DE GRUYTER OPEN

JAISCR, 2016, Vol. 6, No. 2, pp. 119 – 131
10.1515/jaiscr-2016-0010

ENHANCING CONSTRUCTIVE NEURAL NETWORK PERFORMANCE USING FUNCTIONALLY EXPANDED INPUT DATA

João Roberto Bertini Junior[1], Maria do Carmo Nicoletti[2]

[1] *Department of Computer Science, UFSCar, Rodovia Washington Luís, km 235, São Carlos-SP, Brazil*

[2] *Department of Computer Science, UFSCar & FACCAMP, São Carlos & Campo Limpo Paulista - SP, Brazil*

Abstract

Constructive learning algorithms are an efficient way to train feedforward neural networks. Some of their features, such as the automatic definition of the neural network (NN) architecture and its fast training, promote their high adaptive capacity, as well as allow for skipping the usual pre-training phase, known as model selection. However, such advantages usually come with the price of lower accuracy rates, when compared to those obtained with conventional NN learning approaches. This is, perhaps, the reason for conventional NN training algorithms being preferred over constructive NN (CoNN) algorithms. Aiming at enhancing CoNN accuracy performance and, as a result, making them a competitive choice for machine learning based applications, this paper proposes the use of functionally expanded input data. The investigation described in this paper considered six two-class CoNN algorithms, ten data domains and seven polynomial expansions. Results from experiments, followed by a comparative analysis, show that performance rates can be improved when CoNN algorithms learn from functionally expanded input data.

Keywords: Constructive neural networks; Functional link artificial neural networks; Functionally expanded input data.

1 Introduction

Conventional neural network training algorithms (such as the Backpropagation [28]) require the specification of the neural network (NN) architecture before training begins. Given that, the definition of the NN architecture and its subsequent training are two independent processes. However, as pointed out in the literature (see [14], [30], [32]), a modification in the NN architecture will also require changes in the training algorithm. The conventional Backpropagation, for instance, has difficulties when training feedforward networks having many hidden layers; such is the case also when dealing with deep neural networks. Indeed, high computational costs are associated with the process of defining a NN architecture suitable to address a given task. Frequently, a great number of different NN architectures are tried prior to choosing a suitable configuration, for every considered domain, in a process known as model selection.

The approach followed by the so called constructive neural network (CoNN) algorithms is based on the establishment of a symbiotic relation between the two processes: constructing the NN architecture and training it. CoNN algorithms dynamically construct the NN architecture along with (and

as a consequence of) the training process avoiding, in this way, the need for a model selection phase. Taking into account that predefining a suitable NN architecture for solving a certain problem can be a hard task, CoNN algorithms can be a convenient choice when compared to conventional algorithms in that respect.

CoNN algorithms usually start with a NN having the input layer and no hidden layers; the construction of the network's hidden layer(s) occurs simultaneously with training. As pointed out in [26], constructive algorithms overcome the limitation of searching for a solution in the weight space of an a priori fixed network architecture, by extending the search, in a controlled fashion, to the entire space of NN's architectures ... Constructive algorithms search for small solutions first and, by doing that, have the potential for discovering a near-minimal network that suitably matches the complexity of the learning task.

An interesting type of NN called FLANN (*Functional Link Artificial Neural Networks*), was proposed by Pao in [24] (see also [25]) with the intent to overcome the complexities associated with multi-layer NNs, conventionally trained. The FLANN proposal reduces the architecture of a NN to a one-layer feedforward network, where the input layer is a functional-expansion of the original input data. As pointed out in [16], a functional-link NN with a functional-expansion model can be used as a tool for the approximation of nonlinear functions.

The work described in this paper investigates how functionally expanded input data could (or could not) contribute to constructive neural network learning processes, either by improving their performance or, then, producing more concise and smaller NNs. For this purpose, six well-known CoNN algorithms, namely Tower and Pyramid [12], [13], Tiling [20], PTI and Shift [1] and Perceptron Cascade (PC) [5] have been employed in learning experiments using functional expanded input (see Section 4).

The remainder of the paper is organized as follows. Section 2 presents a short introduction to the CoNN research area. Section 3 briefly describes each of the six CoNN algorithms used in the experiments, highlighting some of their distinctive characteristics. Section 4 describes the seven functional expansions employed namely Power Set polynomials, Trigonometric, Chebyshev polynomials of first class, Chebishev polynomials of second class, Fibonacci polynomials, Lucas polynomials and Boubaker polynomials [29]. Section 5 describes experimental setup and the learning experiments conducted using ten knowledge domains, two of them synthetic and eight from the UCI Repository [18], as well as the seven functional expansions, with the intent of evaluating a possible contribution of functionally expanded data input in a context of CoNN based learning. Section 6 summarizes the work done and presents some conclusions. This paper is an extended version of a conference paper [3].

2 Constructive Neural Networks (CoNNs)

While conventional neural network algorithms require the specification of the NN architecture before training begins, constructive neural network (CoNN) algorithms dynamically construct the NN architecture along with the training process [22]. CoNN algorithms dynamically combine two processes: (1) the construction of the NN architecture and (2) learning. Generally both processes alternate and are dependent on each others' performance. CoNN algorithms basically implement two tasks: (1) inserting a single neuron in the neural network (NN) being constructed which, generally, is a threshold logic unit (TLU), and (2) immediately training it.

