# Neurofuzzy Networks With Nonlinear Quantum Learning

Massimo Panella, *Member, IEEE*, and Giuseppe Martinelli, *Life Senior Member, IEEE*

***Abstract*—Nonlinear quantum processing allows the solution of an optimization problem by the exhaustive search on all its possible solutions. Hence, it can replace advantageously the algorithms for learning from a training set. In order to pursue this possibility in the case of neurofuzzy networks, we propose in this paper to tailor their architectures to the requirements of quantum processing. In particular, superposition is introduced to pursue parallelism and entanglement to associate the network performance with each solution present in the superposition. Two aspects of the proposed method are considered in detail: the binary structure of membership functions and fuzzy reasoning and the use of a particular nonlinear quantum algorithm for extracting the optimal neurofuzzy network by exhaustive search.**

***Index Terms*—Exhaustive search, neurofuzzy system, nonlinear quantum processing, quantum neurofuzzy network.**

## I. Introduction

QUANTUM processing allows the solution of an optimization problem through the exhaustive search undertaken on all the possible solutions of the problem itself. The reason for this possibility is the existence of two relevant properties of quantum processing: *parallelism* and *entanglement*. Parallelism is the superposition of an exponential number of states representing the several solutions of the problem, including the best one. Entanglement is the potential for quantum states to exhibit correlations that cannot be accounted for classically, in particular, for associating a fitness to each solution and making a decision.

Quantum processing can be applied to several scientific fields such as physics, mathematics, engineering, and so forth [1]. The research has also been extended to the entire field of computational intelligence, considering, in particular, training and implementation of neural networks [2]–[9]. In this paper, we will focus on the quantum neurofuzzy network (QNFN), where computations are developed using the principles of quantum mechanics. In quantum computation, the information to be processed is stored as quantum bits (or *qubits*), which are physically related to the angular momentum of photons, electrons, or other fundamental particles. The realization of a QNFN is therefore based on a combination of *quantum circuits*, or quantum computers, which can be implemented using different emerging technologies, such as superconducting loops, cavity quantum electrodynamics (QED), and, above all, ion traps and nuclear magnetic resonance [10]–[13].

The architecture of a QNFN is tailored to the specific requirements of quantum processing. Namely, the problem of interest must be formulated in terms of Boolean variables and Boolean functions (BFs). Several problems are already expressed in this form; in the other cases, a preliminary conversion to the Boolean field is necessary, with particular attention to the number of bits required for attaining a proper accuracy [14]. Thus, a QNFN is particularly suited to the application of fuzzy logic and fuzzy systems in the emerging field of granular computing, where data to be processed are often nominal attributes (the labels of classes or clusters, for example) that cannot be represented in an ordered scale. In the following, we will consider this peculiar and interesting application of QNFNs, where binary codes are used to represent the Boolean variables. Nevertheless, we will outline how the proposed approach may also be extended to process integer-valued or ordered variables that are represented by a string of bits.

A QNFN has a classical counterpart in the particular class of networks called binary neurofuzzy networks (BNFNs). The architecture and the computational model of a BNFN are similar to the ones of a QNFN. Namely, a BNFN performs conceptually the same computations of a QNFN. It processes Boolean variables by means of BFs and fuzzy variables assuming integer values only. However, a BNFN can be realized using traditional digital gates and computers, where the computations are obtained exploiting the semiconducting effects typical, for example, of CMOS technologies. Obviously, the information is stored in this case using classical bits, physically realized by the charge of a memory cell's capacitor.

The main advantage of BNFNs, with respect to other classical neurofuzzy paradigms, is their efficient and reliable implementation based on digital hardware [15]–[20]. As will be demonstrated later, the architecture of a BNFN can be easily obtained using the standard tools provided by Boolean logic, fuzzy inference, and data-driven neural learning. Consequently, because of their similitude, it is convenient to preliminarily develop a BNFN and then to convert it into a QNFN. This conversion is always possible, as discussed later in the paper.

The paper is organized as follows. The notations and the basic properties of quantum processing are shortly presented in Section II. The architecture of the proposed BNFN is determined in Section III by means of a suitable fuzzy inference mechanism and a related training procedure. The BNFN is obtained by combining several BFs described in detail in five Appendixes. The conversion of a BNFN into a QNFN is discussed in Section IV; successively, the training procedure of the QNFN is introduced

Manuscript received August 28, 2007; revised April 30, 2008; accepted May 10, 2008. First published July 16, 2008; current version published June 11, 2009.

The authors are with the Department of Information and Communication (INFOCOM), University of Rome "La Sapienza," Rome 00184, Italy (e-mail: panella@infocom.uniroma1.it).

Digital Object Identifier 10.1109/TFUZZ.2008.928603

in Section V, where the properties of entanglement and quantum parallelism are pointed out for the linear quantum gates used in this regard. The optimal QNFN will be determined in Section VI, by using a particular nonlinear quantum algorithm. We will prove that an exhaustive search of the optimal solution is possible; thus, we can obtain an effective learning procedure because of the exponential speedup of the nonlinear quantum algorithm.

The proposed method might be difficult to follow, since it is described in several sections and appendixes. For this reason, a step-by-step guide to its application is presented in Section VII in order to highlight its complexity and the required computational cost. Furthermore, this method is also illustrated by an example of BNFN applied to classification. The details of the several steps involved in this example are distributed all over the paper, where appropriate.

## II. Background of Quantum Computing

We will briefly overview in this section, the basic properties of quantum computation with regard to the physical systems (i.e., the quantum circuits) by which the relevant information is processed [21].

### A. Quantum Data

The data to be handled in a quantum circuit are strings of qubits, which play the same role as bits of the classical logic circuits. A qubit is a unit vector in a 2-D complex vector space. Such vectors are represented by the ket/bra notation; the ket $|\psi\rangle$ denotes a column vector, and the bra $\langle\psi|$ denotes the conjugate transpose of $|\psi\rangle$. The orthonormal basis of the 2-D complex vector space, i.e., the pair of vectors $[1,0]^{\mathrm{t}}$ and $[0,1]^{\mathrm{t}}$, are represented by $|0\rangle$ and $|1\rangle$, respectively.

Unlike classical bits, qubits are in superposition of states $|0\rangle$ and $|1\rangle$, i.e., $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha$ and $\beta$ are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. If the qubit is measured, the probability of observing the state $|0\rangle$ is $|\alpha|^2$, and similarly, the probability of observing $|1\rangle$ is $|\beta|^2$. After the measurement, any superposition of quantum states is destroyed, since only one state is observed with a given probability. The previous properties come out directly from the measurement postulate of quantum mechanics, and therefore, from the physical behavior of quantum particles. A string of $B$ qubits has a state space of $2^B$ dimensions with $2^B$ basis vectors that are represented by the ket/bra notation as well. For instance, a two-qubit vector $|\psi\rangle$ is the superposition of four basis states

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle \tag{1}$$

where the probability to measure the $h$th state $h = 0 \ldots 3$ is $|\alpha_h|^2$ with $\sum_h |\alpha_h|^2 = 1$.

As discussed, the extraction of a specific qubit from the superposition is carried out in a probabilistic manner through the measurement; after that, the original superposition is destroyed. Supposing we wish to measure the first qubit from the superposition (1), the measurement will return $|0\rangle$ with probability $c_0 = |\alpha_0|^2 + |\alpha_1|^2$ and $|1\rangle$ with probability $c_1 = |\alpha_2|^2 + |\alpha_3|^2$. If the outcome is $|0\rangle$, the probability to successively measure $|0\rangle$ for the second qubit will be $|\alpha_0|^2/c_0$, while the probability to measure $|1\rangle$ will be $|\alpha_1|^2/c_0$. Analogous results are obtained by reversing the order of measurements.

### B. Entanglement

Entanglement is the potential for quantum states to exhibit correlations that cannot be accounted for classically. Although the measurement of a qubit in a string may not depend on whether the other qubits have been formerly measured or not, the peculiarity of entanglement is just evidenced in the measurement process, as for example, for the state

$$|\psi\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle . \tag{2}$$

It is an entangled superposition, since the probability of the first qubit to be $|0\rangle$ is 0.5 if the second qubit has not been measured, but this probability would be either 1 or 0 in case the second qubit had been measured and its outcome was $|0\rangle$ or $|1\rangle$, respectively.

A very important consequence of entanglement is the necessity to operate in a $2^B$ dimensional space in the case of a string of length $B$. Instead, if it is fully not entangled, some operations can be performed repeatedly in the 2-D space of each qubit. In fact, only if qubits are not entangled the whole state represented by the string $|\psi\rangle$ can be described in terms of the states of its components (qubits) separately. For instance, the state

$$|\psi\rangle = \frac{1}{\sqrt{2}} |01\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

is not entangled, and it can be represented by the *tensor product* "$\otimes$" of two separated qubits

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle = \left(\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle\right) \otimes (0\,|0\rangle + 1\,|1\rangle)$$

where the probability of the first qubit to be $|0\rangle$ or $|1\rangle$ does not depend on whether the second qubit has been measured or not.

### C. Linear Quantum Gates

The basic components of quantum circuits are linear quantum gates; they implement unitary (and reversible) transformations, which can be thought of as being rotations in the complex qubit vector space. Rotations, in fact, maintain the orthogonality of basis vectors, and hence, the validity of the measurement postulate. In order to represent quantum transformations in matrix notation, we can pick the isomorphism between the orthonormal basis of a $B$-qubit system and the canonical $2^B$-tuple basis in the complex vector space. For clarity, if we consider the 2-qubit string in (2), the equivalent representation of $|\psi\rangle$ is a vector having four components corresponding to the said basis vectors i.e.,

$$\left[\frac{1}{\sqrt{2}} \;\; 0 \;\; 0 \;\; \frac{1}{\sqrt{2}}\right]^t$$

where "$t$" denotes transposition. In the same way, the basis vectors $|00\rangle$ and $|11\rangle$ are represented by the strings $[1\ 0\ 0\ 0]^t$ and $[0\ 0\ 0\ 1]^t$, respectively.

