNs have almost all parameters concentrated in the fully connected neural network at the end, we pay particular attention to keeping c_4 and h low. The parameters used by our classical CNNs are shown in Table A.1. Figure A.3 shows that our quantum neural networks can achieve a worse test accuracy than the classical CNNs with a similar number of parameters, which is probably due to the more general non-linearity of the classical CNN, while our quantum circuit accuracy is upper bounded by the quadratic classifier accuracy, as shown in section A.5.

A.3 Convolutions with a single 1-qubit gate

We argue that, due to amplitude encoding, a 1-qubit gate in our circuit corresponds to a classical convolution operation. Let us imagine for simplicity an operation on y_4 , the higher resolution pixel on the y coordinate of the image (see Figure 2 of the paper). If we apply to that qubit a 1-qubit gate that rotates $|0\rangle$ and $|1\rangle$, then we are mixing all pixels of the image whose y

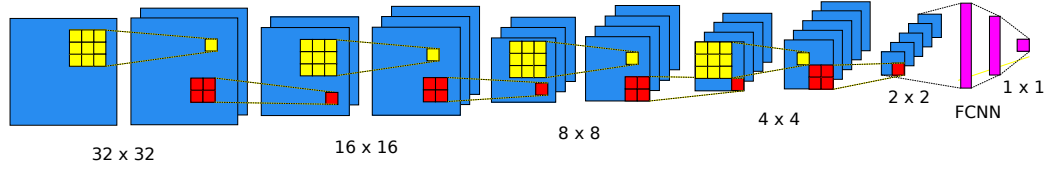


Figure A.2: General structure of our classical Convolutional Neural Network. The image undergoes a series of convolutional layers with a 3×3 kernel matrix (yellow), and pooling layers (red). When a small resolution of 2×2 is achieved, the pixels are flattened and given to a fully connected neural network (purple). To compare the classical CNN with our quantum architecture, the number of parameters of the fully connected NN is kept small.

coordinate ends in 0 with the nearby pixel where it ends in 1. This is in fact a convolution with a 1×2 kernel, whose result is overwritten to the original image. To see it, let us take the general matrix

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

that acts mixing $|s0\rangle$ and $|s1\rangle$ for all bit strings s . This means that the amplitudes c_{s0} and c_{s1} become:

$$c'_{s0} = \alpha c_{s0} + \beta c_{s1}$$

$$c'_{s1} = \gamma c_{s0} + \delta c_{s1}.$$

For a real matrix this corresponds to a convolution by the 1×2 matrix $\begin{pmatrix} \alpha & \beta \end{pmatrix}$ which updates the odd y pixels acting with horizontal stride 2, and $\begin{pmatrix} \gamma & \delta \end{pmatrix}$ which updates on the even y pixels acting with horizontal stride 2 - the stride means simply that we are applying the operation on every other pixel in the horizontal direction.

Classically, this operation would depend on few parameters but would have to be done repeatedly throughout the image. In our quantum circuit, this operation depends on a single gate that, thanks to amplitude encoding, can be applied just once.

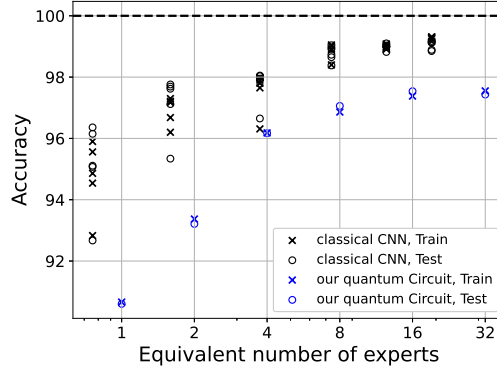


Figure A.3: Comparison between the accuracy of classical and quantum neural networks in the classification of the parity of the MNIST dataset. The equivalent number of experts is simply the number of parameters of a (classical or quantum) model divided by 252, which is the number of parameters of a quantum expert. In this way we can assign a (usually non-integer) equivalent number of expert to our classical circuits to compare them to the quantum ones.

A.4 Extended numerical results

We present here all the data obtained in the simulations. In each epoch, we have computed the **running train accuracy**, *i.e.* the percentage of right predictions on the training images of the selected batches, computed at “run-time” during training when the parameters are evolving. The train accuracy (with the final parameters) is calculated only at the end of training. The **test accuracy** is calculated at the end of each epoch with fixed parameters for all test images. The number of epochs is chosen to saturate the accuracy and varies with the number of experts.

