Approaches to Avoid Overfitting in a Quantum Perceptron

Fernando M. de Paula Neto, Gustavo I. S. Filho, Cláudio A. Monteiro
Centro de Informática - CIn
Universidade Federal de Pernambuco - UFPE
Recife, Cidade Universitária, 50670-901
Email: {fernando, gisf, clam}@cin.ufpe.br

***Abstract*—In this paper, we analyze an existing Quantum Perceptron in terms of its generalization performance and introduce alternative strategies to avoid classification errors, the Circuit Threshold Operator (CTO) and Neuron Threshold Variation (NTV). The adjust of the CTO and NTV parameters increases the probability of the neuron to tolerate differences between input and stored weights allowing some noise level to be acceptable. Experiments were conducted in IBM quantum simulator, showing that our proposals are effective when compared to the original fixed parameter. A hybrid classical-quantum training was executed, using Genetic Algorithm to train the parameters of the model classically and running our model in a quantum simulator. Our solution achieves 78.57% and 76.43% of accuracy on a noisy dataset, while the original neuron approach does not exceed 53.57%.**

***Index Terms*—quantum perceptron, quantum neuron, learning algorithm, quantum machine learning, quantum computing, genetic algorithm**

I. Introduction

Quantum computing is being developed since the 60s with Richard Feynman and in recent years the area has gained strength due to technological advances in the manipulation of matter at the atomic level [1], [2]. Noisy Intermediate Scale Quantum (NISQ) computers are now available for execution of quantum algorithms as well as simulators built with distributed processing on classical computers that allow the use of high level programming languages [3]–[8]. Some benefits of quantum computers are the speed of processing and the multidimensional capability of storing large vector spaces due to the its quantum mechanical properties [9]. Some existing quantum algorithms solve problems involving optimization [5], [10]–[13], which can be applied to develop quantum machine learning applications [14].

Machine learning applications are being widely used around the world for a wide variety of tasks such as speech and facial recognition, process optimization in industries, knowledge discovery, autonomous vehicles and more. With the exponential growth of data storage, availability and complexity, one main challenge is to optimize the processing of machine learning algorithms. Quantum learning algorithms may be a solution for large data mining applications and for the development of new types of learning algorithms based on quantum mechanics.

The authors would like to thank CNPq for financial support.

In this work, we analyze a quantum perceptron in perspective of its generalization ability in a noisy dataset. Based on the limitations found in the neuron, we propose two approaches in which the degree of generalization of the neuron classification increases. In Section II, some concepts of quantum computing are described to explain our proposed approach. Section III presents quantum existing neurons and the quantum perceptron model. In Section IV, we propose two approaches to avoid classification errors in a quantum perceptron learning and, in Section V, we explain and analyze the experiments. Finally, Section VI presents the conclusion of the work and future study directions.

II. Quantum computing

A. Quantum bits

The classic computer enabled humanity to develop complex information systems reaching all fields of life. The most basic unit of classical information is the bit, which takes a value of 1 or 0 based on high and low voltage. With the advance of quantum mechanics, it was possible to conceive a quantum computer that operate using properties subjected to the laws of quantum mechanics. The quantum bit, or *qubit*, is an unit vector in a two-dimensional complex vector space that can store information as a superposition of 0 and 1 at the the same time [1], with a given probability $|\alpha|^2$ to read the '0' state and probability $|\beta|^2$ to read the '1' state. The state normalization ensures that $|\alpha|^2+|\beta|^2=1$. The canonical (or computational) basis is composed of the vectors $|0\rangle = [1,0]^T$ and $|1\rangle = [0,1]^T$, where $|\cdot\rangle$ is a notation introduced by Dirac to represent quantum states [15]. The $\langle\cdot|$ notation represents the complex conjugate of the vector $|\cdot\rangle$. As such any qubit can be seen as a linear combination (usually called *superposition*) of the basis vectors, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers.

The quantum state is the representation of an isolated quantum system with a probability distribution for each observable qubit. A multiple qubit system is mathematically represented by the tensor operator $\otimes$. In a two qubit system for example, $|\psi_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ and $|\psi_2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$, the tensor operator produces the state $|\psi_1\psi_2\rangle = \alpha_1\beta_1|00\rangle$ + $\alpha_1\beta_2|01\rangle$ + $\beta_1\alpha_2|10\rangle$ + $\beta_1\beta_2|11\rangle$.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

B. Quantum operators

Quantum information can be altered with the use of unitary gates that acts as unitary operations over the quantum system. For the purpose of this work, we describe the operators: Identity $\mathbf{I}$, NOT $\mathbf{X}$, Hadamard $\mathbf{H}$ and Pauli-Z Gate $\mathbf{Z}$. The Identity operator $\mathbf{I}$ generates the output exactly as the input; $\mathbf{X}$ operator works as the classic NOT in the computational basis, flipping the value of the qubit; Hadamard $\mathbf{H}$ generates a superposition of states when applied in a computational basis; and the $\mathbf{Z}$ gate inserts a negative in front of the state applied when the qubit is not zero.

