Building a quantum perceptron

Nathan Habib
(supervisor: Nicolas Boutry)

Technical Report n^o2117, June 2021
revision 7ceb9453

With the continuous improvement of quantum technologies, one might wonder if it is possible to take advantage of them for machine learning. In this paper we present a first approach of quantum computing as well as the implementation of a quantum perceptron we then explain the reasoning behind these algorithms.

Les technologies quantiques étant en évolution constante, il est normal de se demander s'il est possible de prendre avantage de ces technologies dans le domaine de l'intelligence artificielle. Dans ce papier, nous présentons les rudimentaires de l'informatique quantique ainsi que l'implémentation d'un perceptron avec comme support une machine quantique et une explication de son fonctionnement.

Keywords
Machine learning; perceptron; quantum computing

Laboratoire de Recherche et Développement de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
nathan.habib@lrde.epita.fr – http://www.lrde.epita.fr/

Copying this document

Contents

Chapter 1

Introduction

Quantum computing technologies are gaining traction, we see heavy investments from governments and private companies as well as a rapid growth in the number of scientific publications each year Elsevier (2021); Inria (2020).

Quantum technologies are found to be very efficient to solve optimization problems. Thus, there is a lot of research today that examine the application of quantum technologies to solve those optimization problems (from portfolio optimization, to classification algorithms). Our goal here is first: to engage in a first approach with quantum technologies and try to apply it to machine learning problems; second: to pave the way to potentially more research by future students and professors by making our work as approachable as possible. We have to remind the reader that quantum technologies are still in a very early stage and are not guaranteed to be the revolution that many speak of, it is however a very promising field with many uses cases already in use, and we believe that it is worth our time and effort to explore this direction.

The motivation behind this research is the potential development in quantum technologies for machine learning applications. We knew very little about quantum technologies, the first step was to learn what was really being done and how. Then we had to figure out in what direction to go to, the building of a perceptron using quantum algorithms seemed to be the best choice as a perceptron is the building block of larger neural networks.

This paper is therefore structured in this way: First what is quantum computing and how can we use it to solve problems, then what is a classical perceptron then how can we theoretically implement a perceptron on a quantum device, to finally go in depth on how to implement it and how can we make it learn a simple function.

Chapter 2

What is quantum computing ?

The notion of quantum computing was first thought of by Richard Feynman in 1982. The idea sprung from the fact that a classical system cannot simulate a quantum system, so the first use case for quantum computers was simulating physics Feynman (1982).

A quantum computer is a device that uses specific properties of quantum mechanics to perform calculation, we call the action of writing and running algorithms on a quantum computer: quantum computing. Therefore, to answer what quantum computing is, we first need to focus on the core components of the quantum physic behind quantum computers, that is: *Superposition*, *Entanglement* and *reversibility* Hidary (2019a). First, we will look at how we can represent one *qubit*.

2.1 Qubit

A qubit is the computational basis of quantum computing. It can be compared to a *bit* on a classical computer. The difference is that instead of being either 0 or 1, it can be both at the same time, we say that it is in a superposition of the two states, 0 and 1. It can be represented using a *Bloch sphere* 2.1 Hidary (2019a).

2.2 Superposition

We can think of the law of quantum mechanics as having two *base states* $|0\rangle$ and $|1\rangle$ where $|0\rangle = \begin{pmatrix}1 & 0\end{pmatrix}^T$ and $|1\rangle = \begin{pmatrix}0 & 1\end{pmatrix}^T$. We will only be using the Dirac notation from now but remember it is only a more compact way of writing a column vector.

When we observe a quantum state, it must be one of those two base states, either $|0\rangle$ or $|1\rangle$. But before it is observed, a quantum state can be in a *superposition* of those two base states. By a superposition we mean a linear combination of those two base states. For example: we can have the quantum state ψ that is composed of 75% of $|0\rangle$ and 25% of $|1\rangle$. We therefore have:

$$\psi = \sqrt{\frac{3}{4}}\,|0\rangle + \sqrt{\frac{1}{4}}\,|1\rangle$$

We say that our quantum state is in a superposition of $|0\rangle$ and $|1\rangle$. When we measure our quantum state, the odds of observing one particular basis state is equal to the modulus of its coefficient squared. That is if we have a quantum state:

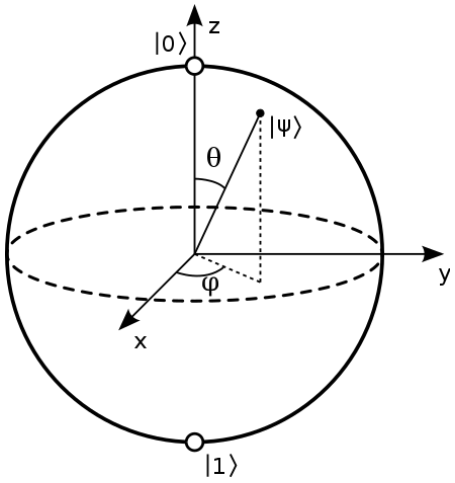


Figure 2.1: Bloch sphere representation of a qubit. The state $|0\rangle$ is at the top and the state $|1\rangle$ at the bottom. Everything in-between is a superposition of those two states.

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

The odds of ending up with $|0\rangle$ when measuring are: $|\,\alpha\,|^2$.

That is what is means for a quantum state to be in a superposition, of course we can take two random quantum state and superpose them again.

One more thing about superposition is that we can have phase differences between our two base states. For this reason, the coefficient are complex numbers.

To wrap your head around this concept, it helps to think about our basis states as waves, light for example. We already saw that light is composed of two basis states, but because it is a wave and not a unit of computation, we will use this notation: $|\uparrow\rangle$ and $|\rightarrow\rangle$. That is to say our light wave is composed of the basic light waves that either points up or left 2.2.

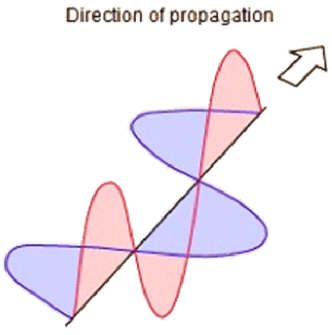


Figure 2.2: Two perpendicular light waves with no phase shift.

In this case, having a complex number as a coefficient means: the modulus represent the "strength" of the base wave in the quantum state and the argument is the phase shift of that

particular wave.

2.3 Entanglement

We say that two quantum states are entangled when measuring one of those states has an impact on the other.

Let's say we have two quantum states:

$$\psi_1 = \sqrt{\frac{1}{2}}\,|0\rangle + \sqrt{\frac{1}{2}}\,|1\rangle$$
$$\psi_2 = \sqrt{\frac{1}{2}}\,|0\rangle + \sqrt{\frac{1}{2}}\,|1\rangle$$

Those two states are composed of base states that have an equal probability of being measured.

If we aggregate those two states into one, we get [1]:

$$\psi_3 = \sqrt{\frac{1}{4}}\,|00\rangle + \sqrt{\frac{1}{4}}\,|01\rangle + \sqrt{\frac{1}{4}}\,|10\rangle + \sqrt{\frac{1}{4}}\,|11\rangle$$

This means we have an equal probability of having any combination of the first two states which is logical because we first measure ψ_1 and then ψ_2.

Now if we take this quantum state also, composed of two sub states:

$$\psi_4{}^{2} = \sqrt{\frac{1}{2}}\,|00\rangle + \sqrt{\frac{1}{2}}\,|11\rangle$$

Here we see that if we obtain $|0\rangle$ when measuring ψ_1 then we *must* obtain $|0\rangle$ when measuring ψ_2. We say that ψ_1 and ψ_2 are entangled.

2.4 Reversibility

All operators used in quantum computation must be reversible except for measurement. This is because we are dealing with quantum mechanics and any non-reversible operation means that we lost information and therefore performed a measurement.

[1] By aggregating, we mean using a tensor product $|0\rangle \otimes |1\rangle = |01\rangle$ Hidary (2019a).

[2] This is called a Bell state

Chapter 3

Quantum computers for machine learning

Now that we know about what makes quantum computing, "quantum", we will shift our focus into what has been done to apply those concepts to machine learning.

It is only natural to ask if quantum computing can give us any advantage in this area. Machine learning is about recognizing patterns in data. Classical computers are limited by the complexity of the patterns that can be found, quantum computers however are capable of producing much more complex patterns with much less resources Biamonte et al. (2017); Moret-Bonillo (2014).

