Modern Applied Science; Vol. 13, No. 2; 2019
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

High-Order Neural Networks are Equivalent to Ordinary Neural Networks

Abdel Latif Abu Dalhoum[1] & Mohammed Al-Rawi[2]

[1] Department of Computer Science, King Abdulla II School for Information Technology, The University of Jordan, Jordan

[2] Centre de Visió per Computador, Universitat Autonoma de Barcelona, Spain

Correspondence: Abdel Latif Abu Dalhoum, Department of Computer Science, King Abdulla II School for Information Technology, The University of Jordan, Amman 11942, Jordan. E-mail: a.latif@ju.edu.jo

Received: Nov. 14, 2018 Accepted: Nov. 28, 2018 Online Published: January 26, 2019

doi:10.5539/mas.v13n2p228 URL: https://doi.org/10.5539/mas.v13n2p228

Abstract

Equivalence of computational systems can assist in obtaining abstract systems, and thus enable better understanding of issues related their design and performance. For more than four decades, artificial neural networks have been used in many scientific applications to solve classification problems as well as other problems. Since the time of their introduction, multilayer feedforward neural network referred as Ordinary Neural Network (ONN), that contains only summation activation (Sigma) neurons, and multilayer feedforward High-order Neural Network (HONN), that contains Sigma neurons, and product activation (Pi) neurons, have been treated in the literature as different entities. In this work, we studied whether HONNs are mathematically equivalent to ONNs. We have proved that every HONN could be converted to some equivalent ONN. In most cases, one just needs to modify the neuronal transfer function of the Pi neuron to convert it to a Sigma neuron. The theorems that we have derived clearly show that the original HONN and its corresponding equivalent ONN would give exactly the same output, which means; they can both be used to perform exactly the same functionality. We also derived equivalence theorems for several other non-standard neural networks, for example, recurrent HONNs and HONNs with translated multiplicative neurons. This work rejects the hypothesis that HONNs and ONNs are different entities, a conclusion that might initiate a new research frontier in artificial neural network research.

Keywords: neural networks equivalence, high-order, neural networks, recurrent neural networks, cascade correlation, Pi-Sigma networks, dual-output neuron, multilayer perceptron

1. Introduction

Inspired by the biological neuronal systems, several computational intelligent based classification systems have been developed in the past few decades and are widely known as computational neural networks. These computational networks, which possess powerful nonlinear classification ability, are also known in the literature with many other names, such as, artificial neural networks, statistical neural networks, first order neural networks, and multi-layer perceptrons (Minsky & Papert, 1969). In this work, we refer to multilayer feedforward artificial neural net-works that contain only summation activation (Sigma) neurons as ordinary neural networks (ONNs), where the term ordinary refers to first order in this work. When ONNs contain product activation (Pi) neurons, they are frequently called high-order neural networks (HONNs).

HONNs were originally proposed in the 1960s for performing nonlinear discrimination but were discarded due to the tremendous amount of high-order terms (Minsky & Papert, 1969). Starting from the mid-nineties of the last century, several researchers relied on HONNs rather than ONNs to solve classification problems (Jeffries, 1995), (Foltyniewicz, 1995), and (Kosmatopoulos, Polycarpou, Christodoulou, & Ioannou, 1995). Prior to that, (Giles & Maxwell, 1987) discussed HONNs and they stated that high-order weights capture high-order correlations in data. They introduced their HONN to solve a challenging computer vision problem known as invariance. (Hughen & Hollon, 1991) stated that HONNs have the advantage of ease training over multilayer perceptrons and claimed to achieve better classification for radar data than a Gaussian classifier. (Jeffries, 1995) employed HONNs for tracking, code recognition and memory management. (Foltyniewicz, 1995) developed a Pi-Sigma-Pi network structure for effective recognition of human faces in gray scale irrespective to their position, orientation and scale, and he claimed that it has small number of adjustable weights, rapid learning convergence, and excellent

generalization properties. (Abdelbar, 1998) showed, and claimed, that Sigma-Pi HONNs perform better than ONNs in classifying age groups of abalone shellfish benchmark. (Kosmatopoulos, Polycarpou, Christodoulou, & Ioannou, 1995) showed that by allowing enough high-order connections to a recurrent HONN, they were able to use it in approximating arbitrary dynamical systems. Their explanation is that the dynamic components distribute throughout the network in the form of dynamic neurons. (Rovithakis, Robustifying nonlinear systems using high-order neural network controllers, 1999) employed HONNs in control and then (Rovithakis, Chalkiadakis, & Zervakis, High-order neural network structure selection for function approximation applications using genetic algorithms, 2004) presented an algorithm to determine the structure of HONNs for function-approximation. A HONN with a prior knowledge on the training of binary patterns reduces computation time and memory (Artyomov & Yadid-Pecht, 2005). (Al-Rawi & Tarakji, An improved neural network builder that includes graphical networks and PI nodes., 2005) developed a software tool that can be used interactively to build HONNs, then, a HONN that learns the recognition of affine transformed images without any prior knowledge of the imaging geometry was presented in (Al-Rawi, Learning affine invariant pattern recognition using high order neural networks., 2006). Recently, HONNs were used in diverse classification applications; see for example (Dunis, Laws, & Sermpinis, 2010).

We can see from the literature that several researchers prefer HONNs on ONNs after performing a comparison on the problem at hand, but how is that justifiable without further mathematical analysis? As we noted earlier, the major difference between HONNs and ONNs is how activations are calculated in each of their neurons, i.e., only Sigma neurons are used to construct ONNs, while Sigma and Pi neurons or just Pi neurons are used to construct HONNs. In computer architecture, however, a multiplication operation can be implemented via an algorithm that implements several addition operations (Knuth, 1997) and (Kulisch, 2002). In fact, multiplications are defined for the whole numbers in terms of repeated addition and even multiplications of real numbers are defined by systematic generalization of this basic idea. With this in mind, one could, theoretically and/or hypothetically, convert a HONN to a very complicated, large-sized, constrained web-like ONN, but does that mean they are equivalent? In such a case, it is unfair to compare the computational cost and/or the expressive-power of some HONN to an ONN that has the same number of neurons and synaptic connections and claim superiority of HONNs over ONNs.

