Quantum Convolutional Neural Networks

Author: Joan Ainaud Fondevila
*Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.**

Advisors: Arnau Rios, Javier Rozalén

Abstract: A combination of Machine Learning techniques with Quantum Computing is presented in the form of a *Quantum Convolutional Neural Network*. This paradigm is implemented with Python to solve two simple problems. Small scaling of the complexity with respect to the input size is observed, and the detriment of architectures with a lot of depth.

I. INTRODUCTION

Machine learning (ML), and especially neural networks, have risen as one of the most popular and versatile approaches to tackle many problems ranging from generic tasks like image recognition [1] to studying quantum many-body systems [2]. Specific architectures of the networks are chosen to better tackle each problem. In studying fermionic many-body systems, neural networks are designed to respect anti-symmetry [2]. Similarly, with image recognition, the most common architecture, that of convolutional neural networks, is invariant to translations. Implementing these symmetries directly not only ensures they are satisfied, but may also reduce the complexity of the neural networks.

At the same time, quantum computing's promise for a quantum advantage is a present interest, especially as advances in quantum hardware keep progressing. But in the current noisy intermediate-scale quantum (NISQ) era, algorithm's like Shor's are still limited by the number of qubits attainable and by their noise. A very popular paradigm has become that of a hybrid design, with a quantum circuit (QC) controlled by a classical routine, for which some calculations are also outsourced [3].

In this context, the idea of combining ML with quantum computing has become quite popular. Quantum neural networks (qNN) have been used for many tasks, but here we focus on their use as classifiers. In particular, quantum convolutional neural networks (qCNNs) have emerged as promising designs [1] which, like their classical counterparts, implement translational symmetry.

In this thesis, a qCNN is implemented, that is, a translationally invariant ansatz for a quantum circuit is described and the full recipe for its implementation and training will be given.

This paper is structured as follows. In section II the quantum machine learning paradigm is presented and its theoretical working evaluated. Special care is given into considering the implementation of the method, including computation of gradients and design limitations. Then, in section III, this design is simulated classically for two simple problems. The implementation using two different libraries (*Qibo*[4] and *Tensorflow Quantum*[5, 6]) is compared to ensure the results are independent. In section IV, the problems are used to characterize the qCNNs behaviour on varying input size and circuit depth. All the code used is accessible in a GitHub repository `https://github.com/joan-ainaud/TFG_QCNN`.

II. QUANTUM MACHINE LEARNING

A quantum neural network (qNN) takes an input quantum state $|\psi(\vec{x})\rangle$, which is then evolved by a trainable parametrized circuit, which we refer to as $U_T(\vec{\theta})$, and the output corresponds to the expectation value of a measurement $\hat{O}$:

$$f(\vec{x},\vec{\theta}) = \langle\psi(\vec{x})|\, U_T^\dagger(\vec{\theta})\hat{O}U_T(\vec{\theta})\,|\psi(\vec{x})\rangle \tag{1}$$

The reason the output is an expected value is that, by quantum mechanics, measurement may yield one from different outcomes, each with some probability. The expectation value is therefore obtained by averaging several samples, each sample being a measurement obtained after rerunning the circuit on the same input. For our purposes, we measure a single qubit in the computational basis, yielding either 0 or 1.

The output is a function of the input and of the parameters $\vec{\theta}$, but the goal is to find $\vec{\theta}_{\text{opt}}$ such that the optimized network $f_{\text{opt}}(\vec{x}) \equiv f(\vec{x},\vec{\theta}_{\text{opt}})$ gives a certain desired behavior, such as classifying the input state from a list of classes. An exterior classical routine is charged with controlling the quantum circuit (QC) to achieve this. The routine can run the circuit, control its parameters and measure its outputs. Our routine attempts to find $\vec{\theta}_{\text{opt}}$ by minimizing a loss $\mathcal{L}$, as defined in section II D.

We only consider QCs that are initialized at $|0\rangle^{\otimes n}$, with the input data being classical $\vec{x}$ and then encoded following some feature map $|\psi(\vec{x})\rangle = U_E(\vec{x})\,|0\rangle^{\otimes n}$.

