Universitat Politècnica de Catalunya
Universitat Rovira i Virgili
Universitat de Barcelona

Master Thesis

Quantum Machine Learning with Hybrid Quantum-Classical Computations

Author:
Albert Cardenete Massip

Supervisor:
Dr. Artur Garcia-Saez

Director:
Dr. Javier Béjar Alonso

A thesis submitted in fulfillment of the requirements for the degree of Master in Artificial Intelligence

October, 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT ROVIRA I VIRGILI
UNIVERSITAT DE BARCELONA

Abstract

Master in Artificial Intelligence

Quantum Machine Learning with Hybrid Quantum-Classical Computations

by Albert Cardenete Massip

This thesis explores variational quantum algorithms in order to solve optimization combinatorial problems using a meta-learning approach. These variational algorithms are promising due to its capability to be used in the near future in the so-called Noisy Intermediate-Scale Quantum (NISQ) era, in which algorithms with high tolerance to noise could perform better than its classical counterparts. In this approach, a Recurrent Neural Network (RNN) optimizer tries to obtain the best set of parameters of a quantum circuit for a given problem using the Qauntum Approximation Optimization Algorithm (QAOA), with the minimal amount of queries possible to a quantum computer. Thanks to the advances of parameter shift methods to compute the gradients of parametrized quantum circuits, these algorithms could be even trained in real quantum computers without the need of highly demanding simulations. The results presented show that this approach is able to generalize to other problem instances which the model has not seen before during training.

Acknowledgements

I would like to thank to some of the people that I have had the privilege to work with during my time at the University. I would like to give a special thanks to Prof. Verònica Ahufinger and Prof. Jordi Mompart for all the support, learnings and opportunities they gave me. Also, I would like to thank to Prof. Javier Béjar for his supervison of the project.

This thesis would have not been possible without Dr. Artur Garcia, who advised me during the project, and introduced me to the world of Quantum Machine Learning. I am sincerely grateful for all the support from him. Also, I want to thank the rest of the QUANTIC team at the Barcelona Supercomputing Center.

Finally, I would like to thank my family. They are the ones who have supported me in all the aspects of my life during all these years, and the main reason I have become who I am today.

Contents

List of Figures

List of Tables

1 Introduction

In the past few years, there has been a rise of quantum computing popularity now that we are closer to develop noisy intermediate-scale quantum (NISQ) devices [20]. With the advent of these devices, a large number of applications have been developed which could be suitable for such devices. The recent achievement of the quantum supremacy [2] has marked a milestone in the field, in which finally a quantum computer has been able to solve a problem that could not be solved using a classical computer.

Quantum computing has shown great potential in many fields, from materials development [7] to decryption of current security with Shor's algorithm [26]. This is thanks to the quantum speedup that has been found in many algorithms in comparison to its classical counterparts.

At the same time, the field of Quantum Machine Learning (QML) has emerged at the intersection between the machine learning and quantum computer fields. There have been two main streams of work in this field. On the one hand, we could use these kinds of algorithms to exploit different tasks for which we only have quantum data, and as such a purely quantum-based algorithm would be the best option to work with. On the other hand, we could also work with classical data, but trying to achieve significant speedups with the use of quantum hardware.

There are several examples of promising quantum machine learning algorithms [4]. Quantum speedups can be achieved in a wide range of algorithms that are very popular in the machine learning field, such as principal component analysis (PCA) [16], support vector machines (SVM) [21], and least squares fitting [31], among others.

A particularly promising category of problems that can be solved using these near-term devices are the so-called quantum variational algorithms [19], which consists of the optimization of a parametrized quantum circuit using classical methods. These algorithms present the advantage that they are robust to noise, and are not very complex in terms of depth of the quantum circuit.

The optimization of the parameters of these algorithms does not differ from the current methods used in the machine learning field when trying to optimize models with a large number of parameters, such as deep neural networks.

There are also more challenges for these types of algorithms. On the one hand, the stochastic nature of quantum computing adds complexity for present classical optimizers, as we need many queries to the circuit to get reliable results. This is even more challenging when the optimization technique used requires the computation of gradients with respect to its parameters. Recent works have proposed the use of analytical gradients to tackle this issue [25]. However, this solution is also very expensive for cases in which we have a large number of parameters, and further advances of these techniques are being developed.

On the other hand, these algorithms heavily rely on the initial set of parameters chosen. Many developments have focused on trying to get good initial values through the development of clever heuristics. Regarding this problem, different approaches have been proposed such as the use of Reinforcement learning to automatically set the parameters [12], or the use of meta-learning techniques with Recurrent Neural Networks [28].

In this work, we are going to focus on the second problem of parameter initialization through the use of meta-learning techniques, which was proposed for general black-box computations [1], and has shown great potential across many different applications.

The structure of the work is divided into two parts. In the first one, we are going to explore the theory of the variational quantum algorithms, with more emphasis on the QAOA algorithm. With these algorithms, we will explore the Max-Cut problem and we will solve it using the current techniques for this problem.

The second part will focus on the meta-learning approach to tackle the parameter initialization problem. Here the theory of the analytical quantum gradients will be developed, which also opens the door to more efficient parameter optimization.

2 Variational Quantum Algorithms

Variational Quantum Algorithms are comprised of an iterative quantum-classical optimization loop between a classical processing unit (CPU) and a quantum processing unit (QPU) that aim to produce approximate solutions to a given problem by making use of an *ansatz*. A general schema of these type of algorithms can be seen in the figure 2.1. These algorithms are based on the implementation of parametrized quantum circuits, in which some of the gates have parameters that allows us to fine-tune their effect.

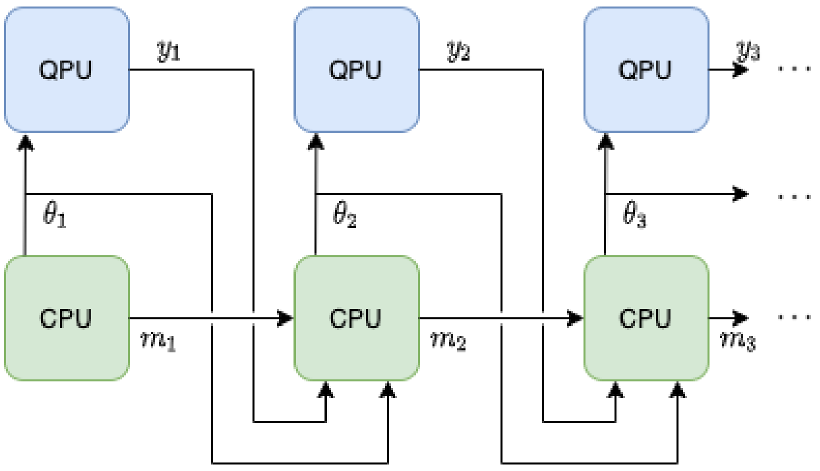


FIGURE 2.1: Hybrid Quantum-Classical graph representing a general Variational Quantum Algorithm. At each iteration of the process t, the CPU returns some candidate parameters θ_t, which are used by the QPU as the parameters for the quantum circuit. With these parameters, the quantum circuit is evaluated and outputs the expected value of a given Hamiltonian H, $y_t = \langle H \rangle_\theta$. At the next step, the CPU can take into consideration the previous proposed states and result of the QPU, as well as its own internal memory m_t.

An iteration begins with the CPU sending the set of candidate parameters $\boldsymbol{\theta}$ to the QPU. The QPU then executes a parametrized circuit $\hat{U}(\boldsymbol{\theta})$, which outputs a state:

$$|\psi_{\boldsymbol{\theta}}\rangle = \hat{U}(\boldsymbol{\theta})|\psi_0\rangle. \tag{2.1}$$

Once we have the prepared state, we are interested of measuring the expected value of a certain Hamiltonian H:

$$f(\boldsymbol{\theta}) = \langle\psi_{\boldsymbol{\theta}}|H|\psi_{\boldsymbol{\theta}}\rangle, \tag{2.2}$$

where $f(\boldsymbol{\theta})$ can be seen as a cost function to be optimized. The expected value of the Hamiltonian has to be estimated by repeating the measurement many times, as in general the output of the measurement will not always be the same. From an optimization point of view, we can think on this problem as one in which we have a black-box function $f : \mathbb{R}^n \to \mathbb{R}$, in which we try to find the parameters $\boldsymbol{\theta}^*$ which minimizes the cost:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^n} f(\boldsymbol{\theta}). \tag{2.3}$$

Finding minimums, or approximate minimums rapidly for these algorithms is quite challenging, as the cost function is stochastic in general. Also, another challenge is that even for perfectly designed quantum gates, there is an inherit noise when performing experiments with real quantum hardware [17].

There have been several successful variational quantum algorithms developed during the last few years. Some of them are intended to be used to approximately solve combinatorial optimization problems (QAOA, [10]), or to approximate the lowest energy level of a givien hamiltonian to solve chemistry problems (VQE, [19]) and even to solve huge linear systems (VQLS, [6]).

2.1 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) [10] is an algorithm that produces approximate solutions for combinatorial optimization problems. The algorithm depends on an integer $p \geq 1$ and the quality of the approximation can improve as p is increased. The quantum circuit that implements the algorithm consists of unitary gates, $\hat{U}$ such that $\hat{U}^\dagger\hat{U} = \hat{I}$, whose locality is at most the locality of the objective function whose optimum is sought. The depth of the circuit grows linearly with p times the number of constraints.

Formulation

consider a combinatorial problem specified by n bits and m clauses. Each clause is a constraint on a subset of the bits which is satisfied for certain assignments of those bits and unsatisfied for the other assignments. The objective function, defined on n bit strings, is the number of satisfied clauses,

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z) \tag{2.4}$$

where,

$$C_\alpha(z) = \begin{cases} 1 & \text{if } z \text{ satisfies a clause,} \\ 0 & \text{otherwise.} \end{cases}$$

And $z = z_1 \ldots z_n$ is the bit string.

These type of problems can be tackled in different ways. Satisfability asks if there is a string that satisfies every clause. MaxSat asks for a string that maximizes the objective function. In the case of QAOA, an approximate optimization asks for a string z for which $C(z)$ is close to the maximum of C. In other words, it tries to find the best possible answer for the combinatorial optimization problem.

We noe define a unitary operator $U(C, \gamma)$ which depends on an angle γ:

$$U(C, \gamma) = e^{-i\gamma C}, \tag{2.5}$$

where in the case in which C is diagonal in the computational basis, we can transform it into:

$$U(C,\gamma) = e^{-i\gamma\sum_{\alpha=1}^{m} C_\alpha} = \prod_{\alpha=1}^{m} e^{-i\gamma C_\alpha}, \tag{2.6}$$

where in the last equality we have used the Baker–Campbell–Hausdorff formula [22] and the fact that every element in $\{C_\alpha\}_\alpha$ commutes with the rest.