As commented in [21], "...the possibility of adapting the network architecture to the given problem is one of the advantages of constructive techniques... [This] has also important effects on the convergence speed of the training process. In most constructive methods, the addition of a new hidden unit implies the updating of a small portion of weights, generally only those regarding the neuron to be added". Considering that the pre-definition of a suitable architecture for a certain problem can be a hard task, CoNN algorithms can be considered a very convenient choice when compared with conventional algorithms in that respect.

As pointed out in [22, 23], in spite of sharing the same basic mechanism, CoNN algorithms differ from each other in many different ways, such as:

- Number of nodes they add per layer at each iteration;
- Direction in which they grow the network: *forward*, from input towards output nodes or *backward*, from output towards input nodes;
- Functionality of the added neurons (do they all play the same role?);
- Stopping criteria (when to stop the NN construction process?);
- Connectivity pattern of the newly added neuron (how do they connect with the other neurons?);
- Algorithm used for training individual neuron, such as the Pocket algorithm and the Pocket with Ratchet Modification (PRM) [12, 13], MinOver [17], Quickprop [7], AdaTron [2], Thermal Perceptron [10], Loss minimization [15], Modified Thermal algorithm [5], Maxover [31], Barycentric Correction Procedure (BCP) [27];
- Type of input patterns they deal with: binary (or bipolar) valued, categorical or real valued attributes;
- Type of problems they solve: *classification (two-class or multi-class)*, where the input is assigned to one of two or a few classes; *regression*, characterized by a continuous mapping from inputs to an output or *clustering*, where the patterns are grouped according to some similarity measure;
- Topology of the connections among neurons (initially fixed or dynamically constructed);
- Shape of the feedforward architecture (*e.g.* tower-like, cascade-like, etc).

Taking into account that the basic step performed by a CoNN algorithm is the addition to the network architecture of a new neuron, usually represented as a TLU (*Threshold Logic Unit*) and its subsequent training, the choice of a suitable TLU training algorithm has become an important issue concerning CoNN algorithms. A great deal of CoNN algorithms use variants of the basic Perceptron algorithm for training the TLU nodes; among them, particularly the Pocket and the Pocket with the Ratchet Modification (PRM) are the most popular. For the experiments described in Section 5 of this paper, the PRM was chosen, due to its good results found in the literature (see for instance [13]).

There are also several CoNN algorithms suitable for two-class learning tasks, such as the Tower and Pyramid [13], Tiling [20], Upstart [11], Perceptron-Cascade [5], Offset [19], PTI and Shift [1], BabCoNN [25], etc. A description of a few well-known CoNN algorithms can be found in [10, 15, 26].

Nicoletti and co-authors in [23] present and discuss thirteen CoNN algorithms suitable for constructing feedforward architectures, aiming at classification tasks involving two classes; their work also briefly approaches the multiclass versions of several two-class algorithms, highlights some of the most popular constructive algorithms for regression problems and refers to several other alternative constructive neural network algorithms scattered among different publications. In [9] Franco and co-workers present a detailed review on CoNN algorithms as well as the descripption of several new CoNN proposals, highlihting and discussing the main achievements in the CoNN research area.

3 A Brief Description of Six CoNN Algorithms

Among the many CoNN algorithms available in the literature, the experiments with functionally expanded training sets focused on six of them: the Tower and the Pyramid [12, 13], the Tiling [20], the PTI and the Shift [1] and the Perceptron Cascade (PC) [5].

The choice of the Tower and the Pyramid algorithms was motivated by the particular way both carry on the constructive process – they only use one hidden neuron per hidden layer.

The Tower and the Pyramid only differ from each other in the way the newly added hidden neuron is connected. The Tower iteratively grows a NN by adding hidden layers where each new hidden layer added has only one TLU, which is connected

to all input neurons as well as to the only neuron that defines the previous hidden layer added to the NN. While in a Tower-NN each new hidden neuron has connections with every input neuron as well as with the last hidden neuron created, in a Pyramid-NN a new hidden neuron has connections with every input neuron as well as with all the previous hidden neurons created.

With the dynamic addition of neurons each algorithm tends to correctly classify a greater number of training instances. A single neuron can learn with 100% precision only from linearly separable training sets. If that is not the case, during the training phase the Tower algorithm continues to add neurons to the network (one per hidden layer) and to train them, in an iterative process controlled by a stopping condition. Generally one of three stopping conditions is used: (1) if the NN correctly classifies the training set; (2) if the addition of new hidden neurons does not improve the NN accuracy and (3) if a predefined maximum number of hidden neurons has been reached. The Tower and Pyramid algorithms have similar performances in classification tasks (see [4]).

The Tiling algorithm [20] was originally proposed for Boolean domains; the algorithm grows a multilayer feedforward NN where hidden nodes are added to a layer similarly to the process of laying tiles. The neurons in each hidden layer have one out of two functionalities and are named accordingly. Each layer has a *master* neuron, that functions as the output neuron for that layer. When the master neuron does not correctly classify all training patterns, however, the algorithm starts to add and train *ancillary* neurons, one at a time, aiming at obtaining a *faithful* representation of the training set. The output layer has only one master neuron. The faithfulness criterion employed by the Tiling algorithm establishes that no two training patterns, belonging to different classes, should produce the same outputs at any given layer. The Tiling constructs a NN as successive layer-by-layer assembly such that each new layer has a smaller number of neurons than the previous layer and the layer L only receives connections from the layer L - 1.