Fig. 1 shows a diagrammatic summary of this design. The QC is also shown, which is read from left to right: first, the input is encoded with U_E, then evolved with U_T and finally measured. The exterior classical routine evaluates the QC and minimizes the loss $\mathcal{L}$ to find $\vec{\theta}_{\text{opt}}$.

In the following subsections, the parts of the qNN are treated in more detail.

*Electronic address: jainaufo35@alumnes.ub.edu

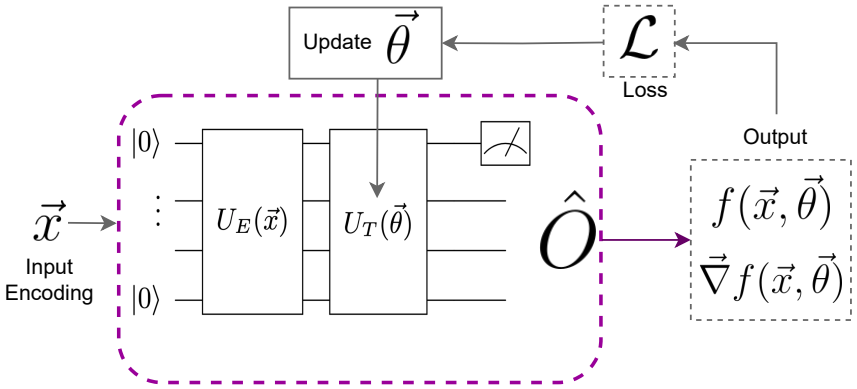


FIG. 1: Diagram of the quantum machine learning paradigm. Within the rounded box, the quantum circuit is shown. $U_E(\vec{x})$ encodes the input $\vec{x}$ into the circuit and $U_T(\vec{\theta})$ operates on this input, parametrized with some trainable parameters $\vec{\theta}$, and the result is measured. Outside, the classical minimization routine is shown, which iteratively updates the parameters $\vec{\theta}$ to minimize a loss $\mathcal{L}$ calculated from the measured output f. Gradients may also be calculated directly from the circuit, as drawn, which is explained in section II D

A. Gates

A QC evolves its qubits using a series of gates, mathematically described as unitary transformations U. The optimization problem equates to finding an optimum gate from the space of all possible gates. Generally, the parametrized gates $U_T(\vec{\theta})$ only span a subspace of all possible gates, therefore, the choice of parametrization is limiting. A good architecture for the circuit is crucial to ensure an optimum gate is achievable, while remaining simple to speed up computations and comply with the current NISQ era limitations.

By considering that in the end these algorithms must run on an actual quantum computer, the layer must be built using directly implementable, elementary, gates. Universality theorems ensure that even if limiting oneself to some elementary gates, as long as certain gates are available, any other gate may be implemented up to any desired precision [7].

In our case, we use as elementary gates the set of rotation gates $R_x(\phi), R_y(\phi), R_z(\phi)$ and the CNOT gate. With these, it is enough to define a general one qubit and two qubit transform (ignoring global phase shifts), which are our basic building blocks and are shown in Fig. 2. We refer to the simple two qubit gate shown there as such, and to the general 2-qubit as Full. They are simple to implement, as the parametrization $\vec{\theta}$ can be directly equated with the angles of the rotations.

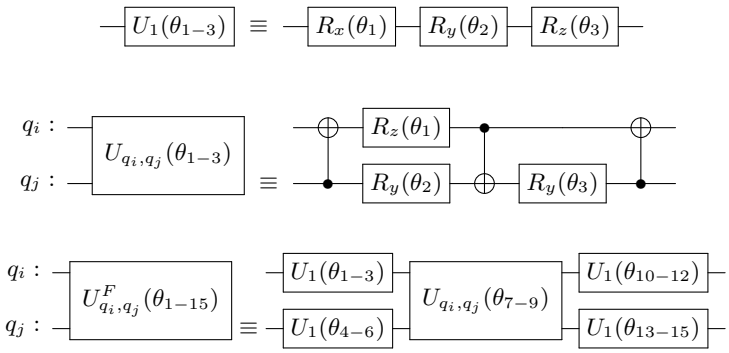
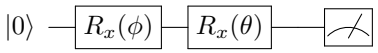


FIG. 2: Basic gate blocks used in this work. Top: Can represent any single qubit gate. Middle: Mixes two qubits. Bottom: Combination of other two, can represent a generic two qubit gate