Despite the fact that many ideas have been proposed to use HONNs in solving various pattern classification problems and compare their classification accuracies to ONNs, there were no theoretical studies of how they are related to each other. Different modifications to HONNs are heuristic, imaginational, and/or model based inspired. The case of many HONNs researches is that experimental results are usually obtained to support the proposed HONN and show its superiority upon ONN, we, however; decided to take the comparison into a mathematical conversion model. Thus, the rational of this work is that, if any HONN is equivalent to some ONN, then we can save lots of unnecessary HONN optimization modeling (that are usually very complicated) by just using its equivalent ONN along with the available ONN optimization algorithms. The equivalence would also provide answers to many controversial comparative issues between HONNs and ONNs. As for the structure of this work, in section 2, we will show how to define the activation of a Pi neuron, that of a Sigma neuron, and we shall discuss how to overcome the numerical instability in the HONN generalized form that we have adopted in this work. The definition of neural net-work equivalence suitable for this work and how a Pi neuron is related to a Sigma neuron will be the topic of section 3. Then, we will show in section 4 using simple mathematical analyses that it is possible to convert any HONN to a similar size equivalent ONN, then, we will give a systematic conversion method in section 5. We will also demonstrate other nonstandard HONNs and few conversion examples in sections 6 and 7 respectively, and finally we wrap up the conclusions in section 8.

2. High-Order Neural Networks Versus Ordinary Neural Networks

ONNs have only Sigma neurons, for example, the output of a $d-2-1$ ONN (i.e., having d inputs with a bias, two hidden neurons, and one output) can be written as (Giles & Maxwell, 1987):

$$z_k = f\left(w_{20} + w_{21} f\left(\sum_{i=1}^{d} w_{1,i} x_i\right) + w_{22} f\left(\sum_{i=1}^{d} w_{2,i} x_i\right)\right), \quad (1)$$

where $\{w_{j,i}\}$ are the weights that connect from neurons in the input layer to neurons in the hidden layer, x_i is the value at the i_{th} input unit, for the input $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$. In contrast, a HONN must at least have one Pi neuron, thus for example, the output of an up to the second order HONN can be written as (Giles & Maxwell, 1987):

$$z_k = f\left(w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} w_{i_1,i_2} x_{i_1} x_{i_2}\right). \quad (2)$$

The above shown equation is a very simple non-general HONN example. The basic units in any HONN, however, are Pi neurons. Many forms of Pi neurons have been used to construct HONNs and this work will adopt a

generalized Pi neuron form that has an adaptable exponent weight, as the one that we shall give below.

2.1 HONNs that We Have Adopted in this Work

Within the context discussed in this work, highly expressive HONNs are given in a form such that their weights are adjustable real-valued numbers (on the contrary to most of the previous works were HONN weights were assumed as non-negative integers). Since HONNs may also contain Sigma neurons, and we will need this Sigma neuron to prove the equivalence later, it is necessary first to give a definition to this type of neuron.

Definition I: The activation of a Sigma neuron

For an input $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$, each Sigma neuron has the following activation:

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = \sum_{i=1}^{d} w_{ji} x_i, \tag{3}$$

where j is the index of Sigma neuron such that $j = 1, \ldots, n$, for a total of n Sigma neurons in a layer, $w_{ji} \in R$ is the synaptic weight between the previous i_{th} neuron and the current j_{th} Sigma neuron, and R is the set of real numbers. The output of a Sigma neuron is usually obtained by modulating its activation by a nonlinear sigmoid activation (transfer) function $f_j(\cdot)$, thus, we can write the output of a Sigma neuron as $y_j = f_j(net_j^{(\text{Sigma})}(\boldsymbol{x}))$.

Definition II: The activation of a Pi neuron

For an input $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]$, each Pi neuron has the following activation:

$$net_j^{(\text{Pi})}(\boldsymbol{x}) = \prod_{i=1}^{d} x_i^{w_{ji}}, \tag{4}$$

where j is the index of Pi neuron such that $j = 1,2, \ldots, n$, for a total of n Pi neurons, $w_{ji} \in R$ is the synaptic weight between the previous i_{th} neuron and the j_{th} Pi neuron and R is the set of real numbers. The output of a Pi neuron is usually obtained by modulating its activation by a nonlinear sigmoid activation function $f_j(\cdot)$, thus, we can write the output of a Pi neuron as $y_j = f_j(net_j^{(\text{Pi})}(\boldsymbol{x}))$.

Many forms of HONNs have been proposed in the literature, they are all based on some structure of Pi and Sigma neurons, and the way their synaptic weights are represented as fixed or adjustable, real-valued or positive integer-valued. To discuss the general case, we shall consider in this work Pi neurons with adaptable real-valued exponent weights, as shown in (4).

2.2 A Complex Valued Synapse in a HONN

The general form used in this work, which (4) shows, to construct HONNs would yield complex values. To demonstrate this issue, let the value of a two dimensional input vector to the Pi order neuron defined in (4) be given by $\boldsymbol{x} = [-1,1]$, then, the its activation would be given by:

$$net_j^{(\text{Pi})}(\boldsymbol{x}) = \prod_{i=1}^{2} x_i^{w_{ji}} = (-1)^{w_{j1}} (1)^{w_{j2}}, \tag{5}$$

and since the weights are real-values, the resultant activation would be complex value, e.g., $w_{j1} = 0.25$ gives $(-1)^{0.25} = 0.707 + 0.707\sqrt{-1}$. If on the other hand, the input vector has a zero value, e.g., $\boldsymbol{x} = [-1,0]$, this will end up in suppressing the activation $net_j^{(\text{Pi})}(\boldsymbol{x}) = (-1)^{w_{j1}} (0)^{w_{j2}} = 0$ if $w_{j2} > 0$, or leading to infinite values if $w_{j2} < 0$. It is therefore necessary for the input to the Pi neuron of the form shown in (4) to be above zero. We shall introduce a normalization scheme to the input to handle this problem, as well as a modification to the activation function so that the output of any neuron is above zero.