As commented by Gallant in [13], "The role of these units (ancillary) is to increase the number of cells for layer L so that no two training examples with different classifications have the same set of activations in layer L. Thus each succeeding layer has a different representation for the inputs, and no two training examples with different classifications have the same representation in any layer. Layers with this property are termed faithful layers, and faithfulness of layers is clearly a necessary condition for a strictly layered network to correctly classify all training examples... Assuming a finite training set with non-contradictory patterns, the Tiling algorithm is guaranteed to converge to zero classification errors (under certain assumptions)". The original Tiling algorithm, as described in [20], uses the same TLU training algorithm for each neuron (master or ancillary) added to the NN; in the same article the authors state and prove a theorem that ensures its convergence.

The Partial Target Inversion (PTI) algorithm [1] shares some similarities with the Tiling algorithm. The PTI also grows a multi-layer network where each layer has one master neuron and a few ancillary neurons. PTI also adds ancillary neurons to a layer in order to satisfy the faithfulness criteria; the neurons in layer L are connected only to neurons in layer L - 1. If the training of the master neuron results in a weight vector that correctly classifies all training patterns or if the master neuron of layer L does not classify a larger number of patterns than the master neuron of layer L - 1, the algorithm stops. If a training pattern, however, was incorrectly classified by the master neuron, and the master neuron correctly classifies a greater number of patterns than the master of the previous layer, the algorithm starts adding ancillary neurons to the current layer aiming at its faithfulness. When the current layer L becomes faithful the algorithm adds a new layer, L + 1, initially only having the master neuron. The process continues until stopping criteria are met, such as when the number of master (or ancillary) neurons has reached a pre-defined threshold. The only noticeable difference between the Tiling and the PTI is the way the training set, used for training the ancillary neurons in the process of turning a layer faithful, is chosen.

Some CoNN algorithms are based on discriminating between two types of errors (*wrongly-on* and *wrongly-off*) and, as a side-effect of this approach, they tend to grow the NN in a backward way, *i.e.* from the output layer towards the input layer. That is the case of both, the Shift [1] and the Perceptron

Cascade (PC) [5] algorithms which, also, have been chosen for the experiments.

An algorithm based on an error-correction strategy generally approaches the growing of the NN as follows: let u_n be a neuron that classifies training patterns but produces *wrongly-off* errors (*i.e.*, positive training patterns are misclassified by u_n as negative). Generally algorithms deal with *wrongly-off* errors by adding a *wrongly-off corrector* as a child neuron, u_{n+}, which will try to correct the errors made by its parent u_n. The main tasks of neuron u_{n+} are 1) to correct the classification of positive training patterns that have been misclassified by u_n as negative and 2) to keep unchanged the other classifications made by u_n (*i.e.* u_{n+} should be inactive for any other pattern). The neuron u_{n+} is trained with the subset of training patterns that were *wrongly-off* plus the set of negative patterns.

Similarly, to deal with *wrongly-on* errors (*i.e.* negative training patterns that are misclassified by u_n as positive) the algorithm creates a *wrongly-on corrector* as a child neuron u_{n-}, aiming at both: to correct the classification of the *wrongly-on* training patterns and to keep unchanged the other classifications made by u_n (*i.e.*, u_{n-} should stay inactive for any other pattern). The neuron u_{n-} is then trained with the subset of training patterns that were *wrongly-on* plus the set of positive patterns. For a child neuron (either a *wrongly-on* or a *wrongly-off* corrector) that has been added in order to correct misclassifications made by its parent, it is mandatory that this child only changes the activations of patterns that provoked the error; for this reason, the inactive output of the neuron should be 0.

The Perceptron Cascade (PC) algorithm [5] adopts the error-correction strategy just described; PC also adopts the architecture of NNs created by the Cascade Correlation algorithm [8]. The PC begins the construction of the network by training the output neuron. If this neuron does not classify all training patterns correctly, the algorithm begins to add hidden neurons to the network. Each new added hidden neuron is connected to all previous hidden neurons as well as to the input neurons. The new hidden neuron is then connected to the output neuron; each time a hidden neuron is added, the output neuron needs to be retrained.

The addition of a new hidden neuron enlarges the space in one dimension. The algorithm has three stopping criteria: 1) the network converges *i.e.*, correctly classifies all training patterns; 2) a pre-defined maximum number of hidden neurons has been achieved and 3) the most common, the addition of a new hidden neuron degrades the networks performance.

The PC adds hidden neurons to correct *wrongly-on* and *wrongly-off* errors. For the algorithm, what distinguishes a neuron created for correcting *wrongly-on* or for correcting *wrongly-off* errors is the training set used for training the neuron. To correct *wrongly-on* errors the training set used should have all negative patterns plus the patterns which produce *wrongly-on* errors. For correcting *wrongly-off* errors, the training set should have all positive patterns plus the negative patterns which produce *wrongly-of* patterns. The PC algorithm only adds and trains one neuron at a time to correct the most frequent error, between the *wrongly-on* and *wrongly-off* errors produced by the output neuron.

Similarly to the PC algorithm, the Shift algorithm [1] also constructs the network beginning with the output neuron. The algorithm, however, creates only one hidden layer, iteratively adding neurons to it; each added neuron is connected to the input neurons and to the output neurons. The error correcting procedure used by the Shift also identifies *wrongly-on* and *wrongly-off* errors. However, it adds and trains a hidden neuron to correct the most frequent between these two types of errors.