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{matrix} \mathbf{I}\,|0\rangle = |0\rangle \\ \mathbf{I}\,|1\rangle = |1\rangle \end{matrix}$$

$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{matrix} \mathbf{X}\,|0\rangle = |1\rangle \\ \mathbf{X}\,|1\rangle = |0\rangle \end{matrix}$$

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \begin{matrix} \mathbf{H}\,|0\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle) \\ \mathbf{H}\,|1\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle) \end{matrix}$$

$$\mathbf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \begin{matrix} \mathbf{Z}\,|0\rangle = |0\rangle \\ \mathbf{Z}\,|1\rangle = -\,|1\rangle \end{matrix}$$

In addition to the single qubit unitary gates, there are controlled operators, which act on several qubits and act as the conditional structures (*i.e.* if / else) of programming languages. The $\mathbf{CNOT}$ is an operation between two qubits where one is the control and the other is the target. When the control is 1, $\mathbf{X}$ is applied on the second one, that is, the target. In the same way, $\mathbf{CZ}$ is a 2-qubit gate that applies the control in the first qubit and $\mathbf{Z}$ to the target.

$$\mathbf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{matrix} \mathbf{CNOT}\,|00\rangle = |00\rangle \\ \mathbf{CNOT}\,|01\rangle = |01\rangle \\ \mathbf{CNOT}\,|10\rangle = |11\rangle \\ \mathbf{CNOT}\,|11\rangle = |10\rangle \end{matrix}$$

$$\mathbf{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \quad \begin{matrix} \mathbf{CZ}\,|00\rangle = |00\rangle \\ \mathbf{CZ}\,|01\rangle = |01\rangle \\ \mathbf{CZ}\,|10\rangle = |10\rangle \\ \mathbf{CZ}\,|11\rangle = -\,|11\rangle \end{matrix}$$

There is also the concept of negative controlled gates where in the $\mathbf{CNOT}$ operator, the control can be set to be 0 instead of 1 for the action of operator X to exist.

C. Quantum circuit

We can represent quantum algorithms by quantum circuits. This graphical representation considers the qubits as wires and quantum operators as boxes. The flow of execution, as in classical circuits, is from left to right. Figure 1 is an example of a quantum circuit composed of $\mathbf{X}$ gate and a $\mathbf{CNOT}$ operator, where the control qubit is depicted by a filled circle and the symbol $\oplus$ indicating the target qubit. The negative controlled gate is drawn on a quantum circuit with an open bullet (indicating control-on-zero), but can be also built using

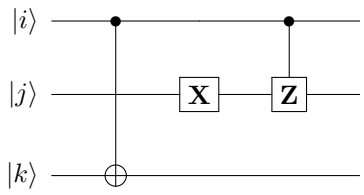

Fig. 1. An example of quantum circuit using some operators. From left to right: $\mathbf{CNOT}$ operator, using the notation of filled circle and $\oplus$ indicating that the first qubit is the control and the third one is the target, a $\mathbf{X}$ operator in the second qubit, and the $\mathbf{CZ}$ operator using the notation of filled circle and $\mathbf{Z}$ gate, between the first and the second qubits.

$\mathbf{CNOT}$ and $\mathbf{X}$ gates. Figure 2 shows the equivalence between the two approaches.

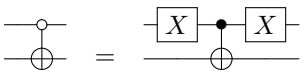

Fig. 2. Equivalence between quantum circuits using negative CNOT and with only CNOT and $\mathbf{X}$ gates.

We call, in this paper, $\mathbf{CNOTs}_j^N$ to generalize a $(N)-$ary $\mathbf{CNOT}$ having $N-1$ control qubits and the last qubit being the target. The control qubits form a control sequence that depends on j and N which are decimal numbers. Turning j into a binary number of size N, the controls-on-one are in the position of that binary number has 1 and the controls-on-zero are in the positions for that binary number has 0. An example of $\mathbf{CNOTs}$ is given in Figure 3, where $j = 20$ (10100 in binary string) and $N = 6$ qubits.

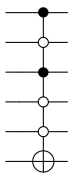

Fig. 3. Example of a $\mathbf{CNOTs}$ operator, $\mathbf{CNOTs}_{j=20}^{N=6}$.

Other N-controlled gate that we use in this work is the $\mathbf{C^N Z}$ operator to generalize an N-multiple $\mathbf{CZ}$ having $N-1$ control qubits, and the last qubit being the target. In practice, when all the qubits involved in the operation are 1, the operator includes a negative phase (negative sign) in front of the quantum basis state.

III. Quantum Neurons

There are proposals for quantum neural networks (qNN) with different approaches that try to combine the advantages of quantum computing with the powerful and parallel processing of neural computing [16]. In general, these approaches use the paradigm of processing information using qubits, such as classical computing, and using quantum logic gates to manipulate their information [14], [17]–[20].