First we show the histograms that represent the distributions of the output of the circuit, given the different train or test images of the dataset. We show it only for the trained circuit with 16 experts. As expected, the outputs of the trained circuit concentrate around $+1$ and -1 for odd and even images, respectively (see Figure A.4). The test accuracy of all simulations as a function of the epoch is shown in Figure A.5. We also present a graph for each quantum neural network we simulated (see Figure A.6), each with a different number of experts, showing the running train accuracy and the test accuracy. We notice that the test accuracy is greater than the running train accuracy. This is due to the fact that the two accuracies are not computed

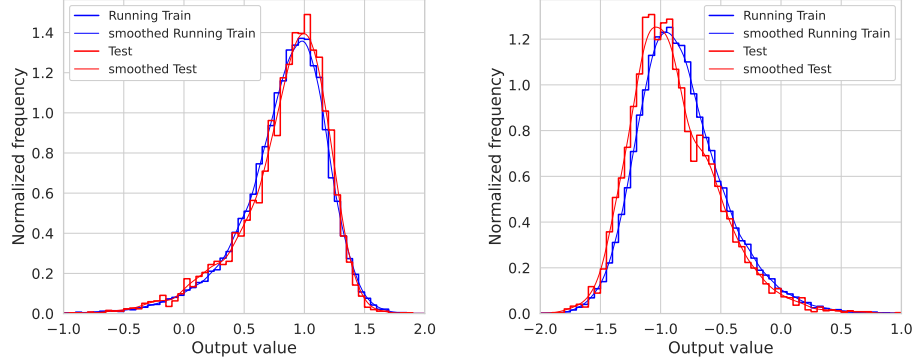


Figure A.4: Histogram of circuit outputs for odd (left) and even (right) images.

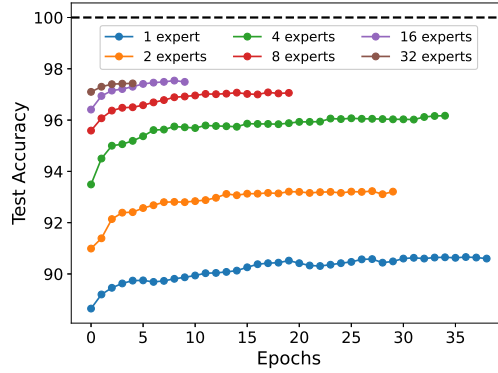


Figure A.5: Test accuracy in function of the epochs. Here we presented the complete data.

with the same parameters, since the running train accuracy is computed during the training while the parameters of the model are changing, while the test accuracy is computed with fixed parameters at the end of each epoch. In fact, we have computed the train accuracy with the parameters at the end of the training, and it turns out to be higher or comparable to the test accuracy as expected (see Table I of the paper).