2.3 Handling numerical extremes in HONN implementation:

Due to multiplications and exponents used in HONNs, several numerical issues might arise at their computation. These extreme cases are likely to occur in any HONN that implements a Pi neuron of the form shown in (4). In this section, we shed light on two of the most important computational issues in this regards.

In any HONN of the form given in (4), it is possible to have complex neuron output, e.g., when the input to a Pi neuron is negative. This problem can be easily resolved by making all signals propagating inside any HONN to have positive values. There are two sides of positifying the signals of any HONN, the first concerns the input patterns, and the second concerns all other signals propagating inside the network. In the first issue, very easy to normalize the inputs to lie within the (nonnegative) hypercube I^n where $(I = (0,1])$, or to be above zero in general. A support of this type of normalization is given in the patter classification bible by (Duda, Hart, & Stork, 2000) where they hinted that; "In any particular problem, we can always scale the input region to lie in a hypercube, and this condition is not limiting." which means that the normalization process is mostly applicable. In any case, the normalization may simply be performed by just shifting the inputs to have above zero values.

To make sure that all signals propagating inside the network are positive, it is important to carefully choose the activation function(s) so that to always have positive value(s) output, preferably above zero. For example, we propose the following positive-valued logistic activation-function:

$$f(net) = \frac{1}{1+exp(-Q_0(net-\mu))} + \varepsilon, \tag{6}$$

where Q_0 is the temperature of the neuron, net is the activation of the neuron, $\varepsilon > 0$ is a parameter that we used to make sure that the range of the function is always above zero, and $\mu > 0$ is a necessary parameter that we use to ensure the sigmoid-shape of the activation function when the input is positive, in this case the input is net. The value of μ can be fixed or variable, which cab be found using backpropagation of error along with the other weights of the network. The value of Q_0 affects the sigmoid function, for very high values its shape is similar to a step function, while, for low values its shape will be close to linear. Typical values for these parameters are $Q_0 = 0.5$, $\mu = 1$, and $\varepsilon = 0.1$. If one chooses another squash function, e.g. based on $tanh(\cdot)$, then, to ensure above zero values, $\varepsilon > 1$ should be used since the lower bound of $tanh(\cdot)$ is -1.

3. On the Equivalence of Neurons

Conversion and/or equivalence methods exist in many mathematical models as well as computer science models. For example, equivalence methods are found in automata theory as the case of the equivalence (or conversion) between nondeterministic finite automata and deterministic finite automata (Linz, 2006). They are also found in studying complex systems (Abu Dalhoum, 2004), and computational models of natural processes (Madain, Abu Dalhoum, & Sleit, 2018). Before diving into artificial neural networks equivalence theorems, it is necessary to give few definitions.

Definition III: Neural networks equivalence

Two neural networks are equivalent if they both satisfy the following criteria; have the same number of weights, have the same total-number of neurons, they give exactly the same output, but their activation (transfer) functions may differ, and, regardless of the neuron types in each network.

Thus, mathematically converting every Pi neuron (that one may encounter in a HONN) to a Sigma neuron may help in proving HONN to ONN equivalence. This can be done by showing that a Pi neuron is related in some way to a sigma neuron, i.e., one is a function of another. In what follows, we shall provide few theorems as well as conversion methodologies that indicate the equivalence between HONN and ONN with respect to Definition III.

Theorem I: The activation of every Pi neuron is nonlinearly related to the activation of a Sigma neuron

Proof: By making use of $\beta = exp(ln(\beta))$, where β is any dummy variable, equation (4) can be rewritten as:

$$net_j^{(\mathrm{Pi})}(\boldsymbol{x}) = exp\left(ln\left(\prod_{i=1}^{d} x_i^{w_{ji}}\right)\right), \tag{7}$$

which we further reduce to:

$$net_j^{(\mathrm{Pi})}(\boldsymbol{x}) = exp\left(\sum_{i=1}^{d} ln\, x_i^{w_{ji}}\right), \tag{8}$$

and if we assume that $X_i = ln\, x_i$ and $\boldsymbol{X} = [ln\, x_1, ln\, x_2, \ldots, ln\, x_d]$, then we can rewrite (8) as follows:

$$net_j^{(\mathrm{Pi})}(\boldsymbol{x}) = exp\left(\sum_{i=1}^{d} w_{ji} X_i\right), \tag{9}$$

or using the more compact form,

$$net_j^{(\text{Pi})}(\boldsymbol{x}) = exp\big(net_j^{(\text{Sigma})}(\boldsymbol{X})\big), \tag{10}$$

which fulfills the proof.

Equation (10) clearly shows that the activation of a Pi neuron relates nonlinearly to the activation of a Sigma neuron. Any Sigma neuron can be converted to a Pi after taking $ln(\cdot)$ to its inputs, and taking $exp(\cdot)$ to the result of the summation.

Corollary I: From **Theorem I**, any Pi neuron can be converted and/or reduced to a Sigma neuron.

Theorem II: The activation of a Sigma neuron is nonlinearly related to the activation of a Pi neuron.

Proof: The proof is similar to that of ***Theorem I*** after making use of $\beta = ln(exp(\beta))$, where β is any dummy variable. In such case, equation (3) can be rewritten as:

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = ln\big(exp\big(\textstyle\sum_{i=1}^{d} w_{ji}x_i\big)\big), \tag{11}$$

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = ln\big(\textstyle\prod_{i=1}^{d} exp\,(w_{ji}x_i)\big), \tag{12}$$

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = ln\big(\textstyle\prod_{i=1}^{d} (exp(x_i))^{w_{ji}}\big), \tag{13}$$

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = ln\big(\textstyle\prod_{i=1}^{d} (X_i)^{w_{ji}}\big), \tag{14}$$

and by making use of the form shown in (4), we can rewrite (14) as follows:

$$net_j^{(\text{Sigma})}(\boldsymbol{x}) = ln\big[net_j^{Pi}(\chi)\big], \tag{15}$$

where $\chi = [X_1, \ldots, X_i, \ldots, X_d]$, such that $X_i = exp(x_i)$. Equation (15) shows that the activation of a Sigma neuron is nonlinearly related to the activation of a Pi neuron, which fulfils the proof.