B. Encoding

The choice of encoding is very important to get good results depending on the problem. As a simple example, a QNN can learn to calculate $\cos(\phi + \alpha)$ as a function of ϕ very easily using the circuit:



Knowing that $R_x(\theta)|0\rangle = \cos(\frac{\theta}{2})|0\rangle - i\sin(\frac{\theta}{2})|1\rangle$, by measuring with respect to Z the output of the QNN is $\langle\psi| Z |\psi\rangle = \cos(\phi+\theta)$. Clearly, this would solve the problem by learning $\theta = \alpha$. This is a very particular case, where a real number is encoded into an angle, and it was key for obtaining such a simple circuit. In this work, *basis encoding* is used, which consists in using the computational basis of a quantum circuit understood as bitstrings $\{|0\ldots00\rangle, |0\ldots01\rangle, \ldots, |1\ldots11\rangle\}$. By converting, and if necessary trimming, classical data into a binary representation, a one-to-one correspondence is made for each bit to a qubit. This is the method we use, which allows to directly compare a quantum circuit with a more intuitive classical (reversible) circuit.

C. Parametrized layers and output qubits

The following known properties: By the reversibility of a quantum circuit, extra ancilla/output qubits must be used if the function to implement is not reversible. Overparametrization leads to worse results [8]. Deeper, more complex, circuits are also worse performing in the NISQ era. On a real implementation of a quantum computer, the topology must be considered. Basic 2-qubit gates may only be directly used on connected qubits.

D. Learning and Gradients

Given and architecture and parametrization, this defines our input to output function in Eq. (1). To use it for solving a problem, a criterion for what defines a good solution must be given. For instance, our circuit output may represent an energy which must be minimized. The usual procedure is to define a loss functional $\mathcal{L}[f]$ to minimize. In our case, we are given some pre-labelled data $(\vec{x_i}, \vec{y_i})_{i=1,N_D}$, that is, we know the desired output of some inputs: $f_{\text{opt}}(\vec{x_i}) = y_i$. An attempt of a solution is given by minimizing an empirical loss. In our case we choose to use the mean squared error (MSE), which reads:

$$\mathcal{L}[f] = \frac{1}{N_D}\sum_{i}^{N_D}\left(f(\vec{x}_i) - \vec{y_i}\right)^2. \tag{2}$$

As $\vec{\theta}$ parametrizes the circuits, this allows to minimize the loss $\mathcal{L}(\theta) = \mathcal{L}[f(\theta)]$ by simple gradient descent:

$$\vec{\theta}^{(k+1)} = \vec{\theta}^{(k)} - \alpha\vec{\nabla}\mathcal{L}(\vec{\theta}^{(k)}), \tag{3}$$

with α being the learning rate, a positive real number. Some other methods may be used for their better convergence properties like speed, such as Adam. Explicitly, the gradient is:

$$\vec{\nabla}\mathcal{L}(\vec{\theta}^{(k)}) = \frac{1}{N_D}\sum_{i}^{N_D} 2\left(f(\vec{x}_i, \vec{\theta}^{(k)}) - y_i\right)\vec{\nabla} f(\vec{x_i}, \vec{\theta}^{(k)}). \tag{4}$$

Parameters are initialized as $\vec{\theta}^{(0)}$, and every iteration $(k) \to (k+1)$ is also called an epoch.

A surprising result of qML is that in some cases the gradient of f can be calculated reusing the same circuit. The derivative with respect to a parameter θ_k can be obtained with two reevaluations of f, shifting θ_k positively and negatively. This warrants its name as the parameter shift rule [9], equivalent to the idea of calculating the derivative of a trigonometric function like $f(x) = \sin(x)$ by exploiting that a cosine is a shifted sine. This is explicit for the example in section II B.