However, some effort has recently been made to represent the information to be processed in the amplitudes of the quantum states, a paradigm referred to as "amplitude encoding". This paradigm concerns a non trivial way of building algorithms, since it deviates from the usual paradigm in computing [9], [21]. The benefits of representing information in the amplitude are diverse, but the most important is to minimize the amount of qubits used, making it possible to run the algorithms on available quantum processors [3]. There is another discussion about storing information in the quantum state as a nonlinear mapping of data, which allows for computations that are classically intractable by kernel functions [22]–[24].

In this work, we analyze the neuron generalization potential proposed in [9], carrying out experiments with available quantum simulators and artificial datasets. In developed experiments, we found that when the neuron has a high acceptance threshold rate, it starts to function as a memory, with no power to generalize and tolerate noise in patterns. In order to increase the generalization power of the circuit, with minimal impact on performance, we propose an operator coupled to the neuron to increase its generalization power against noise bases which we call Circuit Threshold Operator (CTO).

Tacchino *et al.* [9] made a proposal of a quantum neuron based on the classical Rosenblatt's "perceptron", using the paradigm of amplitude enconding, as stated. As the neuron input and weight values are binary, $i_j, w_j \in \{-1, 1\}$, it was proposed that these inputs and weights be encoded in the amplitudes of the quantum states. Equation 1 represents input and weight states where m is a vector size and the amount of amplitudes.

$$
\begin{aligned}
|\psi_i\rangle &= \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} i_j \, |j\rangle \\
|\psi_w\rangle &= \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} w_j \, |j\rangle \\
\vec{i} = \begin{pmatrix} i_0 \\ i_1 \\ \vdots \\ i_{m-1} \end{pmatrix}, &\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{m-1} \end{pmatrix}
\end{aligned}
\tag{1}
$$

This type of quantum state that has -1 and 1 amplitude values is known as *real equally weighted* (REW) state and can be mapped as a quantum hypergraph [25]. In particular, hypergraphs quantum states can be mapped into vertices and hyperedges of generalized graphs and can be prepared using only qubits and multi-controlled **Z** gates. Based on hypergraph generator algorithm, it was possible to propose an algorithm [9] that generates a circuit for a given REW quantum state, without using a brute force search for this representation. In this way, a quantum input and weight state can be efficiently transformed into the quantum circuit. This algorithm was called "Hypergraph states generation subroutine" (HSGS) by its authors and is detailed in Algorithm 1 [9].

Algorithm 1: HSGS Based Algorithm proposed in [9].

```
   Result: Quantum circuit QC;
1  Input: a given v = {−1, 1}^(2^N) input vector;
2  Auxiliary variables: vAux = {1}^(2^N) vector;
3  if v[0] is -1 then
4  |  Flip all the values of v, i.e. v = -1 . v;
5  end
6  for i = 0; i < len(v); i++ do
7  |  if v[i] is -1 and there exist only one qubit is in
   |     IntToBin(i) string then
8  |  |  Put Z gate on the circuit QC in the position of
   |  |    the one qubit of the state IntToBin(i);
9  |  |  Update vAux vector, considering that a signal
   |  |    change was applied in the position i;
10 |  end
11 end
12 for p = 2, p < N, p++ do
13 |  for i = 0; i < len(v); i++ do
14 |  |  if IntToBin(i) has p bits equal to one then
15 |  |  |  if vAux[i] is not equal to v[i] then
16 |  |  |  |  Put C^pZ gate on the circuit QC
   |  |  |  |    between the p qubits that have values
   |  |  |  |    1 in IntToBin(i);
17 |  |  |  |  Update vAux vector considering that a
   |  |  |  |    signal changed was applied in j
   |  |  |  |    positions of vAux which p qubits that
   |  |  |  |    have 1 value in IntToBin(i) are in the
   |  |  |  |    same position in IntToBin(j);
18 |  |  |  end
19 |  |  end
20 |  end
21 end
```

The operator U_i and a part of the U_w, which form the quantum neuron, are quantum operators which generate the neuron input and weight quantum states, respectively.

The role of the HSGS algorithm is to modify the amplitudes of the quantum state with N qubits so that they have the same values as a given vector **v**, of size 2^N, and which has only -1 and 1 as values. To create the quantum operators of this circuit that makes this transformation, the first step is to verify if the value on the first position of this vector is -1 (line 3). If so, change the sign of every position on the **v** vector (line 4). This operation indicates that the neuron with weights **w** has the same functioning as the neuron with weights $-\mathbf{w}$.

Considering each index of the vector in its corresponding representation of binary string, it is verified the binary string representation that has only one '1' and that has its value in the vector equal to -1. In these states, **Z** gates are applied in the position of these binary representation values equal to 1 (lines 6-10). Then, the positions of the vector which the binary string representation have $p \in [2, \cdots, N]$ bits equal to 1 and have value in the vector -1 are checked each quantity p at a time. For each of these states, we check whether it is still necessary

to include a controlled **Z** gate. This operator will only be applied if the state does not have the desired amplitude. This is necessary because some quantum gate previously inserted in the circuit may have already transformed the amplitude of the quantum state to the desired one (lines 12-20).