It is quite interesting, and surprising, to see that starting with the Sigma neuron we ended with the Pi generalized form that we have adopted in this work, this was way beyond our intention.

Corollary II: From Theorem II, any Sigma neuron can be converted and/or reduced to a Pi neuron.

Remark I: Theorem II will not be used in this work, but was stated here for complement. Theorem II can be used to state that any ordinary neural network can be converted to a high order neural network.

In what follows, we shall make use of *Theorem I* to convert high-order synaptic connections to ordinary synaptic connections, which can be used to convert HONNs to ONNs.

4. Converting High-Order Interconnections to Ordinary Interconnections

In this section, we will show how to convert an interconnection containing Pi neurons (that we shall call high-order interconnection) to an interconnection that contains only Sigma neurons (that we shall call ordinary interconnection) and this concept can be used later to convert any HONN to its equivalent ONN. If we consider a HONN with various Pi and Sigma neurons, then, four different interconnections are probable between each pair of neurons: Sigma-Sigma, Sigma-Pi, Pi-Sigma, and Pi-Pi. Accordingly, ONNs should only have Sigma-Sigma interconnections. In theory, one can convert any HONN to an ONN by converting all of its high-order interconnections to Sigma-Sigma interconnection.

Definition IV: For any neuron_1-neuron_2 interconnection, neuron_1 refers to the current-neuron that is connected to neuron_2 which refers to the next-neuron.

Theorem III: Every Sigma-Pi interconnection can be converted to a Sigma-Sigma interconnection.

Proof: Let the input vector to the current Sigma neuron be $\boldsymbol{x}$, then the output of the current Sigma neuron is given by:

$$y_j = f_j\big(net_j^{(\text{Sigma})}(\boldsymbol{x})\big), \tag{16}$$

and the output of the next Pi neuron is given by:

$$z_k = f_k\left(net_k^{(\text{Pi})}(\boldsymbol{y})\right). \tag{17}$$

Now, substituting (10) into (17) yields:

$$z_k = f_k\left(exp\left(net_k^{(\text{Sigma})}(\boldsymbol{Y})\right)\right), \tag{18}$$

or,

$$z_k = \psi_k\left(net_k^{(\text{Sigma})}(\boldsymbol{Y})\right), \tag{19}$$

where $\psi_k = f_k(exp(\cdot))$, $\boldsymbol{Y} = \{Y_1, \ldots, Y_j, \ldots.\}$ such that $Y_j = ln\, y_j$. Now, by moving $ln(\cdot)$ of this Pi neuron to the previous neuron (previous with respect to Pi neuron is current Sigma, see

Figure 1 for illustration), then, the following equation defines Y_j:

$$Y_j = \varphi_j\left(net_j^{(\text{Sigma})}(\boldsymbol{x})\right), \tag{20}$$

where $\varphi_j = ln(f_j(\cdot))$is the function resulted from the composition of $ln(\cdot)$ and $f_j(\cdot)$. It is obvious from (19) and (20) that a Sigma-Pi interconnection is reducible and equivalent to a Sigma-Sigma interconnection.

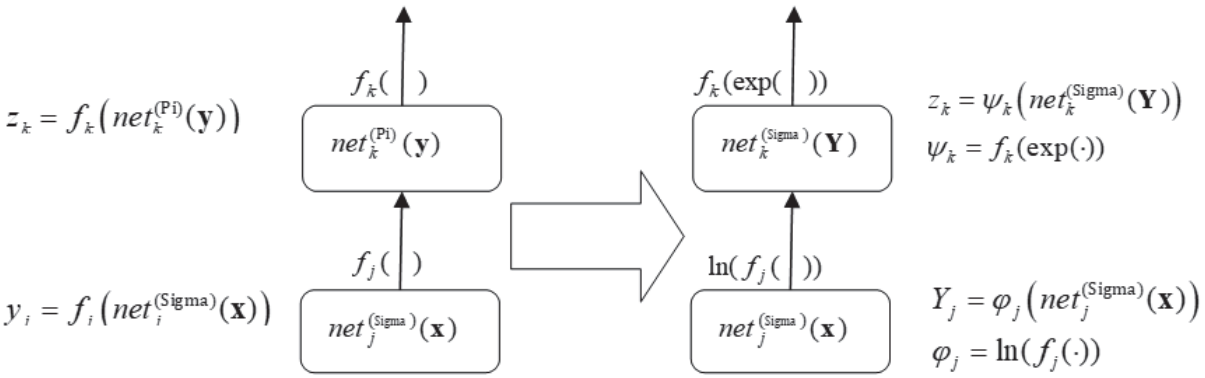

Figure 1. The left part shows Sigma-Pi interconnection converted to a Sigma-Sigma interconnection, which is shown at the right part of the graph

Theorem IV: Every Pi-Sigma interconnection can be converted to a Sigma-Sigma interconnection

Proof: Let the input vector to the current Pi neuron be $\boldsymbol{x}$, then the output of the current Pi neuron will be given by:

$$y_j = f_j(net_j^{(\text{Pi})}(\boldsymbol{x})), \tag{21}$$

the output y_j is then feedforward connected to the next Sigma neuron that has the following output:

$$z_k = f_k(net_k^{(\text{Sigma})}(\boldsymbol{y})). \tag{22}$$

Now, by substituting (10) into (21) we get:

$$y_j = f_j\left(exp\left(net_j^{(\text{Sigma})}(\boldsymbol{X})\right)\right), \tag{23}$$

where as shown previously $X_i = ln\, x_i$. By defining the composite function $\psi_j = f_j(exp(\cdot))$, it would be more appropriate to write (23) as follows:

$$y_j = \psi_j\left(net_j^{(\text{Sigma})}(\boldsymbol{X})\right). \tag{24}$$

It is clear from (24) and (22) that a Pi-Sigma interconnection is equivalent to a Sigma-Sigma interconnection.