Now we define another operator B which is the sum of all single bit σ^x operators:

$$B = \sum_{j=1}^{n} \sigma_j^x. \tag{2.7}$$

Again, we can define a new unitary operator by taking the exponential of this operator and considering a parameter β:

$$U(B,\beta) = e^{-i\beta B}. \tag{2.8}$$

Using again the Baker–Campbell–Hausdorff formula and the commutation rule of the Pauli matrices in $SU(2)$:

$$[\sigma_i, \sigma_j] = 2i\varepsilon_{ijk}\sigma_k, \tag{2.9}$$

we can transform the operator into:

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}. \tag{2.10}$$

Now we consider an initial state $|s\rangle$, which will be a uniform superposition over the computational basis state:

$$|s\rangle = \frac{1}{\sqrt{2^n}}\sum^{z}|z\rangle. \tag{2.11}$$

For any integer p and $2p$ parameters $\gamma_1 \dots \gamma_p \equiv \boldsymbol{\gamma}$ and $\beta_1 \dots \beta_p \equiv \boldsymbol{\beta}$ we define the following angle dependent quantum state:

$$|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle = U(B,\beta_p)U(C,\gamma_p)\dots U(B,\beta_1)U(C,\gamma_1)|s\rangle. \tag{2.12}$$

This state can be produced by a quantum circuit of depth at most $mp + p$. Let F_p be the expectation of C in this state:

$$F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) = \langle\boldsymbol{\gamma},\boldsymbol{\beta}|C|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle, \tag{2.13}$$

and let M_p be the maximum of F_p over the possible parameters:

$$M_p = \max_{\boldsymbol{\gamma},\boldsymbol{\beta}} F_p(\boldsymbol{\gamma},\boldsymbol{\beta}). \tag{2.14}$$

From here we can notice that the maximization at $p-1$ can be viewed as a constrained maximization at p, for instance by taking the parameters $\beta_p = \gamma_p = 0$, so:

$$M_p \geq M_{p-1}. \tag{2.15}$$

These results suggest the QAOA algorithm. Pick up a p and start with a set of angles $(\boldsymbol{\gamma},\boldsymbol{\beta})$ and somehow make F_p as large as possible. Use the quantum computer to get the state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$. Then measure in the computational basis to get a string z and evaluate $C(z)$. Enough repetitions of this procedure will give a string z with $C(z)$ very near or greater than $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$. The problem here is that it is no obvious in advance how to pick good parameters.

Now that we have the general formulation of the algorithm, we will use it to explore different problems

Max-Cut problem

We will start by exploring a classical combinatorial optimization problem that tires to find the maximum cut for a given graph. Given a graph $G = (\mathcal{V}, E)$ with a set of vertices V, and edges E, a cut is a partition of the set $\mathcal{V}$ into two disjoint sets $\mathcal{V} = S \cup T$. The size of the cut is the number of edges such that one of its vertices is in the set S, while the other is in the set T. The maximum cut then the one with maximal size.

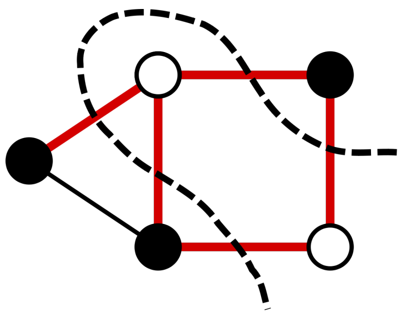


FIGURE 2.2: Representation of the max-cut problem. For a given graph, we want to fins the two disjoint sets (here represented as the black and white sets) of nodes which generates the maximum possible cut.

This is an example of an NP-hard problem, and thus there is no classical algorithm that is able to solve this general problem efficiently. For small instances of the problem, solutions can be found by brute force. However, for larger problems we have to make use of heuristics and give approximate results.

Now we are going to formulate this problem in terms of the QAOA formulation. To translate this problem into a quantum algorithm, we can assign each vertex $i \in \mathcal{V}$ to a qubit in the circuit. Then, we can represent the state of each qubit $|a\rangle_i, a \in \{0, 1\}$ as being in S or T depending on its value. This representation defines the partition we are looking for.

Once we have a given state, we can write the cost function for the max-cut problem as:

$$C = \frac{1}{2} \sum_{\langle j,k \rangle} (1 - \sigma_j^z \sigma_k^z), \tag{2.16}$$

where we are summing across all the edges $\langle j, k\rangle \in E$, and σ_j^z represents the one qubit Pauli-Z quantum gate applied on the j-th qubit. with this representation, each term of the sum will be 2 if and only if the qubit j belongs to one partition and the qubit

k belongs to the other one, otherwise the term is going to be zero.

Using the equation (2.6), we will have to implement the following unitary transormation for the QAOA algorithm:

$$U(C,\gamma) = \prod_{\langle j,k\rangle} e^{-i\gamma\frac{1-\sigma_j^z\sigma_k^z}{2}}. \tag{2.17}$$

The question now is how we can implement this equation into a quantum circuit. Considering the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, the factor for a given edge $\langle j,k\rangle$ is:

$$\begin{aligned} e^{-i\frac{1-\sigma_j^z\sigma_k^z}{2}} &= \exp\left[\frac{-i\gamma}{2}\begin{pmatrix}1&0\\0&1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&1\end{pmatrix}+\frac{i\gamma}{2}\begin{pmatrix}1&0\\0&-1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&-1\end{pmatrix}\right] \\ &= \exp\left[\frac{-i\gamma}{2}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}+\frac{i\gamma}{2}\begin{pmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{pmatrix}\right] \\ &= \exp\left[\begin{pmatrix}0&0&0&0\\0&-i\gamma&0&0\\0&0&-i\gamma&0\\0&0&0&0\end{pmatrix}\right] \\ &= \begin{pmatrix}1&0&0&0\\0&e^{-i\gamma}&0&0\\0&0&e^{-i\gamma}&0\\0&0&0&1\end{pmatrix} \end{aligned} \tag{2.18}$$

Now we would like to get the same result using elementary quantum gates. Considering the same computational basis, the $CNOT$ gate is the one that flips the second qubit if and only if the first qubit is 1. Its matrix representation is the following:

$$CNOT = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}, \tag{2.19}$$

in additon, we can consider the single qubit gate $R_z(\alpha)$, which rotates the state along the z axis. In the computational basis $\{|0\rangle, |1\rangle\}$, this gate has the following matrix form:

$$R_z(\alpha) = \begin{pmatrix}e^{\frac{-i\alpha}{2}}&0\\0&e^{\frac{i\alpha}{2}}\end{pmatrix}. \tag{2.20}$$

With this two quantum gates, we are able to implement the previous exponential function in the following way:

$$\begin{aligned}
CNOT \circ \hat{I} \otimes R_z(\gamma) \circ CNOT &= CNOT \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} e^{\frac{-i\gamma}{2}} & 0 \\ 0 & e^{\frac{i\gamma}{2}} \end{pmatrix} CNOT \\
&= CNOT \begin{pmatrix} e^{\frac{-i\gamma}{2}} & 0 & 0 & 0 \\ 0 & e^{\frac{i\gamma}{2}} & 0 & 0 \\ 0 & 0 & e^{\frac{-i\gamma}{2}} & 0 \\ 0 & 0 & 0 & e^{\frac{i\gamma}{2}} \end{pmatrix} CNOT \\
&= CNOT \begin{pmatrix} e^{\frac{-i\gamma}{2}} & 0 & 0 & 0 \\ 0 & e^{\frac{i\gamma}{2}} & 0 & 0 \\ 0 & 0 & 0 & e^{\frac{-i\gamma}{2}} \\ 0 & 0 & e^{\frac{i\gamma}{2}} & 0 \end{pmatrix} \\
&= \begin{pmatrix} e^{\frac{-i\gamma}{2}} & 0 & 0 & 0 \\ 0 & e^{\frac{i\gamma}{2}} & 0 & 0 \\ 0 & 0 & e^{\frac{i\gamma}{2}} & 0 \\ 0 & 0 & 0 & e^{\frac{-i\gamma}{2}} \end{pmatrix} \\
&= e^{\frac{-i\gamma}{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\gamma} & 0 & 0 \\ 0 & 0 & e^{i\gamma} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (2.21)
\end{aligned}$$

This concatenation of gates generates the same operator, with the addition of a global phase that will not affect the final measurement. By sequentially adding these gates for each pair of connected nodes in the graph, we can finally implement the operator in (2.17) using elementary quantum gates. A diagram of the implementation for a given factor can be seen in the figure 2.3.

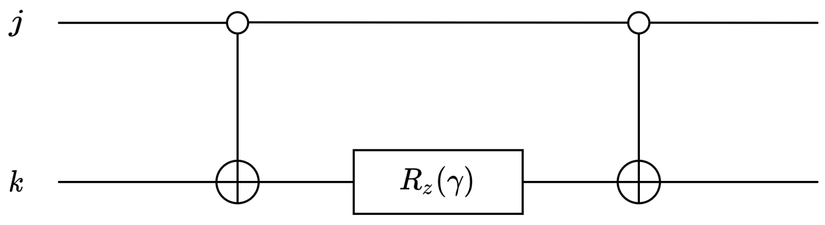


FIGURE 2.3: Implementation of a quantum circuit of the cost operator (2.6) factor between the nodes j and k.

The implementation of the operator (2.10) is easier, as the exponential of the Pauli matrix is a rotation in the same axis. Also, in order to generate the initial state (2.11) we have to apply Hadamard gates to all the qubits from an initial state $\otimes_n |0\rangle$. These are the matrix representation of both one-qubit gates considering the same computational basis as before:

$$R_x(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \qquad (2.22)$$

With all these gates, we are able to build the final circuit for a given graph. In the figure 2.4 we can see an example of a graph and its corresponding QAOA circuit with $P = 1$.

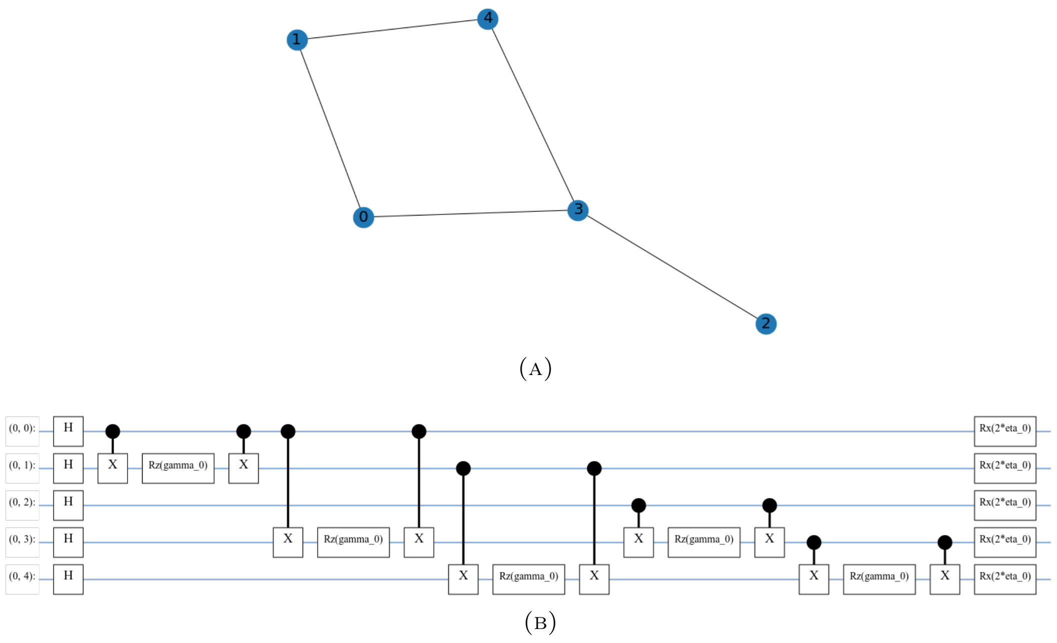


FIGURE 2.4: Representation of a graph (a) and the QAOA circuit for the corresponding graph with $P = 1$ (b). The way in which the operator $U(C, \gamma)$ is applied is irrelevant for the circuit, we can interchange the operators in a pair of adjacent nodes and we will still get the same quantum state.