For example, the K-nearest neighbors algorithm can be improved using quantum technologies:

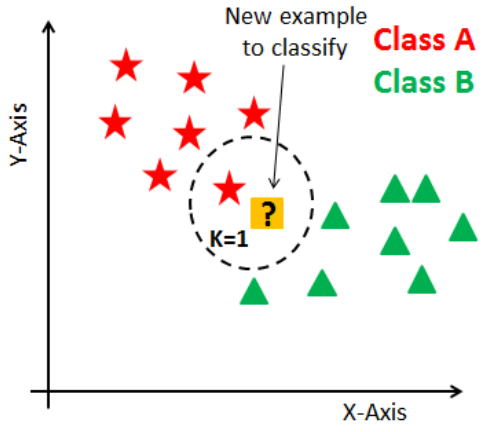


Figure 3.1: K-nearest neighbors algorithm.

This algorithm classifies points according to the class of its closets points. On a classical computer, the complexity needed to calculate the distance is polynomial but, Lloyd et al. (2013) claim that their quantum algorithm is more efficient.

Another domain where quantum computing could be beneficial is, neural networks. A neural network is a graph where each node is called a neuron and its connections to other neurons are weighted. Each neuron also has an activation function which decides on the value of a neuron based on the value of the neurons connected to it Goodfellow et al. (2016). A neural network is then a computational device with input, the original state of a subset of neurons and output, the final state of another subset Schuld et al. (2015a).

There are a number of proposal to simulate neural networks on quantum devices, however as of now, very few practical cases have been found, and there are still many challenges notably how to make a quantum neural net converge toward a solution Abbas (2020).

One first step would be to simulate a perceptron. Indeed, the perceptron is the building block of neural networks Minsky and Papert (2017).

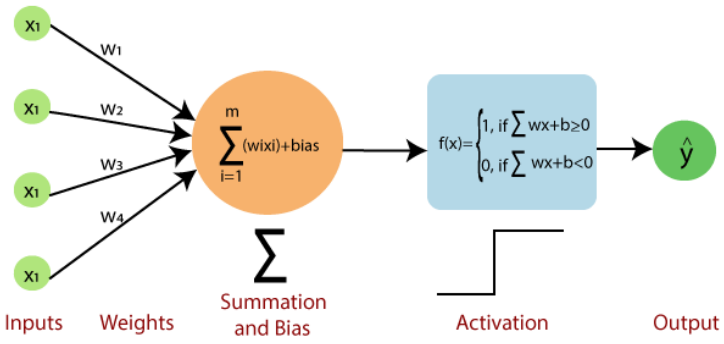


Figure 3.2: Classical Perceptron javatpoint (2021).

One of the challenges of reproducing classical technics with quantum devices is to find a way to encode our classical data and algorithms into quantum information.

Chapter 4

Simulating a perceptron.

We talked about the phase in Section 2.2 and that is how we will encode the output of the perceptron Schuld et al. (2015b).

The idea is to take a binary vector as input, each component of that vector has a weight associated to it. We make a quantum state from that vector and we shift the phase of that quantum state depending on the weights and the value of the input. We later evaluate that phase shift using a quantum algorithm named *Quantum Phase Estimation*. If the phase shift is more than 180 degrees, the output is one, otherwise it is 0.