The HSGS algorithm creates the circuit that generates the amplitudes of a given vector as amplitudes of a quantum state. To create the neuron defined in [9], the circuit is defined as follows. For N qubits, N Hadamard gates are added in order to generate a superposition of basis quantum states with REW amplitude. Then, the U_i, which is the circuit generated by the HSGS algorithm for input vector $\vec{i}$, is coupled. Next, the U_w operator is coupled. This operator is composed of the circuit generated by the HSGS algorithm for the weight vector $\vec{w}$, coupled with N Hadamard and N **X** gates. The result of the inner product of the weight with the input will be as state amplitude $|11...1\rangle$. To capture this amplitude information, the authors suggest inserting a $CNOT_{j=N-1}^{N}$ gate and applying it to a qubit that will be read. As result, the probability of reading that last auxiliary qubit will be read with the value 1 proportional to the value of the calculated inner product.

In Figure 4, it is shown an example of a quantum neuron where the input is $i = \{1,-1,-1,-1,1,1,1,1,1,1,1,1,1,1,1,1\}$ and $w = \{1,1,1,-1,1,1,1,1,1,1,1,1,1,1,1,1\}$ generated using the HSGS algorithm defined in Algorithm 21.

In their experiments [9], the authors trained a quantum neuron with 4 qubits (*i.e.* 16 amplitude states) to recognize a cross in a 4x4 pixels image using a quantum training algorithm to do a random adjust of the weights. This algorithm was based in the amount of different values between the input and the weight operators.

IV. Avoiding Classification Errors in Quantum Perceptron Model

Analyzing the the proposed inner product's behavior, we noticed that when the difference between weight and input increases linearly, the probability of the inner product been still positive decreases quadratically. In Figure 5, it is possible to see what happens when the input vectors and the weight become increasingly different (horizontal axis). In this graph, the line below represents the original neuron, which drastically decreases its probability of measuring 1 at the end of its operation, as input and weight are more different. For each distance value, 1000 random possible states with respective distances were generated and the average value between them was calculated.

Given a real problem in which a pattern has noise, as shown in Figure 8, the neuron behavior would cause a very self-adjusted (data memorization) behavior to a certain value, preventing it from generalizing. We found that the probability of being correctly positively classified in the case where the cross's image has only two noises decreases more than 40%. This means that the quantum neuron only has ability to accept an input image with few differences in relation with its weight. As consequence of overfitting implies not tolerate noise and hence not being able to achieve good levels of generalization.

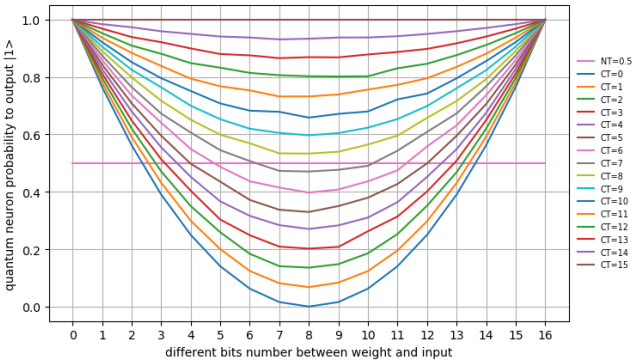

Fig. 5. The addition of different numbers of pixels versus the probability of neuron output being one. An illustration that what occurs with the inner product when the different bits number increases. The "CT" means "Circuit Threshold" and it is equal to zero in the original proposal. In that case we have a zero toleration that take us to overfit, since it does not tolerate many different input values in relation to weight.

A. Neuron threshold variation

In order for the neuron to become tolerant to noise, the threshold for accepting the output of the neuron can be varied during its execution. In the presence of significant noise, the neuron probability of giving output 1 becomes very low. Thus, it is necessary that its execution must be repeated several times, since the probability value is really low, and can be neglected if it is executed only a few times.

B. Circuit threshold operator

Considering the $|\psi_w\rangle$ state defined in Equation 1 applied to the U_w operator, built using Algorithm 21, coupled with $H^{\otimes N}$ and $X^{\otimes N}$ quantum gates, it is possible to verify that $\langle\psi_w|\,|\psi_i\rangle = c_{m-1}$, which is the value of the internal product between the two states (more details can be found in [9]). What happens at the end of the neuron operations is that this amplitude is concentrated in the amplitude of the state $|m-1\rangle$, which can be a low value, even with an insignificant distance between the two vectors.

Therefore, our other proposal is that the amplitude to be considered is not only that of the quantum state $|m-1\rangle$, but of some other quantum states, therefore the neuron decreases its data memorization and starts to accept a tolerable noise level. In this alternative, the probability of giving as output 1 is increased, preventing errors from being considered when the number of executions of the neuron is not significant to model the probability of its output.