Each quantum gate is therefore represented by a suitable unitary matrix $\mathbf{U}$ such that $\mathbf{U}\mathbf{U}^* = \mathbf{E}$, where the asterisk represents

transpose conjugation, and $\mathbf{E}$ is the identity matrix of order $2^B$. For instance, a quantum gate $\mathbf{U}$ operating on the generic 2-qubit string (1) will yield as a result the 2-qubit string

$$|\zeta\rangle = \beta_0 |00\rangle + \beta_1 |01\rangle + \beta_2 |10\rangle + \beta_3 |11\rangle$$

where

$$|\zeta\rangle = \mathbf{U} |\psi\rangle \tag{3}$$

or, equivalently

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \mathbf{U} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} . \tag{4}$$

As a consequence of linearity for matrix-vector multiplication, the gate operation is equivalently represented by the transformation of every basis vector in the quantum state space.

The unitary property implies that quantum states cannot be copied or cloned; this is also known as the *no cloning* property. Nevertheless, the parallelism of quantum processing is implicitly represented by (3), since the gate operates concurrently on every qubit basis vector. It is also important to remark that, even when input qubits are not entangled, some quantum gates may produce entangled qubits at their output.

### D. Nonlinear Quantum Operators

Nonlinear quantum operators allow enhanced algorithms to overcome the difficulties connected to the use of ordinary components. In particular, the power of quantum nonlinear computing can be fully exploited in order to reduce to a minimum the computational complexity. We will consider in Section VI an algorithm of this kind, which makes use of a specific nonlinear quantum gate.

## III. Architecture of the Proposed BNFN

As previously mentioned, the architecture of a QNFN can be obtained by preliminarily determining that of a BNFN. In this regard, we will consider a BNFN constituted by $R$ rules of zero-order Sugeno type [22], [23], where the $k$th rule, $k = 1 \dots R$, has the form

$$\texttt{if}\, x_1 \,\texttt{is}\, A_{k1} \,\texttt{and}\, x_2 \,\texttt{is} A_{k2} \,\dots \texttt{and}\, x_N \,\texttt{is}\, A_{kN} \quad \texttt{then}\, \text{output}\, \texttt{is}\, c_k \,. \tag{5}$$

Each input variable $x_j$, $j = 1 \dots N$, coincides with a string of $W_j$ bits. In the same way, each input fuzzy quantity $A_{kj}$ is associated with a membership function (MF) defined with respect to a binary string $a_{kj}$ of $W_j$ bits. The output value $c_k$ is a string of $L$ bits. Thus, the whole network can be represented by a string of $B$ bits, where

$$B = R \left( L + \sum_{j=1}^{N} W_j \right) . \tag{6}$$

The properties of BNFNs will be introduced in the following considering generic values for $R$, $N$, $L$, and $W_j$. However, in order to improve the understandability of the paper, the proposed approach will be illustrated, where necessary, by also considering an example of BNFN applied to binary classification. In this case, the output of each rule is a string of only $L = 1$ bit, discriminating between the two classes of the problem. In addition, this BNFN is constituted by $R = 8$ zero-order Sugeno rules and $N = 3$ input variables. Each input variable is a string of 4 bits, hence $W_1 = W_2 = W_3 = 4$, and the BNFN is represented by a string of $B = 8 \times 13 = 104$ bits. The said example will be referred to in the following as the "reference case."

### A. Fuzzy Inference and Output Determination

When a generic pattern $\underline{x}_i = [x_{i1}\, x_{i2} \dots x_{iN}]$ is presented to the input of a BNFN, a fuzzy inference mechanism is invoked in order to assign the pattern a specific output. Several options are possible for the fuzzy reasoning (i.e., T-norms, T-conorms, compositions, implications, etc.); we will consider in the following a specific type of reasoning suited to the Boolean variables herein involved.

First of all, when $\underline{x}_i$ is presented to the BNFN, then the MF of each input fuzzy quantity $A_{kj}$ must be calculated. In the proposed architecture, the MF is obtained by means of a suitable distance between the component $x_{ij}$, $j = 1 \dots N$, of the input and the string $a_{kj}$ associated with the $j$th antecedent of the $k$th rule. Evidently, the component $x_{ij}$ is an input variable coded by $W_j$ bits. The value of the MF, which is denoted by $\mu_{ikj}$, decreases with distance and is implemented as a BF $F_1(\cdot)$

$$\mu_{ikj} = F_1 \left( x_{ij}, a_{kj} \right) . \tag{7}$$

The BF $F_1$ is described in Appendix A and illustrated in detail for the reference case.

The number of MFs $\mu_{ikj}$ of the antecedents of the $k$th rule is equal to $N$. Consequently, there are $N$ values to be combined together for evaluating the reliability $\gamma_{ik}$ of the $k$th rule with respect to the input $\underline{x}_i$. This can be achieved by implementing a suitable BF $F_2(\cdot)$, described and illustrated in Appendix B

$$\gamma_{ik} = F_2 \left( \mu_{ik1}, \mu_{ik2}, \dots, \mu_{ikN} \right) . \tag{8}$$

The output $u_i$ of the BNFN, when $\underline{x}_i$ is applied to its input, is an integer coded by $L$ bits as for the rule outputs; it is determined by taking into account both the reliabilities of the rules and their outputs. Let us denote with $u_i^{(q)}$ the $q$th bit, $q = 1 \dots L$, of the output $u_i$. The procedure based on the following steps will determine every bit $u_i^{(q)}$ independently of each other.

*Procedure I:*

1) Consider the current value of $q$ and then the $q$th bit $c_k^{(q)}$ of the $k$th rule's output $c_k$, $k = 1 \dots R$.
2) Divide the rules into two groups: The first group refers to the rules for which $c_k^{(q)} = 1$ and the second group to $c_k^{(q)} = 0$. We outline that this division is independent of the current input and it could be carried out once the rule outputs have been established.
3) Determine the sum of reliabilities of the rules belonging to each group. They will be denoted by $S_{iq}^1$ and $S_{iq}^0$ for the first and second group, respectively. Such sums are integers coded by a proper number of bits, since the

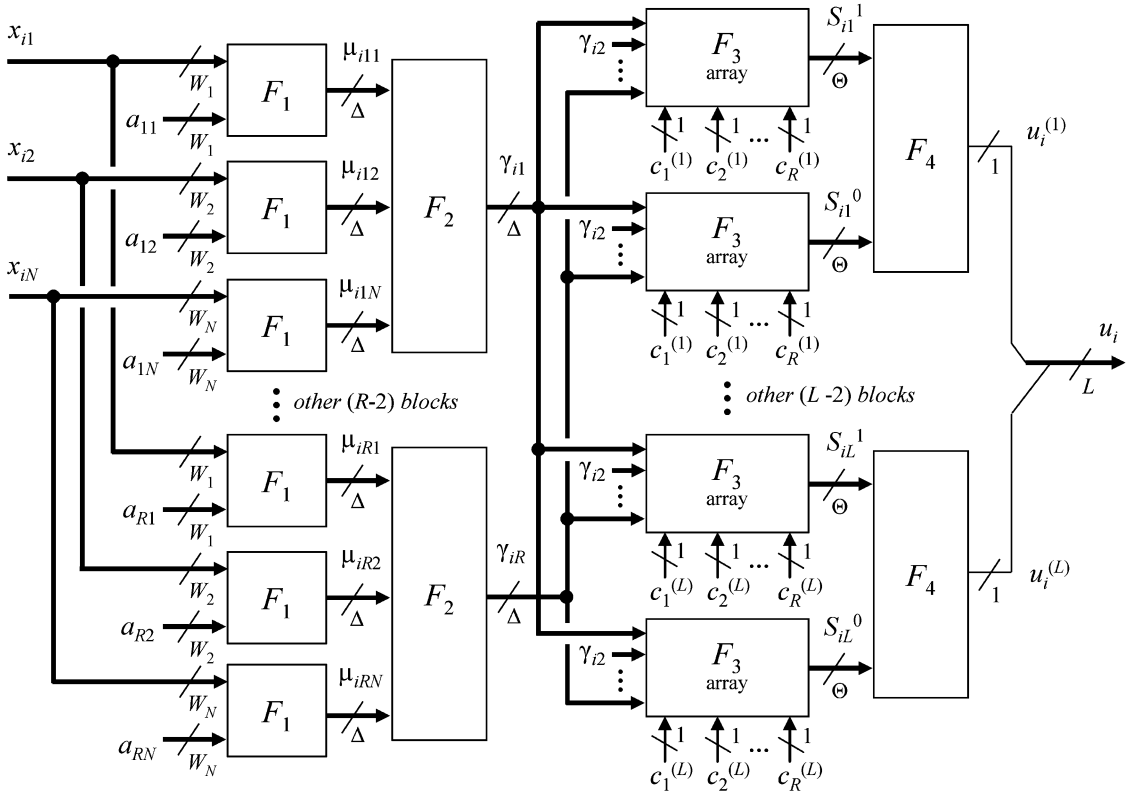



Fig. 1. Architecture of the proposed BNFN. Each quantity is a string of bits, whose number is indicated in correspondence of transverse lines. The values of Δ and Θ are introduced in Appendix A and Appendix C, respectively. The "$F_3$ array" combines several applications of the BF $F_3$.

reliabilities calculated by (8) are integers as well. The determination of $S_{iq}^1$ and $S_{iq}^0$ is carried out by the repeated application of a suitable BF $F_3(\cdot)$, as described and illustrated in Appendix C.

4) The output bit $u_i^{(q)}$ is then calculated as follows:

$$u_i^{(q)} = \begin{cases} 1, & \text{if } S_{iq}^1 \geq S_{iq}^0 \\ 0, & \text{if } S_{iq}^1 < S_{iq}^0. \end{cases} \tag{9}$$

The value of $u_i^{(q)}$ can be obtained by means of a BF $F_4(\cdot)$, which depends on the binary strings representing both $S_{iq}^1$ and $S_{iq}^0$. The BF is discussed in Appendix D considering, for instance, the reference case.

*end of Procedure I*

On the basis of the previous fuzzy reasoning, the architecture of the proposed BNFN assumes the structure illustrated in Fig. 1. Using the combination of functions $F_1$, $F_2$, $F_3$, and $F_4$, the BNFN implements an overall BF $F_u(\cdot)$, for which the actual output $u_i$ of any input pattern $\underline{x}_i$ is given by

$$u_i = F_u\left(\underline{x}_i\right). \tag{10}$$

As discussed in Appendixes A and C, the functional blocks implementing the BFs $F_1$ and $F_3$ can be simplified, if all the rule parameters $a_{kj}$ and $c_k^{(q)}$ are fixed in advance, and we are not interested in a reconfigurable BNFN, where such parameters can be varied like generic inputs.