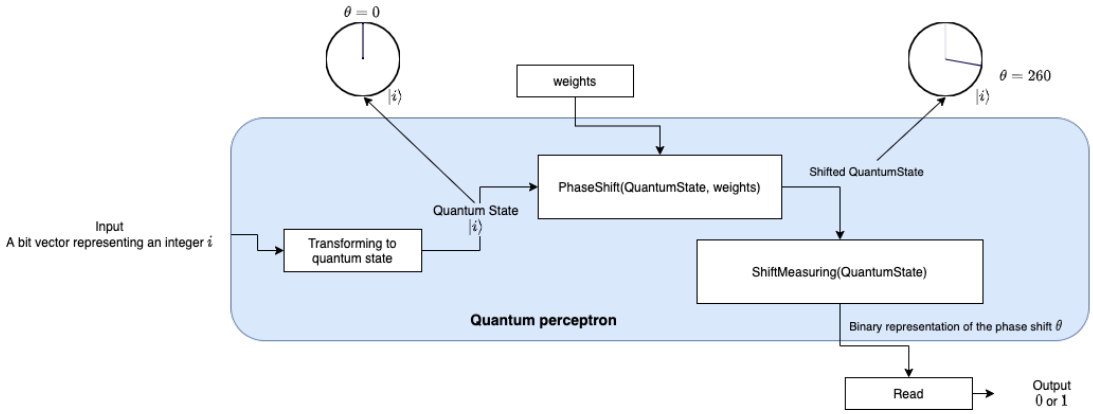


Figure 4.1: Abstraction of the workings of the quantum perceptron.

4.1 How to shift the phase.

What we want is a quantum operator that depends on the input, and a weight and, it will encode its output in the phase of the quantum state used as input Hidary (2019b); Johnston (2019a).

We have:

$$|\text{input}\rangle \leftarrow e^{2\pi\phi}\,|\text{input}\rangle$$

For example, let's say we have an input vector, $\begin{pmatrix}1 & 1\end{pmatrix}$, we first need to encode it in a quantum state: $|11\rangle$, the technics for doing this are hardware dependent, and we will not go in detail in this report.

The way we shift the phase of a quantum state is the following:

Because all quantum operations are linear, they can be represented as a matrix acting on the two basis states:

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

The superposition operation can be written as seen in Section 2.2, this is called the *Hadamard* operation or quantum gate.

$$\text{Hadamard} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$$

Now that we know that, to shift the phase of our quantum state by an amount $2\pi\phi$ we need an operation of the form.

$$\begin{pmatrix}e^{i2\pi\phi} & 0\\ 0 & e^{i2\pi\phi}\end{pmatrix}$$

However, we said that we wanted this phase shift to depend on the input, that means that when the input is $|0\rangle$ we shift by $-2\pi\phi$ and when the input is $|1\rangle$ we shift by $2\pi\phi$, the operation then becomes:

$$\begin{pmatrix}e^{-i2\pi\phi} & 0\\ 0 & e^{i2\pi\phi}\end{pmatrix}$$

We also need the phase shift to depend on the weight associated with the input quantum state, for a perceptron with N inputs, there is N PhaseShift operations each being defined as:

$$\text{PhaseShift}(w_k) = \begin{pmatrix}e^{\frac{-i2\pi w_k}{2N}} & 0\\ 0 & e^{\frac{i2\pi w_k}{2N}}\end{pmatrix}$$

The gate PhaseShift_k is applied to the k^{th} input, along with a global phase shift of π. In the end, all the inputs have received a phase shift of π plus their respective shift depending on the weight associated with them (w_k).

We end up with a quantum gate, PhaseShift, that will shift the phase of a one qubit quantum state by an amount depending on the value of that quantum state and the weight associated to that state.

Next we will see how to manually find the weights to reproduce the *NOT* function.

4.2 Circuits parameters.

Suppose we want to reproduce the *NOT* function. That is, when input is 1 we want 0 as output, and when input is 0 we want 1 as output.

To do this we will want a weight associated with our input that produces a phase shift greater than 180 degrees when our input is 0 and, lower when the input is 1.

We use two quantum gates on our input, the first one shifts the phase by pi radian and the second by either $\frac{-2\pi w_1}{2N}$ or $\frac{2\pi w_1}{2N}$

The total shift then is:

$$i\pi - \frac{i2\pi w_1}{2N} = 2i\pi(\frac{w_1 - 1}{2}) \text{ when } x = 0$$
$$\text{and}$$
$$i\pi + \frac{i2\pi w_1}{2N} = 2i\pi(\frac{w_1 + 1}{2}) \text{ when } x = 1$$
$$\text{We take, } \frac{w_1 - 1}{2} = \phi_0$$
$$\text{and } \frac{w_1 + 1}{2} = \phi_1$$

Now remember we want the phase shift to be greater than half a turn (180 degrees) when $x = 0$ than means that ϕ_0 needs to be greater than $\frac{1}{2}$.

For the same reason, we want ϕ_1 to be lower than $\frac{1}{2}$.