2.1.1 State-of-the-art

In order to solve these type of problems, authors have developed different hand-crafted algorithms. Focusing on the max-cut problem, we find many developments in the literature [10, 5, 32, 9, 13].

After the QAOA algorithm was proposed, the authors proposed a simple strategy to find the optimal parameters [10]. Given that the expected value of the cost function (2.13) can be seen as a sum of a cost function (2.16) for each edge in the graph:

$$F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \sum_{\langle j,k \rangle} \langle \boldsymbol{\gamma}, \boldsymbol{\beta} | C_{\langle j,k \rangle} | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle, \tag{2.23}$$

where

$$C = \sum_{\langle j,k \rangle} C_{\langle j,k \rangle}, \quad C_{\langle j,k \rangle} = \frac{1}{2}(1 - \sigma_j^z \sigma_k^z) \tag{2.24}$$

we can consider the operator associated with an edge $\langle j, k \rangle$:

$$U^\dagger(C, \gamma_1) \ldots U^\dagger(B, \beta_p) C_{\langle j,k \rangle} U(B, \beta_p) \ldots U(C, \gamma_1). \tag{2.25}$$

As we can see, this operator only involves the qubits j and k, and the qubits that are at most at p distance from these two qubits, due to the operator C. In particular, for the case $p = 1$, we can simplify this expression as:

$$\begin{aligned}
\langle C_{\langle j,k\rangle}\rangle &= U^\dagger(C,\gamma_1)U^\dagger(B,\beta_1)C_{\langle j,k\rangle}U(B,\beta_1)U(C,\gamma_1) \\
&= U^\dagger(C,\gamma_1)\prod_{l=1}^{n}\left[e^{i\beta_1\sigma_l^x}\right]C_{\langle j,k\rangle}\prod_{m=1}^{n}\left[e^{-i\beta_1\sigma_m^x}\right]U(C,\gamma_1) \\
&= U^\dagger(C,\gamma_1)e^{i\beta_1(\sigma_j^x+\sigma_k^x)}C_{\langle j,k\rangle}e^{-i\beta(\sigma_j^x+\sigma_k^x)}\prod_{m\neq j,k}^{n}\left[e^{i\beta_1(\sigma_m^x-\sigma_m^x)}\right]U(C,\gamma_1) \\
&= U^\dagger(C,\gamma_1)e^{i\beta_1(\sigma_j^x+\sigma_k^x)}C_{\langle j,k\rangle}e^{-i\beta(\sigma_j^x+\sigma_k^x)}U(C,\gamma_1), \qquad (2.26)
\end{aligned}$$

where in the third line we have used the fact that the σ^x commutes with σ^z if both operators do not apply to the same qubits. Now the operator (2.26) only involves the edge $\langle j,k\rangle$ and the qubits adjacent to these edges. In a similar way, when considering circuits with a larger p, we can see that this operator only involves the subset of qubits that are at most p distance from the edge considered.

With this consideration, we can then evaluate F_p in each of these subgraphs that contain a number of qubits that is independent of the number of edges n, and thus simplifies the search of the optimal parameters for a fixed p.

Numerical results on classical hardware shows that in the case of two-regular graphs, the quantum algorithm is able to produce an approximation ratio that can be arbitrarily close to 1 by making p large enough, independent of n. In the case of three-regular graphs, it can be seen that for $p = 1$, the worst case approximation of the ratio between the output of the algorithm and the maximum of C is 0.6924 [10].

Following with circuits with a low-level p, the authors in [29] studied analytically and numerically the case for which $p = 1$. For this case they derived an analytical expression for a general graph, in which for each edge $\langle j,k\rangle$, the cost is the following:

$$\begin{aligned}
\langle C_{\langle j,k\rangle}\rangle &= \frac{1}{2}+\frac{1}{4}(\sin(4\beta_1)\sin(\gamma_1))(\cos^{d_j}(\gamma)+\cos^{d_k}(\gamma)) \\
&\quad -\frac{1}{4}(\sin^2(\beta_1)\cos^{d_j+d_k-2\lambda_{jk}}(\gamma_1))(1-\cos^{\lambda_{jk}}(2\gamma_1)), \qquad (2.27)
\end{aligned}$$

where $d_j + 1$ and $d_k + 1$ are the degrees of the vertices j and k respectively, and λ_{jk} is the number of triangles in the graph containing the edge $\langle j,k\rangle$. Using this result, we can optimize the cost function in a fully classical manner. In addition, this result can be further simplified in the case in which we have a triangle-free n-regular graph:

$$F_1(\gamma_1,\beta_1) = \frac{|E|}{2}(1+\sin(4\beta_1)\sin(\gamma_1)\cos^{n-1}(\gamma_1)). \qquad (2.28)$$

For this particular type of graphs, the optimal pair of angles is:

$$\gamma_1^* = \arctan\left(\frac{1}{\sqrt{n-1}}\right), \quad \beta_1^* = \frac{\pi}{8}, \qquad (2.29)$$

which leads to the maximum expectation value of:

$$F_1^* = \frac{|E|}{2}\left(1+\frac{1}{\sqrt{n}}\left(\frac{n-1}{n}\right)^{\frac{n-1}{2}}\right). \qquad (2.30)$$

In [32], the authors explore QAOA's performance beyond its lowest depth variant, and proposed heuristic strategies to find quasi-optimal parameters for a depth p algorithm in polynomial time with respect to p. In comparison, the strategy in which the circuit starts using random parameters requires $2^{(p)}$ optimization runs to achieve a similar performance.

In this work, the authors observed that the optimal parameters obtained by brute force indicate that in general, there is a slowly varying continuous curve that underlies the parameters γ_i^* and β^*. This observations conduced the authors to propose that the parameters on the level $p+1$ can be based on optimized parameters from the previous levels.

A first heuristic strategy called INTERP, uses linear interpolation to chose the initial parameters. The parameters for $p = 1$ are optimised as a first step, and then the subsequent parameters $p+1$ are guessed from the interpolation from the previous ones. The circuit is then optimised from these parameters and we can keep iteratively applying the strategy to get deeper circuits.

The second heuristic strategy proposed, called FOURIER, uses a different parametrization in which we change from $2p$ parameters to $2q$ parameters in the following way:

$$\gamma_i = \sum_{k=1}^{q} u_k \sin\left[\left(k-\frac{1}{2}\right)\left(i-\frac{1}{2}\right)\frac{\pi}{p}\right], \tag{2.31}$$

$$\beta_i = \sum_{k=1}^{q} v_k \cos\left[\left(k-\frac{1}{2}\right)\left(i-\frac{1}{2}\right)\frac{\pi}{p}\right]. \tag{2.32}$$

These transformations are known as the Discrete Sine and Cosine Transformations respectively. In this strategy, a similar procedure to the previous one is performed, where instead of optimizing γ_i and β_i, we optimize u_k and v_k.

In all these cases explored, one has to classically optimize each graph for $p > 1$, which requires an ah-hoc solution for each possible instance of the problem. In a recent work, reinforcement learning has been applied to figure out the set of parameters at the level $p+1$ from the previously obtained parameters p and measurements of the qubits at the different levels [12].

This work is based on a classical agent which is able to propose new parameters and is trained to maximize the final outcome. This results in an algorithm which is able to be trained with circuits with low p, while then achieving for test instances optimal results in cases for which $p = 21$.

2.2 Experiments and Results

Now that we have seen the strategies to find the different parameters, let's focus on the optimization of such circuits, which is the main procedure in most of the strategies we have explored. To do so, we will explore the optimization procedure for 3-regular graphs and random connected graphs.

Fore the first type of graphs, we have seen that the QAOA circuit in the case of $p = 1$ should theoretically yield results over 0.6924 for the approximation ratio with

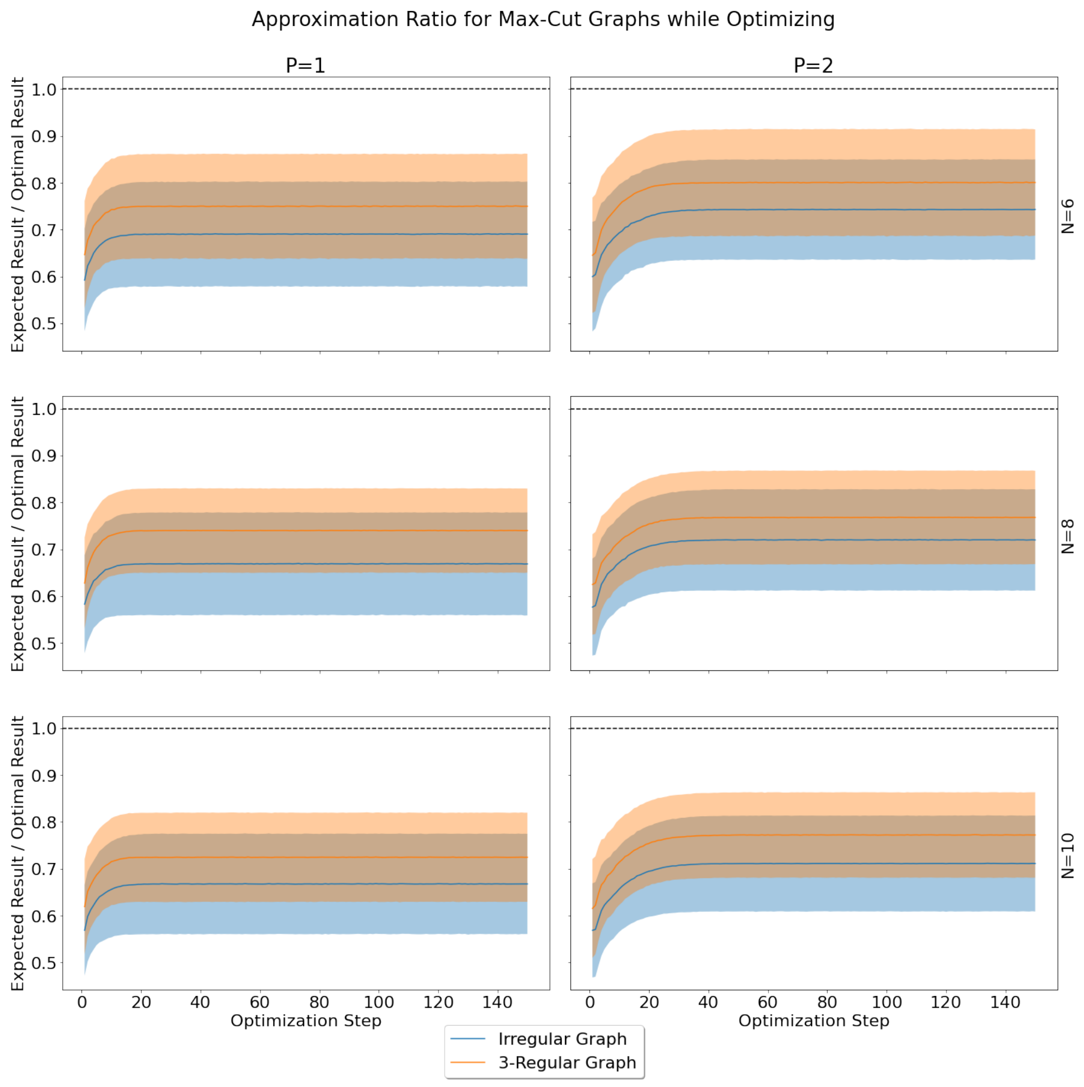


FIGURE 2.5: Performance of the Nelder-Mead optimizer in different problems and QAOA parametrizations.

the optimal parameters, thus this being a simple case.