### B. Training Process and Performance Evaluation

The training process of a BNFN requires the determination of the number $R$ of rules and the computation of the bits defining each rule. The right number of rules is essential in order to obtain a good generalization capability of the BNFN. The optimization of $R$ can be carried out in a separate procedure, using well-known constructive or pruning methods such as early stopping, cross validation, or learning theory [24]–[26]. However, the basic learning step that we will focus in the following is the BNFN training for a fixed number of rules.

When dealing with a data-driven learning, the problem to be solved by a BNFN is usually defined through a given training set (TS) of numerical examples. The TS is constituted by $P$ patterns $\underline{\xi}_i = (\underline{x}_i, t_i)$, $i = 1 \ldots P$, where $\underline{x}_i$ is the joint vector of inputs previously considered, and $t_i$ is the output value, which is coded by $L$ bits as for rule outputs. Each pattern $\underline{\xi}_i$ defines the input–output relation between the binary strings $\underline{x}_i$ and $t_i$; consequently, the TS represents the sampling of an unknown BF $f(\cdot)$ i.e.,

$$t_i = f\left(\underline{x}_i\right), \; i = 1 \ldots P. \tag{11}$$

As for any other learning process of neurofuzzy networks, it is necessary to define and calculate the performance of the BNFN with respect to the TS. Evidently, the performance depends on the matching between the desired output $t_i$ of each pattern and the actual output $u_i$, determined using the architecture illustrated in Fig. 1. In other words, the BNFN will approximate $f(\cdot)$

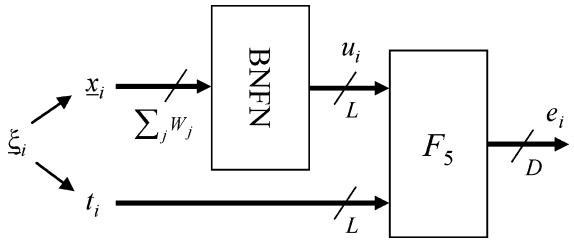



Fig. 2. Architecture for computing the BNFN's performance in correspondence to any pattern $\underline{\xi}_i$ of the TS. The BNFN block assembles the system illustrated in Fig. 1. The number of bits $D$ is discussed in Appendix E.

by the BF $F_u(\cdot)$ defined in (10), whose sampling is obtained using the patterns of the TS. The BNFN's performance $e_i$, in correspondence to the pattern $\underline{\xi}_i$ of the TS, is obtained through the BF $F_5(\cdot)$ described in Appendix E:

$$e_i = F_5\left(u_i, t_i\right) . \tag{12}$$

Consequently, the performance can be computed by the architecture shown in Fig. 2, which while associating block $F_5$ to the BNFN of Fig. 1 is able to compute $e_i$ directly. In this regard, it is important to notice that the value of $e_i$, which is an integer coded by a string of $D$ bits, is obtained as an overall BF using the functional blocks $F_1$, $F_2$, $F_3$, $F_4$, and $F_5$ introduced up until now.

The architectures proposed in Figs. 1 and 2 can be realized directly on a digital computing machine. In fact, each functional block is associated with a BF that we have defined in terms of its truth table. Thus, each block can be implemented using basic logic gates and widely used CMOS technologies, like, for example, the programmable logic array (PLA) or the field-programmable gate array (FPGA). On the other hand, we may consider the whole procedure proposed in this section to calculate the BNFN's output and the TS performance. Also, in this case, such algorithms can be implemented on digital signal processor (DSP) boards or general-purpose computers.

## IV. Mapping a BNFN into the Quantum Field: The QNFN Definition

Using the TS, the learning process aims to determine the rule parameters so that the resulting BNFN may approximate the underlying function $f(\cdot)$ above a given accuracy. Several approaches can be followed in this regard; a first family comes from the adaptation to binary integer programming of classical derivative-based methods (such as clustering and backpropagation); conversely, the family of direct search methods (such as genetic or swarm-based algorithms) can be directly applied to the BNFN training.

For both the previous categories, the computational complexity of training depends on the number of different BNFNs for which the TS performance is evaluated. Since no assumptions are usually made about the structure of the search problem, an $\mathrm{O}(M)$ complexity is mandatory to find an optimal BNFN, with $M = 2^B$ being the total number of possible solutions. The less BNFNs are evaluated the more the probability to find a poor solution increases. Unfortunately, the value of $M$ grows exponentially with $B$, and it can be very large even for very simple architectures; for instance, in the reference case, we have $M = 2^{104} \cong 2 \times 10^{31}$ possible solutions.

As a critical observation, one may notice that a BNFN could be implemented even considering directly the truth table of the BF $F_u(\cdot)$, realized by the whole system of Fig. 1, instead of using the rules and their parameters, as done in this paper. However, the truth table of $F_u$ should be identical to the one of $f(\cdot)$, which is only partially known through the TS. In this way, the BNFN's synthesis would not use any inference mechanism to extend knowledge from the partial available data, as obtained by the use of rules in neurofuzzy networks.

The use of quantum computing is able to overcome the previous drawbacks; we will demonstrate that the optimal neurofuzzy network can be extracted by the exhaustive search on all the possible $M$ solutions in a time drastically reduced with respect to the traditional training approaches. This is obtained by converting the BNFN into a QNFN, taking into account two major aspects. The computational structure of a QNFN is identical to a BNFN; namely, it can be obtained from the schemes of Figs. 1 and 2, considering that each functional block (i.e., from $F_1$ to $F_5$) represents a computable function that can always be converted into a valid linear quantum gate [27]. As discussed in Section I, the main difference between QNFNs and BNFNs will consist of their physical realization. In the former, the functional blocks are implemented as quantum circuits to process quantum particles (photons, electrons, etc.); in BNFNs, the functional blocks are based on common digital circuits working on electric signals.

Let us consider the $h$th BNFN, $h = 1 \ldots M$, in the set of all possible $M$ solutions. As said, it can be associated with a string $\psi_h$ of $B$ bits representing the parameters of all the rules

$$\psi_h = [\varphi_{h1}\ \varphi_{h2}\ \cdots\ \varphi_{hR}] \tag{13}$$

where

$$\varphi_{hk} = \left[a^h_{k1}\ a^h_{k2}\ \cdots\ a^h_{kN}\ c^h_k\right], \qquad k = 1 \ldots R \tag{14}$$

and the superscript "$h$" has been added in order to indicate that rule parameters refer to the $h$th network. The QNFN corresponding to this BNFN can be represented by the string $\left|\psi_h\right\rangle$ of $B$ qubits

$$\left|\psi_h\right\rangle = \left|\varphi_{h1}\ \varphi_{h2}\ \ldots\ \varphi_{hR}\right\rangle. \tag{15}$$

For instance, if a BNFN is represented by $\psi_h = 0101 \ldots 1100$, then the corresponding QNFN will be represented by $\left|\psi_h\right\rangle = \left|0101 \ldots 1100\right\rangle$. Consequently, as defined in Section II, every QNFN can be associated with exactly one basis vector of a $B$-qubit quantum system.

## V. Setup of QNFN Training: The Quantum Oracle

On the basis of the previous definitions, a QNFN can be trained by adapting the procedure described in Section III for BNFNs. Also in this case, the basic step consists in the computation of the network performance using the available TS. The procedure, which is illustrated in Section III, allows the computation of the performance $e_i$ considering a generic pattern $\underline{\xi}_i$ of the TS and using a given BNFN (i.e., using fixed rule

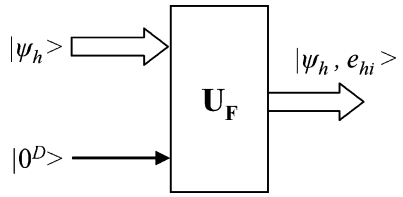



Fig. 3. Linear quantum gate used to calculate the QNFN's performance for a pattern of the TS. Wide arrows indicate entangled qubits in strings $|\psi_h\rangle$ and $|\psi_h, e_{hi}\rangle$.

parameters). However, the same procedure may be applied for different rule parameters in such a way that it can be considered as the application of an overall BF calculating the performance for the pattern $\underline{\xi}_i$ in function of different networks. Let us denote with $F_e$ this BF

$$e_{hi} = F_e(\psi_h; \underline{\xi}_i) \tag{16}$$

where $e_{hi}$ is computed as specified in Section III, in particular, using (12). By the way, we added a further subscript "$h$" for indicating that this computation is now dependent on the particular BNFN described by the bit string $\psi_h$.

Looking at Figs. 1 and 2, it is easy to prove that $F_e$ is a combination of the well-defined BFs previously introduced. Hence, it can be mapped into the quantum field to calculate the TS performance for QNFNs as well. More precisely, $F_e$ can be implemented following the procedure suggested in [21], i.e., by using the linear quantum gate shown in Fig. 3. The gate is driven by any qubit basis vector $\big|\psi_h\big\rangle$, representing a QNFN, and by a string $\big|0^D\big\rangle = \big|00\ldots0\big\rangle$ of $D$ qubits all equal to zero. The output of the gate is a string $\big|\psi_h, e_{hi}\big\rangle$ of $(B+D)$ qubits, where $\big|e_{hi}\big\rangle$ is the performance of the QNFN. For example, let $\psi_h$ be a BNFN having a performance $e_{hi}$ on $\underline{\xi}_i$ such that

$$\psi_h = 0101\ldots1100$$
$$e_{hi} = 101\ldots010$$

where $e_{hi}$ is obtained by (16) and is represented by $D$ bits. Then, the corresponding QNFN $\big|\psi_h\big\rangle$ will be characterized into the quantum field by

$$\big|\psi_h\big\rangle = \big|0101\ldots1100\big\rangle$$
$$\big|e_{hi}\big\rangle = \big|101\ldots010\big\rangle$$
$$\big|\psi_h, e_{hi}\big\rangle = \big|0101\ldots1100\ \ 101\ldots010\big\rangle$$

where $\big|e_{hi}\big\rangle$ is a string of $D$ qubits.