With that we find that w_1 needs to be between -1 and 0. We find that $w_1 = -0.5$.

Because we are talking about rotations around the unit circle, the weights are modulo 1.

4.3 How to measure the phase.

When measuring a quantum bit, we will only get 0 or 1, there is no way of having information about the phase of a quantum state by measuring it Johnston (2019b).

A solution is to use an algorithm that translates the phase shift of or quantum gate "Phase-Shit" to the amplitude of our quantum phase. Because we can actually measure the amplitude of a quantum state, we will be capable of measuring the phase shift caused by our quantum gate.

This algorithm is called *Quantum Phase Estimation (QPE)* Hidary (2019c); Johnston (2019b).

It is a very complex algorithm and understanding it fully takes time, if you want all the details we encourage you to read: Hidary (2019c); Johnston (2019b); Kang (2020).

The main idea of the algorithm is to take the gate we want to measure the phase shift from and use it to shift the phase of a quantum state but in such a way that the individual quantum states underlying the main one see their phase shifted with a certain frequency. This frequency can then be read using the *Quantum Fourier Transform algorithm*. Because this frequency is directly linked to the amount by with our quantum gate shifts the phase, we can determine it easily.

4.4 Determine the precision.

When Using the QPE, we need to choose the precision we want to be using when evaluating our phase shift, Schuld et al. (2015b) goes in detail about that issue. We will here briefly explain why we need a precision of at least 2 qubits.

We chose a weight equal to -0.5, when we do the calculation we see that the rotation will then be a quarter of a turn and three-quarter of a turn depending on the value of our input.

The output of the QPE is the binary fraction representation of our phase shift.

For example if our phase shift is: $\frac{1}{4}$ then the output of the QPE with precision two, meaning using two qubits to encode the output, will be 1 or 01 in binary. This is because with two qubits we can encode 4 values and 1 is a fourth of 4. If our phase shift was $\frac{3}{4}$, the result would have been 3 or 11 in binary.

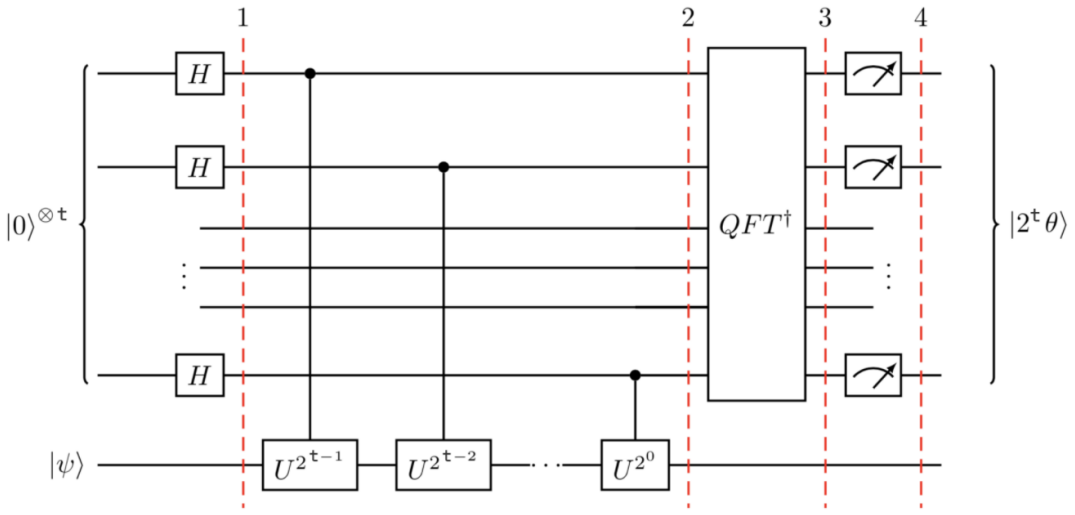


Figure 4.2: Quantum Phase Estimation circuit. This cicruit make use of the tensor product notation .

A precision of 1 qubit will not work, because with 1 bit we can only have two different values therefore we cannot have a quarter of the total number of values (we can only have one half).

That explains why we need to have our output encoded on a minimum of two qubits.

Chapter 5

The implementation

We know have the main building blocks to construct our perceptron on a real machine.