Explicitly, for our case where all parameters correspond to Pauli rotations, the gradient due to the parameter θ_k of a single Pauli Rotation is given by:

$$\partial_k f(\vec{x}, \vec{\theta}) = \frac{1}{2}\left[f(\vec{x}, \vec{\theta} + \frac{\pi}{2}\hat{e}_k) - f(\vec{x}, \vec{\theta} - \frac{\pi}{2}\hat{e}_k)\right]. \tag{5}$$

It must be understood that $\vec{\theta} = (\theta_1, \ldots, \theta_k, \ldots, \theta_{N_p})$, so $\hat{e}_k = (0, \ldots, 1, \ldots, 0)$ changes just the θ_k value. A proof of said rule is given in the Appendix. Interestingly, this implies the maximum eigenvalue of the measure operator $\hat{O}$ is an upper bound for the gradient. An apparent problem that comes when choosing a symmetric structure for the quantum neural network is that parameters will be reused for different rotation gates. In terms of implementation, this is not a problem, as although conceptually the parameter that determines the rotation of these different gates is the same, in terms of implementation each gate can be shifted independently. By the product rule, the total gradient will be obtained by adding these up.

III. IMPLEMENTATION

Two simple problems are devised to test our implementation. These consist in a parity check and an excitation check.

- *Parity*: Given n input bits, return the parity of the input. That is, return 0 (1) if an even (odd) number of them are 1.
- *Excitation*: Given n input bits, return 1 if more than half of them are 1.

These problems are simple and can be solved classically, so they lend well to compare results. These have also been chosen for their simple symmetry. Both are only functions the number of 1s and 0s, they are permutation invariant.

Our quantum neural network, strictly speaking differs from a canonical quantum convolutional neural network as first defined in [1]. Our circuit only implements the analogue of convolutional layers, which implement the classical idea of translationally symmetric layers where a same kernel (our 1 qubit or 2 qubit gates) are applied equally to neighboring qubits. A pooling layer should also be implemented, which mixes qubits together like our 2 qubit gate, but discards one of them from that point onward, effectively reducing the dimension. This pooling layer was not necessary for our simple problems, but it is very useful when thinking about big inputs like images.

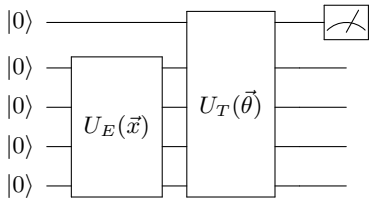


FIG. 3: Structure of our circuit for $N = 4$ and one readout qubit.

A. Comparing Libraries

These designs are implemented, as a simulation, with Python. Two libraries are used, *Qibo* and *Tensorflow Quantum* (TFQ), which differ in their approach. Qibo is more generic while TFQ is more focused on the machine learning aspect. As this is a simulation, the circuit is implemented with tensor operations (state vector and gate matrices), which TFQ exploits to offer gradients by automatically differentiating the circuit. Alternatively, TFQ also provides the option to calculate the gradient via the Parameter Shift Rule, which gives the same results but would also let us take sampling into account. This allows to compare this abstracted implementation with the more grounded Qibo implementation, where the Parameter Shift Rule is implemented manually. Fig.4 shows the evolution of the loss function in terms of the epoch for the encoding problem. Dashed lines represent the TFQ simulation and full lines the Qibo implementation. Being more precise, their difference is also plotted, and we find that the two implementations coincide within numerical precision of 32-bit floats.

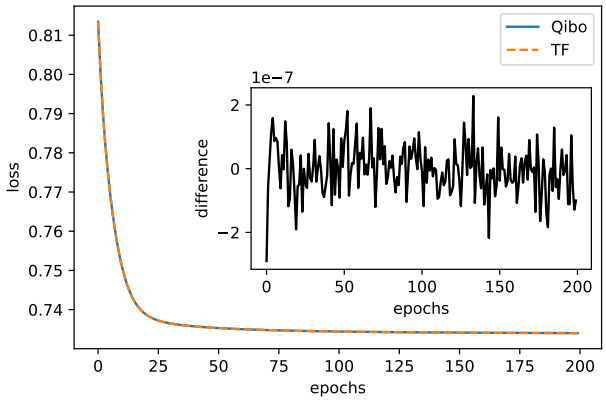


FIG. 4: Evolution of loss, dashed Tensorflow and filled Qibo, for each epoch during training of excitation problem with $N = 4$ input qubits and all initial parameters set to 1. The inside figure shows the difference between the two.

IV. RESULTS

We choose N input qubits and one output qubit (readout) to allow non-reversible outputs in terms of the input. Our problems are very classical, so we first explore what solution they may have.