On the other hand, we are also going to explore the case in which we have a random connected graph, a much more challenging case that will resemble a more realistic case.

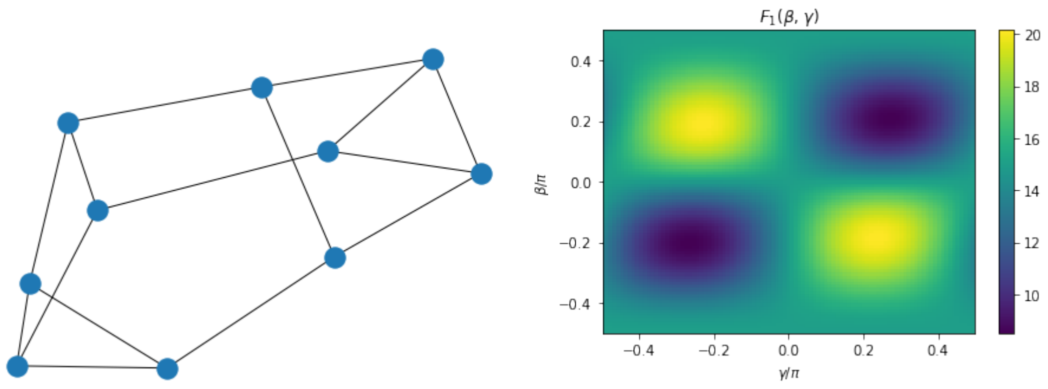
For each experiment, we are going to simulate the result with 500 randomized graphs that fulfill the requirements. Then, using a Nelder-Mead optimizer [15] that queries the quantum circuit 1000 times for each measure we are going to optimize the parameters for up to 150 optimization steps.

3-Regular Graphs

In this first set of experiments, we are going to focus on the optimization procedure for the Max-Cut problem in 3-Regular graphs. These type of graphs are the ones in which all the nodes have exactly 3 neighbours.

		Approximation Ratio of 3-Regular Graphs			
n	p	5 Steps	10 Steps	20 Steps	100 Steps
6	1	0.72 ± 0.10	0.74 ± 0.11	0.75 ± 0.11	0.75 ± 0.11
	2	0.71 ± 0.11	0.75 ± 0.11	0.79 ± 0.11	0.80 ± 0.11
8	1	0.703 ± 0.083	0.730 ± 0.087	0.740 ± 0.089	0.740 ± 0.090
	2	0.679 ± 0.096	0.716 ± 0.095	0.754 ± 0.097	0.77 ± 0.10
10	1	0.687 ± 0.091	0.715 ± 0.093	0.725 ± 0.095	0.725 ± 0.096
	2	0.673 ± 0.091	0.714 ± 0.091	0.754 ± 0.090	0.772 ± 0.091

TABLE 2.1: Approximation ratio for different 3-Regular Graphs for $n = 6, 8, 10$ and $p = 1, 2$ at different optimization steps using the Nelder-Mead optimizer.



(A) (B)

FIGURE 2.6: 3-Regular graph (a) with its corresponding expected cost value of the QAOA circuit with $p = 1$ as a function of β_1 and γ_1.

On the figure 2.5 we can see the result of the optimization procedure for different values of $p = 1, 2$ and $n = 6, 8, 10$. This graph shows us some interesting facts about this algorithm. First of all, we see how the optimization stabilizes after 40 steps in the observed cases, and from that point the variance across different graphs is stable.

Another thing to notice is how the performance of the QAOA circuit under the optimal parameters is better on average with larger p, and that this average tends to decay with larger graphs. A more detailed view of the results can be seen on the table 2.1. An example of the lattice of F_1 for a 3-Regular graph can be seen in the figure 2.6.

A final thing to notice is that the expected approximation ratio shown here is obtained by repeatedly querying the quantum circuit and computing the Max-Cut of those solutions, which can give different results in different measures. This means that some measures of this circuit could yield in better solutions to the problem.

Finding the optimal parameters is quite challenging, as it needs several thousands of queries to the quantum circuit. When considering larger values of p, it is even more difficult to reach good parameters, and we encounter the so-called *barren plateau* problem.

Irregular Graphs

In the second set of experiments, we want to observe the optimization performance of irregular graphs in the same way we have done in the previous case. Even though we want to have graphs without any particular shape, we will constrain the set of graphs generated to connected graphs. The reason for this is that if we have an unconnected graph, the optimal result for the Max-Cut problem is the union of the optimal results for the different connected subgraphs.

In order to build the different graphs for these experiments, we are going to use the Watts-Strogatz small-world graph method to generate a random connected graph [30]. The way this algorithm works is by creating a first ring graph over n nodes. Then, each node in the ring is joined to its k nearest neighbours and finally edges are replaced by other edges to other nodes with probability p. For these experiments, we have used the values $k = 3$ and $p = 0.4$.

The results of the optimization procedure can be seen in the figure 2.5. In comparison to the 3-Regular graphs, we see that the average approximation ratio of Irregular graphs is lower than the one with 3-Regular graphs across all the experiments. However, we see that these types of graphs also exhibit a similar behaviour, in terms of the response of the p and n parameters. Also, we can see how the standard deviation is comparable to the previous case.

A more detailed view of the results of the optimization procedure can be seen in the table 2.2. There we can see how in the first 20 optimization steps we manage to get to similar results than the ones after 100 steps.

		Approximation Ratio of Irregular Graphs			
n	p	5 Steps	10 Steps	20 Steps	100 Steps
6	1	0.66 ± 0.10	0.68 ± 0.11	0.69 ± 0.11	0.69 ± 0.11
	2	0.66 ± 0.10	0.69 ± 0.10	0.73 ± 0.10	0.74 ± 0.11
8	1	0.637 ± 0.099	0.66 ± 0.10	0.67 ± 0.11	0.67 ± 0.11
	2	0.635 ± 0.097	0.671 ± 0.098	0.71 ± 0.10	0.72 ± 0.11
10	1	0.632 ± 0.095	0.66 ± 0.10	0.67 ± 0.11	0.67 ± 0.11
	2	0.622 ± 0.094	0.657 ± 0.091	0.695 ± 0.097	0.71 ± 0.10

TABLE 2.2: Approximation ratio for different Irregular Graphs for $n = 6, 8, 10$ and $p = 1, 2$ at different optimization steps using the Nelder-Mead optimizer.

All these initial steps to get us closer to good parameters are very expensive for a quantum computer, and in order to achieve feasible results with near-term quantum devices we would need to reduce the number of queries to get a good set of initial parameters. In the next section we are going to explore exactly this, how we can reduce the initial exploration stage to get closer to the optimal parameters with a much less demanding number of queries to the quantum device with the use of *meta-learning* techniques.

3 Quantum Recurrent Neural Networks

In the proposed quantum problems, one can think of them as an optimization problem, in which we want to find the optimal parameters $\theta \in \Theta$ for a given objective function, $f(\theta)$. The goal is to find the parameters that minimize this function:

$$\theta^* = \arg\min_{\theta \in \Theta} \{f(\theta)\}. \tag{3.1}$$

In machine learning, a standard way of solving this problem is to apply gradient descent, in case that f is a continuous function, that can be differentiated. With this approach, one would have to keep updating the parameters according to the following formula until convergence:

$$\theta_{t+1} = \theta_t - \lambda_t \nabla f(\theta_t), \tag{3.2}$$

where λ_t is a scalar that receives the name of learning rate.

In a recent work in mate-learning [1], a different strategy was proposed to replace hand-designed update rules with a learned update rule, which we can set as a function g, with his own set of parameters ϕ. This results in updates of the objective function f to optimize of the form

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi_t). \tag{3.3}$$

The goal of meta-learning is to develop a learning algorithm which performs well on a particular subset of optimization problems. In our case, we would like to have an algorithm that is able to achieve a good performance in different instances of a quantum problem expressed in the form of a graph.

This approach is different from the previous hand-designed algorithms for variational quantum algorithms, in the sense that we have to query the quantum computer and then the learned algorithm gives the next guesses for the parameters. On the other hand, classic algorithms would treat every problem independent from the rest, and we won't be exploiting possible patterns learned in some instances.

In the different meta-learning studies, a RNN has been used as the optimizer function [8, 1, 11, 18, 24, 28]. An schema of this architecture can be found on the figure 3.1.

In this architecture, the RNN gives a proposal set of parameters θ_t at each timestep t. Then, the proposed parameters, together with the result of the objective function are aggregated as the inputs of the following step in the RNN. In addition, the results of the objective function are finally aggregated to compute the loss function.

With this loss function, we can train the architecture in order to learn the correct parameters ϕ of the RNN. The problem then becomes which loss function to use.

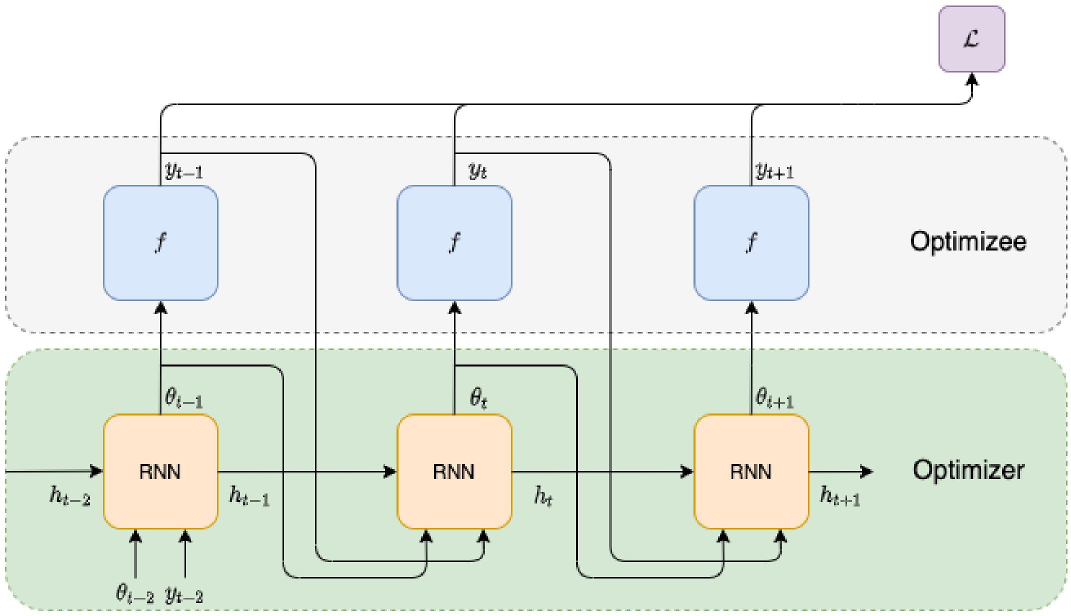


FIGURE 3.1: Basic architecture of the meta-learning proposal. The grey region represents the function f to be optimized, while the green region is the optimizee g that is used to update the parameters θ. The results of the function f are finally aggregated into a loss function $\mathcal{L}$.