The linear quantum gate of Fig. 3 can be associated with a unitary matrix $\mathbf{U_F}$ of order $2^{(B+D)}$ such that

$$\big|\psi_h, e_{hi}\big\rangle = \mathbf{U_F}\big|\psi_h, 0^D\big\rangle. \tag{17}$$

As evidenced in (17), the gate $\mathbf{U_F}$ exploits the entanglement property of quantum processing. In fact, the output is a qubit string where the QNFN representation is entangled with its performance for a specific pattern of the training set. In addition, this gate agrees with the crucial property of quantum parallelism. Let us consider a superposition $|\Psi\rangle$ of all the QNFNs having a same amplitude probability $1/M$

$$\big|\Psi\big\rangle = \frac{1}{\sqrt{M}}\sum_{h=1}^{M}\big|\psi_h\big\rangle. \tag{18}$$

When the superposition $|\Psi\rangle$ is applied to the input of the gate $\mathbf{U_F}$, together with the string $\big|0^D\big\rangle$, its output will be the superposition $|\Omega_i\rangle$ where each QNFN $\big|\psi_h\big\rangle$ is entangled with its performance $\big|e_{hi}\big\rangle$ on $\underline{\xi}_i$

$$\big|\Omega_i\big\rangle = \frac{1}{\sqrt{M}}\sum_{h=1}^{M}\big|\psi_h, e_{hi}\big\rangle. \tag{19}$$

From now on, all the QNFNs will be processed in parallel using superpositions as in (18) and (19). This is the main reason for which quantum processing can speed up conventional computations.

The performance of the QNFN must be determined with respect to the entire TS. Once again, it can be obtained from the related computation used for BNFNs. The whole TS performance for the $h$th BNFN is

$$E_h = \sum_{i=1}^{P} e_{hi}. \tag{20}$$

Consequently, the corresponding QNFN will have a performance on the TS represented by the qubit string $\big|E_h\big\rangle$, obtained by means of the usual convention for associating $\big|E_h\big\rangle$ with the corresponding bit string representing the integer $E_h$. Taking into account that performances $e_{hi}$ are positive integers assuming values between 0 and $2^D-1$, the maximum value for $E_h$ is $P(2^D-1)$. Thus, $E_h$ must be represented by a string of $D_E$ bits and $\big|E_h\big\rangle$ by $D_E$ qubits, where

$$D_E = 1 + \big\lfloor \log_2\big(P(2^D-1)\big)\big\rfloor \tag{21}$$

and $\lfloor\cdot\rfloor$ denotes the greatest integer less than or equal to the argument.

For the $h$th QNFN, the TS performance $\big|E_h\big\rangle$ is obtained into the quantum field by adding the qubits $\big|e_{hi}\big\rangle$, $i=1\ldots P$, to each other. However, by exploiting the quantum parallelism, the performances $\big|E_h\big\rangle$, $h=1\ldots M$, can be obtained all together using the superpositions (19). We report the procedure proposed in this regard, which is based on a suitable initialization followed by $P$ steps, each related to a pattern of the TS.

*Procedure II:*

1) *Initialization*: Let $\big|\Gamma\big\rangle$ be a register of $(B+D_E)$ qubits, where the first (leftmost) $B$ qubits are used for storing the QNFN representations $\big|\psi_h\big\rangle$, while the latter $D_E$ qubits will be used to accumulate the sum of performances $\big|e_{hi}\big\rangle$. Initialize the register with the superposition

$$\big|\Gamma\big\rangle = \frac{1}{\sqrt{M}}\sum_{h=1}^{M}\big|\psi_h, 0^{D_E}\big\rangle \tag{22}$$

where the last $D_E$ qubits represent the cumulative sum now initialized to zero.

2) *Step 1:* Compute, using $\mathbf{U_F}$, the superposition $|\Omega_1\rangle$ corresponding to the performances of all the QNFNs with

respect to the first pattern $\underline{\xi}_1$ of the TS. Then, using the linear quantum gates introduced in [28] for the sum of two qubit strings, add the performances of $|\Omega_1\rangle$ to $\left|\Gamma\right\rangle$ in such a way that

$$\left|\Gamma\right\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \left|\psi_h, e_{h1}\right\rangle. \tag{23}$$

3) *Step 2:* Compute the superposition $|\Omega_2\rangle$ for the second pattern $\underline{\xi}_2$ of the TS, and using the same quantum gates, add the performances of $|\Omega_2\rangle$ to $\left|\Gamma\right\rangle$

$$\left|\Gamma\right\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \left|\psi_h, e_{h1} + e_{h2}\right\rangle. \tag{24}$$

4) *Step $P$*: Compute the superposition $|\Omega_P\rangle$ for the last pattern $\underline{\xi}_P$ of the TS, and using the same quantum gates, add the performances of $|\Omega_P\rangle$ to $\left|\Gamma\right\rangle$

$$\left|\Gamma\right\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \left|\psi_h, e_{h1} + e_{h2} + \cdots + e_{hP}\right\rangle. \tag{25}$$

Once we have considered all the patterns of the TS, the register $\left|\Gamma\right\rangle$ will contain the superposition (25) where each QNFN is entangled with its whole performance $\left|E_h\right\rangle$ on the TS, i.e.,

$$\left|\Gamma\right\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \left|\psi_h, E_h\right\rangle. \tag{26}$$

*end of Procedure II*

*Remark*: A plain sum of two quantum superpositions consisting of $M$ elements, such as, for example, $|\Omega_1\rangle$ and $|\Omega_2\rangle$, produces as a result a superposition with $M^2$ elements. Consequently, there exist $M(M-1)$ undesirable elements taking the sum of performances of different networks; in the case of $|\Omega_1\rangle$ and $|\Omega_2\rangle$, such elements would be like $\left|\psi_h, e_{h1} + e_{k2}\right\rangle$, with $k \neq h$. As a well-known result of quantum computing, the linear gates proposed in [28] ensure that such elements do not exist in the sum of superpositions. In fact, the superposition resulting in (24) from $|\Omega_1\rangle$ and $|\Omega_2\rangle$ consists of $M$ elements only, each corresponding to exactly one QNFN and whose performance is just the sum of the performances related to the same network.

As illustrated in the following, the training algorithm consists in determining those states of superposition (26) that are characterized by a value of $E_h$ less than a given threshold. It is possible to mark these states by means of a very simple quantum gate. Let the said threshold be $2^d$, $0 \leq d \leq (D_E - 1)$, and the binary representation of $E_h$ be the string

$$E_h^{(D_E)} \;\; E_h^{(D_E-1)} \;\; \cdots \;\; E_h^{(2)} \;\; E_h^{(1)}$$

where

$$E_h = \sum_{q=1}^{D_E} E_h^{(q)} 2^{q-1}, \quad E_h^{(q)} \in \{0, 1\}.$$

In order to verify whether $E_h < 2^d$ or not, it is sufficient to verify that bits $E_h^{(D_E)}$, $E_h^{(D_E-1)}, \ldots, E_h^{(d+1)}$ are all equal to zero. This is obtained by the following BF $G(\cdot)$:

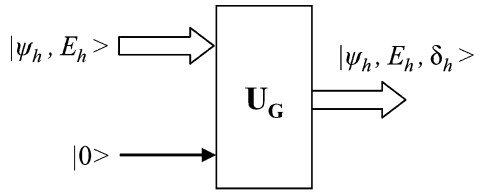



Fig. 4. Linear quantum gate used to verify if the QNFN's performance is less than a given threshold. Wide arrows indicate entangled qubits in strings $|\psi_h, E_h\rangle$ and $|\psi_h, E_h, \delta_h\rangle$.

$$\delta_h = G(E_h) = \text{NOT}\left[\text{OR}\left(E_h^{(D_E)}, E_h^{(D_E-1)}, \ldots, E_h^{(d+1)}\right)\right]. \tag{27}$$

The function performs the logical OR of the bits to be controlled and then the result is negated. The output of the BF is a single bit $\delta_h$, which is set to one if $E_h < 2^d$ and zero otherwise.

The function $G$ can be mapped into the quantum field following a procedure similar to the one previously developed for $F_e$. In this case, the BF is implemented by the linear quantum gate shown in Fig. 4. The gate is driven by any element of superposition (26) and by a single qubit equal to zero. The output of the gate is a string $\left|\psi_h, E_h, \delta_h\right\rangle$ of $(B + D_E + 1)$ qubits, where $\left|\delta_h\right\rangle$ is the flag qubit indicating whether the QNFN's performance is less than the given threshold or not.

The linear quantum gate of Fig. 4 can be associated with a unitary matrix $\mathbf{U_G}$ of order $2^{(B+D_E+1)}$ such that

$$\left|\psi_h, E_h, \delta_h\right\rangle = \mathbf{U_G}\left|\psi_h, E_h, 0\right\rangle. \tag{28}$$

Thus, the gate $\mathbf{U_G}$ entangles every QNFN with its flag qubit $\left|\delta_h\right\rangle$. Moreover, the quantum parallelism assures that, when the whole superposition (26) is present at the input of $\mathbf{U_G}$, then the output will be the superposition $\left|\Xi\right\rangle$ of all the QNFNs entangled with the performance and the flag qubit

$$\left|\Xi\right\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \left|\psi_h, E_h, \delta_h\right\rangle. \tag{29}$$

The procedure illustrated in this section is based on a subsequent application of simple and well-known linear quantum gates. Starting from the superposition (18) of all the possible QNFNs, we obtain the superposition (29) where each network is marked with a qubit indicating if it is a solution of the problem (i.e., if its performance on the TS is acceptable or not). This procedure is the basic step that any quantum search problem is based on. It corresponds to calling the *Oracle* of the problem and solving the question that can be reformulated as "find some QNFN in a set of all the possible solutions such that its performance satisfies the TS under a given error performance."

## VI. Exhaustive Search by Nonlinear Quantum Circuits

In a classical computational framework, we can consider an equivalent Oracle that is called every time we evaluate the TS performance by a BNFN. As discussed in Section IV, there

are necessary $O(M)$ evaluations to solve the search problem using BNFNs. A fundamental result of quantum computing is that, by means of the quantum Oracle described in Section V and other linear quantum gates [29], the upper bound for the number of Oracle's evaluations can be reduced to $O(\sqrt{M})$. The determination of the optimal QNFN is attained in this way by an exhaustive search, since all the possible $M$ solutions are present in the processed superpositions.