***Theorem V**: Every Pi-Pi interconnection can be converted to a Sigma-Sigma interconnection*

Proof: Let $\boldsymbol{x}$ be a vector denoting inputs to the current Pi neuron. Analog to the proof shown above that

demonstrated the conversion of Pi-Sigma and Sigma-Pi interconnections to a Sigma-Sigma interconnection each, it is straightforward to deduce using (10) that a Pi-Pi interconnection, which has current-neuron that has an output $y_j = f_j(net_j^{(\mathrm{Pi})}(\boldsymbol{x}))$ connected to a next-neuron that has an output $z_k = f_k(net_k^{(\mathrm{Pi})}(\boldsymbol{y}))$, can be written as follows:

$$Y_j = \zeta_j\left(net_j^{(\mathrm{Sigma})}(\boldsymbol{X})\right), \tag{25}$$

$$z_k = \psi_k\left(net_k^{(\mathrm{Sigma})}(\boldsymbol{Y})\right), \tag{26}$$

where $\zeta_j = \ln(f_j(\exp(\cdot)))$, $\psi_k = f_k(\exp(\cdot))$, Y_j is the output of the current-neuron after conversion, and z_k is the output of the next-neuron after conversion. Thus, equations (25) and (26) clearly show that a Pi-Pi interconnection can be converted to a Sigma-Sigma interconnection, which fulfills the proof.

Above we showed that it is possible to convert any high-order interconnection to ordinary interconnection. Does this mean, however, that every HONN has an equivalent ONN? In the next section, we will give a very simple method to convert any high-order interconnection to an ordinary interconnection.

5. A Simple Method to Convert A High-Order Neural Network to An Ordinary Neural Network

We proved in section 4 that one could directly convert any high-order interconnection to an equivalent ordinary interconnection by just manipulating the activation function acting at each Pi and/or Sigma neurons.

5.1 The Conversion Method

To convert a HONN to an equivalent ONN (let it be denoted as $\mathrm{ONN}^{(\mathrm{E})}$) in an easy manner, we propose using the following method:

Define a network with the same number of layers and neurons as that of a HONN that you want to convert. Let γ_k be the set containing the activation functions acting at each neuron in the k_{th} layer of the newly defined network, and $\{f_k(\cdot)\}$ be the set of the activation functions acting at each neuron in the k_{th} layer of the HONN that we seek to convert to $\mathrm{ONN}^{(\mathrm{E})}$. Hence, according to the previous theorems, the major goal is finding γ_k as a composite function of elements from the set $\{\ldots, f_{k-1}(\cdot), f_k(\cdot), f_{k+1}(\cdot), \ldots.\}$, and by just doing this, the conversion is complete. Using the equivalence theorems presented in the previous section, the activation function at each Sigma neuron in the newly defined $\mathrm{ONN}^{(\mathrm{E})}$ network can be found according to the following criteria:

Case 1: Current neuron in the k_{th} layer is Sigma, connected to a next Pi neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = ln(f_k(\cdot)). \tag{27}$$

Case 2: Current neuron in the k_{th} layer is Pi, connected to a next Sigma neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = f_k(exp(\cdot)). \tag{28}$$

Case 3: Current neuron in the k_{th} layer is Pi, connected to a next Pi neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = ln(f_k(exp(\cdot))). \tag{29}$$

Thus, the newly defined network after finding γ_k is $\mathrm{ONN}^{(\mathrm{E})}$. We note that $ln(\cdot)$ should be taken to the input patterns whenever they are connected to a Pi neuron at the first hidden layer. Another important issue is that the conversion criteria is a bottom-up approach, i.e., one should start from the input layer and sequentially move to the output layer.

5.2 The Conversion Method and Training with Backpropagation of Error

In training with backpropagation-of-error, one needs the first derivative of each of the above activation functions. For a transfer function denoted as γ_k, let its derivative be denoted by γ_k'. For convince to other interested researchers, we list the derivatives of the conversion cases shown in (27), (28), and (29) below:

Case 1: Current neuron in the k_{th} layer is Sigma, connected to a next Pi neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = ln(f_k(\cdot)) \quad \Rightarrow \quad \gamma_k' = [1/f_k(\cdot)]f_k{}'(\cdot). \tag{30}$$

Case 2: Current neuron in the k_{th} layer is Pi, connected to a next Sigma neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = f_k(exp(\cdot)) \Rightarrow \gamma_k' = [f_k{}'(exp(\cdot))]\, exp(\cdot). \tag{31}$$

Case 3: Current neuron in the k_{th} layer is Pi, connected to a next Pi neuron in the $(k+1)_{\mathrm{th}}$ layer:

$$\gamma_k = ln(f_k(exp(\cdot))) \Rightarrow \gamma_k{}' = [1/f_k(exp(\cdot))][f_k'(exp(\cdot))]\, exp(\cdot). \tag{32}$$

If the output layer however contains a Pi neuron, one can assume that a hypothetical next neuron is Sigma thus uses a Pi to Sigma conversion rule. Finally yet importantly, there is no need to do any conversion for any Sigma-to-Sigma interconnection if the HONN that we want to convert contains any.

5.3 Sigma and Pi Neurons in the Same Layer

Although uncommon in the literature, any HONN may have both Pi and Sigma neurons in the same layer. Fortunately, it is possible to convert HONNs to ONNs when they have this kind of mixed-Sigma-Pi layer (MSP-HONN) topology. The major problem of having two types of neurons in the same layer is that our proposed conversion methodology is based on the current and the next neuron types, thus, it is not possible to have one type of activation function in the resultant ONN$^{(E)}$ (since it is possible that the next neuron be either Sigma or Pi). To illustrate this issue, let us assume that the current neuron is Pi, and the next layer contains both Pi and Sigma neurons. In this case, one will need two conversions for the current neuron, one is Pi-Pi, and the other is Pi-Sigma that results of two possible types of activation functions acting at the current converted neuron. To resolve this issue, we propose using a new type of neurons which we shall call dual-output neurons. Thus, we will have to find the equivalent activation functions for all the neurons of ONN$^{(E)}$ in a way similar to what we have described in section 5. The following method shows how one can directly convert MSP-HONN to ONN$^{(E)}$:

- Define a network with the same number of layers and neurons as the MSP-HONN.