3.1 Loss functions

Given an optimizee like the one previously described, we must choose an adequate loss function $\mathcal{L}(\phi)$ with respect to the parameters of the RNN ϕ. For a given quantum circuit in which the cost function is the expected value of the Hamiltonian:

$$f(\theta) = \langle \hat{H} \rangle_\theta, \tag{3.4}$$

the loss function of the RNN can depend in general on all the intermediate times $\{\mathbb{E}_{f,\boldsymbol{y}}[f(\theta_t)]\}_{t=1}^{T}$ of the network. We would like to have a loss function that is general and can be applied to a variety of problems, as well as a function that can learn to rapidly find optimum parameters and is constantly driven to find higher quality parameters.

One of the most simple loss functions we could use is the loss at the final iteration T of the RNN:

$$\mathcal{L}_{\text{final}}(\phi) = \mathbb{E}_{f,y_T}[f(\theta_T)], \tag{3.5}$$

which would be averaged over all the different samples. This loss was considered by [1] when learning first-order optimizers, but ultimately found that a better alternative was the sum of the losses at different times:

$$\mathcal{L}_{\text{sum}}(\phi) = \sum_{t=1}^{T} \mathbb{E}_{f,y_t}[f(\theta_t)]. \tag{3.6}$$

The main advantage of the latter approach is that $\mathcal{L}_{\text{final}}$ carries a temporally sparse signal, while $\mathcal{L}_{\text{sum}}$ is able to provide information form every step of the optimizer while training. This strategy would be equivalent to find the strategy which minimizes the expected cumulative regret. In general, we would be interested in the best possible outcome at any given time of the optimizer, $\min_t f(\theta_t)$. However, taking the last observation of the cumulative regret can be seen as a good proxy for this.

The previous loss function is limited to exploration though. This function may prioritize rapidly finding an approximate optimum and staying there, instead of trying to explore and encouraging the optimizer to explore better minima.

We can enforce exploration of the optimizer parameter space by encoding an exploratory force into the meta-learning loss. One example of these exploration strategies is to use the observed improvement:

$$\mathcal{L}_{\text{sum}}(\phi) = \mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T} \min\left(f(\theta_t) - \min_{i<t}\left(f(\theta_i)\right), 0\right)\right]. \tag{3.7}$$

The observed improvement (OP) at the step t, is given by the difference between the value at the current time $f(\theta_t)$, and the best obtained value from the past of the optimizer until that point $\min_{i<t}(f(\theta_i))$. In case there is no improvement with respect to the past values, the loss for this step is zero. This is a loss function that could explore better parameters than the previous one, as it can give intermediate steps with worse performance if at a later stage it gives a better minima, in contrast to the other losses.

Once we have a loss function, we have an objective function that can be used to assess whether or not the parameters of a given architecture are good. In order to train the architecture, and thus obtain a good set of parameters for the RNN, we would need the gradient of the architecture with respect to all its parameters. Training of this networks are topically done through back-propagation algorithms, and these are only defined for classical computers, not quantum ones. In this next section we are going to discuss how back-propagation can be implemented in a hybrid quantum-classical computation.

3.2 Quantum Gradients

After establishing the convention for a Quantum Neural Networks in the first section, we can discuss how to obtain gradients of the cost function with respect to the parameters when we have quantum computations. Here we will not consider the case of hybrid quantum-classical computations, as we can think this case as simply considering the classical and quantum back-propagation's depending on the computation block we are dealing with at the moment.

Being able to rapidly compute differentiate error functionals has been one of the main keys of the success of modern deep learning, thanks to the back-propagation algorithm [23], a special case of an Auto-differnetiation algorithm (AutoDiff) [3]. In order to extend these algorithms to accept quantum computations, there are two options which we could follow: finite difference or parameter shift methods.

Finite difference methods

A trivial way in which we could obtain the derivative of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with respect to a parameter $k \in \mathbb{R}$ would be to use a finite difference method.

An example of a finite difference method is the central difference:

$$\partial_k f(\boldsymbol{\theta}) = \frac{f(\boldsymbol{\theta} + h\boldsymbol{I}_{\Gamma}30059) - f(\boldsymbol{\theta} - h\boldsymbol{I}_{\Gamma}30059)}{2h} + \mathcal{O}(h^2), \tag{3.8}$$

where $(\boldsymbol{I}_{\Gamma}30059)_i = \delta_i^k$ is a vector which is zero everywhere except on the position of the k-th parameter, which is 1.

With this approach, for this parameter we would have to evaluate the function f twice, every time varying the parameter by a small h value. Also the precision of this method in particular would be $\mathcal{O}(h^2)$, and in case we would like to have more precision, we would need to query the function more times.

However, this is not the only problem of this approach. As the function f in this case is a quantum measurement as in the equation 3.4, we would have to perform multiple measures for each parameter, as we are interested in the expected value of that function, not a single query which would not be representative of the quantity we are looking for.

Parameter shift methods

More recently, a proposal for a novel way to compute the exact gradient of a quantum computation with respect to its parameters known as *parameter shift* has been developed [25]. In this section we are going to explain this method.

First of all, we have to rewrite the way we think a quantum neural network. An schematic view of the decomposition can be seen in the figure 3.2.

The Quantum Neural Network $\hat{U}(\boldsymbol{\theta})$ can be generally written as a product of L layers of unitary operators:

$$\hat{U}(\boldsymbol{\theta}) = \prod_{l=1}^{L} \hat{V}^l \hat{U}^l(\boldsymbol{\theta}^l), \tag{3.9}$$

where the l-th layer of the QNN consists of the product of a constant unitary $\hat{V}^l$, and a parametrized unitary operator $\hat{U}^l(\boldsymbol{\theta}^l)$. The latter operator can itself be comprised in general of multiple unitaries applied in parallel:

$$\hat{U}^l(\boldsymbol{\theta}^l) = \bigotimes_{j=1}^{M_l} \hat{U}_j^l(\theta_j^l), \tag{3.10}$$

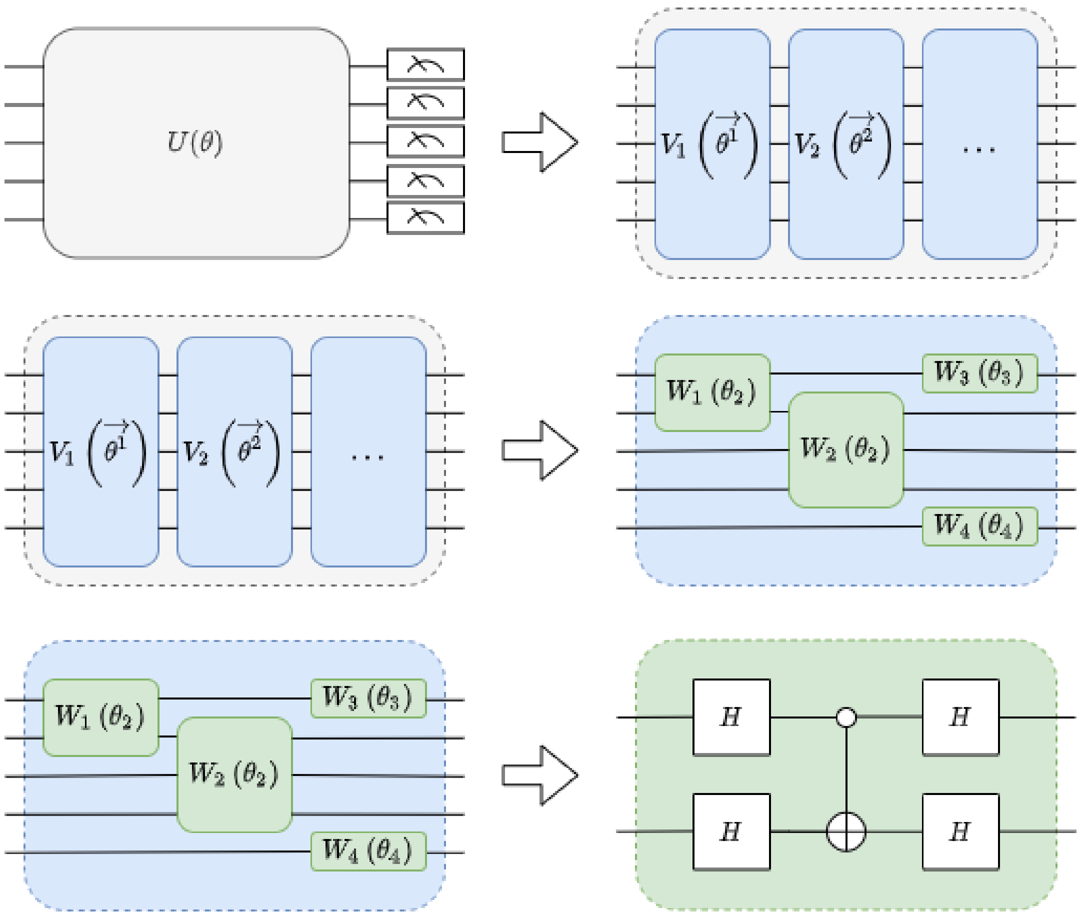


FIGURE 3.2: Decomposition of a Quantum Neural Network.

in which each unitary $\hat{U}_j^l(\theta_j^l)$ only has one parameter.

Finally, each one of these unitaries $\hat{U}_j^l$ can be expressed as the exponential of some generator $\hat{g}_j^l$, which at the same time can be described by a Hermitian operator on n qubits:

$$\hat{U}_j^l(\theta_j^l) = e^{-i\theta_j^l \hat{g}_j^l}, \tag{3.11}$$

$$\hat{g}_j^l = \sum_{k=1}^{K_{jl}} \beta_k^{jl} \hat{P}_k. \tag{3.12}$$

In these equations, $\hat{P}_k \in \mathcal{P}_n$ denotes a Pauli matrix on n qubits. We can express any Hamiltonian on n qubits as a sum of Paulis, as it is a base of the same space, for a given set of $\beta_k^{jl} \in R, \forall j, k, l$.

In the case in which all of the Pauli terms in a given single parameter unitary $\hat{U}_j^l$ commute:

$$[\hat{P}_k, \hat{P}_m] = 0, \quad \forall m, k,$$

we can simplify the formula for a product of exponentials:

$$\hat{U}_j^l(\theta_j^l) = \prod_{k=1}^{K_{jl}} e^{-i\theta_j^l \beta_k^{jl} \hat{P}_k}. \tag{3.13}$$

In the cases in which the above does not hold, we still can approximate the exponential of the sum with a product of exponentials using a Trotter-Suzuki decomposition [27].