The main limitation of quantum search problems is that, after calling the Oracle, the flag qubits $|\delta_h\rangle$ in (29) are entangled with the QNFN strings $|\psi_h\rangle$. Moreover, a basic property of quantum mechanics is that, when we measure a qubit in a superposition, the latter is destroyed and reduced to *only one* string $|\psi_h, E_h, \delta_h\rangle$, measured with its own probability. Thus, we can measure the flag qubit only once in superpositions like $|\Xi\rangle$. If a solution is found, i.e., $|\delta_h\rangle = |1\rangle$, then the entangled qubits $|\psi_h\rangle$ will represent a solution of the problem, and hence, an optimal QNFN. Conversely, when $|\delta_h\rangle = |0\rangle$, we must recall the Oracle to build a new superposition.

The Grover's algorithm proposed in [29] and all its derivations make use of linear quantum gates to change the probability amplitude of the states in the superposition (18), and consequently, in the superposition (29) as well. In this way, the probability of states representing a solution will tend to 1 after $O(\sqrt{M})$ successive applications of the Oracle procedure. However, the implementation of such algorithms on a quantum hardware may still require a huge amount of computational time, even for very simple QNFNs. Considering the reference case, where $M \cong 2 \times 10^{31}$, the overall complexity would be reduced to about $4.5 \times 10^{15}$ possible evaluations. Thus, linear quantum gates only are not sufficient to make the QNFN training feasible.

The complexity of search problems is heavily reduced with respect to classical quantum algorithms when nonlinear quantum circuits are used. This result follows from the quantum algorithms suggested in [30], regarding the implication of a nonlinear regime in quantum processing. Namely, one of these algorithms can ascertain with certainty if a problem has either optimal solutions or not, by requiring only a single evaluation of the Oracle and the iterative application of a suitable two-qubit nonlinear operator. The nonlinear operator is moreover feasible by the help of ordinary linear quantum gates and much simpler and more natural single qubit nonlinear operators, justified using the model of nonlinear quantum mechanics put forth by Weinberg. An improved justification about the use of nonlinear quantum gates is further available in [6], [31], and [32].

By using the algorithms proposed in [30], we will consider the nonlinear quantum gate denoted as "Nonlinear Abrams & Lloyd" (NAL), which is able to determine if a solution (i.e., an optimal QNFN) is present in a superposition of states. For instance, let us consider the superposition $|\Xi\rangle$ obtained in (29) after the Oracle's evaluation. When $|\Xi\rangle$ is at the input of the NAL gate, its output will be the following superposition $|\Upsilon\rangle$

$$|\Upsilon\rangle = \frac{1}{\sqrt{M}} \sum_{h=1}^{M} |\psi_h, E_h\rangle \otimes |\chi\rangle . \tag{30}$$

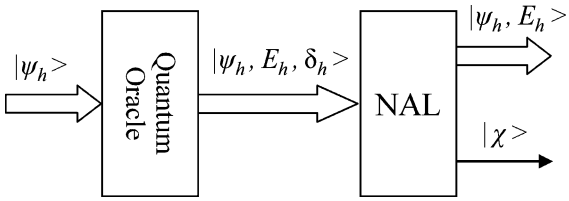



Fig. 5. Example of application of the NAL quantum gate to solve the exhaustive search problem. The "Quantum Oracle" is implemented using the linear circuits and the procedures illustrated in Section V. The qubit $|\chi\rangle$ at the output of the NAL gate is disentangled from $|\psi_h, E_h\rangle$.

The tensor product reveals that the QNFN strings $|\psi_h, E_h\rangle$, $h = 1 \dots M$, are disentangled from the last qubit $|\chi\rangle$. More precisely, $|\chi\rangle = |1\rangle$ for every element of $|\Upsilon\rangle$ if there exists *at least one* element of $|\Xi\rangle$ for which $|\delta_h\rangle = |1\rangle$, i.e., if at least one QNFN is optimal. Otherwise, $|\chi\rangle = |0\rangle$ for every element of the superposition. The application of the whole quantum circuit, using the Oracle and NAL gates, is summarized in Fig. 5.

We propose in the following an algorithm for the exhaustive search of the optimal QNFN by means of the quantum operators introduced up until now. For convenience, we make explicit the binary representation of a generic QNFN $|\psi_h\rangle$

$$|\psi_h\rangle = |\psi_h^{(B)}\ \psi_h^{(B-1)}\ \cdots\ \psi_h^{(2)}\ \psi_h^{(1)}\rangle$$
$$|\psi_h^{(q)}\rangle \in \{|0\rangle, |1\rangle\}, \quad q = 1 \dots B .$$

Similarly, the optimal QNFN $|\psi_{\text{opt}}\rangle$ found by the following algorithm will be represented as

$$|\psi_{\text{opt}}\rangle = |\psi_{\text{opt}}^{(B)}\ \psi_{\text{opt}}^{(B-1)}\ \cdots\ \psi_{\text{opt}}^{(2)}\ \psi_{\text{opt}}^{(1)}\rangle$$
$$|\psi_{\text{opt}}^{(q)}\rangle \in \{|0\rangle, |1\rangle\}, \qquad q = 1 \dots B .$$

Using these notations, the proposed algorithm is based on the iterative application of the following steps.

*Procedure III:*

1) Consider the superposition (18) and divide it into two subsets $|\Psi_1\rangle$ and $|\Psi_0\rangle$ on the basis of the first qubit $|\psi_h^{(B)}\rangle$, respectively, set to $|1\rangle$ or $|0\rangle$. That is

$$|\Psi_1\rangle = \frac{1}{\sqrt{M_1}} \sum_{h=1}^{M_1} |1\ \psi_h^{(B-1)}\ \cdots\ \psi_h^{(1)}\rangle$$
$$|\Psi_0\rangle = \frac{1}{\sqrt{M_1}} \sum_{h=1}^{M_1} |0\ \psi_h^{(B-1)}\ \cdots\ \psi_h^{(1)}\rangle \tag{31}$$

where each superposition consists of $M_1$ states determined by the remaining $B-1$ qubits. Hence, $M_1 = M/2 = 2^{(B-1)}$.

Apply the superposition $|\Psi_1\rangle$ to the cascade of quantum Oracle and NAL gates illustrated in Fig. 5; its output will be the superposition denoted as $|\Upsilon_1\rangle$. In the same manner, apply the superposition $|\Psi_0\rangle$, and obtain the superposition $|\Upsilon_0\rangle$. The following two cases are possible:

   a) The qubit $|\chi\rangle$ is $|0\rangle$ for both $|\Upsilon_1\rangle$ and $|\Upsilon_0\rangle$. This means that no optimal solutions are present, and

hence, no BNFNs satisfy the TS for the given accuracy. In this case, the algorithm is stopped, and we should consider if the threshold used for discriminating the TS performance has been set too low and it may be increased.

b) Optimal solutions are present in one or both the two subsets. In this case, the procedure continues and it is applied to *only one* of the two subsets where an optimal solution is present. Namely, let $\left|\psi_{\text{opt}}^{(B)}\right\rangle = \left|1\right\rangle$ if an optimal solution is found in $\left|\Psi_1\right\rangle$, because $\left|\chi\right\rangle = \left|1\right\rangle$ in the superposition $\left|\Upsilon_1\right\rangle$. Otherwise, let $\left|\psi_{\text{opt}}^{(B)}\right\rangle = \left|0\right\rangle$.

2) Consider the superposition of QNFN states, where the first qubit is set to $\left|\psi_{\text{opt}}^{(B)}\right\rangle$. Then, divide it into two subsets, by rebuilding the superpositions $\left|\Psi_1\right\rangle$ and $\left|\Psi_0\right\rangle$ on the basis of the second qubit $\left|\psi_h^{(B-1)}\right\rangle$, respectively, set to $|1\rangle$ or $|0\rangle$. That is

$$\left|\Psi_1\right\rangle = \frac{1}{\sqrt{M_2}} \sum_{h=1}^{M_2} \left|\psi_{\text{opt}}^{(B)}\ 1\ \cdots\ \psi_h^{(1)}\right\rangle$$

$$\left|\Psi_0\right\rangle = \frac{1}{\sqrt{M_2}} \sum_{h=1}^{M_2} \left|\psi_{\text{opt}}^{(B)}\ 0\ \cdots\ \psi_h^{(1)}\right\rangle \qquad (32)$$

where each superposition consists now of $M_2$ states determined by the remaining $B-2$ qubits. Hence, $M_2 = M/4 = 2^{(B-2)}$.

Apply *only* $\left|\Psi_1\right\rangle$ to the Oracle and NAL gates so that the superposition denoted as $\left|\Upsilon_1\right\rangle$ is obtained. If $\left|\chi\right\rangle = \left|1\right\rangle$ in $\left|\Upsilon_1\right\rangle$, then the algorithm will continue by setting $\left|\psi_{\text{opt}}^{(B-1)}\right\rangle = \left|1\right\rangle$. If $\left|\chi\right\rangle = \left|0\right\rangle$, we are sure that an optimal QNFN exists in $\left|\Psi_0\right\rangle$, since the algorithm has not been stopped in the previous step. Thus, the algorithm will continue by setting $\left|\psi_{\text{opt}}^{(B-1)}\right\rangle = \left|0\right\rangle$, with no more calls to the Oracle in this step.

3) The procedure continues in the same manner by determining iteratively the other $B-2$ qubits of the optimal solution, i.e., $\left|\psi_{\text{opt}}^{(B-2)}\right\rangle$, $\left|\psi_{\text{opt}}^{(B-3)}\right\rangle, \ldots, \left|\psi_{\text{opt}}^{(1)}\right\rangle$. At each step, the superpositions are characterized by $M_Q = 2^{(B-Q)}$ elements, with $Q = 3, 4, \ldots, B$.

*end of Procedure III*

The total cost of the proposed procedure is polynomial. It is able to find an optimal QNFN using $B+1$ iterations. In fact, each step requires one evaluation only of the quantum Oracle and of the NAL gate, but for the first step, it applies the procedure to two separate superpositions. Thus, the resulting complexity for the proposed algorithm is $\mathrm{O}(B) = \mathrm{O}(\log_2 M)$. Considering the reference case, for which $M \cong 2 \times 10^{31}$ and $B = 104$, the search problem would be completed after about 100 evaluations; therefore, it requires much less time than using linear quantum gates only.