We decided to use the platform Qiskit IBM (2021), developed by IBM. Qiskit is a python library used for the development of quantum algorithm and applications. We chose it because it has a large community and the visualization tools it offers are the best from any other platform.

The code for the perceptron can be found on github.

When working with actual code, we face the limitations of the machine, and the implementation details of the library.

Notably, we had to find a way to implement the gates in a way that would make it easy for us to experiment, scale, read and comprehend for when future researcher need to use our code.

Also, the QPE algorithm makes use of the quantum Fourier transform, however in qiksit this algorithm and its inverse are swapped in comparison to the ones used in Schuld et al. (2015b). It does not matter as long as we stay coherent in the way we handle the Quantum Fourier Transform and its inverse. Although this seems trivial now, it caused a lot of confusion when writing down the code for the perceptron.

When running the circuit, we can choose to either run on an emulator or to run on an actual quantum device, hosted by IBM. The notebook is made so that we run on an emulator because of the flexibility it offers.

5.1 Implementing the PhaseShift Gate

Because we have access to only a few quantum gates when working on physical devices, we have assembled them in order to make the gates we need.

In the case of the PhaseShift gate, Qiskit only gives us access to a gate that applies a shift to the second component of the quantum state:

$$\mathrm{U1} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$

We also have access to the X gate:

$$\mathrm{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

By combining those two gates we can make our PhaseShift gate. $XU1(-\lambda)XU1(\lambda)\,|\psi\rangle = \mathrm{PhaseShift}\,|\psi\rangle$

Then, if we want to be able to use our gate with the QPE algorithm, we need to make it *controlled*. That means that the gate will be active, if and only if, another qubit is set to $|1\rangle$.

In code that gives us:

```
def Cperceptron_U(quantum_circuit, w, n, qubit, control_qubit, control_count):
    theta = ((2 * pi * w) / (2 * n)) * control_count
    quantum_circuit.cp(theta, control_qubit, qubit)
    quantum_circuit.cx(control_qubit, qubit)
    quantum_circuit.cp(-theta, control_qubit, qubit)
    quantum_circuit.cx(control_qubit, qubit)
    return quantum_circuit.to_gate(label="U" + str(qubit+1))
```

5.2 Putting it all together

Now that we have our function PhaseShift gate, we only need to use it with the QPE algorithm. Here is the fully detailed circuit, for an input of size 1 and a precision (output) of size 2.

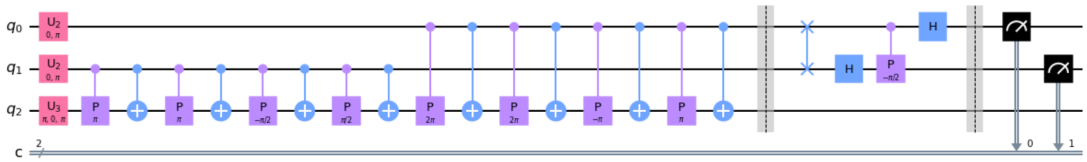


Figure 5.1: Fully detailed circuit. One line represents one qubit. The two U2 gates are Hadamard gates used to put the qubits in superposition, the U3 gate at the bottom is a X gate used to make the qubit go from $|0\rangle$ to $|1\rangle$. As we saw in 5.1 one PhaseShift gate is composed of fourunderlying controlled gates. The control qubit is the

5.3 Walkthrough

In this section, we will take an example and go through the algorithm step by step.

1. First we put our two top qubits in a superposition with the two Hadamard gates.
2. We then apply our controlled PhaseShift gate (U(w)) to get our top two qubits in the state 2 in 5.3.
3. Then we do the same but, we apply U(w) two times and, we end up in the state 3 in 5.3.
4. We then analyze the frequency at which the phase changes in the quantum state made up of the first two qubits.
5. Finally, we read the first qubit, this will tell us if the phase shift greater than 180 degrees.

Now that we know that our perceptron works with hard-coded weights, we need to make it learn by itself.