Now, in order to decompose the exponential, we have to take a look at the Taylor expansion of the exponential function:

$$e^x = \sum_{i=0}^{\infty} \frac{x^n}{n!}. \tag{3.14}$$

Plugging the exponential from (3.13) into (3.14), we have:

$$\begin{aligned}
e^{-i\theta_j^l \beta_k^{jl} \hat{P}_k} &= \sum_{i=0}^{\infty} \frac{\left(-i\theta_j^l \beta_k^{jl} \hat{P}_k\right)^n}{n!} \\
&= \sum_{n=0}^{\infty} \frac{(-i)^{2n}}{(2n)!} (\theta_j^l \beta_k^{jl})^{2n} \hat{P}_k^{2n} + \sum_{n=0}^{\infty} \frac{(-i)^{2n+1}}{(2n+1)!} (\theta_j^l \beta_k^{jl})^{2n+1} \hat{P}_k^{2n+1} \\
&= \sum_{n=0}^{\infty} \frac{((-1)^2 i^2)^n}{(2n)!} (\theta_j^l \beta_k^{jl})^{2n} \hat{P}_k^{2n} - i \sum_{n=0}^{\infty} \frac{((-1)^2 i^2)^n}{(2n+1)!} (\theta_j^l \beta_k^{jl})^{2n+1} \hat{P}_k^{2n+1} \\
&= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} (\theta_j^l \beta_k^{jl})^{2n} \hat{P}_k^{2n} - i\hat{P}_k \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} (\theta_j^l \beta_k^{jl})^{2n+1} \hat{P}_k^{2n}
\end{aligned}$$

Then, by considering the general property that the square of any Pauli matrix is equal to the identity matrix, $\hat{P}_k^2 = \hat{I}$, we have:

$$e^{-i\theta_j^l \beta_k^{jl} \hat{P}_k} = \hat{I} \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} (\theta_j^l \beta_k^{jl})^{2n} - i\hat{P}_k \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} (\theta_j^l \beta_k^{jl})^{2n+1}. \tag{3.15}$$

Considering the Taylor expansion of the sine and cosine functions:

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}, \quad \cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}, \tag{3.16}$$

we can plug these results into (3.15) to get the final decomposed exponential:

$$e^{-i\theta_j^l \beta_k^{jl} \hat{P}_k} = \hat{I} \cos\left(\theta_j^l \beta_k^{jl}\right) - i\hat{P}_k \sin\left(\theta_j^l \beta_k^{jl}\right). \tag{3.17}$$

Finally, we can express the single parameter unitary (3.13) as:

$$\hat{U}_j^l(\theta_j^l) = \prod_{k=1}^{K_{jl}} \left[\hat{I} \cos\left(\theta_j^l \beta_k^{jl}\right) - i\hat{P}_k \sin\left(\theta_j^l \beta_k^{jl}\right)\right]. \tag{3.18}$$

Now that we have these expressions for the representation of a quantum circuit, we can proceed to compute the gradients of the circuit with respect to its parameters. Once we have this, we will be able to back-propagate the gradients and train the whole hybrid quantum-classical architecture.

Lets consider now a parameter of interest θ_j^l, appearing in a parametric unitary of the form (3.13). Considering the following change of variables:

$$\eta_k^{jl} = \theta_j^l \beta_k^{jl}, \quad \forall k \in \{1, \dots, K_{jl}\}, \tag{3.19}$$

the derivative of a function f with respect to θ_j^l becomes, under the chain rule:

$$\begin{aligned} \frac{\partial f\left(\eta_1^{jl}(\theta_j^l, \beta_1^{jl}), \dots, \eta_{K_{jl}}^{jl}(\theta_j^l, \beta_{K_{jl}}^{jl})\right)}{\partial \theta_j^l} &= \sum_{k=1}^{K_{jl}} \frac{\partial \eta_k^{jl}}{\partial \theta_j^l} \frac{\partial f}{\partial \eta_k^{jl}} \\ &= \sum_{k=1}^{K_{jl}} \beta_k^{jl} \frac{\partial f}{\partial \eta_k^{jl}}. \end{aligned} \tag{3.20}$$

As we are working with this transformation of variables, we can now express the multi-parameter unitary gate (3.10) as:

$$\hat{U}^l(\boldsymbol{\eta}^l) = \bigotimes_{j=1}^{M_l} \left(\prod_{k=1}^{K_{jl}} e^{-i\eta_k^{jl} \hat{P}_k} \right). \tag{3.21}$$

In a similar way, we can also express the single parameter unitary gate as a product of sines and cosines by plugging the equation (3.17) into (3.21). With this change, we end up with the final representation of the multi-parameter gate:

$$\hat{U}^l(\boldsymbol{\eta}^l) = \bigotimes_{j=1}^{M_l} \left(\prod_{k=1}^{K_{jl}} \left[\hat{I} \cos\left(\eta_k^{jl}\right) - i\hat{P}_k \sin\left(\eta_k^{jl}\right) \right] \right). \tag{3.22}$$

Now that we have the quantum circuit parametrised as a function of the new parameter η_k^{jl}, we are ready to ask for the derivative of the expected value of a given Hamiltonian (3.4) with respect to any of its parameters. Considering that thanks to the equation (3.20) we only need the derivatives with respect to η_k^{jl} to get the values of the gradients with respect the parameters of the quantum circuit θ_j^l, we need to find the following derivatives:

$$\begin{aligned}
\frac{\partial f(\boldsymbol{\eta})}{\partial \eta_k^{jl}} &= \frac{\partial \langle \hat{H} \rangle}{\partial \eta_k^{jl}} \\
&= \frac{\partial}{\partial \eta_k^{jl}} \left[\langle \Psi | H | \Psi \rangle \right] \\
&= \frac{\partial}{\partial \eta_k^{jl}} \left[\langle \Psi_0 | \hat{U}^\dagger(\boldsymbol{\eta}) H \hat{U}(\boldsymbol{\eta}) | \Psi_0 \rangle \right] \\
&= \langle \Psi_0 | \hat{U}^\dagger H \frac{\partial \hat{U}(\boldsymbol{\eta})}{\partial \eta_k^{jl}} | \Psi_0 \rangle + \langle \Psi_0 | \frac{\partial \hat{U}^\dagger(\boldsymbol{\eta})}{\partial \eta_k^{jl}} H \hat{U}(\boldsymbol{\eta}) | \Psi_0 \rangle \\
&= \frac{1}{2} \left[\langle \Psi_0 | \underbrace{\left(\hat{U}^\dagger + \frac{\partial \hat{U}^\dagger(\boldsymbol{\eta})}{\partial \eta_k^{jl}} \right)}_{(I)} H \underbrace{\left(\hat{U}(\boldsymbol{\eta}) + \frac{\partial \hat{U}(\boldsymbol{\eta})}{\partial \eta_k^{jl}} \right)}_{(II)} | \Psi_0 \rangle \right. \\
&\qquad \left. - \langle \Psi_0 | \underbrace{\left(\hat{U}^\dagger - \frac{\partial \hat{U}^\dagger(\boldsymbol{\eta})}{\partial \eta_k^{jl}} \right)}_{(III)} H \underbrace{\left(\hat{U}(\boldsymbol{\eta}) - \frac{\partial \hat{U}(\boldsymbol{\eta})}{\partial \eta_k^{jl}} \right)}_{(IV)} | \Psi_0 \rangle \right]
\end{aligned} \tag{3.23}$$

The last equality can be easily seen by expanding all the terms in parenthesis. We will start by finding the first term of (3.23). Although the unitary operator of a quantum circuit seems complex, the derivatives with respect to one of its parameters can be simplified thanks to the fact that each parameter only appears once in a given layer. Considering the factorization of the circuit as a product of layers (3.9), we have:

$$\frac{\partial \hat{U}(\boldsymbol{\eta})}{\partial \eta_k^{jl}} = \hat{V}^1 \hat{U}^1(\boldsymbol{\eta}^1) \ldots \hat{V}^{l-1} \hat{U}^{l-1}(\boldsymbol{\eta}^{l-1}) \hat{V}^l \frac{\partial \hat{U}^l(\boldsymbol{\eta}^l)}{\partial \eta_k^{jl}} \hat{V}^{l+1} \hat{U}^{l+1}(\boldsymbol{\eta}^{l+1}) \ldots \hat{V}^L \hat{U}^L(\boldsymbol{\eta}^L). \tag{3.24}$$

As we can see, we are only interested in the derivative of a single multi-parameter gate. In the same way, when deriving a multi-parameter gate (3.22), we are only interested in the derivative of a single term:

$$\frac{\partial}{\partial \eta_k^{jl}} \left[\hat{I} \cos\left(\eta_k^{jl}\right) - i \hat{P}_k \sin\left(\eta_k^{jl}\right) \right] = -\hat{I} \sin\left(\eta_k^{jl}\right) - i \hat{P}_k \cos\left(\eta_k^{jl}\right) \tag{3.25}$$

Following a similar procedure, we find that in the Hermitian conjugate case we are interested only in a single parameter as well:

$$\frac{\partial}{\partial \eta_k^{jl}} \left[\hat{I} \cos\left(\eta_k^{jl}\right) + i \hat{P}_k \sin\left(\eta_k^{jl}\right) \right] = -\hat{I} \sin\left(\eta_k^{jl}\right) + i \hat{P}_k \cos\left(\eta_k^{jl}\right) \tag{3.26}$$

So, when computing (I), we can see that all the terms of $\hat{U}^\dagger$ and $\frac{\partial}{\partial \eta_k^{jl}} U^\dagger$ are exactly the same, except the factor containing η_k^{jl}. This factor can be simplified as:

$$\begin{aligned} (I)_k^{jl} &= \hat{I}\left(\cos\left(\eta_k^{jl}\right) - \sin\left(\eta_k^{jl}\right)\right) + i\hat{P}_k\left(\sin\left(\eta_k^{jl}\right) + \cos\left(\eta_k^{jl}\right)\right) \\ &= \sqrt{2}\left[\hat{I}\cos\left(\eta_k^{jl} + \frac{\pi}{4}\right) + i\hat{P}_k \sin\left(\eta_k^{jl} + \frac{\pi}{4}\right)\right], \end{aligned} \tag{3.27}$$

where $(I)_k^{jl}$ is the factor that contains η_k^{jl} in (I). We can see that this result has the same functional form as $U^\dagger$, but shifting by $\pi/4$ the parameter of interest and adding a global scaling factor. Thus we can simplify (I) as:

$$(I) = \sqrt{2}\hat{U}^\dagger(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4}), \tag{3.28}$$

where $\boldsymbol{I}_k^{jl}$ is a vector which is zero everywhere except on the position of the η_k^{jl} parameter, which is 1. Similarly, we can simplify the factor of interest in (II) as:

$$\begin{aligned} (II)_k^{jl} &= \hat{I}\left(\cos\left(\eta_k^{jl}\right) - \sin\left(\eta_k^{jl}\right)\right) - i\hat{P}_k\left(\sin\left(\eta_k^{jl}\right) + \cos\left(\eta_k^{jl}\right)\right) \\ &= \sqrt{2}\left[\hat{I}\cos\left(\eta_k^{jl} + \frac{\pi}{4}\right) - i\hat{P}_k \sin\left(\eta_k^{jl} + \frac{\pi}{4}\right)\right], \end{aligned} \tag{3.29}$$