4 Functional Expansions Used

In learning experiments a functional expansion can be approached as a pre-processing task which 'horizontally' enlarges a given training set, by introducing functionally expanded attribute values. In this paper the functional explansion of the training data has been implemented using seven polynomial expansions, as described in this section. For each of the expansions it is shown up to its seventh term.

For the experiments conducted in this study, the following expansions were used: (1) Power Series (PW); (2) Trigonometric (TR); (3) Chebyshev polynomials (1^{st} class) (CH1); (4) Chebyshev polynomials (2^{nd} class) (CH2); (5) Fibonacci (FI); (6) Lucas (LU) and (7) Boubaker (BO). In what follows, each original data set, without being pre-processed

to be functionally expanded, is referred in the text as Original (OG). Figure 1 presents a diagram of a generic functional expansion process.

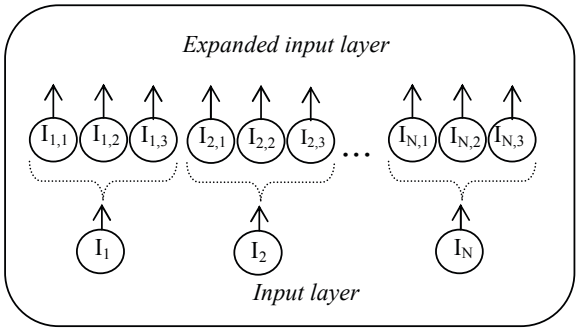

Figure 1. A view of a generic functional expansion of the input layer, considering an expansion of N = 3 values

Power Series(PW)
$F_1(x) = 1$
$F_2(x) = x$
$F_3(x) = x^2$
$F_4(x) = x^3$
$F_5(x) = x^4$
$F_6(x) = x^5$
$F_7(x) = x^6$

Trigonometric(TR)
$F_1(x) = x$
$F_2(x) = cos(\pi x)$
$F_3(x) = sin(\pi x)$
$F_4(x) = cos(2\pi x)$
$F_5(x) = sin(2\pi x)$
$F_6(x) = cos(3\pi x)$
$F_7(x) = sin(3\pi x)$

Chebyshev 1^{st} *class* $(CH1)$
$F_1(x) = x$
$F_2(x) = 2x^2 - 1$
$F_3(x) = 4x^3 - 3x$
$F_4(x) = 8x^4 - 8x^2 + 1$
$F_5(x) = 16x^5 - 30x^3 + 5x$
$F_6(x) = 32x^6 - 48x^4 + 18x^2 - 1$
$F_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x$

Chebyshev 2^{nd} *class* $(CH2)$
$F_1(x) = 2x$
$F_2(x) = 4x^2 - 1$
$F_3(x) = 8x^3 - 4x$
$F_4(x) = 16x^4 - 12x^2 + 1$
$F_5(x) = 32x^5 - 32x^3 + 6x$
$F_6(x) = 64x^6 - 80x^4 + 24x^2 - 1$
$F_7(x) = 128x^7 - 192x^5 + 80x^3 - 8x$

Fibonacci(FI)
$F_1(x) = 1$
$F_2(x) = x$
$F_3(x) = x^2 + 1$
$F_4(x) = x^3 + 2x$
$F_5(x) = x^4 + 3x^2 + 1$
$F_6(x) = x^5 + 4x^3 + 3x$
$F_7(x) = x^6 + 5x^4 + 6x^2 + 1$

Lucas(LU)
$F_1(x) = x$
$F_2(x) = x^2 + 2$
$F_3(x) = x^2 + 3x$
$F_4(x) = x^4 + 4x^2 + 2$
$F_5(x) = x^5 + 5x^3 + 5x$
$F_6(x) = x^6 + 6x^4 + 9x^2 + 2$
$F_7(x) = x^7 + 7x^5 + 14x^3 + 7x$

Boubaker(BO)
$F_1(x) = x$
$F_2(x) = x^2 + 2$
$F_3(x) = x^3 + x$
$F_4(x) = x^4 - 2$
$F_5(x) = x^5 - x^3 - 3x$
$F_6(x) = x^6 - 2x^4 - 3x^2 + 2$
$F_7(x) = x^7 - 3x^5 - 2x^3 + 5x$

5 Experiments and Results

For the experiments a total of 10 data sets have been chosen; 2 of them (named Artificial and Spirals) were artificially generated and are shown in Figure 2. The other 8 were downloaded from the UCI repository [18]. The chosen UCI data sets are typically data sets for supervised learning tasks and have been employed in a great number of machine learning related tasks reported in the literature. Table 1 presents a summary of their main characteristics. The chosen data sets have a variable number of data patterns and they all have their data patterns described by numerical attributes. The algorithms were implemented using Java and all the experiments were run under a Windows environment.