- For the parity problem, the only thing needed is to CNOT all input qubits into the readout qubit, which when measured gives the parity. The CNOT gate is in the end a reversible XOR gate, strongly related to parity.
- For the excitation problem, the solution is less trivial. Thinking classically, an approach would be to use $\log_2(n)$ ancilla qubits where to add the number of excitations to then see if they amount to more than half. But these extra ancilla bits are not considered, as we want to minimize resources.

The architecture of our circuit will consist of 1-qubit gates for each of the input qubits, all using the same parameter, and then of 2-qubit gates connecting each qubit with the readout qubit. In the following sections, depth will refer to the amount of 2-qubit gates used, and when referring to the circuit as Full, it means general 2-qubit gates are used instead of the simple ones. Then, these problems are solved using our quantum neural network. First, for different N, two different training layers, one more complete than the other, are used to compare the convergence of the problems. The results are shown in Fig.5, which show the evolution of the loss for different ammount of qubits and depending on problem (colum) and architecture (row). The parity problem illustrates how the symmetries have made it so with the same number of parameters, our circuit can learn the solution to the problem for any number of qubits using the Full architecture. This is an ideal solution: the cost in terms of parameters, and therefore of depth too, is constant with the input size. As can be seen, with a layer of our choice of single qubit gates and simple two qubit gates, neither of the problems converge. By changing the two qubit gates into full ones, the parity problem converges, while the excitation one doesn't. This is coherent with the expected behaviour. The solution to the first problem just needs to find the equivalent of a CNOT cascade, while the second one has no direct solution with just two bit gates without more ancillary qubits. A certain scaling of the minimum loss achieved is observed.

The excitation problem showcases a different reality. As heuristically expected, just our general two gate convolutional layer is not enough to solve the problem, but it is much better. it is difficult to make general conclusions from these kinds of results, as we don't know the full shape of $\mathcal{L}[f]$, but this is why the next thing to be done is look at how the depth of our circuit, by adding more layers, can change this minimum observed loss. This is showcased in table IV, where it is observed how the minimum loss achieved needs a depth of 3, but with higher depth more local minima appear and we don't converge to the solution.

The results coincide with theoretical results on quantum classifiers. By itself, increasing the depth and the number of parameters clearly may only increase the subspace of accessible unitary gates U_T, as our gates can imitate the identity matrix. In spite of that, while this means that the minima attainable for $\mathcal{L}$ could be lower, the risk of getting stuck in local minima is much greater.

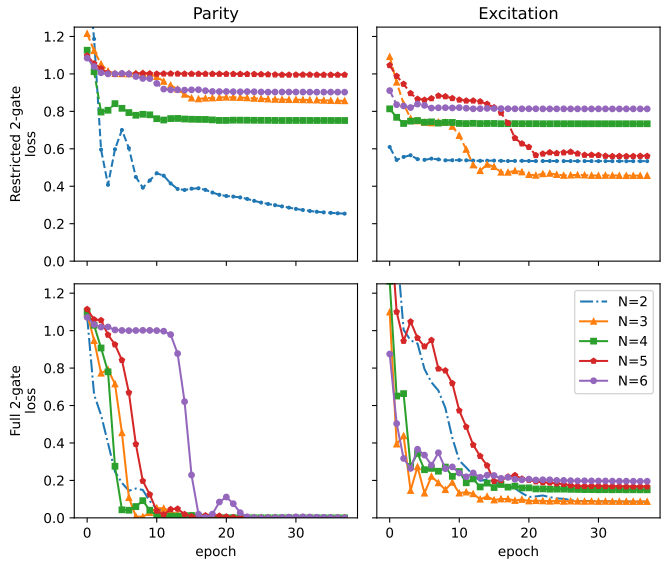


FIG. 5: Evolution of loss with epochs showing convergence for the two problems. First column corresponds to the parity problem and the second column to the excitation problem. First row corresponds to using simple two qubit gates and second row to using full two qubit gates

Run\Depth	1	2	3	4	5
#1	0.16388	**0.01682**	0.28086	0.19210	0.20765
#2	**0.14961**	0.24070	0.07499	**0.00136**	0.16617
#3	0.16387	0.03736	**0.00007**	0.17856	0.21864
#4	0.16402	0.02338	0.01088	0.23033	0.09737
#5	0.16391	0.05819	0.05413	0.00362	**0.00163**

TABLE I: Table showing loss values converged to after 1000 epochs for the excitation problem with $N = 4$ input qubits. The column indicates the depth of the circuit, 5 different runs are shown, and the lowest loss for each depth is given.