For this, other **CNOTs** are coupled to the neuron so that more amplitude values are added to the amplitude of the final state to be measured. The operator that has **CNOTs** was called **CTO** (Equation 2) and has two parameters that are N and CT. N is the number of qubits manipulated (considering the output qubit) and CT is how many **CNOTs**

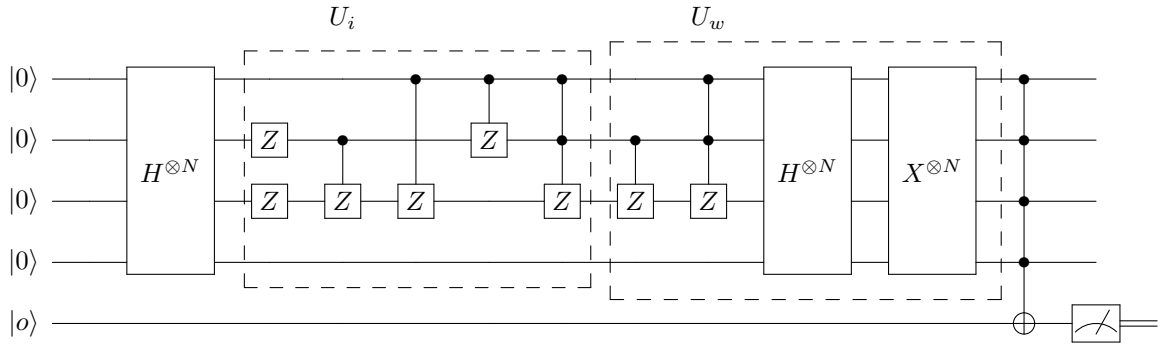

Fig. 4. Quantum perceptron circuit of a N=4, where the input $i_1 = i_2 = i_3 = -1$, and $i_j = 1$, for $j = 0, 4, 5, 6, 7$ and the weight is $w_3 = -1$, and 1 for all other values, using HSGS algorithm to generate U_i and U_w circuits.

are considered in the calculation, i.e. what the circuit threshold (CT) is.

$$CTO_{CT}^{N} = \prod_{j=0}^{CT} \mathbf{CNOTs}_{N-j}^{N} \tag{2}$$

The final quantum circuit of the quantum neuron with a circuit threshold (CT) is shown in Figure 6. Figure 7 shows the quantum state amplitudes considered when the CT parameter is applied in quantum neuron circuit building. The CT parameter and the final amplitude of the output state are positively correlated. It is possible to accept noise levels in the original neuron (when CT = 0), it is necessary to significantly lower the neuron threshold (NT) to values below 10% or 5% for example.

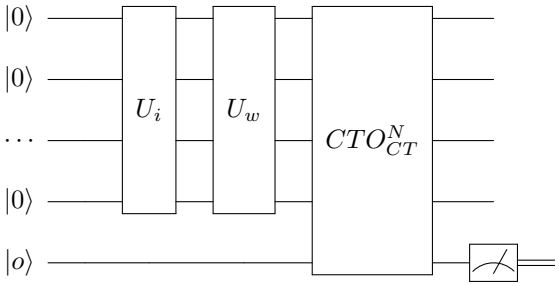

Fig. 6. Quantum perceptron circuit with parameter CT for circuit threshold.

CT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Fig. 7. The quantum states (top) considered to increase the final amplitude of the neuron output when the circuit threshold (CT) parameter is set.

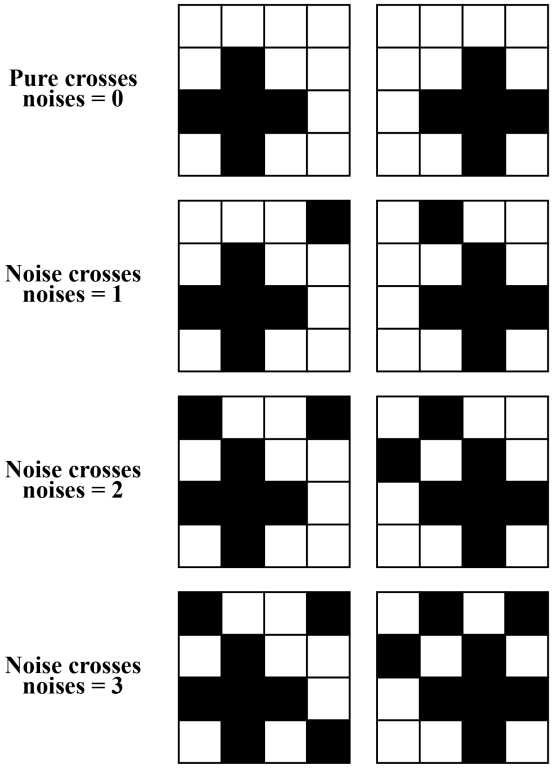

Fig. 8. The computation of different numbers of noises. When we have the pure crosses, there is no noise. In the basic implementations using 4x4 image there is only 4 possibility of crosses in different positions. When we add noises, even if there is some noise, the cross is still there and those possibilities must be consider.

V. Experiments

In this section, we detail the experiments conducted to test the two alternatives to avoid the data memorization mentioned in the previous sections. From Figure 5, it is possible to identify the NT values at which the neuron accepts more and more noise: 0.01, 0.06, 0.12, 0.20, 0.37, 0.55, 0.75, 0.9. For a given training set, how to choose the values of the CT and NT parameters? Training the CT or NT together with the neuron

weights can be considered. To validate this training hypothesis, we have a dataset with randomly distributed noises in 140 images with balanced classes (50% of cases each).