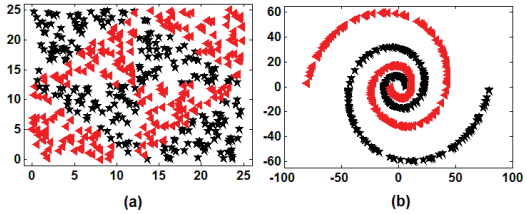

Figure 2. Scatter plots of the artificially generated data. (a) Artificial (b) Spirals. In both, stars (black) stand for class 1 and left-pointing triangles (red) for class 2

Table 1. Summary of data domains, where #NA: number of attributes; #C: number of classes and #NI: number of instances

Domain	#NA	#C	#NI
Artificial	2	2	500
Spiral	2	2	500
WPBC	34	2	198
Liver	6	2	345
Pima	8	2	768
Heart	13	2	270
Mammography	5	2	961
Blood	5	2	748
Ionosphere	32	2	351
Credit (German)	20	2	1,000

In the experiments each functional expansion was ran for a varying number of terms $\{N = 3,4,\ldots,7\}$ and the best result was selected. The results from the experiments have been organized in ten tables, Table 2 to 11, each of them associated to a particular data domain. The tables show the average classification results as well as the average number of neurons obtained through a repeated 30 time cross-validation process; both numbers are followed by the associated standard deviation values. Each result also shows, inside parentheses, the number of terms which yielded the best result. In the tables, the best result for each CoNN algorithm is bold faced.

Based on results it can be said that some improvement has been obtained in practically every single domain enlarged by a functional expansion. In spite of that, the results also show that some domains are more sensitive to the enlargement by a functional expansion than others. Moreover, a particular expansion may work better for certain domains than that for others. For example, the Spiral domain (Table 3) was the one with the highest enhancement in performance, when any functional expansion was used. On the other hand, the experiments with the WPBC (Table 4) have shown no significant difference whether the training data had been functionally expanded or not. Now, in the experiments concerning the Mammography and the Blood transfusion domains (Table 8 and 9), a particular functional expansion worked better than the others, namely the Trigonometric and the Fibonacci, respectively.

To aid further analyses, Figure 5 groups the results by domain *vs.* functional expansions, by CoNN learning algorithm *vs.* functional expansions and by CoNN *vs.* domain, respectively. In Figure 5 among the UCI domains, the Liver domain was the one that most benefited from the expansions, presenting over than 4%, on average, better performance than when considering the original data. Good results have also been observed in the domains Heart, Mammography, Blood, Ionosphere and Credit. As far as functional expansions are concerned, it can be said that the Trigonometric not only has yielded the highest variation among all the expansions considered, but also was the expansion which produced the highest accuracy values.

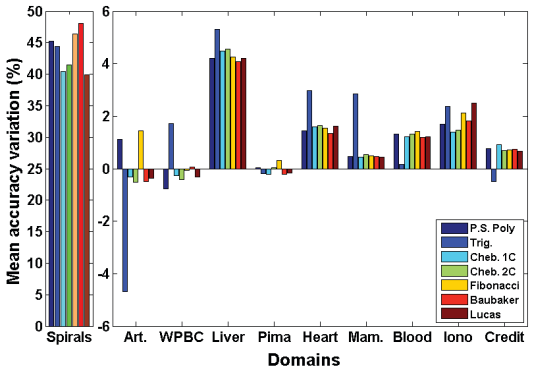

Figure 3. Mean accuracy variation between the results obtained when considering functional expansions and the results obtained with the original data. The results are grouped by domains and functional expansions

Figure 5 shows the impact of functional expansions on each of the CoNN algorithms, taking into account each data domain. It can be seen that the Tiling was one with the highest improvement; functional expansions usually help the algorithm to

Table 2. Artificial

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	**66.5**±**6.6** 11.8±4.0	63.8±6.9(3) 11.8±6.3	57.2±7.4(3) 5.6±3.2	63.7±7.2(3) 11.3±7.0	62.8±6.3(3) 11.6±5.8	64.6±7.4(3) 12.8±7.0	62.6±7.4(3) 11.0±5.7	64.2±8.0(3) 11.0±6.3
Pyramid	65.2±6.3 12,6±4,8	**65.3**±**7.1**(4) 16.6±8.7	61.9±9.8(5) 10.4±8.2	64.0±6.3(3) 16.6±7.7	63.6±8.3(3) 17.6±6.4	65.2±8.0(4) 15.9±7.6	63.6±7.0(3) 14.9±7.4	63.0±7.6(3) 16.2±7.1
Tiling	**65.2**±**6.9** 219.3±62.9	62.8±7.2(7) 216.7±82.1	57.2±7.5(5) 156.9±106.6	63.3±6.0(4) 208.4±83.5	63.3±5.6(4) 233.2±71.4	60.6±7.2(7) 232.4±82.4	62.7±8.0(7) 252.2±78.2	63.0±6.7(3) 230.3±55.7
PTI	53.8±7.6 1.0±0.0	54.7±5.6(4) 1.0±0.0	56.2±.94(7) 22.4±24.6	54.1±6.5(5) 1.7±4.3	53.0±8.2(7) 4.8±14.7	**55.0**±**6.0**(5) 2.2±7.9	53.2±9.0(5) 3.0±7.1	53.6±9.3(7) 9.0±19.6
PC	67.8±6.8 3.4±1,5	68.8±6.7(6) 6.2±3.1	62.6±8.8(6) 5.4±3.1	67.4±7.8(4) 5.3±2.6	68.6±8.6(4) 5.8±3.4	**70.8**±**6.1**(3) 6.0±3.8	68.2±7.9(3) 5.8±3.3	68.8±6.3(3) 6.4±3.6
Shift	65.8±6,6 7.2±3.0	73.1±6.8(3) 26.8±7.3	70.0±7.1(7) 31.6±8.2	70.5±7.9(3) 30.3±10.6	71.2±5.8(3) 32.8±10.8	**73.6**±**6.1**(3) 27.4±7.1	72.2±7.0(3) 27.5±5.7	70.4±5.0(4) 32.8±9.5