## VII. Step-by-Step Guide to the Proposed Method

1) Training set data $\underline{\xi}_i = (\underline{x}_i, t_i)$, $i = 1 \ldots P$:
   a) $\underline{x}_i$ is the input consisting of $N$ components $x_{ij}$, $j = 1 \ldots N$, each coincident with a string of $W_j$ bits.
   b) $t_i$ is the output consisting of a string of $L$ bits.
2) Architecture of the BNFN (see Fig. 1):
   a) $R$ rules of zero-order Sugeno type (5).
   b) The fuzzy quantities $A_{kj}$, $k = 1 \ldots R$, appearing in (5) are defined by strings $a_{kj}$ of $W_j$ bits.
   c) The output $c_k$ of the $k$th rule is a string of $L$ bits.
3) Representation of the BNFN (see Fig. 1):
   a) String $\psi_h$, representing the $h$th BNFN, defined in (13) and (14).
   b) Number $B$ of bits, composing $\psi_h$, defined in (6).
   c) Number $M$ of all the possible BNFNs represented by $B$ bits: $M = 2^B$.
4) Implementation of the BNFN yielding the output $u_i = F_u\ (\underline{x}_i)$:
   a) Determination using (7) of the MF $\mu_{ikj}$ of the input component $x_{ij}$ with respect to the $j$th antecedent of the $k$th rule: *details in Appendix A*.
   b) Determination using (8) of the reliability $\gamma_{ik}$ of the $k$th rule with respect to the input $\underline{x}_i$, by handling the MFs of the antecedents: *details in Appendix B*.
   c) Determination of the bits of the output $u_i$: *details in Procedure I, which involves Fig. 6, Appendixes C, and D*.
5) Determination using (12) and Fig. 2 of the performance $e_i$ of the BNFN with respect to the pattern $\underline{\xi}_i$: *details in Appendix E*.
6) Mapping (18) of the entire set of BNFNs into the quantum field, i.e., into the QNFNs $\left|\Psi\right\rangle$.

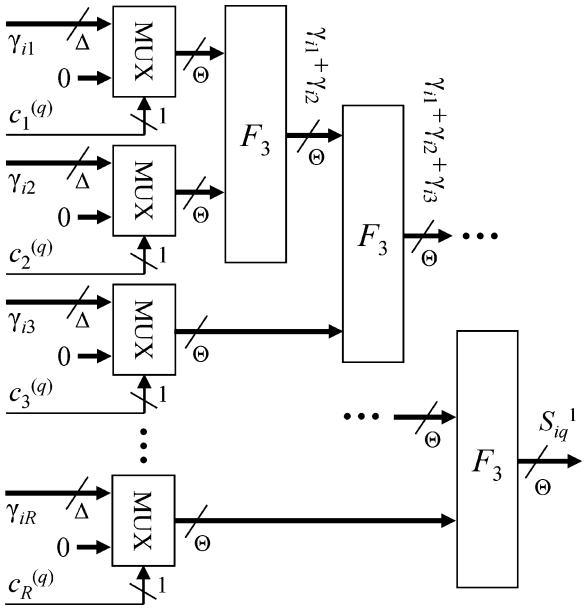



Fig. 6. "$F_3$ array" for the computation of $S_{iq}^1$ (or $S_{iq}^0$). On the basis of the control bit $c_k^{(q)}$, each multiplexer selects its output between the reliability $\gamma_{ik}$ and 0. The representation of reliabilities is extended from $\Delta$ to $\Theta$ bits in order to comply with the requirements of the $F_3$ adders for carry saving.

7) Entanglement (19) of $|\psi_h\rangle$ to its performance $|e_{hi}\rangle$ on pattern $\underline{\xi}_i$: *details in (17) and Fig. 3*.
8) Entanglement (26) of $|\psi_h\rangle$ to its performance $|E_h\rangle$ on the entire training set. The performance is represented by a string of $D_E$ qubits (21): *details in Procedure II*.
9) Entanglement (29) of $|\psi_h\rangle$ to a single qubit $|\delta_h\rangle$, which is set to $|1\rangle$ if $|\psi_h\rangle$ yields an optimal solution. This situation arises when $E_h$ is less than a threshold $2^d$, $0 \leq d \leq (D_E - 1)$: *details in (27), (28), and Fig. 4*.
10) Determination of an optimal solution, if it exists, by exhaustive search: *details in Procedure III*.

## VIII. Conclusion

In the present paper, we investigate the application of quantum processing to neurofuzzy networks. Since these networks are based on the use of rules, preliminarily the rules must be tailored to quantum processing. This result is attained by introducing a suitable BNFN, characterized by a particular membership function in the antecedent of each rule of zero-order Sugeno type and by a suitable inference mechanism.

The determination of the optimal neurofuzzy network is undertaken by transforming a BNFN into a QNFN. After this step, a quantum Oracle is defined in order to obtain a superposition, where optimal solutions are marked by a dedicated qubit. The extraction of these optimal solutions is carried out by using a specific nonlinear quantum algorithm. Hence, the resulting learning algorithm applies no optimization method. It is replaced by an exhaustive search of the optimal solutions. It is important to note that such a learning procedure is infeasible with any traditional algorithm because of the computational cost required, which is too large, to consider all the possible solutions. Exhaustive search is possible as a consequence of the exponential speedup of the nonlinear quantum algorithm, which carries out the extraction of the optimal solution from the superposition of an exponential number of different QNFNs.

The method proposed in the paper is optimal with respect to the values of its parameters but not with respect to its complexity, i.e., the number of rules. This last optimization could be carried out as in the case of classical neurofuzzy networks by cross validation, following a constructive approach. In this case, the proposed algorithm would only be a part of the entire learning procedure, starting from a small number of rules up to the optimal one; this learning procedure can be guided, for example, by the performance on a validation set.

## Appendix A

### Boolean Function $F_1$ Calculating the Similarity of an Input Value to an Antecedent Fuzzy Set

The determination of the MF $\mu_{ikj}$ of the input component $x_{ij}$, with respect to the $j$th antecedent of the $k$th rule, preliminarily requires to define the BF that relates the value of the MF to $x_{ij}$ and $a_{kj}$. It is also necessary to establish the number of bits used for representing this MF. As said, in this paper, we mainly consider neurofuzzy networks dealing with Boolean variables representing nominal attributes. Thus, they are represented by strings of bits, where the position of each bit does not have a relevance with respect to the position of the other bits. In this case, a simple distance between $x_{ij}$ and $a_{kj}$, which are both represented by $W$ bits[1] can be based on the well-known Hamming distance $H_{ikj}$ between these strings. This metric is based on a temporary string of $W$ bits, where every bit is set to zero if the bits in the corresponding positions of $x_{ij}$ and $a_{kj}$ are identical; otherwise, the bit is set to one. This is logically obtained by the bitwise XOR between the two strings. The Hamming distance is the number of bits equal to one in the string calculated this way. Thus, $H_{ikj}$ is an integer ranging from 0, when $x_{ij}$ and $a_{kj}$ are perfectly identical, up to $W$.

TABLE I
Definition of the MF in the Reference Case

| Hamming distance $0 \leq H_{ikj} \leq 4$ | MF dec. | MF bin. ($\Delta = 2$) |
|---|---|---|
| 0 | 3 | 11 |
| 1 | 2 | 10 |
| 2 | 1 | 01 |
| $> 2$ | 0 | 00 |

TABLE II
Some Rows of the Truth Table $\mathbf{T}_1$

| $x_{ij}$ | $a_{kj}$ | $H_{ikj}$ | $\mu_{ikj}$ | $\mathbf{T_1}$ row $x_{ij}$ | $a_{kj}$ | $\mu_{ikj}$ |
|---|---|---|---|---|---|---|
| 0000 | 0111 | 3 | 0 | 0000 | 0111 | 00 |
| 0001 | 0000 | 1 | 2 | 0001 | 0000 | 10 |
| 0010 | 0010 | 0 | 3 | 0010 | 0010 | 11 |
| 0110 | 1001 | 4 | 0 | 0110 | 1001 | 00 |
| 1111 | 1001 | 2 | 1 | 1111 | 1001 | 01 |

The MF $\mu_{ikj}$ will be represented by a positive integer coincident with a string of $\Delta$ bits: it is maximum, i.e., it coincides with a string of bits all equal to one, when $H_{ikj}$ is zero; it decreases to zero when $H_{ikj}$ is larger than a suitable threshold (an integer). In the reference case, where $W = 4$, we assume $\Delta = 2$ and a possible MF can be obtained, as indicated in Table I. It is important to remark that it is not necessary to normalize the values of MFs between 0 and 1, since the determination of the BNFN's output is based on (9) without using specific values of MFs and rule reliabilities.

The value of $\mu_{ikj}$ can be expressed by a BF $F_1(\cdot)$, depending on $x_{ij}$ and $a_{kj}$, as evidenced in (7). The BF has $2W$ input bits ($W$ related to $x_{ij}$ and $W$ to $a_{kj}$) and $\Delta$ output bits (for representing $\mu_{ikj}$). Thus, it can be represented by a truth table, which is a binary matrix $\mathbf{T}_1$ of order $2^{2W} \times (2W + \Delta)$. In the reference case, $F_1$ is a BF with eight input and two output bits, having chosen $\Delta = 2$. The order of matrix $\mathbf{T}_1$ is therefore $256 \times 10$ and it is easy to derive. For illustration, we report in Table II five indicative rows of $\mathbf{T}_1$.

We outline that $F_1$ depends on $a_{kj}$ for two reasons. First, $a_{kj}$ is varied during the training process where, as discussed in Section III-B, the BNFN is used to calculate the TS performance. On the other hand, if the value of $a_{kj}$ is handled as a generic

[1] In the following, we will consider for simplicity that every rule's antecedent will by represented by a same number $W$ of bits.