We did five sets of experiments, performing a classical-quantum training on the models. First, a genetic algorithm (GA) was executed to find the values of the quantum neuron weights and its CT value. In the second experiment, GA was executed to search for weight and NT values – next section details the parameters of the genetic algorithm. In the third set of experiments, we searched weights for some fixed values of NT=0.05, 0.1, 0.15, 0.3, 0.4, 0.6, in the original neuron, *i.e.* when CT = 0, using GA. We also performed the experiment where CT = 0 and NT = 0.5, where the GA searches only for weights. The last experiment attempted to perform the GA to search for weights in fixed CT configurations.

Tests were conducted on Google Cloud Platform in a virtual machine with 24 vCPUs Intel Skylake and 21.75 GB of memory, using the Qiskit programming language [26] that builds executable quantum circuits in quantum processor simulators and in real quantum processors from IBM. In our analysis we focus on two parameters: the accuracy and the time of each run. For each example of the training set, the HSGS algorithm was used to generate the circuit and coupled with the operators of each strategy (CTO or NTV).

A. Genetic Algorithm

Genetic algorithm was applied to perform weight, NT and CT searches since they have already been used in weight searches of neural networks and are fast for prototyping [27], [28]. We follow an usual GA implementation. Initially, we generate a population of *nPopulation* individuals, who encodes the binary weights and, if applicable, NT and CT parameter. CT is an integer that can vary between 0 (included) and 16, so we can represent it as a 4-position binary vector. NT value was modeled to be represented in 3 bits, since there are 8 possible values. The Algorithm 2 details the general functioning of the GA performed in our experiment. The crossover and mutation operator is detailed in Figure 9. The size of population in our experiment was 20 individuals. In each generation 19 new individuals are created, considering the roulette algorithm to choose the best performing parents. The most accurate weight of the current iteration is maintained in the next iteration.

B. Experiments analysis

It is possible to verify in Figure 10 that training only the weights for some fixed values of neuron threshold we found good results. For example, for NT = 0.05, the hit rate was 83.57%. However, training weights and NT parameter with GA, good rates were found, 76.43%, but not greater than training weights with the CT parameter, which was 78.57%. Probably, this experimental result was due to the fact that it may be easier to find weights by increasing the level of noise tolerance (*i.e*, varying CT) than to look for weight and NT.

It is also found that high values of NT have low accuracy in training their weights, which was expected, since it has little tolerance in noise. It was not placed in the graph for simplicity,

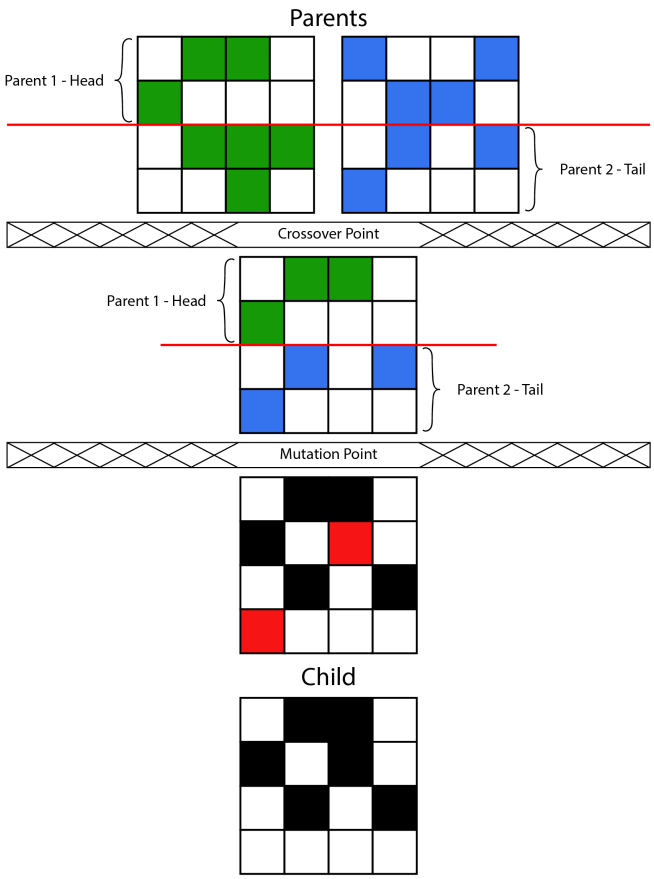

Fig. 9. Given two parents, in the crossover operator, a new child is created by copping the head from one (blue) and merging to a tail from the other (blue). The mutation flips some elements (red) probabilistically, with 10% of probability.

but all the values found for GA, training weights and with fixed values of NT = 0.5 and CT = 0, that is, the original neuron with fixed threshold, the accuracy result remains constant 53.57% in iterations. This means that the $NT = 0.5$ was not a good value for this base and the GA did not find weights that would satisfy the acceptance condition for the entire noisy base.