- Find the activation function γ_k of each neuron of the newly defined network topology in a way similar to that of section 5. However, by converting MSP-HONN to ONN, each neuron will have dual-output activation function depending on the current neuron and the next neuron of MSP-HONN. For feasible and easy conversion, we deduced from the rules of section 5 the following:

Case 1: Current neuron is Sigma

$$\gamma_k = \begin{bmatrix} f_k(\cdot) & \text{if the next neuron is Sigma} \\ ln(f_k(\cdot)) & \text{if the next neuron is Pi} \end{bmatrix}, \tag{33}$$

Case 2: Current neuron is Pi

$$\gamma_k = \begin{bmatrix} f_k(exp(\cdot)) & \text{if the next neuron is Sigma} \\ ln(f_k(exp(\cdot))) & \text{if the next neuron is Pi} \end{bmatrix}, \tag{34}$$

Thus, we showed that even a HONN with mixed Sigma and Pi neurons is equivalent to an ONN and the conversion can be performed easily with the method we proposed here. The derivatives for these forms can be found from (30)-(32), for example, case 1 will have the following derivative form:

$$\gamma' = \begin{bmatrix} f_k'(\cdot) & \text{if the next neuron is Sigma} \\ [1/f_k(\cdot)]f_k'(\cdot) & \text{if the next neuron is Pi} \end{bmatrix}. \tag{35}$$

5.3 Does the neuron output in the equivalent ONN have a sigmoid shape?

The last issue we discus in this section, is to illustrate the characteristics of a typical activation function, such as the one we proposed in (6) in relation to the conversion methodology that we propose. Let us consider a neuron that appears after converting a Sigma-Pi interconnection into a Sigma-Sigma interconnection; $\gamma(net) = ln[1/(1 + exp(-Q_0(net - \mu)) + \varepsilon]$. Fig. 2 shows the characteristics of this function for different Q_0 and ε values where we can see in Fig. 2-a that the activation function kept its sigmoid shape for $\mu = 1$ and $\varepsilon = 1$. For $\mu = 1$ and $\varepsilon = 2$ Fig. 2-b shows that the lower part is smoother than the upper part. A rather more complicated activation function is found after converting a Pi-Pi interconnection into a Sigma-Sigma interconnection which is $\gamma(net) = ln[1/(1 + exp(-Q_0\, exp(net - \mu))) + \varepsilon]$, its characteristic graph is shown in Fig. 2-c, where in this case the activation function is asymmetric squash function.

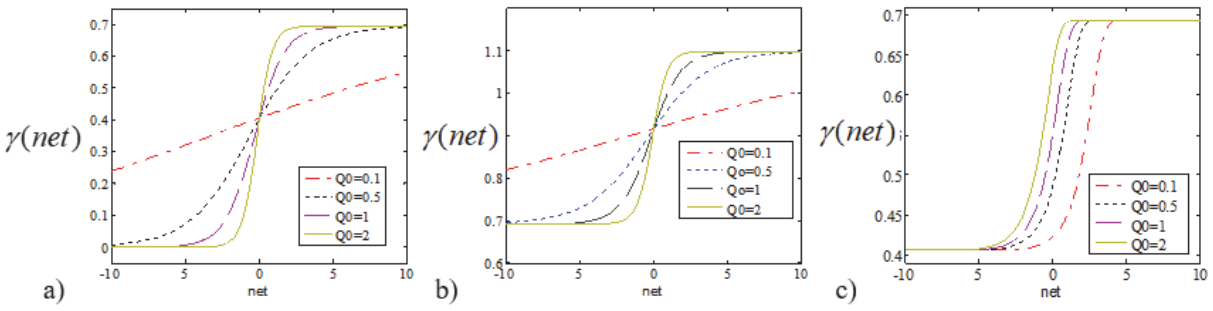

Figure 2. Activation functions that can be used to yield positive outputs. a) The activation function that may result from converting a Sigma-Pi interconnection into a Sigma-Sigma interconnection is $\gamma(net) = ln[1/(1 + exp(-Q_0(net - \mu)) + \varepsilon]$ using $\mu = 1$ and $\varepsilon = 1$, b) the same activation function of (a) is shown

using $\mu = 1$ and $\varepsilon = 2$, c) the activation function that may result from converting a Pi-Pi interconnection into a Sigma-Sigma interconnection is $\gamma(net) = ln[1/(1 + exp(-Q_0\, exp(net - \mu))) + \varepsilon]$ using $\mu = 1$ and $\varepsilon = 1$ we can see that this is an asymmetric sigmoid-shaped

6. Other Non-Standard Honns From the Literature

In this section, we provide few equivalence theorems on nonstandard HONNs and show how they can be converted to ONNs.

Theorem VI*: Every high-order recurrent neural network can be converted to an ordinary recurrent neural network*

Proof: The conversion method demonstrated in section 5.3 would be directly useful to convert a recurrent HONN to a recurrent ONN$^{(E)}$. This is because the activation functions at any of ONN$^{(E)}$'s feedback connections could be found using MSP-HONN conversion rules.

This case is depicted in Fig. 3 where the product neuron P_1 has an input from x_1 and a feedback from S_2. This S_2 neuron should have dual-output activation function after conversion due to S_2-S_3 and S_2-P_1 interconnections. If, however; S_2 is feedforward connected to a next layer that only contains Pi neurons, say P_3, then S_2 will have the usual unary-output activation neuron.

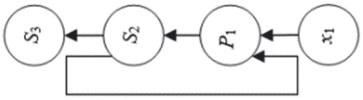

Figure 3. A recurrent high-order interconnection that may appear in recurrent high-order neural networks. Here we have a case of connection from S_2 to S_3 and S_2 to P_1. This means that converting a recurrent high-order neural network is similar in a way similar to that of converting mixed Sigma and Pi neurons in the same layer as previously shown.