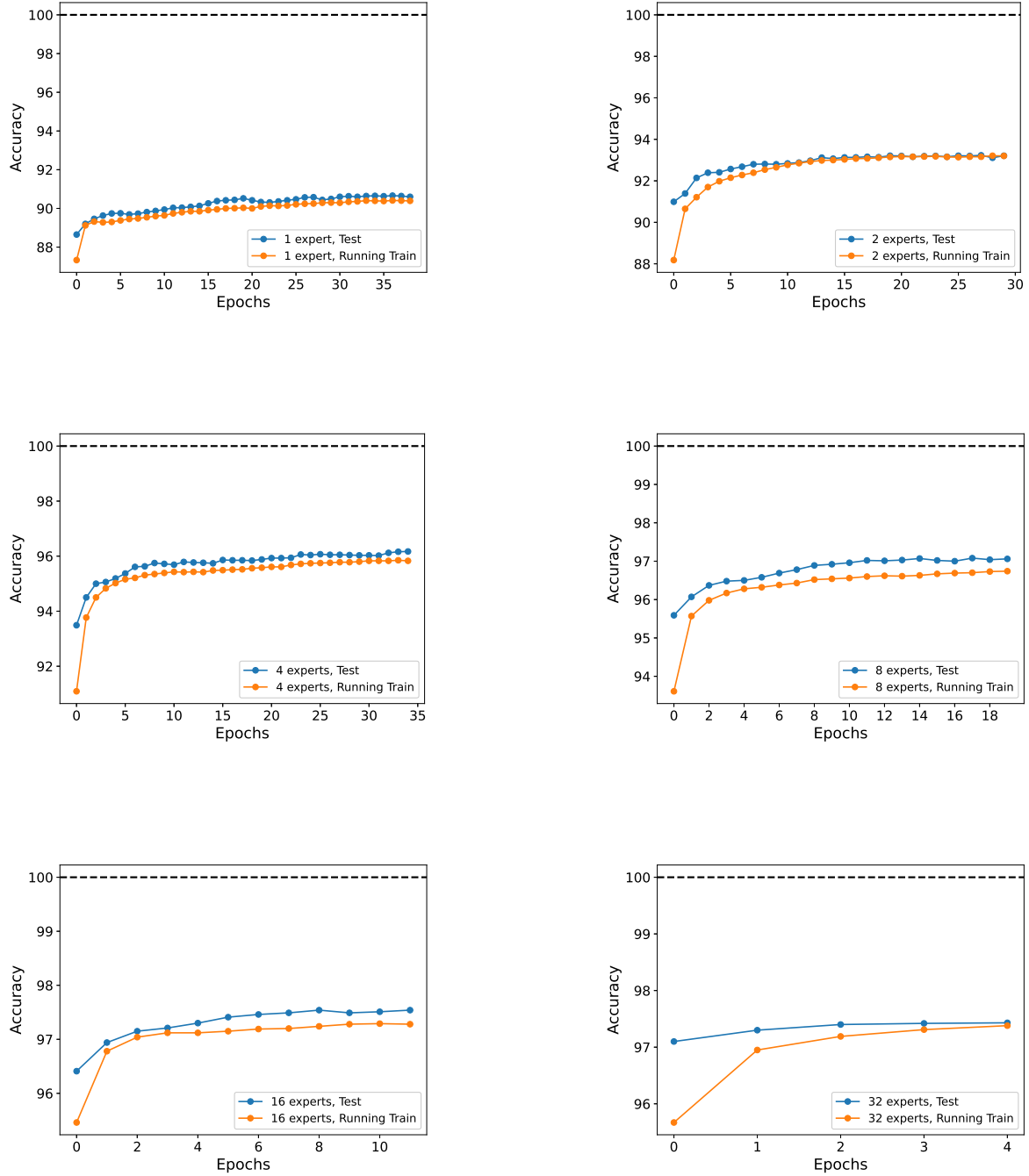


Figure A.6: Running and Test Accuracy using circuits with different experts.

A.5 Classical Quadratic classifier

We notice that the function generated by each expert is quadratic in the wavefunction of the input of the expert, which is the vector of the intensities of the pixels (except for the normalization factor). Therefore, the model function of our quantum neural network is a quadratic polynomial of the vector of the intensities of the pixels. Thus, we compare our results with a **classical quadratic classifier**, which for 784 pixels has $1 + 784 + \binom{784+1}{2} = 308,505$ parameters, and achieves a test accuracy of about 98%, which is right above the maximum accuracy reached by our model (97.5%).

Bibliography

- [1] Yunseok Kwak, Won Joon Yun, Soyi Jung, and Joongheon Kim. Quantum neural networks: Concepts, applications, and challenges, 2021.
- [2] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors, 2018.
- [3] El Amine Cherrat, Iordanis Kerenidis, Natansh Mathur, Jonas Landman, Martin Strahm, and Yun Yvonna Li. Quantum vision transformers. *Quantum*, 8:1265, February 2024.
- [4] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), November 2018.
- [5] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1), March 2021.
- [6] John Napp. Quantifying the barren plateau phenomenon for a model of unstructured variational ansätze, 2022.
- [7] Zoë Holmes, Kunal Sharma, M. Cerezo, and Patrick J. Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1), January 2022.
- [8] Paolo Alessandro Xavier Tognini, Leonardo Banchi, and Giacomo De Palma. Solving mnist with a globally trained mixture of quantum experts, 2025.
- [9] PennyLane. Quantum embedding.

- [10] Seth Lloyd, Giacomo De Palma, Can Gokler, Bobak Kiani, Zi-Wen Liu, Milad Marvian, Felix Tennie, and Tim Palmer. Quantum algorithm for nonlinear differential equations, 2020.
- [11] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), October 2009.
- [12] George Gamow. *Thirty Years that Shook Physics*. Dover Publications, 1966.
- [13] Richard P. Feynman, Robert B. Leighton, Matthew Sands. *The Feynman Lectures on Physics, Volume III, Chapter 1. Quantum Behavior*. Addison-Wesley, 1964.
- [14] Francesco Campaioli, Stefano Gherardini, James Q. Quach, Marco Polini, and Gian Marcello Andolina. Colloquium : Quantum batteries. *Reviews of Modern Physics*, 96(3), July 2024.
- [15] Manoj Mathews. A study on quantum radar technology developments and design consideration for its integration, 2022.
- [16] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, May 1935.
- [17] J. S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics Physique Fizika*, 1:195–200, 1964.
- [18] JOHN S. BELL. On the problem of hidden variables in quantum mechanics. *Rev. Mod. Phys.*, 38:447–452, Jul 1966.
- [19] Richard P. Feynman. Simulating physics with computers.
- [20] Richard P. Feynman. Quantum mechanical computers.
- [21] John Preskill. Quantum computing and the entanglement frontier, 2012.
- [22] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