Table I shows the results found when fixing the value of CT in a quantum neuron, and using GA to search for weights in the neuron, in a noisy dataset. Table II shows the results of GA execution to search for weights when NT has some fixed value.

CT	Best accuracy	Time	Generation
0	54.29%	04h04m	116
2	60%	05h05m	193
4	65.71%	06h06m	1
6	82.86%	06h06m	157
8	82.14%	07h07m	94

TABLE I
BEST ACCURACY OF WEIGHTS FOUND IN TRAINING CONDUCTED BY THE GA WITH CT=0,2,4,6,8 FIXED, IN A NOISY DATASET.

From these tables, it is possible to verify that the GA searching for weights and CT or NT, as shown in Figure 10,

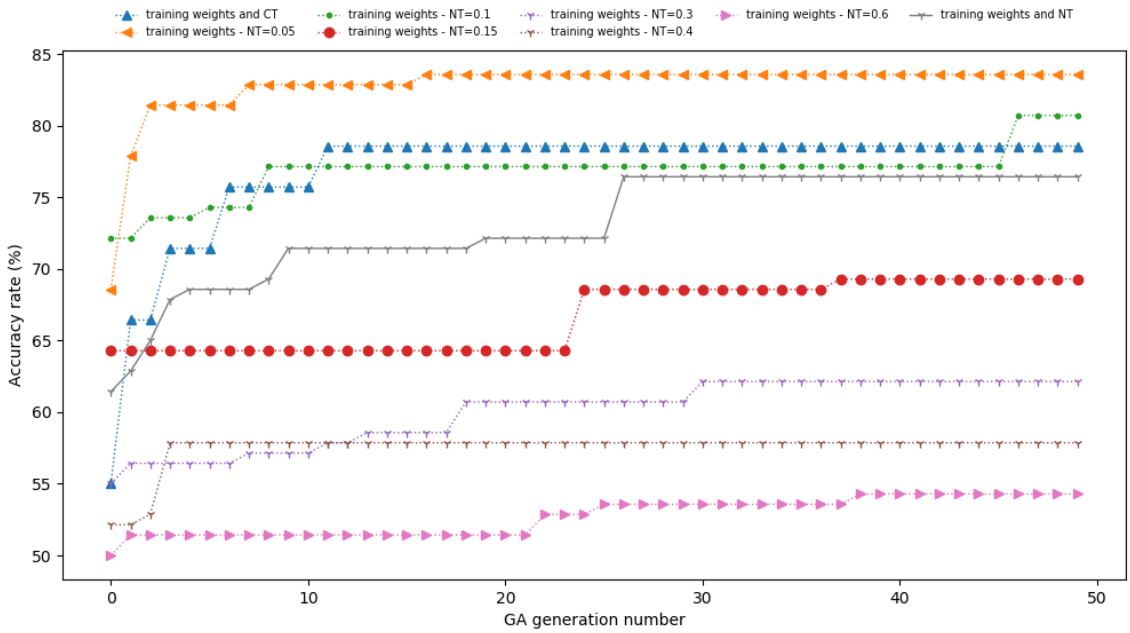

Fig. 10. Accuracy of the best weight found in each GA generation for the eight experiments: GA searching for weights and CT parameter of the neurons; GA searching for weights and NT parameter; and GA searching for weights when NT is 0.05, 0.1, 0.15, 0.3, 0.4, 0.6.

Algorithm 2: Genetic Algorithm to train any kind of N size population

Result: Best neuron with weight and parameters

```
Parameters: nPopulation;
population ← newPopulation(nPopulation);
population ← rank(population);
generation ← 1;
while (generation != 50) do
    lastBestWeight ← bestWeight(population);
    population ← makeCrossover(population);
    population ← makeMutation(population);
    population ← population + lastBestWeight;
    population ← rank(population);
    generation++;
end
```

have not yet managed to achieve better general configurations, probably because it needs more iterations.

NT	Best accuracy	Time	Generation
0 .05	83.57%	04h04m	16
0.1	80.71%	04h15m	46
0.15	68.57%	04h06m	37
0.3	62.14%	04h06m	30
0.4	57.14%	04h05m	3
0.6	54.29%	04h08m	38

TABLE II
BEST ACCURACY OF WEIGHTS FOUND IN TRAINING CONDUCTED BY THE GA WITH NT=0.05, 0.1, 0.15, 0.3, 0.4, 0.6 IN A NOISY DATASET.

VI. CONCLUSION

This work presents initial results to understand the behavior of a quantum Perceptron proposed in [9] from the point of view of learning and generalization in situations that require the presence of noise, running it on the IBM quantum simulator. It was verified that this model has the possibility to over-adjust its weights to a dataset, causing overfitting and therefore being unable to tolerate noise for fixed parameters. It has been proposed to include a quantum operator, Circuit Threshold Operator (**CTO**), which increases the functioning threshold of the circuit threshold (**CT**), adding degrees of tolerance to its classification, amplifying the amplitude of the expected output.

Another existing alternative is the adjustment of the neuron threshold (**NT**). This approach implies that the adjustment value of this threshold may be very low, which require that the repetition of the neuron be rigorous to capture the statistical behavior of its output.