V. CONCLUSIONS

We have successfully implemented a convolutional quantum neural network. Implementing the whole while giving thought to each of its parts. These simple problems have served as tests to illustrate the power in using the symmetries of the problem, while also serving as simple starting points to expand further. For instance, these classically simple problems can be complicated by thinking less of the 1s and more about the X gates used to implement them. Thinking about the problem as excitations X of the ground state $|00\ldots0\rangle$ illustrates how if instead of the 0 state the initial state is prepared to be some other state $|\psi\rangle$, the problem becomes much less trivial and much more quantum. We can train to classify whether an input state is close to ψ or has been excited with respect to it.

On another note, while the performance of the quantum circuit has been evaluated by itself, there has been no direct classical analogue to compare. These problems were very classical, and if anything, a slower performance is to be expected. This is specially true for large n, where unless some alternative simulation methods are used, no matter how simple the circuit gets like in our parity case, simulating the 2^n Hilbert space becomes redundant and prohibitively large. Still, some alternative classical methods could be used to give some benchmarks.

Finally, the most relevant NISQ test would be to implement this circuit in an actual NISQ quantum computer. And to prepare for it, the formalism could be developed in terms of density matrices and implementing noise, to make concrete how it propagates, and to what extent the symmetries and learning process would compensate for it.

Acknowledgments

I would like to thank my advisor Dr. Arnau Rios Huguet for his dedication and for taking me into his wing. Moreover, I am very grateful for his and Javier Rozalén's weekly feedback, which has been crucial to shape this work as it has become. Finally, I would also like to thank Alejandro Romero for his comments, and my family and friends for their support.

[1] I. Cong, S. Choi, and M. D. Lukin, Nat. Phys **15**, 1273 (2019).
[2] J. W. T. Keeble, M. Drissi, A. Rojo-Francàs, B. Juliá-Díaz, and A. Rios, Phys. Rev. A **108**, 10.1103/physreva.108.063320 (2023).
[3] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New J. Phys. **18**, 023023 (2016).
[4] S. Efthymiou, S. Ramos-Calderer, C. Bravo-Prieto, A. Pérez-Salinas, D. García-Martín, A. Garcia-Saez, J. I. Latorre, and S. Carrazza, Quantum Sci. Technol. **7**, 015018 (2021).
[5] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, *et al.*, Tensorflow quantum: A software framework for quantum machine learning (2021), arXiv:2003.02989 [quant-ph] .
[6] Cirq Developers, Cirq (v1.4.0) (2024).
[7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, 2022 [11th edition]).
[8] Y. Du, Y. Yang, D. Tao, and M.-H. Hsieh, Phys. Rev. Lett. **131**, 140601 (2023).
[9] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, Phys. Rev. A **99**, 032331 (2019).

VI. APPENDIX

A. Parameter Shift Rule

Here the parameter shift rule is proven for unitary gates that can be expressed as $e^{-i\theta G}$, with G a Hermitian matrix with only 2 distinct eigenvalues.

Consider a parametrized quantum circuit $U_T(\vec{\theta})$ with measure operators $\hat{O}$. In the process of training our neural network, we evaluate the expectation value

$$f = \langle 0|^{\otimes n} U_T^\dagger \hat{O} U_T |0\rangle^{\otimes n}, \tag{6}$$

of which we want to calculate the gradient with respect to the training parameters. If the total circuit is built, or can be expressed, as the composition of gates depending on no more than one parameter: $U_T(\vec{\theta}) = \prod_k U_k(\theta_k)$, we may split the total unitary gate with respect to a single parameter θ as $U_T = LU(\theta)F$. Thus we write:

$$f = \langle \psi | U(\theta)^\dagger \hat{M} U(\theta) | \psi \rangle \tag{7}$$