Table 3. Spirals

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	58.3±6.2 4.5±1.3	85.7±9.3(3) 7.2±1.8	**89.2**±**7.6**(4) 7.5±2.3	78.3±9.8(7) 5.7±3.1	77.8±11.0(5) 5.7±2.7	86.5±8.1(3) 6.8±2.1	86.2±10.6(3) 7.2±2.4	80.5±11.7(7) 5.8±2.6
Pyramid	60.0±7.3 5.2±2.7	88.7±13.1(3) 10.2±4.0	88.8±7.8(4) 9.0±2.8	79.0±13.1(3) 8.8±5.4	82.1±13.2(7) 7.1±2.9	90.2±7.7(3) 10.3±4.1	**92.6**±**9.0**(4) 10.9±2.8	79.1±10.7(7) 6.6±3.3
Tiling	73.6±20.6 24.5±32.2	95.2±12.6(3) 20.9±13.7	95.1±12.7(6) 57.8±34.2	94.7±9.3(4) 24.3±13.2	93.5±9.7(5) 27.0±17.3	96.2±8.2(6) 29.3±14.8	**97.2**±**7.5**(3) 24.2±16.6	96.8±7.1(7) 26.6±12.3
PTI	57.2±5.5 1.0±0.0	82.3±14.2(4) 11.3±11.0	72.4±14.2(5) 8.4±11.9	80.7±15.1(3) 9.7±10.6	82.8±15.3(6) 11.2±11.2	78.7±14.3(7) 8.1±10.2	**83.2**±**14.4**(6) 9.9±10.7	78.6±13.6(7) 9.5±10.6
PC	59.8±7.1 3.0±1.1	90.3±9.3(3) 7.8±3.0	**91.6**±**5.8**(6) 9.2±2.9	90.4±7.3(4) 8.3±2.9	89.3±8.5(4) 8.4±3.6	90.5±7.9(3) 7.0±3.2	90.0±9.7(4) 7.6±3.4	87.7±10.2(4) 9.1±3.9
Shift	58.0±5.4 3.6±1.6	88.5±8.5(3) 7.3±2.3	90.8±5.0(6) 8.9±2.7	90.2±7.5(5) 6.2±1.7	91.4±8.1(6) 6.6±2.0	**92.7**±**6.9**(3) 7.9±1.9	91.8±7.8(6) 7.1±1.9	89.3±9.5(6) 5.9±1.9

Table 4. WPBC

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	76.8±9.8 7.9±2.5	75.8±9.9(7) 8.9±2.4	**77.1**±**9.2**(3) 9.2±3.1	75.2±9.8(4) 8.6±2.8	75.7±9.3(4) 9.2±3.0	76.7±10.4(7) 8.0±2.4	75.7±8.7(4) 7.7±2.7	75.4±11.5(3) 7.8±2.8
Pyramid	76.2±10.5 8.0±2.4	74.4±11.2(3) 7.8±2.2	76.3±8.8(5) 2.5±0.5	75.0±8.8(6) 9.6±2.8	**76.7**±**8.7**(5) 8.5±2.8	76.4±10.9(3) 7.6±2.4	76.5±8.9(5) 8.7±2.7	74.8±11.2(6) 8.3±2.6
Tiling	75.6±8.5 14.0±8.9	76.7±9.2(3) 10.7±7.3	**78.1**±**9.0**(7) 5.3±6.1	76.8±10.1(5) 15.5±12.7	75.4±9.8(5) 16.6±9.1	76.9±8.0(4) 12.1±8.3	77.2±11.2(4) 13.9±8.0	76.6±8.4(6) 13.2±9.9
PTI	77.8±8.5 1.0±0.0	**78.4**±**7.8**(6) 1.4±2.5	78.3±9.3(3) 1.7±4.3	78.3±8.3(4) 2.8±5.5	77.2±9.3(5) 3.5±7.6	77.3±9.5(3) 2.6±5.1	78.3±10.4(4) 1.9±4.1	77.9±8.3(3) 4.0±6.4
PC	**77.6**±**8.9** 4.7±2.8	75.3±9.8(4) 6.0±2.4	80.4±9.2(3) 3.4±1.6	76.9±9.4(7) 5.2±2.4	76.4±10.4(5) 5.8±2.6	74.8±10.7(7) 6.0±2.5	75.3±8.1(5) 6.2±3.0	76.6±9.4(7) 4.9±2.3
Shift	76.7±7.0 5.7±2.8	76.3±9.5(5) 8.6±2.1	**78.4**±**6.8**(3) 4.6±1.7	77.3±9.0(3) 7.7±2.5	77.4±9.1(4) 6.1±2.8	78.2±9.6(3) 8.2±2.6	78.0±8.5(3) 8.4±2.7	77.9±9.0(3) 8.3±2.4