Theorem VII*: Every higher- order Functional Neural Network (Giles & Maxwell, 1987) can be converted to a four-layer ordinary neural network.*

Proof. Higher Order Functional Neural Network HOFNN (Giles & Maxwell, 1987) is used as a one-layer neural network, a multiplication of the inputs that are connected to output units. The output of a neuron of a second order HOFNN, which is the same as the definition in (2), can be written as follows.

$$z_k = f(w_0 + y_1 + y_2) \qquad \text{(output Sigma layer)}, \tag{36}$$

where the hidden layer contains one Sigma neuron y_1,

$$y_1 = \sum_{i=1}^{d} w_i x_i, \tag{37}$$

and a weighted sum of several Pi neurons represented as y_2,

$$y_2 = \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} w_{i_1,i_2} x_{i_1} x_{i_2}. \tag{38}$$

For the proof purpose, we have converted the Pi neuron to Sigma neuron using the method we have shown in Theorem I, thus, we can rewrite (38) as follows:

$$y_2 = \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} w_{i_1,i_2} u_{i_1,i_2} \qquad \text{(second hidden Sigma layer)} \tag{39}$$

$$u_{i_1,i_2} = f_1\left(\sum_{i=\{i_1,i_2\}} X_i\right) \qquad \text{(first hidden Sigma layer)}, \tag{40}$$

where $\boldsymbol{X} = [ln\, x_1, ln\, x_2, \ldots, ln\, x_d]$ is the augmented input vector, and $f_1 = exp(\cdot)$. This fulfills the proof that a second order HOFNN is a four layer ONN. We can see from (39) and (40) that we needed to restructure the network to have another hidden layer. Moreover, the double summation that appears in (39) adds to problem at all, it is always possible to expand it linearly as $y_2 = w_{1,1}u_{1,1} + w_{1,2}u_{1,2} + \ldots + w_{d,d}u_{d,d}$, which is the normal layer shape in any artificial neural network.

The output layer of an up to the d -th order HOFFN has the form $z_k = f\left(w_0 + y_1 + y_2 + y_3 + \ldots + y_p, \ldots, y_d\right)$. Using the same method above for up to the second order HOFFN, we can find y_3, y_4,…etc, then using mathematical induction we found that the HOFNN up to the d -th order will have the following form:

$$z_k = f\left(w_0 + y_1 + y_2 + y_3 + \ldots + y_p, \ldots, y_d\right) \quad \text{(output Sigma layer)}, \tag{41}$$

$$y_p = \sum_{i_1=1}^{d} \sum_{i_2=1}^{d} \ldots \sum_{i_p=1}^{d} w_{i_1,i_2,\ldots,i_p} u_{i_1,i_2,\ldots,i_p} \quad \text{(second hidden Sigma layer)}, \tag{42}$$

$$u_{i_1,i_2,\ldots,i_p} = f_1\left(\sum_{i=\{i_1,i_2,\ldots,i_p\}}^{d} X_i\right) \quad \text{(first hidden Sigma layer)}. \tag{43}$$

This fulfills the proof that even up to higher orders; HOFNN can be converted to an ordinary neural network.

Theorem VIII*: Every HONN with n translated multiplicative neurons (Iyoda, Nobuhara, & Hirota, 2003), with has an output given by:*

$$z_k = f\left(b_k \prod_{j=1}^{n} (y_j - w_{kj})\right), \tag{44}$$

$$y_j = f\left(\sum_{i=1}^{d} (w_{ji} x_j - w_{0i})\right), \tag{45}$$

is equivalent to an ONN.

Proof: According to Theorem I, we can write z_k as follows:

$$z_k = f_k\left(w_k + \sum_{j=1}^{n} u_{kj}\right), \tag{46}$$

where w_k is the weight (that mimics $w_k = ln(b_k)$), $f_k = f(exp(\cdot))$, and u_{kj}, is the output of the previous node which has the value:

$$u_{kj} = ln(y_j - w_{kj}). \tag{47}$$

Thus, after manipulation, a HONN with translated multiplicative neurons contains only Sigma neurons.

7. Conversion Examples

The aim of this work was to show, mathematically, the equivalence of HONNs and ONNs. For more illustration, we give below practical conversion examples as well as a simulation experiment.

Example 1: Suppose that a HONN is consisted of three inputs, first hidden layer containing five Pi neurons with activation function P_1 each, and an output layer containing three Sigma neurons with activation function S_2 each. To obtain an equivalent ONN$^{(E)}$, construct an ONN with three inputs, first hidden layer with five Sigma neurons and an output layer with three Sigma neurons. Using the method shown in section 5, we find the activation functions γ_1 and γ_2 for each neuron in the first hidden layer and the output layer respectively:

$$\gamma_1 = ln(P_1()),$$

$$\gamma_2 = S_2(exp()).$$

Example 2: Suppose a HONN has two inputs connected to the first two-neuron hidden layer which has Pi and Sigma neuron having P_1 and S_1 activation functions respectively. The first hidden layer connects to another hidden layer that contains Pi and Sigma neurons having P_2 and S_2 as activation functions respectively. The second hidden layer connects to a single Sigma neuron output layer with activation function S_3. To convert this HONN to ONN$^{(E)}$, we shall use the method described in section 5.3, the activation functions of the equivalent ONN$^{(E)}$ are as shown below:

$$\gamma_{11} = \begin{cases} S_1(\cdot) & \text{for the connection between } S_1 \text{ and } S_2 \\ ln(S_1(\cdot)) & \text{for the connection between } S_1 \text{ and } P_2 \end{cases},$$

$$\gamma_{12} = \begin{cases} P_1(exp(\cdot)) & \text{for the connection between } P_1 \text{ and } S_2 \\ ln(P_1(exp(\cdot))) & \text{for the connection between } P_1 \text{ and } P_2 \end{cases},$$

$$\gamma_{21} = S_2(\cdot),$$

$$\gamma_{22} = P_2(exp(\cdot)),$$

$$\gamma_{31} = S_3(\cdot).$$

Despite the fact that Fig. 4-a contains mixed Pi and Sigma neurons in each hidden layer, the graph shown in Fig. 4-b that depicts the equivalent ONN contains only Sigma neurons.