In order to show the effectiveness of the proposed model, experiments to search for weights, **CT** and **NT** were performed using a classical genetic algorithm. The genetic algorithm finding weights and **CT** reaches a hit rate value of 78.57% while looking for weights and **NT** reached 76.43% hit rate. If only the weight was searched with NT = 0.5, CT = 0 fixed (original neuron), the hit rate is not higher than 53.57% in all generations. Better results were found by searching for weights for some fixed and low **NT** values. The need to vary circuit or neuron thresholds is evident in order to generalize training sets.

Some level of effort is being made for the maturation of quantum models and their simulations. This work has exper-

imental contributions that can serve as a basis for classifiers whose applications require noise levels. Future work can be directed to the use of a quantum training algorithm instead of a classic, which allows the use of quantum mechanical properties to solve optimization problems; and to the comparison of different NISQ computers available for experimentation.

Data availability

All the data that support the experiments shown in this study alongside with the code that have been created are available from the corresponding author upon a simple request.

Acknowledgment

We thank IBM for making the IBM Quantum Experience platform available and Google for making credits available to our research group on the Google Cloud Platform that enabled the simulation to be efficient and faster. We also have to thank the Brazilian National Council for Scientific and Technological Development (CNPq) for financial support.

References

[1] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[2] C. Wang, G. Yvonne, L. Frunzio, M. Devoret, and R. J. Schoelkopf III, "Techniques for manipulation of two-qubit quantum states and related systems and methods," Jan. 17 2019, uS Patent App. 16/068,405.

[3] A. Cross, "The IBM Q experience and QISKit open-source quantum computing software," in *APS March Meeting Abstracts*, ser. APS Meeting Abstracts, vol. 2018, Jan 2018, p. L58.003.

[4] Accessed: 2019-06-05.

[5] J. A. Jones, M. Mosca, and R. H. Hansen, "Implementation of a quantum search algorithm on a quantum computer," *Nature*, vol. 393, no. 6683, pp. 344–346, 1998.

[6] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh *et al.*, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017.

[7] P. D. Nation and J. Johansson, "Qutip: Quantum toolbox in python," *online at http://qutip. org*, 2011.

[8] H. A. et al, "Qiskit: An open-source framework for quantum computing," 2019.

[9] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni, "An artificial neuron implemented on an actual quantum processor," *npj Quantum Information*, vol. 5, pp. 1–8, 2019.

[10] R. Orus, S. Mugel, and E. Lizaso, "Quantum computing for finance: Overview and prospects," *Reviews in Physics*, vol. 4, p. 100028, 02 2019.

[11] N. Shenvi, J. Kempe, and K. B. Whaley, "Quantum random-walk search algorithm," *Physical Review A*, vol. 67, no. 5, p. 052307, 2003.

[12] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, vol. 2. IEEE, 2000, pp. 1354–1360.

[13] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik*, vol. 46, no. 4-5, pp. 493–505, 1998. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/

[14] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, pp. 195–202, 2017.

[15] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[16] M. Schuld, I. Sinayskiy, and F. Petruccione, "The quest for a quantum neural network," *Quantum Information Processing*, vol. 13, no. 11, p. 2567–2586, Aug 2014. [Online]. Available: http://dx.doi.org/10.1007/s11128-014-0809-8

[17] M. Panella and G. Martinelli, "Neural networks with quantum architecture and quantum learning," *International Journal of Circuit Theory and Applications*, vol. 39, no. 1, pp. 61–77, 2011.

[18] W. de Oliveira, "Quantum ram based neural netoworks." European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning, 01 2009.

[19] M. Schuld, I. Sinayskiy, and F. Petruccione, "Simulating a perceptron on a quantum computer," *Physics Letters A*, vol. 379, no. 7, pp. 660 – 663, 2015.

[20] F. M. de Paula Neto, T. B. Ludermir, W. R. de Oliveira, and A. J. da Silva, "Implementing any nonlinear quantum neuron," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–6, 2019.

[21] M. Schuld, M. Fingerhuth, and F. Petruccione, "Implementing a distance-based classifier with a quantum interference circuit," *EPL (Europhysics Letters)*, vol. 119, no. 6, p. 60002, sep 2017.

[22] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[23] N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, "Continuous-variable quantum neural networks," *Physical Review Research*, vol. 1, no. 3, p. 033063, 2019.

[24] R. B. de Sousa, E. J. Pereira, M. P. Cipolletti, and T. A. Ferreira, "A proposal of quantum data representation to improve the discrimination power," *Natural Computing*, pp. 1–15, 2019.

[25] M. Rossi, M. Huber, D. Bruss, and C. Macchiavello, "Quantum hypergraph states," *New Journal of Physics*, vol. 15, no. 11, 2013.

[26] A. Cross, "The ibm q experience and qiskit open-source quantum computing software," *Bulletin of the American Physical Society*, vol. 63, 2018.

[27] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[28] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms." in *IJCAI*, vol. 89, 1989, pp. 762–767.