- [23] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.
- [24] Oded Regev. An efficient quantum factoring algorithm, 2024.
- [25] John Proos and Christof Zalka. Shor’s discrete logarithm quantum algorithm for elliptic curves, 2004.
- [26] Craig Gidney. How to factor 2048 bit rsa integers with less than a million noisy qubits, 2025.
- [27] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.
- [28] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, October 1997.
- [29] C. Figgatt, D. Maslov, K. A. Landsman, N. M. Linke, S. Debnath, and C. Monroe. Complete 3-qubit grover search on a programmable quantum computer. *Nature Communications*, 8(1), December 2017.
- [30] David Jones and Benjamin Varcoe. A method for application of a quantum search algorithm to classical databases, 2022.
- [31] Scott Aaronson. Read the fine print. *Nature Phys.*, 11(4):291–293, 2015.
- [32] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2(4), December 2021.
- [33] Scott Aaronson. How much structure is needed for huge quantum speedups?, 2022.
- [34] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004.
- [35] Simon Forest, David Gosset, Vadym Kliuchnikov, and David McKinnon. Exact synthesis of single-qubit unitaries over clifford-cyclotomic gate sets. *Journal of Mathematical Physics*, 56(8), August 2015.

- [36] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical Review Letters*, 116(25), June 2016.
- [37] Yifan Zhang and Yuxuan Zhang. Classical simulability of quantum circuits with shallow magic depth, 2025.
- [38] Yuzhen Zhang and Yingfei Gu. Quantum magic dynamics in random circuits, 2024.
- [39] Natalie Wolchover. Physicists aim to classify all possible phases of matter. January 2018.
- [40] Tyler D. Ellison, Kohtaro Kato, Zi-Wen Liu, and Timothy H. Hsieh. Symmetry-protected sign problem and magic in quantum phases of matter. *Quantum*, 5:612, December 2021.
- [41] Xhek Turkeshi, Emanuele Tirrito, and Piotr Sierant. Magic spreading in random quantum circuits. *Nature Communications*, 16(1), March 2025.
- [42] Salvatore F. E. Oliviero, Lorenzo Leone, Seth Lloyd, and Alioscia Hamma. Unscrambling quantum information with clifford decoders. *Physical Review Letters*, 132(8), February 2024.
- [43] Christopher David White, ChunJun Cao, and Brian Swingle. Conformal field theories are magical. *Physical Review B*, 103(7), February 2021.
- [44] Qiaofeng Liu, Ian Low, and Zhewei Yin. Quantum magic in quantum electrodynamics, 2025.
- [45] Andi Gu, Salvatore F. E. Oliviero, and Lorenzo Leone. Magic-induced computational separation in entanglement theory, 2024.
- [46] Dongheng Qian and Jing Wang. Quantum nonlocal nonstabilizerness. *Physical Review A*, 111(5), May 2025.
- [47] ChunJun Cao, Gong Cheng, Alioscia Hamma, Lorenzo Leone, William Munizzi, and Salvatore F. E. Oliviero. Gravitational back-reaction is magical. 2025.

- [48] Natalie Wolchover. How space and time could be a quantum error-correcting code, January 2019.
- [49] Kanato Goto, Tomoki Nosaka, and Masahiro Nozaki. Chaos by magic, 2021.
- [50] Lorenzo Leone, Salvatore F. E. Oliviero, You Zhou, and Alioscia Hama. Quantum chaos is quantum. *Quantum*, 5:453, May 2021.
- [51] Leslie G. Valiant. Quantum computers that can be simulated classically in polynomial time. In *Symposium on the Theory of Computing*, 2001.
- [52] Barbara M. Terhal and David P. DiVincenzo. Classical simulation of noninteracting-fermion quantum circuits. *Physical Review A*, 65(3), March 2002.
- [53] Joshua Cudby and Sergii Strelchuk. Gaussian decomposition of magic states for matchgate computations, 2025.
- [54] Beatriz Dias and Robert Koenig. Classical simulation of non-gaussian fermionic circuits. *Quantum*, 8:1350, May 2024.
- [55] Oliver Reardon-Smith, Michał Oszmaniec, and Kamil Korzekwa. Improved simulation of quantum circuits dominated by free fermionic operations. *Quantum*, 8:1549, December 2024.
- [56] Zhi Li and Latham Boyle. The penrose tiling is a quantum error-correcting code, 2024.
- [57] Jonathan Bain. Spacetime as a quantum error-correcting code? *Stud. Hist. Phil. Sci. B*, 71:26–36, 2020.
- [58] Rajeev Acharya, Dmitry A. Abanin, Laleh Aghababaie-Beni, Igor Aleiner, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Nikita Astrakhantsev, Juan Atalaya, Ryan Babbush, Dave Bacon, Brian Ballard, Joseph C. Bardin, Johannes Bausch, Andreas Bengtsson, Alexander Bilmes, Sam Blackwell, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, David A. Browne, Brett Buchea, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Anthony Cabrera, Juan Campero, Hung-Shen Chang, Yu Chen, Zijun Chen,