TABLE III
SOME ROWS OF THE TRUTH TABLE $\mathbf{T}_2$

| $\mu_{ik1}$ dec. (bin.) | $\mu_{ik2}$ dec. (bin.) | $\mu_{ik3}$ dec. (bin.) | $\gamma_{ik}$ dec. (bin.) | $\mathbf{T_2}$ row $\mu_{ik1}$ | $\mu_{ik2}$ | $\mu_{ik3}$ | $\gamma_{ik}$ |
|---|---|---|---|---|---|---|---|
| 1 (01) | 1 (01) | 0 (00) | 0 (00) | 01 | 01 | 00 | 00 |
| 2 (10) | 1 (01) | 2 (10) | 1 (01) | 10 | 01 | 10 | 01 |
| 2 (10) | 3 (11) | 2 (10) | 2 (10) | 10 | 11 | 10 | 10 |
| 3 (11) | 3 (11) | 3 (11) | 3 (11) | 11 | 11 | 11 | 11 |

input, then the same implementation of $F_1$ can be used for any antecedent of every rule (supposing that the same number of bits is used). Conversely, the complexity of $F_1$ can be reduced once the BNFN's parameters have been established and the values of $a_{kj}$ have been fixed. In this case, the BF can be represented by a smaller truth table of order $2^W \times (W + \Delta)$, although it will be different for each rule and each antecedent.

The proposed BNFN may also operate on integer-valued or ordered variables. In this case, the bit strings $x_{ij}$ and $a_{kj}$ can be considered to be integers between 0 and $2^W - 1$. That requires the Hamming distance to be replaced by a more suited metric in order to take into account that bits represent now binary digits. Namely, a difference of two bits has a weight that depends on their position in the strings. For instance, the value of $H_{ikj}$ can be replaced by the absolute difference $|x_{ij} - a_{kj}|$ between the two integers. Besides this change, and further adjustments in the thresholds, the value of $\mu_{ikj}$ can be obtained using the same procedure previously illustrated and the same structure for $\mathbf{T}_1$.

## APPENDIX B
## BOOLEAN FUNCTION $F_2$ CALCULATING THE RELIABILITY OF A RULE WITH RESPECT TO AN INPUT VALUE

The reliability $\gamma_{ik}$ of the $k$th rule with respect to the input $\underline{x}_i$ is determined by handling the MFs $\mu_{ikj}$, $j = 1 \dots N$, of the antecedents. The inference will be based in the following on the MIN operator. Consequently, $\gamma_{ik}$ will assume a value selected among the integers of the MFs and it can be represented by $\Delta$ bits as well. The resulting BF $F_2(\cdot)$ is characterized, as shown in (8), by $N\Delta$ input bits ($\Delta$ for every MF) and $\Delta$ output bits (for $\gamma_{ik}$). The related truth table can be represented as a binary matrix $\mathbf{T}_2$ of order $2^{N\Delta} \times (N + 1)\Delta$. In the reference case, where $N = 3$, $F_2$ is a BF with six input and two output bits, having chosen $\Delta = 2$ in Appendix A. The matrix $\mathbf{T}_2$ can be determined easily, taking into account that its order is $64 \times 8$. We report, for example, in Table III about four rows of $\mathbf{T}_2$.

## APPENDIX C
## BOOLEAN FUNCTION $F_3$ CALCULATING THE BINARY SUM OF TWO INTEGERS

For every bit of rule outputs, the determination of the overall reliabilities $S_{iq}^1$ and $S_{iq}^0$ of the two groups of rules is undertaken by the sum of integers (i.e., the reliabilities) coincident with a string of $\Delta$ bits. In fact, as discussed in Section III, each group can contain a number of rules from 0 up to $R$, depending on the value of the $q$th bit of rule outputs, $q = 1 \dots L$. A simple and effective implementation of the overall sum can be obtained by the adder tree shown in Fig. 6, which is based on a cascade of identical basic blocks. Each of them implements the BF $F_3(\cdot)$, which is simply the binary sum of two integers. The whole adder tree will be called in the following "$F_3$ array."

For the sum of $R$ reliabilities, there are, in general, necessary $R - 1$ levels in the tree. However, some reliabilities should not be added since the related rules do not belong to that group. In this regard, a multiplexer (MUX) is used in order to select if the reliability $\gamma_{ik}$, $k = 1 \dots R$, has to be used or not. If the $q$th output bit $c_k^{(q)}$ of the $k$th rule is 1, then the MUX's output will be $\gamma_{ik}$ and it will be summed by the successive $F_3$ block. Otherwise, if $c_k^{(q)}$ is zero, the MUX's output will be a string of zeros, which will not contribute to the whole sum. That way, the output of the array will be $S_{iq}^1$. The same array can be used for calculating $S_{iq}^0$, by simply inverting the logic of the MUX (i.e., its output is $\gamma_{ik}$ when $c_k^{(q)}$ is zero).

The $F_3$ blocks into the array must use a proper number of bits for carry saving. Taking into account that the reliabilities are positive integers between 0 and $2^\Delta - 1$, and each group can contain at most $R$ rules, the maximum value for any sum is $R(2^\Delta - 1)$. Thus, the function $F_3$ should use the same number of bits for both the operands and for the output. This value, which is denoted as $\Theta$, is obtained by the following expression:

$$\Theta = 1 + \left\lfloor \log_2 \left( R(2^\Delta - 1) \right) \right\rfloor \tag{33}$$

where $\lfloor \cdot \rfloor$ denotes the greatest integer less than or equal to the argument; the BF has therefore $2\Theta$ input bits ($\Theta$ for each operand) and $\Theta$ output bits. Its truth table is a binary matrix $\mathbf{T}_3$ of order $2^{2\Theta} \times 3\Theta$ and its derivation is trivial, since it represents the binary sum of two integers. In the reference case, where $\Delta = 2$ and $R = 8$, we have $\Theta = 1 + \lfloor \log_2 (8 \cdot 3) \rfloor = 5$, and hence, the order of $\mathbf{T}_3$ is $1024 \times 15$.

As for the BF $F_1(\cdot)$, the $F_3$ array can be simplified once the coefficients $c_k^{(q)}$ have been determined by the learning procedure. In addition, the array can be customized for the computation of each sum by using only the reliabilities that actually contribute to it. Leaving the $F_3$ blocks identical for all the arrays, they can also use a reduced number of bits. This value, which is denoted as $\Theta_{\mathrm{red}}$, is given by

$$\Theta_{\mathrm{red}} = 1 + \left\lfloor \log_2 \left( \rho_{\max} (2^\Delta - 1) \right) \right\rfloor \tag{34}$$

where $\rho_{\max}$ is the maximum number of rules contained among all the possible groups (there are on the whole $2L$ groups to be considered: two of them for each output bit). Consequently, the BF $F_3(\cdot)$ is simplified to $2\Theta_{\mathrm{red}}$ input bits and $\Theta_{\mathrm{red}}$ output bits.

For instance, we may consider the reference case where there is only one output bit per rule. Let the training procedure determine the output bits in such a way that there are two groups of rules: The first group contains the first four rules having $c_k^{(1)} = 1$, $k = 1 \dots 4$; the second group contains the other rules for which $c_k^{(1)} = 0, k = 5 \dots 8$. Consequently, the value of $S_{i1}^1$ is given by

$$S_{i1}^1 = \gamma_{i1} + \gamma_{i2} + \gamma_{i3} + \gamma_{i4}\,. \tag{35}$$

The simplified $F_3$ array consists in this case of three levels, as shown in Fig. 7, and the MUXs are no longer

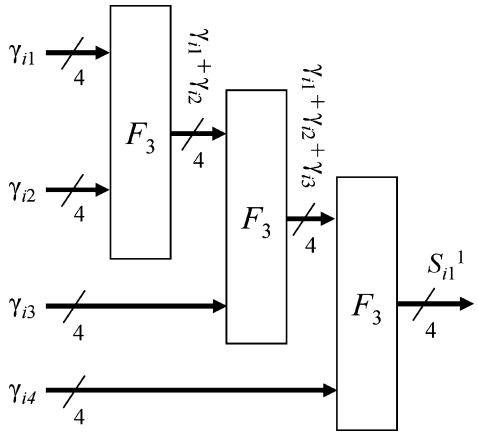



Fig. 7. Simplified $F_3$ array for the computation of $S^1_{i1}$ in the reference case. The representation of reliabilities is extended from $\Delta = 2$ to $\Theta_{\rm red} = 4$ bits.

TABLE IV
SOME ROWS OF THE TRUTH TABLE $\mathbf{T}_4$

| $S^1_{iq}$ dec. (bin.) | $S^0_{iq}$ dec. (bin.) | $u_i^{(q)}$ | $\mathbf{T_4}$ row $S^1_{iq}$ | $S^0_{iq}$ | $u_i^{(q)}$ |
|---|---|---|---|---|---|
| 0 (00000) | 7 (00111) | 0 | 00000 | 00111 | 0 |
| 3 (00011) | 1 (00001) | 1 | 00011 | 00001 | 1 |
| 6 (00110) | 5 (00101) | 1 | 00110 | 00101 | 1 |
| 8 (01000) | 9 (01001) | 0 | 01000 | 01001 | 0 |

necessary. By this subdivision of rules, we have $\rho_{\max} = 4$, $\Theta_{\rm red} = 1 + \lfloor \log_2 (4 \cdot 3) \rfloor = 4$, and $\mathbf{T}_3$ is simplified to a $256 \times 12$ matrix.

## APPENDIX D

## BOOLEAN FUNCTION $F_4$ CALCULATING THE GREATEST BETWEEN TWO INTEGERS

The value of the output bit $u_i^{(q)}$ assigned by the BNFN to the input pattern can be obtained by means of the BF $F_4(\cdot)$, which determines the greatest between integers $S^1_{iq}$ and $S^0_{iq}$, as defined in (9). This BF is characterized by $2\Theta$ input bits ($\Theta$ related to $S^1_{iq}$ and $\Theta$ to $S^0_{iq}$) and one output bit (for $u_i^{(q)}$). The truth table of $F_4$ can be represented by a binary matrix $\mathbf{T}_4$ of order $2^{2\Theta} \times (1 + 2\Theta)$. In the reference case, we have $\Theta = 5$, and consequently, $\mathbf{T}_4$ will be of order $1024 \times 11$. We report in Table IV four illustrative rows of $\mathbf{T}_4$.