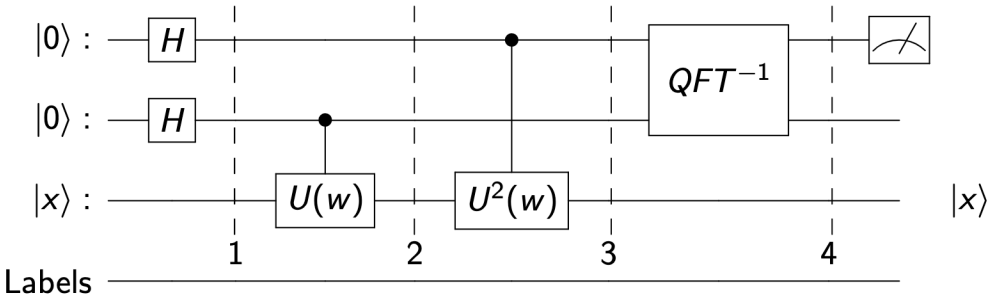


Figure 5.2: Circuit of the perceptron simulating the *NOT* function.

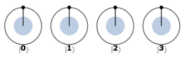
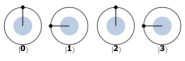
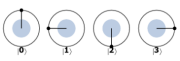
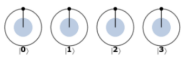
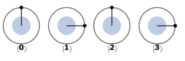
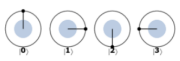
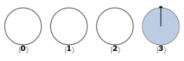
$\|x\rangle$	1	2	3	4	Output
1					0
0					1

Figure 5.3: Table representing the state of the top two qubits for the two different values of $|x\rangle$ in 5.2 at moment $k \in [1; 4]$.

Chapter 6

Make it learn

One thing we know about quantum operations is that they are all differentiable.

We chose to use a python library called Pennylane PennyLane (2021). Pennylane allows us to build quantum circuit, differentiate them and make them learn and input output relation, just like a neural network model.

To do this, we then needed to rebuild the circuit in the Pennylane framework. Build an error function, and choose an optimizer.

The code is available here.

6.1 Automatic differentiation

Pennylane uses automatic differentiation on the quantum circuit it is given Mitarai et al. (2018); Schuld et al. (2019).

The technic used is called *Parameter-shift rules*. The idea is that the gradient of a quantum circuit can be expressed as a linear combination this same quantum circuit but with a shift in its arguments. This means that partial derivatives of a variational circuit can be computed by using the same variational circuit architecture.

This is similar to how the derivative of the function $f(x) = sin(x)$ is identical to

$$\frac{1}{2}sin(x + \frac{\pi}{2}) - \frac{1}{2}sin(x - \frac{\pi}{2})$$

The same, function, sin, can be used to compute $sin(x)$ and its derivative.

6.2 Error function

We use the mean square error function.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} prediction_i - label_i{}^2$$

It is important to not that the mean square error is a classical function. Pennylane uses the quantum circuit as component of a larger circuit composed of classical and quantum components, and calculate the gradient of the whole circuit.

6.3 Optimizer

The optimizer is a simple gradient descent optimizer.

$$w^{t+1} = w^t - \eta \nabla_w f(x; w^t)$$

This means that the next iteration of weights simply is equal the previous iteration minus the gradient times a learning rate.

6.4 Results

For training, we chose to simulate the *NOT* function so that we can later compare the set of weights found for our model.

The circuit and training parameters are:

- input length = 1
- precision (output) = 2
- learning rate = 0.05

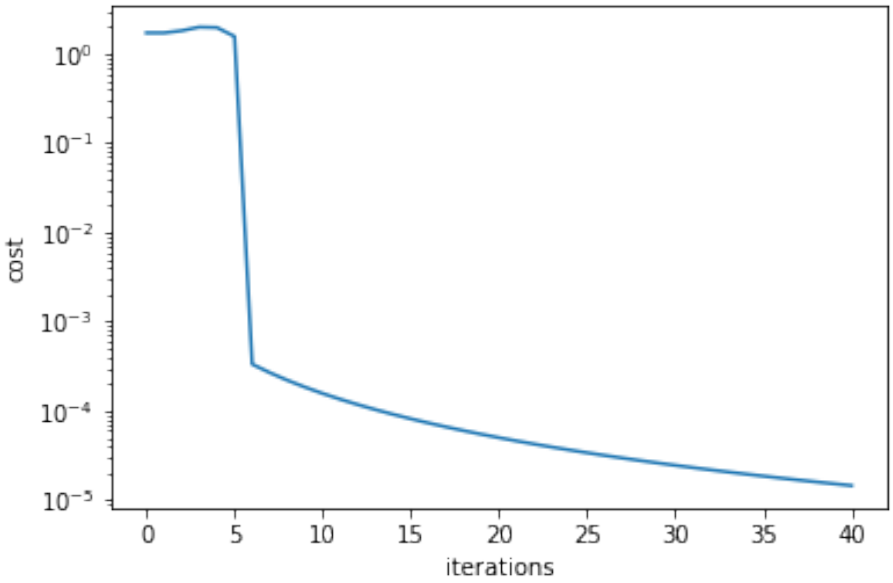


Figure 6.1: Evolution of cost with respect to the number of iterations.

We can see from the graph, 6.1, that we get very good results for that simple function. A next step would be to try and simulate more complicated functions. And compare the performance to existing methods both classical and quantum.

Chapter 7

Conclusion

This research had four goals: To have a first contact with the world of quantum computing and understand its basic principles; To see what was being done in the field of quantum AI; To get to know a few of the existing frameworks and finally to lay a foundation for future research.

Even though it took more time and effort than we first imagine, all those goals were met.

It is important to note that as of now, we do not recognize any advantages for quantum technics compared to classical ones. That is because all the gain that we potentially gain are wiped up by the process of making our classical data into quantum data.

Further research would be to use our perceptron to learn more complex functions, and to compare our model to the state of the art and compare to its classical equivalent. Our perceptron also does not seem to use the potential of quantum technologies, by that we mean that one could imagine a quantum perceptron that could learn about all the possible input output pairs at the same time, by putting them in superposition. Even though we have no idea how to implement such an algorithm, it is certain that such an advantage would make quantum computing even more interesting for machine learning.

Chapter 8

Bibliography

Abbas, A. (2020). Are quantum neural networks actually relevant? by amira abbas. https://www.youtube.com/watch?v=aU8XBjG5tAw&t=2193s. (page 9)

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671):195–202. (page 8)

Elsevier (2021). Quantum computing research trends report. (page 4)

Feynman, R. P. (1982). Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7). (page 5)

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. (page 9)

Hidary (2019a). *Quantum Computing: An Applied Approach*. Springer International Publishing. (pages 5 and 7)

Hidary (2019b). *Quantum Computing: An Applied Approach*, chapter 3. Springer International Publishing. (page 10)

Hidary (2019c). *Quantum Computing: An Applied Approach*, chapter 9. Springer International Publishing. (page 12)

IBM (2021). Qiksit documentation. (page 14)

Inria (2020). Who are the main players in the world of quantum computing? (page 4)

javatpoint (2021). Single layer perceptron in tensorflow - javatpoint. (page 9)

Johnston, Harrigan, G.-S. (2019a). *Programming Quantum Computers*, chapter 2. O'Reilly Media, Inc. (page 10)

Johnston, Harrigan, G.-S. (2019b). *Programming Quantum Computers*, chapter 8. O'Reilly Media, Inc. (page 12)

Kang, H. (2020). Quantum phase estimation. (page 12)

Lloyd, S., Mohseni, M., and Rebentrost, P. (2013). Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*. (page 8)

Minsky, M. and Papert, S. A. (2017). *Perceptrons: An introduction to computational geometry*. MIT press. (page 9)

Mitarai, K., Negoro, M., Kitagawa, M., and Fujii, K. (2018). Quantum circuit learning. *Physical Review A*, 98(3). (page 17)

Moret-Bonillo, V. (2014). Can artificial intelligence benefit from quantum computing? *Progress in Artificial Intelligence-Springer*. (page 8)

PennyLane (2021). Pennylane documentation. https://pennylane.readthedocs.io/en/stable/. (page 17)

Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., and Killoran, N. (2019). Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3). (page 17)

Schuld, M., Sinayskiy, I., and Petruccione, F. (2015a). An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185. (page 9)

Schuld, M., Sinayskiy, I., and Petruccione, F. (2015b). Simulating a perceptron on a quantum computer. *Physics Letters A*, 379(7):660–663. (pages 10, 12, and 14)