Ben Chiaro, Desmond Chik, Charina Chou, Jahan Claes, Agnetta Y. Cleland, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Sayan Das, Alex Davies, Laura De Lorenzo, Dripto M. Debroy, Sean Demura, Michel Devoret, Agustin Di Paolo, Paul Donohoe, Ilya Drozdov, Andrew Dunsworth, Clint Earle, Thomas Edlich, Alec Eickbusch, Aviv Moshe Elbag, Mahmoud Elzouka, Catherine Erickson, Lara Faoro, Edward Farhi, Vinicius S. Ferreira, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, Suhas Ganjam, Gonzalo Garcia, Robert Gasca, Élie Genois, William Giang, Craig Gidney, Dar Gilboa, Raja Gosula, Alejandro Grajales Dau, Dietrich Graumann, Alex Greene, Jonathan A. Gross, Steve Habegger, John Hall, Michael C. Hamilton, Monica Hansen, Matthew P. Harrigan, Sean D. Harrington, Francisco J. H. Heras, Stephen Heslin, Paula Heu, Oscar Higgott, Gordon Hill, Jeremy Hilton, George Holland, Sabrina Hong, Hsin-Yuan Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Stephen Jordan, Chaitali Joshi, Pavol Juhas, Dvir Kafri, Hui Kang, Amir H. Karamlou, Kostyantyn Kechedzhi, Julian Kelly, Trupti Khaire, Tanuj Khattar, Mostafa Khezri, Seon Kim, Paul V. Klimov, Andrey R. Klots, Bryce Kobrin, Pushmeet Kohli, Alexander N. Korotkov, Fedor Kostritsa, Robin Kothari, Borislav Kozlovskii, John Mark Kreikebaum, Vladislav D. Kurilovich, Nathan Lacroix, David Landhuis, Tiano Lange-Dei, Brandon W. Langley, Pavel Laptev, Kim-Ming Lau, Loïck Le Guevel, Justin Ledford, Joonho Lee, Kenny Lee, Yuri D. Lensky, Shannon Leon, Brian J. Lester, Wing Yan Li, Yin Li, Alexander T. Lill, Wayne Liu, William P. Livingston, Aditya Locharla, Erik Lucero, Daniel Lundahl, Aaron Lunt, Sid Madhuk, Fionn D. Malone, Ashley Maloney, Salvatore Mandrà, James Manyika, Leigh S. Martin, Orion Martin, Steven Martin, Cameron Maxfield, Jarrod R. McClean, Matt McEwen, Seneca Meeks, Anthony Megrant, Xiao Mi, Kevin C. Miao, Amanda Mieszala, Reza Molavi, Sebastian Molina, Shirin Montazeri, Alexis Morvan, Ramis Movassagh, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Chia-Hung Ni, Murphy Yuezhen Niu, Thomas E. O'Brien, William D. Oliver, Alex Opremcak, Kristoffer Ottosson, Andre Petukhov, Alex Pizzuto, John Platt, Rebecca Potter, Orion Pritchard, Leonid P. Pryadko, Chris Quintana, Ganesh Ramachandran, Matthew J. Reagor, John Red-

- ding, David M. Rhodes, Gabrielle Roberts, Elliott Rosenberg, Emma Rosenfeld, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Andrew W. Senior, Michael J. Shearn, Aaron Shorter, Noah Shutty, Vladimir Shvarts, Shraddha Singh, Volodymyr Sivak, Jindra Skruzny, Spencer Small, Vadim Smelyanskiy, W. Clarke Smith, Rolando D. Somma, Sofia Springer, George Sterling, Doug Strain, Jordan Suchard, Aaron Szasz, Alex Sztein, Douglas Thor, Alfredo Torres, M. Mert Torunbalci, Abeer Vaishnav, Justin Vargas, Sergey Vdovichev, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Steven Waltman, Shannon X. Wang, Brayden Ware, Kate Weber, Travis Weidel, Theodore White, Kristi Wong, Bryan W. K. Woo, Cheng Xing, Z. Jamie Yao, Ping Yeh, Bicheng Ying, Juhwan Yoo, Noureldin Yosri, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Nicholas Zobrist. Quantum error correction below the surface code threshold. *Nature*, 638(8052):920–926, December 2024.
- [59] Ben Brubaker. Quantum computers cross critical error threshold, December 2024.
- [60] Marin Ivezic. 4,099 Qubits: The Myth and Reality of Breaking RSA-2048 with Quantum Computers, 2024.
- [61] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, August 2018.
- [62] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harri-gan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel

- Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019.
- [63] A. P. Lund, Michael J. Bremner, and T. C. Ralph. Quantum sampling problems, bosonsampling and quantum supremacy. *npj Quantum Information*, 3(1), April 2017.
- [64] Aram W. Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203–209, September 2017.
- [65] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstern, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.
- [66] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. Artificial intelligence index report 2025, 2025.
- [67] David Owen. What will ai look like in 2030?, September 2025.
- [68] Andrei Potlogea and Anson Ho. Ai and explosive growth redux, 2025.