## APPENDIX E

## BOOLEAN FUNCTION $F_5$ CALCULATING THE PERFORMANCE OF THE BNFN ON AN EXAMPLE OF THE TS

As defined in (12), the BNFN's performance $e_i$ can be obtained by a BF $F_5(\cdot)$ measuring the matching between the actual BNFN's output and the desired TS output. More precisely, this can be obtained using the Hamming distance between the strings $u_i$ and $t_i$, similarly to what is illustrated in Appendix A. Since output values are strings of $L$ bits, $e_i$ will be an integer between 0 and $L$; hence, it can be represented by a binary string of $D$ bits, where

$$D = 1 + \lfloor \log_2 L \rfloor . \tag{36}$$

TABLE V
SOME ROWS OF THE TRUTH TABLE $\mathbf{T}_5$

| $u_i$ | $t_i$ | $e_i$ dec. (bin.) | $\mathbf{T_5}$ row $u_i$ | $t_i$ | $e_i$ |
|---|---|---|---|---|---|
| 0000 | 0111 | 3 (011) | 0000 | 0111 | 011 |
| 0001 | 1110 | 4 (100) | 0001 | 1110 | 100 |
| 0110 | 0110 | 0 (000) | 0110 | 0110 | 000 |
| 1100 | 1111 | 2 (010) | 1100 | 1111 | 010 |
| 1110 | 1111 | 1 (001) | 1110 | 1111 | 001 |

In this way, the BF $F_5(\cdot)$ has $2L$ input bits ($L$ bits for $u_i$ and $L$ for $t_i$) and $D$ output bits (for coding $e_i$). The BF can be associated with a truth table represented by a binary matrix $\mathbf{T}_5$ of order $2^{2L} \times (2L + D)$. In the reference case, it is $L = 1$, by which $D = 1 + \lfloor \log_2 1 \rfloor = 1$, and $\mathbf{T}_5$ is a trivial matrix of order $4 \times 3$. For the sake of illustration, we also consider a more general case in which $L = 4$, $D = 1 + \lfloor \log_2 4 \rfloor = 3$, and $\mathbf{T}_5$ has order $256 \times 11$. Five indicative rows of this $\mathbf{T}_5$ are illustrated in Table V.

The proposed BNFN can also handle output strings representing integer-valued or ordered variables. We may consider in this regard the remarks of Appendix A concerning the input variables. Consequently, the Hamming distance between $u_i$ and $t_i$ can be replaced by the absolute difference $|u_i - t_i|$. As a consequence of this replacement, since $u_i$ and $t_i$ represent now integers from 0 up to $2^L - 1$, the value of $e_i$ will have the same range. The number of bits $D$ and the order of $\mathbf{T}_5$ should be therefore extended accordingly. In spite of this, the procedure to develop the BF $F_5(\cdot)$ remains identical.

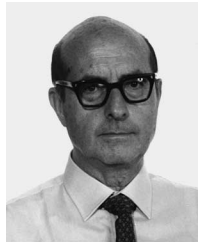
## REFERENCES

[1] Los Alamos preprint server. (2006). [Online]. Available: http://xxx.lanl.gov/archive/quant-ph
[2] S. C. Kak, "Quantum neural computing," *Adv. Imag. Electron Phys.*, vol. 94, pp. 259–314, 1995.
[3] G. Purushothaman and N. B. Karayiannis, "Quantum neural networks (QNNs): Inherently fuzzy feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 679–693, May 1997.
[4] A. Narayanan and T. Menneer, "Quantum artificial neural network architectures and components," *Inf. Sci.*, vol. 128, pp. 231–255, 2000.
[5] A. A. Ezhov and D. Ventura, "Quantum neural networks," in *Future Directions for Intelligent Systems and Information Sciences*, N. Kasabov, Ed. Berlin, Germany: Springer-Verlag, 2000, pp. 213–234.
[6] S. Gupta and R. K. P. Zia, "Quantum neural networks," *J. Comput. Syst. Sci.*, vol. 63, pp. 355–383, 2001.
[7] M. A. Perkowski, "Multiple-valued quantum circuits and research challenges for logic design and computational intelligence communities," *IEEE Connections*, vol. 3, no. 4, pp. 6–12, Nov. 2005.
[8] M. Panella and G. Martinelli "Binary neuro-fuzzy classifiers trained by nonlinear quantum circuits," in *Applications of Fuzzy Sets Theory, Lecture Notes in Artificial Intelligence*, vol. 4578, F. Masulli, S. Mitra, and G. Pasi, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 237–244.
[9] A. Malossini, E. Blanzieri, and T. Calarco, "Quantum genetic optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 231–241, Apr. 2008.
[10] G. J. Iafrate and M. A. Scorsio, "Application of quantum-based devices: Trends and challenges," *IEEE Trans. Electron Devices*, vol. 43, no. 10, pp. 1621–1625, Oct. 1996.
[11] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 1995.

[12] D. Stick, W. Hensinger, S. Olmschenk, M. Madsen, K. Schwab, and C. Monroe, "Ion trap in a semiconductor chip," *Nat. Phys.*, vol. 2, pp. 36–39, Jan 2006.
[13] N. A. Gershenfeld and I. L. Chuang, "Bulk spin resonance quantum computing," *Sci.*, vol. 275, pp. 350–356, 1997.
[14] A. Papageorgiou and J. F. Traub. (2006, Apr.) Qbit complexity of continuos problems. ArXiv:quant-ph/0512082. [Online]. Available: http://www.lanl.gov/archive/quant-ph
[15] M. J. Patyra, "Guest editorial: Fuzzy logic hardware implementations," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 4, pp. 401–402, Nov. 1996.
[16] F. Salam and T. Yamakawa, Eds., "*Special issue on microelectronic hardware implementation of soft computing: Neural and fuzzy networks with learning*," (Comput. Electron. Eng.), vol. 25, no. 5, 1999.
[17] I. Baturone and S. Sanchez-Solano, "Microelectronic design of universal fuzzy controllers," *Mathware Soft Comput.*, vol. 8, pp. 303–319, 2001.
[18] D. Anguita, I. Baturone, and J. Miller, Eds. "*Special Issue on Hardware Implementations of Soft Computing Techniques*," (Appl. Soft Comput.), vol. 4, 2004.
[19] K. Basterretxea, J. M. Tarela, I. del Campo, and G. Bosque, "An experimental study on nonlinear function computation for neural/fuzzy hardware design," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 266–283, Jan. 2007.
[20] P. Echevarria, M. V. Martinez, J. Echanobe, I. del Campo, and J. M. Tarela, "Digital hardware implementation of high dimensional fuzzy systems," in *Applications of Fuzzy Sets Theory, Lecture Notes in Artificial Intelligence*, vol. 4578, F. Masulli, S. Mitra, and G. Pasi, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 245–252.
[21] E. Rieffel and W. Polak, "An introduction to quantum computing for nonphysicists," *ACM Comput. Surv.*, vol. 32, pp. 300–335, 2000.
[22] J. S. Jang, C. T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Upper Saddle River, NJ: Prentice–Hall, 1997.
[23] M. Brown and C. Harris, *Neurofuzzy Adaptive Modeling and Control.* (International Series in Systems and Control Engineering). Upper Saddle River, NJ: Prentice–Hall, 1995.
[24] F. M. Frattale Mascioli and G. Martinelli, "A constructive approach to neurofuzzy networks," *Signal Process.*, vol. 64, pp. 347–358, 1998.
[25] S. Haykin, *Neural Networks (A Comprehensive Foundation)*, 2nd ed. Englewood Cliffs, NJ: Prentice–Hall, 1999.
[26] M. Panella and A. S. Gallo, "An input–output clustering approach to the synthesis of ANFIS networks," *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 1, pp. 69–81, Feb. 2005.
[27] D. Deutsch, "Quantum theory, the church-turing principle, and the universal quantum computer," *Proc. Roy. Soc. Ser. A*, London, U.K., vol. 400, pp. 97–117, 1985.
[28] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Phys. Rev. A, Gen. Phy.*, vol. 54, pp. 147–153, 1996.
[29] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, Philadelphia, PA, May 1996, pp. 212–219.
[30] D. Abrams and S. Lloyd, "Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems," *Phys. Rev. Lett.*, vol. 81, pp. 3992–3995, 1998.
[31] M. Czachor. (1998, Feb.). Notes on nonlinear quantum algorithms. ArXiv:quant-ph/9802051v2., [Online]. Available: http://www.lanl.gov/archive/quant-ph
[32] M. Czachor. (1998, Mar.). Local modification of the Abrams–Lloyd nonlinear algorithm. ArXiv:quant-ph/9803019v1. [Online]. Available: http://www.lanl.gov/archive/quant-ph

**Massimo Panella** (S'01–M'02) was born in Rome, Italy, in 1971. He received the Dr. Eng. degree (with honors) in electronic engineering in 1998 and the Ph.D. degree in information and communication engineering in 2002 from the University of Rome "La Sapienza," Rome, Italy.

In 2001, he joined the Department of Information and Communication (INFOCOM), University of Rome "La Sapienza," where he is currently an Assistant Professor of circuit theory and neural networks. His research activity is related to the use of neural networks, fuzzy logic, and evolutionary algorithms for the solution of both supervised and unsupervised learning problems. His current research interests include pattern recognition, function approximation, nonlinear prediction, and nonlinear system identification for intelligent signal processing and multimedia applications.



**Giuseppe Martinelli** (M'68–SM'74–LS'01) received the Laurea degree in electrical engineering from the University of Rome "La Sapienza," Rome, Italy, in 1958, and specialized in telecommunications with Istituto Superiore delle Poste e delle Telecomunicazioni (ISPT), Rome, in 1959.

During 1963–1964, he was a North Atlantic Treaty Organization (NATO) Fellow at Polytechnic Institute of Brooklyn, Brooklyn, NY, working on pathological circuits. From 1960 to 1967, he was a Researcher with Fondazione Ugo Bordoni (FUB), Rome. Since 1967, he has been with the University of Rome "La Sapienza," where he is a Full Professor of circuit theory. He has authored or coauthored of more than 200 papers published in international journals. His current research interests include circuit theory and applications and general aspects of neural networks.

Prof. Martinelli was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUIT AND SYSTEMS—PART 1, from 1997 to 1999.