In this case we get the same functional form as $\hat{U}$, shifted and scaled by the same quantity. We can then simplify (II) as:

$$(II) = \sqrt{2}\hat{U}(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4}). \tag{3.30}$$

Once we have computed (I) and (II), we can plug these results into the first term of the equation (3.23). This results in the following term:

$$\begin{aligned} \langle\Psi_0|(I)H(II)|\Psi_0\rangle &= 2\langle\Psi_0|\hat{U}^\dagger(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4})H\hat{U}(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4})|\Psi_0\rangle \\ &= 2f\left(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4}\right). \end{aligned} \tag{3.31}$$

Following a similar procedure, we get the following results for the terms (III) and (IV) of the equation (3.23):

$$(III) = \sqrt{2}\hat{U}^\dagger(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4}), \tag{3.32}$$

$$(IV) = \sqrt{2}\hat{U}(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4}), \tag{3.33}$$

which in turn gives us the following result for the second term of (3.23):

$$\begin{aligned} \langle\Psi_0|(III)H(IV)|\Psi_0\rangle &= 2\langle\Psi_0|\hat{U}^\dagger(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4})H\hat{U}(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4})|\Psi_0\rangle \\ &= 2f\left(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4}\right). \end{aligned} \tag{3.34}$$

Finally, plugging (3.31) and (3.34) into (3.23), we get to the derivative from the expected value of a quantum circuit with respect to its parameters:

$$\frac{\partial f(\boldsymbol{\eta})}{\partial \eta_k^{jl}} = f\left(\boldsymbol{\eta} + \boldsymbol{I}_k^{jl}\frac{\pi}{4}\right) - f\left(\boldsymbol{\eta} - \boldsymbol{I}_k^{jl}\frac{\pi}{4}\right). \tag{3.35}$$

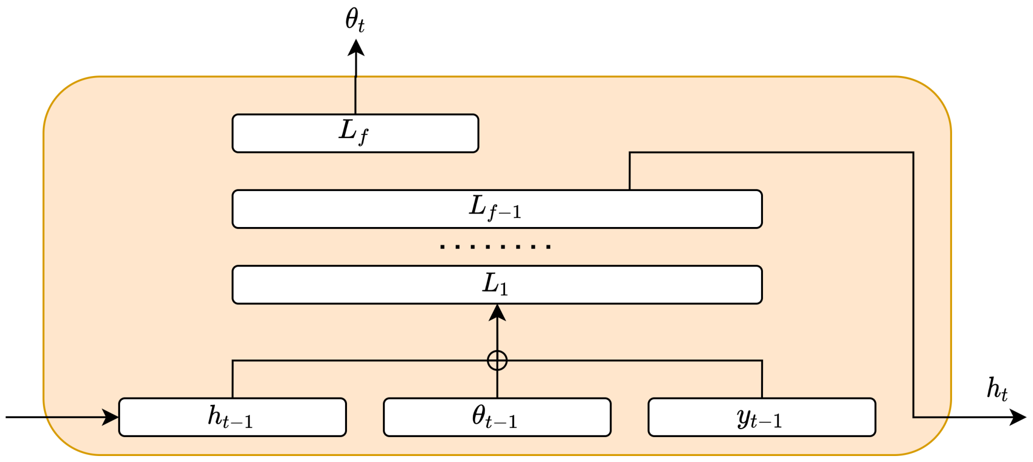


FIGURE 3.3: Architecture of the RNN used for the meta-learning model. The $L_1, \ldots, {}_f$ layers represent fully connected layers of artificial neural neurons.

With this result, we can see that in order to obtain the derivative of the expected value of the circuit, we only have to measure two times the result with a shift of the parameter. This is a much more robust approach than the finite difference method for two reasons: first of all, here we are computing the exact gradient, not an approach. Secondly, as the parameter shift is not an infinitesimal one, we require much less runs to achieve a sufficiently precise estimate of the value of the gradient.

One thing to consider though, is that according to the chain rule from the equation (3.20), in order to obtain the gradient of the parameter θ_j^l, we need to compute $2K_{jl}$ expectation values. These computations can be prohibitive in some cases. To overcome this issue, a method of stoachastically selecting only some terms according to a distribution weighted by the coefficients of each term to estimate the gradient of a given θ_j^l parameter have been proposed [14], showing good results with a reduced number of computations.

3.3 Experiments and Results

In order to test the QRNNs, we are going to perform different experiments on 3-Regular Graphs and Irregular graphs as in the previous part to compare the results. As this approach is independent of the number of n, we will also be able to test the generalization capabilities when training with instances of different size.

The RNN cell architecture used for the experiments will be based on the meta-learning approach, in which we will consider the hidden state for the recurrent unit h_{t-1} to be the concatenation of the previous proposed parameters θ_{t-1}, the expected value of the quantum circuit with the proposed parameters t_{t-1} and a hidden layer of a multi-layer perceptron. An schema of the RNN cell used can be seen in the figure 3.3.

For training, we will use the observed improvement loss (3.7), but using a slight modification of the Max-Cut loss function (2.16). As when we optimize RNN models we usually try to minimize a loss function, we will consider the objective function to minimize the negative expected cost of the Max-Cut operator.

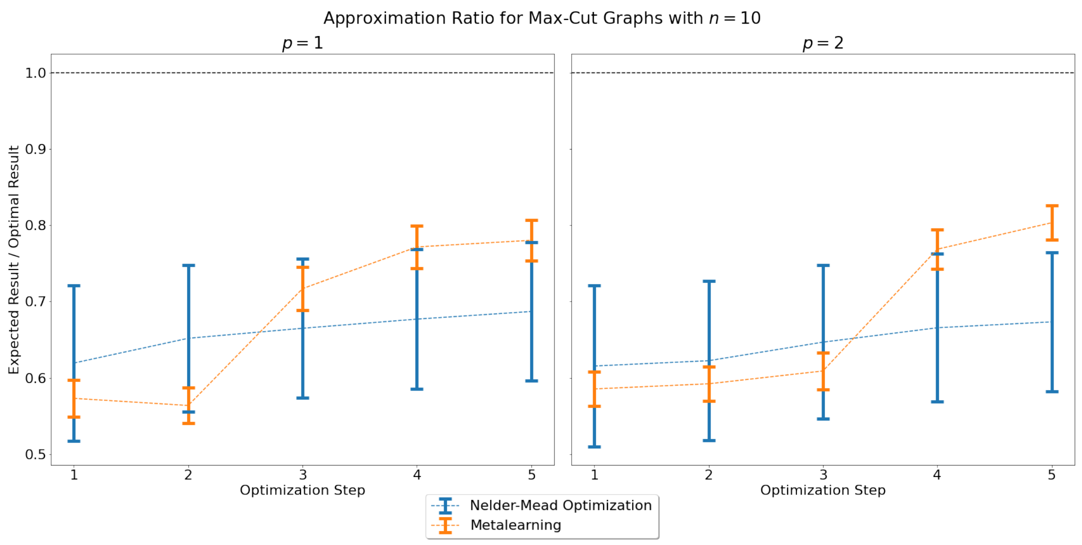


FIGURE 3.4: Comparison of the 5 first steps between the Nelder-Mead and the Meta-Learning optimization processes for $n = 10$ on 3-Regular graphs.

3-Regular Graphs

In this first set of experiments we are going to measure the performance of the Meta-Learning approach for 3-Regular graphs as we did on the previous section to compare this methodology to an optimization of the objective function directly. To do so, we will generate the meta-learning model with 5 timesteps and compare to the first steps of the previous method. Also, we will be focusing on the case $n = 10$, as is a more challenging problem.

To train the QRNN, we will use 1000 random instances of 3-Regular graphs, and use 500 more instances as a validation set. In addition, we will train the model until we see no further improvements of the validation graphs for 20 epochs. The comparison of the performance on a new generated test set in comparison to the previous methodology used for the 5 first optimization steps can be seen in the figure 3.4.

Here we can see how the meta-learning approach is able to get much better results then the classical optimization method in the first 5 steps. The real advantage for real quantum computers is even bigger, as the optimization method measures more than once the expected value of the quantum circuit, whereas the meta-learning model only computes the expectancy value of the model once every step. A more detailed set of results can be seen in the table 3.1.

An interesting thing to notice is the behaviour during the first two steps of the QRNN, where it shows similar results and after the third step the Meta-learning approach rapidly increases its performance. This could be the result of the implemented Loss function, which lets the model to explore and then it tries to get the best set of parameters. Also, the results obtained with this approach are less volatile than the ones with the other approaches, as it can be seen from the standard deviation of this method.

Finally, we can also see from these results how the average result for this approach

Step	$p = 1$		$p = 2$	
	QRNN	NM	QRNN	NM
1	0.573 ± 0.024	0.62 ± 0.10	0.585 ± 0.023	0.62 ± 0.11
2	0.564 ± 0.023	0.652 ± 0.096	0.592 ± 0.023	0.62 ± 0.10
3	0.717 ± 0.028	0.665 ± 0.091	0.609 ± 0.024	0.65 ± 0.10
4	0.772 ± 0.028	0.677 ± 0.092	0.769 ± 0.026	0.666 ± 0.097
5	0.780 ± 0.026	0.687 ± 0.091	0.803 ± 0.023	0.673 ± 0.091

TABLE 3.1: Comparison of the approximation ratio of Nelder-Mead (NM) and Meta-Learning (QRNN) methods for different 3-Regular Graphs for $n = 10$ and $p = 1, 2$ at different optimization steps.

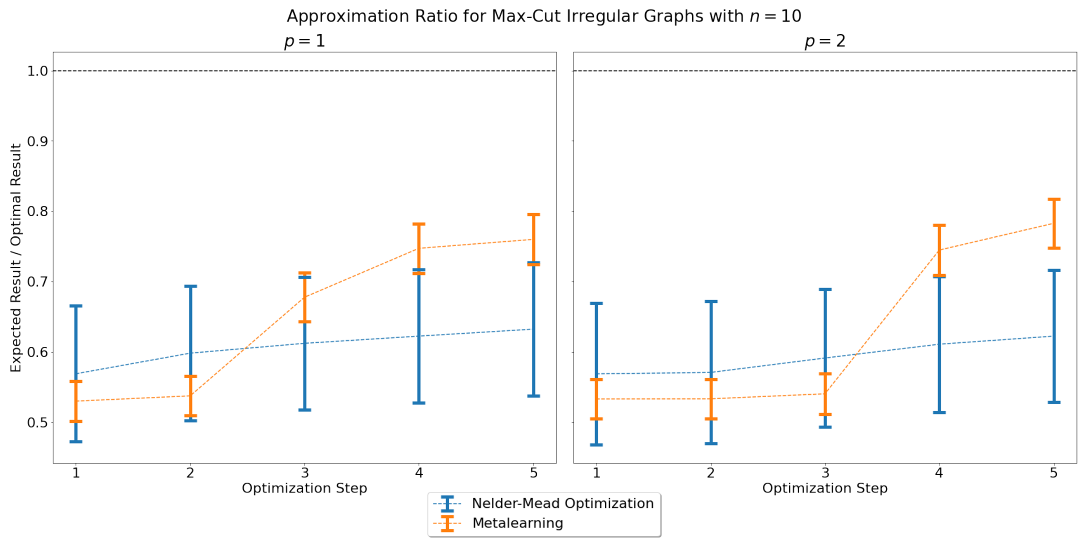


FIGURE 3.5: Comparison of the 5 first steps between the Nelder-Mead and the Meta-Learning optimization processes for $n = 10$ on irregular graphs.

is higher than the one obtained with the classical optimization method after achieving a stable regime. This shows that this approach is also able to obtain better minima for the QAOA circuit.