- [69] Dario Amodei. Machines of loving grace, October 2024.
- [70] Severin Field. Why do experts disagree on existential risk and p(doom)? a survey of ai experts, 2025.
- [71] Terrence J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences*, 117(48):30033–30038, January 2020.
- [72] Henry W. Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, July 2017.
- [73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [74] Michael Phi. Illustrated guide to lstm’s and gru’s: A step by step explanation, September 2018.
- [75] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 52:99–115, 1990.
- [76] Richard P. Feynman. The perceptron: A perceiving and recognizing automaton.
- [77] Richard Sutton. The bitter lesson, March 2019.
- [78] Patrick Rebentrost and Seth Lloyd. Quantum computational finance: quantum algorithm for portfolio optimization, 2018.
- [79] Romy Müller, Marcel Dürschmidt, Julian Ullrich, Carsten Knoll, Sascha Weber, and Steffen Seitz. Do humans and convolutional neural networks attend to similar areas during scene classification: Effects of task and image type. *Applied Sciences*, 14(6):2648, March 2024.
- [80] Minkyu Choi, Yizhen Zhang, Kuan Han, Xiaokai Wang, and Zhongming Liu. Human eyes inspired recurrent neural networks are more robust against adversarial noises, 2024.

- [81] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2018.
- [82] Ethan Dyer and Guy Gur-Ari. Asymptotics of wide networks from feynman diagrams, 2019.
- [83] Kevin T. Grosvenor and Ro Jefferson. The edge of chaos: quantum field theory and deep neural networks, 2022.
- [84] Ian Banta, Tianji Cai, Nathaniel Craig, and Zhengkang Zhang. Structures of neural network effective theories, 2023.
- [85] Mehmet Demirtas, James Halverson, Anindita Maiti, Matthew D. Schwartz, and Keegan Stoner. Neural network field theories: Non-gaussianity, actions, and locality, 2023.
- [86] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation, 2020.
- [87] Greg Yang. Tensor programs i: Wide feedforward or recurrent neural networks of any architecture are gaussian processes, 2021.
- [88] Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture, 2020.
- [89] Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics, 2021.
- [90] Greg Yang. Tensor programs iii: Neural matrix laws, 2021.
- [91] Greg Yang and Edward J. Hu. Feature learning in infinite-width neural networks, 2022.
- [92] Greg Yang and Etai Littwin. Tensor programs ivb: Adaptive optimization in the infinite-width limit, 2023.
- [93] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022.

- [94] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite-depth neural networks, 2023.
- [95] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, September 2017.
- [96] M. Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J. Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, September 2022.
- [97] Mario Krenn, Jonas Landgraf, Thomas Foesel, and Florian Marquardt. Artificial intelligence and machine learning for quantum technologies. *Physical Review A*, 107(1), January 2023.
- [98] A. Termanova, Ar. Melnikov, E. Mamenchikov, N. Belokonev, S. Dolgov, A. Berezutskii, R. Ellerbrock, C. Mansell, and M. Perelshtein. Tensor quantum programming, 2024.
- [99] Hannah Lange, Anka Van de Walle, Atiye Abedinnia, and Annabelle Bohrdt. From architectures to applications: A review of neural quantum states, 2024.
- [100] Yuan-Hang Zhang and Massimiliano Di Ventra. Transformer quantum state: A multipurpose model for quantum many-body problems. *Physical Review B*, 107(7), February 2023.
- [101] Yangjun Wu, Chu Guo, Yi Fan, Pengyu Zhou, and Honghui Shang. Nnqs-transformer: an efficient and scalable neural network quantum states approach for ab initio quantum chemistry, 2023.
- [102] J. D. Lykken D. K. Kolch-meyer S. I. Davis N. Lauk H. Neven D. Jafferis, A. Zlokapa and M. Spiropulu. Traversable wormhole dynamics on a quantum processor.
- [103] Bryce Kobrin, Thomas Schuster, and Norman Y. Yao. Comment on "traversable wormhole dynamics on a quantum processor", 2023.
- [104] Carlos Outeiral, Martin Strahm, Jiye Shi, Garrett M. Morris, Simon C. Benjamin, and Charlotte M. Deane. The prospects of quantum computing in computational molecular biology. *WIREs Computational Molecular Science*, 11(1), May 2020.