Table 5. Liver

CoNN	OG	PW	TR	CH	CH2	FI	BO	LU
Tower	69.2±7.2 9.9±2.7	71.0±8.3(3) 9.1±2.4	71.6±6.5(5) 8.1±4.3	71.0±7.9(5) 7.8±2.6	70.7±7.7(5) 8.6±2.8	**71.7±7.9**(3) 9.7±2.4	70.8±8.5(3) 8.6±2.5	71.2±6.8(4) 8.4±2.4
Pyramid	69.6±7.4 11.7±3.8	71.6±9.0(4) 9.6±3.3	71.4±8.5(3) 8.0±2.9	71.6±6.0(4) 10.0±3.4	**71.7±8.0**(3) 9.7±3.1	70.9±7.4(3) 10.4±2.7	71.1±8.9(3) 9.8±3.5	71.6±7.1(5) 9.8±3.7
Tiling	60.4±9.0 133.4±35.3	64.2±8.4(4) 131.8±26.8	**70.4±7.4**(4) 1.0±0.0	64.9±7.4(7) 127.8±37.8	65.0±8.5(7) 117.8±38.7	64.3±6.6(4) 125.3±27.8	64.0±8.1(7) 118.8±31.4	64.6±7.2(3) 129.6±29.7
PTI	68.6±7.3 1.0±0.0	72.6±8.2(5) 1.0±0.0	70.6±7.4(4) 1.0±0.0	**72.9±7.9**(5) 1.0±0.0	72.1±6.6(6) 1.0±0.0	72.5±7.6(4) 1.0±0.0	72.4±6.2(6) 1.0±0.0	72.2±6.7(5) 1.0±0.0
PC	68.1±6.9 3.0±1.4	71.4±7.0(4) 2.6±1.9	70.9±6.4(3) 3.6±2.7	71.9±6.9(4) 2.4±1.4	**72.6±8.4**(4) 2.5±1.5	71.5±6.0(4) 2.7±1.3	71.7±7.6(3) 2.2±1.5	71.3±8.1(3) 2.7±1.9
Shift	68.9±7.9 6.2±2.4	70.8±6.0(5) 7.4±3.6	70.4±6.5(6) 8.7±3.3	70.4±7.1(4) 8.6±4.2	70.8±7.3(5) 8.7±3.8	70.9±8.6(4) 7.3±3.8	**71.1±6.5**(6) 10.2±3.8	70.7±5.9(5) 9.3±4.5

Table 6. Pima

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	76.1±4.4 14.8±3.4	76.3±3.7(4) 12.0±4.3	**77.0±5.1**(5) 4.9±2.3	75.8±4.7(5) 11.0±4.0	76.2±4.2(3) 13.3±4.2	76.2±4.9(7) 9.4±3.0	76.3±4.4(6) 9.8±3.7	76.1±5.4(3) 12.6±4.1
Pyramid	76.1±5.0 18.4±4.5	76.4±5.2(3) 18.4±5.0	76.5±4.5 6.0±2.7	76.3±4.8(6) 14.7±6.6	**76.8±4.2**(4) 15.1±5.9	76.1±5.7(4) 17.6±5.3	76.0±5.2(4) 16.5±5.5	76.0±4.8(4) 17.6±7.6
Tiling	68.3±5.9 323.8±101.3	67.8±5.4(5) 321.9±75.0	66.1±6.5(3) 297.3±149.9	67.8±5.8(4) 313.7±99.6	67.9±6.2(5) 310.7±101.3	**69.9±7.4**(3) 303.7±109.8	67.9±6.2(7) 288.8±130.6	68.6±6.0(4) 300.8±121.5
PTI	76.8±4.8 1.0±0.0	76.5±4.6(3) 1.0±0.0	75.0±5.9(7) 1.0±0.0	**77.0±3.8**(5) 1.0±0.0	76.5±4.9(4) 1.0±0.0	76.0±4.5(3) 1.0±0.0	76.1±5.2(3) 1.0±0.0	76.1±5.0(3) 1.0±0.0
PC	76.3±4.7 2.4±1.2	76.1±5.3(4) 4.9±2.8	**76.7±4.7**(3) 2.5±1.6	75.8±5.1(5) 5.8±3.0	76.4±4.9(5) 4.7±2.5	76.3±4.3(7) 4.8±3.1	76.2±4.3(3) 3.4±2.0	75.7±5.0(6) 5.0±2.5
Shift	75.5±4.3 8.6±2.7	76.2±4.9(4) 22.5±9.5	**77.2±4.4**(5) 12.2±4.6	75.5±4.7(3) 19.6±11.5	75.5±4.2(5) 26.6±12.4	75.8±4.2(3) 14.8±8.0	75.7±5.0(4) 22.1±11.8	75.8±4.2(5) 24.1±12.7

Table 7. Heart

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	81.0±7.6 11.8±4.7	82.3±6.7(3) 11.0±4.5	82.9±8.2(5) 4.4±2.1	82.3±6.7(6) 9.6±4.1	81.0±8.2(4) 10.6±3.8	82.2±8.4(3) 11.8±4.6	81.4±6.9(6) 9.9±3.8	**83.2±8.8**(4) 10.3±4.4
Pyramid	81.7±7.4 9.4±2.6	82.4±6.4(3) 10.1±3.2	83.7±8.0(4) 4.0±1.5	81.8±7.4(3) 10.6±2.9	83.2±7.2(4) 10.2±3.4	81.7±7.6(5) 10.8±3.1	82.0±7.6(3) 11.0±3.7	82.0±7.0(5) 10.0±3.5
Tiling	80.6±9.1 30.8±13.9	79.4±6.0(6) 93.5±34.6	**81.9±6.5**(3) 38.1±24.5	79.4±7.1(6) 95.4±33.2	79.5±8.5(3) 79.8±40.6	80.1±7.5(3) 93.4±37.3	79.7±6.4(3) 74.5±38.9	79.4±7.0(5) 83.9±41.8
PTI	81.1±7.6 2.1±5.0	82.7±6.6(7) 1.0±0.0	83.3±7.2(6) 7.9±11.6	**83.4±6.0**(4) 1.0±0.0	83.8±6.4(4) 1.0±0.0	82.8±6.8(3) 1.0±0.0	83.1±6.5(5) 1.0±0.0	83.1±7.5(5) 1.0±0.0
PC	79.9±7.9 4.6±2.9	82.5±6.2(6) 1.8±0.8	**84.2±6.3**(3) 2.2±1.2	83.0±6.5(4) 2.3±1.1	81.9±8.8(6) 2.0±0.9	82.6±7.1(6) 2.4±1.1	82.7±7.4(4) 2.1±1.0	82.4±5.7(3) 2.4±1.3
Shift	80.7±7.0 13.0±4.8	82.7±7.9(3) 6.8±2.5	83.4±7.2(6) 9.7±3.9	82.8±8.3(4) 7.3±2.7	**83.6±8.1**(3) 7.1±2.2	83.1±8.1(5) 7.0±2.7	82.6±7.6(6) 7.2±2.4	82.7±7.4(7) 6.7±2.5