With $|\psi\rangle = F|0\rangle^{\otimes n}$, $\hat{M} = L^\dagger \hat{O} L$

Using the chain rule, the gradient becomes

$$\partial_\theta f = \langle \psi | (\partial_\theta U^\dagger) \hat{M} U | \psi \rangle + \langle \psi | U^\dagger \hat{M} (\partial_\theta U) | \psi \rangle, \tag{8}$$

where, given our representation, $\partial_\theta U = -iGU = -iUG$. Due to G being a hermitian matrix with only 2 distinct eigenvalues $\lambda_m < \lambda_M$, we may center these around 0 by reexpressing G as:

$$G = \left(G - \frac{\lambda_m + \lambda_M}{2} I \right) + \frac{\lambda_m + \lambda_M}{2} I.$$

The first summand has eigenvalues $\pm\frac{\lambda_M - \lambda_m}{2} \equiv \pm r$, and we may neglect the second summand by considering how it affects U:

$$U = e^{-i\theta G} = e^{-i\theta \left[\left(G - \frac{\lambda_m + \lambda_M}{2} I \right) + \frac{\lambda_m + \lambda_M}{2} \right]} = e^{-i\theta G'} e^{i\theta \left[\frac{\lambda_m + \lambda_M}{2} \right]}$$

The rightmost term is a global phase, which is irrelevant. More explicitly, it cancels in equation 7. Thus, we can consider without loss of generality our 2 distinct eigenvalue hermitian matrices as having said eigenvalues $\pm r$.

Under this assumption, it follows that:

$$G^2 = r^2 I.$$

This, in turn, allows us to rewrite after Taylor expanding

$$U(\theta) = e^{-i\theta G} = \cos(r\theta) I - i r^{-1} \sin(r\theta) G, \tag{9}$$

which we use to develop 8, where we write $|\psi'\rangle = U|\psi\rangle$, as

$$\begin{aligned} \partial_\theta f &= \langle \psi' | iG\hat{M} | \psi' \rangle + \langle \psi' | \hat{M}(-iG) | \psi' \rangle \\ &= r \left(\langle \psi' | i r^{-1} G \hat{M} | \psi' \rangle + \langle \psi' | \hat{M}(-i r^{-1} G) | \psi' \rangle \right). \end{aligned}$$

Finally, rewriting this expression with a little algebraic trick, we obtain:

$$\begin{aligned} \frac{r}{2} \Big(&+ \langle \psi' | (I - i r^{-1} G)^\dagger \hat{M} (I - i r^{-1} G) | \psi' \rangle \\ &- \langle \psi' | (I + i r^{-1} G)^\dagger \hat{M} (I + i r^{-1} G) | \psi' \rangle \Big) \end{aligned}$$

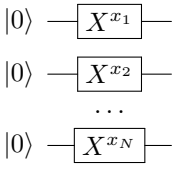
Using equation 9, we may express $\frac{1}{\sqrt{2}}(I \mp i r^{-1} G) = U\left(\pm\frac{\pi}{4r}\right)$, and bringing back our definition of $|\psi'\rangle = U|\psi\rangle$, using that $U(x)U(y) = U(x+y)$ given our exponential form and defining $s = \frac{\pi}{4r}$, we finally obtain:

$$\begin{aligned} \partial_\theta f = r \Big(&+ \langle \psi' | U(\theta + s)^\dagger \hat{M} U(\theta + s) | \psi' \rangle \\ &- \langle \psi' | U(\theta - s)^\dagger \hat{M} U(\theta - s) | \psi' \rangle \Big). \end{aligned}$$

Now this expression for the gradient is very useful, as it corresponds to using the same circuit to calculate f with θ shifted positively and negatively and then adequately combining them. This is similar to a two point finite difference formula, but it differs a bit with the scaling factor and it is analytical. For our single qubit rotation gates, $r = \frac{1}{2}$, $s = \frac{\pi}{2}$

B. Basis Encoding

While the basis encoding directly equates the input with a computational basis state, the actual initial state is still taken to be the $|00\ldots00\rangle = |0\rangle^{\otimes N}$ state. As a reminder, this is because $|0\rangle$ usually corresponds to the easier state to prepare, such as the ground state of an atom, while in comparison $|1\rangle$ usually represents an excited state. Therefore, the encoding circuit $U_E(\vec{x})$ must be implemented, which is simply:



Where it must be understood that each x_i is a bit.