- [105] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), September 2018.
- [106] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), March 2019.
- [107] PennyLane.ai. Parameter-shift Rule, 2025.
- [108] Maria Schuld and Nathan Killoran. Is quantum advantage the right goal for quantum machine learning? *PRX Quantum*, 3(3), July 2022.
- [109] Sohun Thakkar, Skander Kazdaghli, Natansh Mathur, Iordanis Kerenidis, André J. Ferreira-Martins, and Samurai Brito. Improved financial forecasting via quantum machine learning, 2024.
- [110] Abtin Ameri, Erika Ye, Paola Cappellaro, Hari Krovi, and Nuno F. Loureiro. Quantum algorithm for the linear vlasov equation with collisions. *Physical Review A*, 107(6), June 2023.
- [111] I. Joseph, Y. Shi, M. D. Porter, A. R. Castelli, V. I. Geyko, F. R. Graziani, S. B. Libby, and J. L. DuBois. Quantum computing for fusion energy science applications. *Physics of Plasmas*, 30(1), January 2023.
- [112] Ryan Babbush, Jarrod R. McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2(1), March 2021.
- [113] Stephen P. Jordan, Keith S. M. Lee, and John Preskill. Quantum algorithms for fermionic quantum field theories, 2014.
- [114] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems, 2016.
- [115] Annie E. Paine, Vincent E. Elfving, and Oleksandr Kyriienko. Physics-informed quantum machine learning: Solving nonlinear differential equations in latent spaces without costly grid evaluations, 2023.
- [116] Chi-Fang Chen, Hsin-Yuan Huang, John Preskill, and Leo Zhou. Local minima in quantum systems, 2023.

- [117] Chi-Fang Chen, Michael J. Kastoryano, Fernando G. S. L. Brandão, and András Gilyén. Quantum thermal state preparation, 2023.
- [118] Sofiene Jerbi, Casper Gyurik, Simon C. Marshall, Riccardo Molteni, and Vedran Dunjko. Shadows of quantum machine learning. *Nature Communications*, 15(1), July 2024.
- [119] Tensorflow quantum, mnist classification.
- [120] Mnist database, classifiers.
- [121] The path to quantum transformers.
- [122] Dominic Pasquali, Michele Grossi, and Sofia Vallecorsa. Measurements With A Quantum Vision Transformer: A Naive Approach. *EPJ Web Conf.*, 295:12003, 2024.
- [123] Nikhil Khatri, Gabriel Matos, Luuk Coopmans, and Stephen Clark. Quixer: A quantum transformer model, 2024.
- [124] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.
- [125] Naixu Guo, Zhan Yu, Matthew Choi, Aman Agrawal, Kouhei Nakaji, Alán Aspuru-Guzik, and Patrick Rebentrost. Quantum linear algebra is all you need for transformer architectures, 2024.
- [126] Hui Zhang, Qinglin Zhao, Mengchu Zhou, Li Feng, Dusit Niyato, Shenggen Zheng, and Lin Chen. A survey of quantum transformers: Architectures, challenges and outlooks, 2025.
- [127] Aram W. Harrow and Richard A. Low. Random quantum circuits are approximate 2-designs. *Communications in Mathematical Physics*, 291(1):257–302, July 2009.
- [128] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1), March 2021.

- [129] Roeland Wiersema, Cunlu Zhou, Yvette de Sereville, Juan Felipe Carrasquilla, Yong Baek Kim, and Henry Yuen. Exploring entanglement and optimization within the hamiltonian variational ansatz. *PRX Quantum*, 1(2), December 2020.
- [130] John Napp. Quantifying the barren plateau phenomenon for a model of unstructured variational ansätze, 2022.
- [131] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. Entanglement induced barren plateaus, 2021.
- [132] Eric R. Anschuetz and Bobak T. Kiani. Quantum variational algorithms are swamped with traps. *Nature Communications*, 13(1), December 2022.
- [133] See Supplemental Material at URL-will-be-inserted-by-publisher for details of preparation and extended numerical results.
- [134] Sanghyeon An, Minjun Lee, Sanglee Park, Heerin Yang, and Jungmin So. An ensemble of simple convolutional neural network models for mnist digit recognition, 2020.
- [135] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures, 2023.
- [136] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, August 2019.
- [137] Seunghyeok Oh, Jaeho Choi, and Joongheon Kim. A tutorial on quantum convolutional neural networks (qcnn), 2020.
- [138] Iordanis Kerenidis and Alessandro Luongo. Classification of the mnist data set with quantum slow feature analysis. *Physical Review A*, 101(6), June 2020.
- [139] C. M. Wilson, J. S. Otterbach, N. Tezak, R. S. Smith, A. M. Polloreno, Peter J. Karalekas, S. Heidel, M. Sohaib Alam, G. E. Crooks, and M. P. da Silva. Quantum kitchen sinks: An algorithm for machine learning on near-term quantum computers, 2019.