Irregular Graphs

Following the same methodology than the previous section to generate the irregular graphs, we are going to perform a similar set of experiments to compare the performance of this methodology with a more complex set of Max-Cut problem instances.

The comparison between the two QRNN and the classical optimization method can be seen in the figure 3.5, which again shows a superior performance than the classical optimization technique.

The comparison of the results can be seen in the table 3.2. This comparison shows similar results to what we have seen in the case of the 3-Regular graphs, and the average improvement of the QRNN approach is even larger after 5 timesteps when compared with the classical optimization method than in the case of 3-Regular graphs. This could be because with more complex graph instances, it is easier to get trapped

Step	$p = 1$		$p = 2$	
	QRNN	NM	QRNN	NM
1	0.530 ± 0.028	0.569 ± 0.096	0.533 ± 0.028	0.57 ± 0.10
2	0.537 ± 0.028	0.598 ± 0.096	0.533 ± 0.028	0.57 ± 0.10
3	0.678 ± 0.035	0.612 ± 0.094	0.540 ± 0.029	0.591 ± 0.098
4	0.747 ± 0.035	0.622 ± 0.094	0.745 ± 0.036	0.611 ± 0.096
5	0.760 ± 0.036	0.632 ± 0.095	0.783 ± 0.035	0.622 ± 0.094

TABLE 3.2: Comparison of the approximation ratio of Nelder-Mead (NM) and Meta-Learning (QRNN) methods for different Irregular Graphs for $n = 10$ and $p = 1, 2$ at different optimization steps.

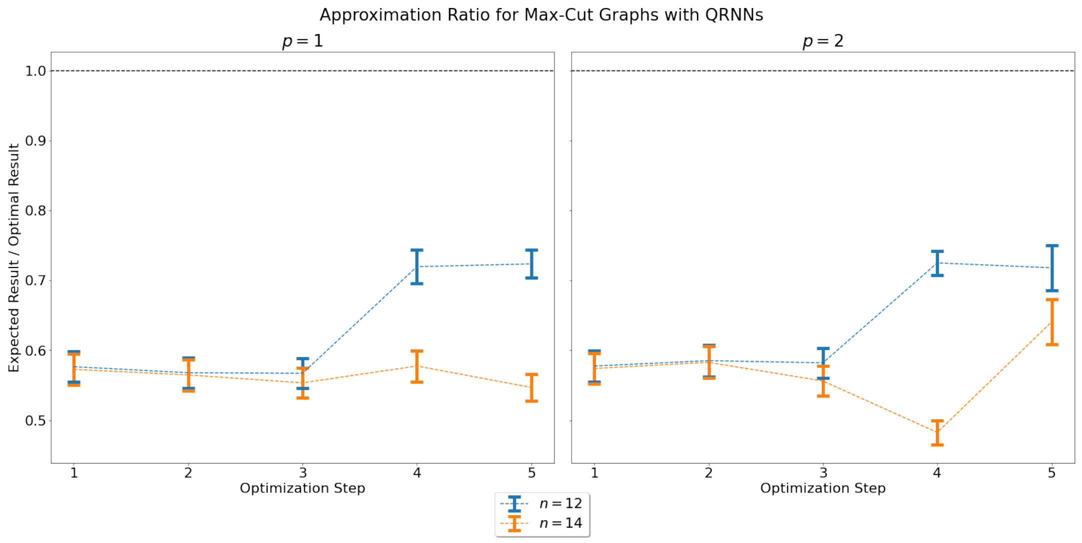


FIGURE 3.6: Approximation ratio of the meta-learning approach on instances bigger than one ones the model has used for training ($n = 10$) at different values of p.

in poor local minima with the classical approach.

Generalization to larger n

Finally, we are going to use the meta-learning models trained on a certain type of graphs to see if they can still perform well with other type of instances. In particular, we are going to use the models trained with instances in which $n = 10$ to see the performance of problems in which $n = 12, 14$. With these experiments, we hoe to see the generalization capability of these meta-learning methods applied to quantum circuits.

One thing to notice though, is that this approach does not let us generalize to higher values of p, as each internal RNN cell from the QRNN is forced to output exactly the number of values for which it has been trained. Then, in order to use this model with other p values, we would have to create them ad-hoc.

We can see the results from the optimization in the figure 3.6. In these plots we can see how the QRNN trained on instances with $n = 10$ is able to propose good initial parameters for the case $n = 12$. We can also see for this case that the behaviour is similar to the previous one, in which the optimizer uses the first steps to explore and then it applies the best parameters for the final steps. However, we can also see that this behaviour will not always happen. When increasing the size of the problem, we can see how the performance of the model gets diminished. This could be explained because these larger instances are very different than the cases explored while training, and thus the patterns discovered for these cases do not hold anymore.

This results shows that we can transfer the learning from this architecture to other similar problems, and that this meta-learning method is able to detect some general patterns from these type of problems. An open question is whether or not this learned knowledge could be transferred to other problems than the Max-Cut using the same QAOA framework.

4 Conclusions

In this work we have explored some of the main problems from the variational quantum algorithms, and in particular the Quantum Approximation Optimization Algorithm method applied to the Max-Cut problem. When solving these kind of problems, we mainly need to find the best set of parameters for a parametrized quantum circuit. There are two main challenges to solve this problem though. On the one hand, most optimization methods involve the computation of gradients or higher order differentials of the model with respect to its parameters, something which would require a large amount of measurements on an actual quantum device. Secondly, these optimization methods are highly dependent on the initial parameters, and good initialization techniques are required to keep a low amount of calculations in order to be able to use this methods in near-term quantum devices.

In the first part of this thesis we have seen the theoretical framework of the variational methods and we have seen how we can implement combinatorial optimization problems in quantum circuits. In order to highlight the main challenges when optimizing these circuits, we have used a classical optimization method to get the optimal parameters for different sets of problems. Here we have seen that with these approaches a high number of measurements are needed in order to obtain good sets of parameters. Several clever methods to obtain deeper circuits exist in the literature, which shows to improve the approximation with these algorithms, however, these methods rely on Greedy optimizations and we can not get rid of this initial overhead without other heuristics.

In the second part we have explored the meta-learning optimization approach which has shown great success in different optimization areas. We have seen how the use of this approach is able to provide good initial parameters for quantum optimization problems. In the experiments that we have performed, we have seen how these approaches are able to explore different parameters, and are able to provide parameters that are more robust to get near poor local minima.

This approach is much more less demanding than the classical optimization methods, in terms of the number of queries required on an actual quantum device. At the moment of inference, this method is only required to compute the expectation value of the circuit a few times, and no gradient computations are needed. This makes the studied approach a good candidate of a method that could be trained on a near-term quantum device.

Although for training we would require to compute the gradients to back-propagate the errors, we have seen how these can be computed efficiently in a quantum device using the analytical gradients of the circuit. This is a much more robust approach than finite differences methods, given that the measurement of a quantum circuit has a stochastic nature. Although we could compute the analytical gradient with the parameter shift method, we would still need to compute approximations of these gradients given the large amount of measurements needed in large circuits.

Finally, we have seen how this approach is able to generalize to instances which the model has not seen before. This shows that QRNNs are able to discover general patterns in quantum circuits and apply this knowledge to other quantum problems, even though the nature of the optimizer is classical.

Bibliography

[1] Marcin Andrychowicz et al. *Learning to learn by gradient descent by gradient descent*. 2016. arXiv: 1606.04474 [cs.NE].

[2] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510.

[3] Atılım Günes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 5595–5637.

[4] Jacob Biamonte et al. "Quantum machine learning". In: *Nature* 549.7671 (2017), pp. 195–202.

[5] Fernando GSL Brandao et al. "For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances". In: *arXiv preprint arXiv:1812.04170* (2018).

[6] Carlos Bravo-Prieto et al. *Variational Quantum Linear Solver*. 2020. arXiv: 1909.05820 [quant-ph].

[7] Yudong Cao et al. "Quantum chemistry in the age of quantum computing". In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

[8] Yutian Chen et al. "Learning to learn without gradient descent by gradient descent". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 748–756.

[9] Gavin E Crooks. "Performance of the quantum approximate optimization algorithm on the maximum cut problem". In: *arXiv preprint arXiv:1811.08419* (2018).

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A quantum approximate optimization algorithm". In: *arXiv preprint arXiv:1411.4028* (2014).

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *arXiv preprint arXiv:1703.03400* (2017).

[12] Artur Garcia-Saez and Jordi Riu. "Quantum Observables for continuous control of the Quantum Approximate Optimization Algorithm via Reinforcement Learning". In: *arXiv preprint arXiv:1911.09682* (2019).

[13] Gian Giacomo Guerreschi and Anne Y Matsuura. "QAOA for Max-Cut requires hundreds of qubits for quantum speed-up". In: *Scientific reports* 9.1 (2019), p. 6903.

[14] Aram Harrow and John Napp. "Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms". In: *arXiv preprint arXiv:1901.05374* (2019).

[15] Jeffrey C Lagarias et al. "Convergence properties of the Nelder–Mead simplex method in low dimensions". In: *SIAM Journal on optimization* 9.1 (1998), pp. 112–147.

[16] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. "Quantum principal component analysis". In: *Nature Physics* 10.9 (2014), pp. 631–633.

[17] Jarrod R McClean et al. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2 (2016), p. 023023.

[18] Alex Nichol, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms". In: *arXiv preprint arXiv:1803.02999* (2018).

[19] Alberto Peruzzo et al. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature communications* 5 (2014), p. 4213.

[20] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79.

[21] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. "Quantum support vector machine for big data classification". In: *Physical review letters* 113.13 (2014), p. 130503.

[22] W. Rossmann. *Lie Groups: An Introduction Through Linear Groups.* Oxford graduate texts in mathematics. Oxford University Press, 2006. ISBN: 9780199202515.

[23] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[24] Andrei A Rusu et al. "Meta-learning with latent embedding optimization". In: *arXiv preprint arXiv:1807.05960* (2018).

[25] Maria Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (2019), p. 032331.

[26] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science.* Ieee. 1994, pp. 124–134.

[27] Masuo Suzuki. "Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations". In: *Physics Letters A* 146.6 (1990), pp. 319–323.

[28] Guillaume Verdon et al. "Learning to learn with quantum neural networks via classical neural networks". In: *arXiv preprint arXiv:1907.05415* (2019).

[29] Zhihui Wang et al. "Quantum approximate optimization algorithm for MaxCut: A fermionic view". In: *Physical Review A* 97.2 (2018), p. 022304.

[30] Duncan J Watts and Steven H Strogatz. "Collective dynamics of 'small-world'networks". In: *nature* 393.6684 (1998), pp. 440–442.

[31] Nathan Wiebe, Daniel Braun, and Seth Lloyd. "Quantum algorithm for data fitting". In: *Physical review letters* 109.5 (2012), p. 050505.

[32] Leo Zhou et al. "Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices". In: *preprint arXiv:1812.01041* (2018).