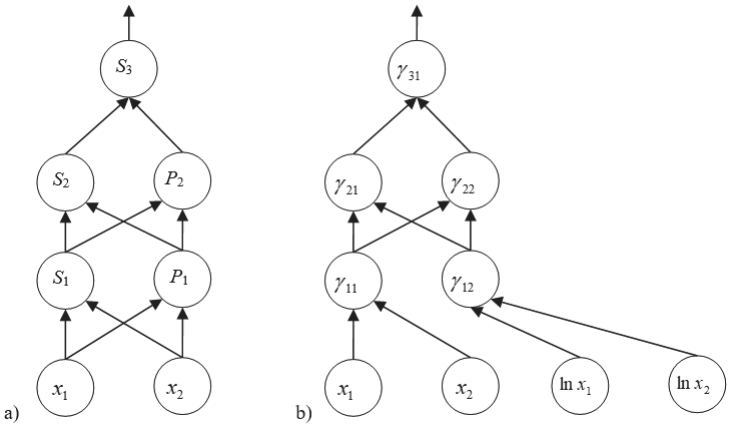

Figure. 4. a) High-order neural network with mixed Pi and Sigma neurons in each layer, b) Converting the network in (a) to an equivalent ordinary neural network using the method described in this work, $\{\gamma_{11}, \gamma_{12}, \gamma_{21}, \gamma_{22}, \gamma_{31}\}$is the set of activation functions that depend on $\{S_1, S_2, P_1, P_2, S_3\}$.

8. Conclusions

HONNs are equivalent to ONNs, thus, this work rejects the hypothesis that HONNs and ONNs are different entities. The proposed conversion of HONN to ONN would permit using the huge amount of optimization algorithms to speed up the convergence of HONN and/or finding better topologies. Recurrent HONNs and cascaded correlation HONNs can be simply defined via their equivalent ONNs and then trained with backpropagation-of-error. The equivalence proof shade new ideas on neural networks mechanisms and topological issues, such as, dual-output neurons. One might generalize this idea on new triple-output or quadruple-output neurons to propose a new type of artificial neural networks, i.e., new ONNs. Mathematically, the converted ONN posses the same features of the original HONN and they both have exactly the same functionality and output. Any of the equivalent ONN will run faster than the HONN since the former uses only additions while the later uses additions and multiplications. A HONN and its equivalent ONN will both have the same number of neurons and synaptic connections and they will only differ by the activation function acting at each neuron. Further equivalence analysis may open a new era for artificial neural networks.

References

Abdelbar, A. M. (1998, 6 01). Achieving superior generalisation with a high order neural network. *Neural Computing {\&} Applications, 7*, 141-146. https://doi.org/10.1007/BF01414166

Abu Dalhoum, A. l. (2004). *Genetic Evolution and equivalence of some complex systems: Fractals, Cellular Automata and Lindenmayer Systems.* Ph.D. dissertation, Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid.

Al-Rawi, M. (2006). Learning affine invariant pattern recognition using high order neural networks. *International Conference on Artificial Intelligence and Machine Learning*, (pp. 24-29). Sharm El Sheikh, Egypt.

Al-Rawi, M., & Tarakji, B. B. (2005). An improved neural network builder that includes graphical networks and PI nodes. *Human Computer Interaction. International Conference on Human Computer Interaction* (pp. 229-237). Al-Zaytoona university, Amman, Jordan.

Artyomov, E., & Yadid-Pecht, O. (2005). Modified high-order neural network for invariant pattern recognition. *Pattern Recognition Letters, 26*, 843-851. https://doi.org/10.1016/j.patrec.2004.09.029

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification (2Nd Edition).* New York, NY, USA: Wiley-Interscience.

Dunis, C. L., Laws, J., & Sermpinis, G. (2010). Modelling and trading the EUR/USD exchange rate at the ECB fixing. *The European Journal of Finance, 16*, 541-560.

Foltyniewicz, R. (1995). Efficient high order neural network for rotation, translation and distance invariant recognition of gray scale images. In V. Hlav{\'a}{\v{c}}, & R. {\v{S}}{\'a}ra (Ed.), *Computer Analysis of Images and Patterns* (pp. 424-431). Berlin: Springer Berlin Heidelberg.

Giles, C. L., & Maxwell, T. (1987, 12). Learning, invariance, and generalization in high-order neural networks.

Appl. Opt., 26, 4972-4978. https://doi.org/10.1364/AO.26.004972

Hughen, J. H., & Hollon, K. R. (1991). Millimeter wave radar stationary-target classification using a high-order neural network., *1469*, 341-350.

Iyoda, E. M., Nobuhara, H., & Hirota, K. (2003, 12 01). A Solution for the N-bit Parity Problem Using a Single Translated Multiplicative Neuron. *Neural Processing Letters, 18*, 233-238. https://doi.org/10.1023/B:NEPL.0000011147.74207.8c

Jeffries, C. D. (1995). Tracking, code recognition, and memory management with high-order neural networks. *2492*, 2492 - 2492 - 10. https://doi.org/10.1117/12.205207

Knuth, D. (1997). *Fundamental Algorithms* (3rd ed.). Addison-Wesley, Reading.

Kosmatopoulos, E. B., Polycarpou, M. M., Christodoulou, M. A., & Ioannou, P. A. (1995, 3). High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks, 6*, 422-431. https://doi.org/10.1109/72.363477

Kulisch, U. (2002). *Advanced Arithmetic for the Digital Computer* (1 ed.). Springer-Verlag Wien.

Linz, P. (2006). *An Introduction to Formal Language and Automata* (4 ed.). MA: Jones and Bartlett Publishing.

Madain, A., Abu Dalhoum, A. L., & Sleit, A. (2018, 3 30). Application of local rules and cellular automata in representing protein translation and enhancing protein folding approximation. *Progress in Artificial Intelligence*. https://doi.org/10.1007/s13748-018-0146-8

Minsky, M., & Papert, S. (1969). *Perceptrons.* MIT Press, Cambridge.

Rovithakis, G. A. (1999, 1). Robustifying nonlinear systems using high-order neural network controllers. *IEEE Transactions on Automatic Control, 44*, 102-108. https://doi.org/10.1109/9.739082

Rovithakis, G. A., Chalkiadakis, I., & Zervakis, M. E. (2004, 2). High-order neural network structure selection for function approximation applications using genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34*, 150-158. https://doi.org/10.1109/TSMCB.2003.811767