- [140] Po-Wei Huang and Patrick Rebentrost. Post-variational quantum neural networks, 2024.
- [141] Jose I Latorre. Image compression and entanglement. *arXiv preprint quant-ph/0510031*, 2005.
- [142] Marcus Cramer, Martin B Plenio, Steven T Flammia, Rolando Somma, David Gross, Stephen D Bartlett, Olivier Landon-Cardinal, David Poulin, and Yi-Kai Liu. Efficient quantum state tomography. *Nature communications*, 1(1):149, 2010.
- [143] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts, 2024.
- [144] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- [145] Filippo Girardi and Giacomo De Palma. Trained quantum neural networks are gaussian processes. *Communications in Mathematical Physics*, 406(4), April 2025.
- [146] Anderson Melchor Hernandez, Filippo Girardi, Davide Pastorello, and Giacomo De Palma. Quantitative convergence of trained quantum neural networks to a gaussian process, 2024.
- [147] Erfan Abedi, Salman Beigi, and Leila Taghavi. Quantum lazy training. *Quantum*, 7:989, April 2023.
- [148] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O’Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel

- Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. PennyLane: Automatic differentiation of hybrid quantum-classical computations, 2022.
- [149] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [150] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, 2020.
- [151] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification. *CVPR 2012*, p. 3642–3649, 2012.
- [152] Dominic W Berry. High-order quantum algorithm for solving linear differential equations. *Journal of Physics A: Mathematical and Theoretical*, 47(10):105301, February 2014.
- [153] Andrew M. Childs, Robin Kothari, and Rolando D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, January 2017.
- [154] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, page 792–809. IEEE, October 2015.
- [155] Yonghae Lee, Jaewoo Joo, and Soojoon Lee. Hybrid quantum linear equation algorithm and its experimental test on ibm quantum experience. *Scientific Reports*, 9(1), March 2019.
- [156] Antonio Macaluso. *A Novel Framework for Quantum Machine Learning*. PhD thesis, alma, May 2021.

- [157] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. Preconditioned quantum linear system algorithm. *Physical Review Letters*, 110(25), June 2013.
- [158] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, 2013.
- [159] Scott Aaronson. Bqp and the polynomial hierarchy, 2009.
- [160] Scott Aaronson and Andris Ambainis. Forrelation: A problem that optimally separates quantum from classical computing, 2014.
- [161] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of big data, 2015.
- [162] Alexander Schmidhuber, Michele Reilly, Paolo Zanardi, Seth Lloyd, and Aaron Lauda. A quantum algorithm for khovanov homology, 2025.
- [163] Dominic W. Berry, Andrew M. Childs, Aaron Ostrander, and Guoming Wang. Quantum algorithm for linear differential equations with exponentially improved dependence on precision. *Communications in Mathematical Physics*, 356(3):1057–1081, October 2017.
- [164] Bobak Toussi Kiani, Giacomo De Palma, Dirk Englund, William Kaminsky, Milad Marvian, and Seth Lloyd. Quantum advantage for differential equation analysis. *Physical Review A*, 105(2), February 2022.
- [165] Sauro Succi, Wael Itani, Katepalli Sreenivasan, and René Steijl. Quantum computing for fluids: where do we stand?, 2023.
- [166] Jin-Peng Liu, Herman Øie Kolden, Hari K. Krovi, Nuno F. Loureiro, Konstantina Trivisa, and Andrew M. Childs. Efficient quantum algorithm for dissipative nonlinear differential equations. *Proceedings of the National Academy of Sciences*, 118(35), August 2021.
- [167] Claudio Sanavio and Sauro Succi. Lattice boltzmann–carleman quantum algorithm and circuit for fluid flows at moderate reynolds number. *AVS Quantum Science*, 6(2), April 2024.
- [168] C. Sanavio, R. Scatamacchia, C. de Falco, and S. Succi. Three carleman routes to the quantum simulation of classical fluids. *Physics of Fluids*, 36(5), May 2024.

- [169] Claudio Sanavio, Enea Mauri, and Sauro Succi. Carleman-grad approach to the quantum simulation of fluids, 2024.
- [170] Hari Krovi. Improved quantum algorithms for linear and nonlinear differential equations. *Quantum*, 7:913, February 2023.
- [171] Claudio Sanavio, Enea Mauri, and Sauro Succi. Explicit quantum circuit for simulating the advection–diffusion–reaction dynamics. *IEEE Transactions on Quantum Engineering*, 6:1–12, 2025.
- [172] Wael Itani, Katepalli R. Sreenivasan, and Sauro Succi. Quantum algorithm for lattice boltzmann (qalb) simulation of incompressible fluids with a nonlinear collision term, 2023.
- [173] Oleksandr Kyriienko, Annie E. Paine, and Vincent E. Elfving. Solving nonlinear differential equations with differentiable quantum circuits. *Phys. Rev. A*, 103:052416, May 2021.
- [174] Sauro Succi, Wael Itani, Katepalli R. Sreenivasan, and Rene Steijl. Ensemble fluid simulations on quantum computers, 2023.
- [175] Alexis Ralli Sauro Succi Claudio Sanavio, William A. Simon. Carleman-lattice-boltzmann quantum circuit with matrix access oracles, 2025.
- [176] Felix Tennie, Sylvain Laizet, Seth Lloyd, and Luca Magri. Quantum computing for nonlinear differential equations and turbulence, 2024.
- [177] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.