Table 8. Mammography

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	81.6±3.6 7.2±2.3	81.6±4.2(3) 7.3±2.5	**83.7±3.8**(7) 5.8±1.8	81.4±3.7(3) 6.8±2.1	81.7±3.6(3) 6.8±2.3	81.5±3.7(3) 7.3±2.4	81.5±4.9(3) 6.8±2.4	81.7±4.0(7) 4.5±1.8
Pyramid	81.6±4.3 6.3±1.8	81.6±4.7(5) 5.4±2.0	**83.7±3.5**(5) 7.0±2.7	81.4±2.8(5) 5.7±2.4	81.6±3.2(3) 6.0±2.2	81.6±4.5(6) 5.8±2.2	81.5±3.2(7) 4.2±1.9	81.7±4.0(3) 6.0±2.3
Tiling	81.2±3.7 1.0±0.0	81.5±3.9(7) 1.0±0.0	**84.1±3.0**(7) 1.0±0.0	81.5±3.7(5) 1.0±0.0	81.7±3.4(4) 1.0±0.0	81.9±3.5(3) 1.0±0.0	81.5±4.6(3) 1.0±0.0	81.4±4.1(3) 1.0±0.0
PTI	81.0±4.1 1.0±0.0	81.9±3.5(3) 1.0±0.0	**83.6±4.3**(5) 1.0±0.0	81.8±4.6(3) 1.0±0.0	81.7±4.3(4) 1.0±0.0	81.6±4.3(3) 1.0±0.0	81.8±5.1(4) 1.0±0.0	81.4±3.1(7) 1.0±0.0
PC	81.1±3.3 2.3±1.2	81.6±4.1(4) 2.2±1.1	**83.3±3.4**(6) 2.4±1.3	81.7±3.7(4) 2.2±1.0	81.6±4.1(5) 2.3±0.9	81.6±3.8(3) 2.5±1.2	81.5±3.1(4) 2.2±1.3	81.9±4.0(3) 2.5±1.2
Shift	81.3±4.2 3.9±1.3	81.9±4.2(3) 6.7±2.9	**83.3±4.1**(5) 4.9±1.6	82.1±2.9(7) 4.8±1.3	82.1±3.1(6) 4.0±1.3	82.0±3.6(4) 5.0±1.8	82.2±3.8(6) 5.0±1.9	81.9±4.2(4) 4.5±1.4

Table 9. Blood

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	78.0±4.3 4.1±1.0	78.9±5.1(5) 4.9±1.6	77.9±5.0(4) 8.6±2.9	78.7±4.4(4) 4.3±1.5	78.7±4.3(3) 4.9±1.4	**79.0±3.6**(5) 5.2±1.8	78.8±3.5(4) 4.9±1.3	78.8±4.9(5) 4.8±1.4
Pyramid	77.5±4.6 4.6±1.2	78.6±5.0(7) 4.8±1.7	78.0±5.1(6) 7.8±2.2	78.7±3.8(7) 4.5±1.7	78.7±5.1(7) 4.7±1.6	**78.9±4.0**(6) 4.7±1.4	78.6±4.5(4) 4.6±1.3	78.7±4.2(3) 4.9±1.6
Tiling	78.1±4.2 1.0±0.0	78.6±4.6(4) 1.0±0.0	77.9±4.3(6) 1.0±0.0	78.6±4.1(5) 1.0±0.0	78.8±5.4(6) 1.0±0.0	**79.0±4.5**(6) 1.0±0.0	78.6±5.1(3) 1.0±0.0	78.5±4.2(6) 1.0±0.0
PTI	77.7±4.3 1.0±0.0	78.7±4.4(6) 1.0±0.0	77.8±4.8(4) 1.0±0.0	78.7±3.8(7) 1.0±0.0	78.5±4.4(5) 1.0±0.0	**78.8±4.9**(4) 1.0±0.0	78.4±3.8(7) 1.0±0.0	78.6±3.8(6) 1.0±0.0
PC	77.2±4.1 1.9±0.7	78.7±5.1(7) 1.8±1.0	77.9±4.8(7) 2.3±1.4	78.4±4.4(5) 2.0±0.7	78.7±4.3(3) 1.8±0.9	78.4±4.5(3) 1.7±0.7	**78.9±5.3**(6) 2.2±1.0	78.6±4.4(4) 2.0±0.8
Shift	77.5±4.4 1.6±0.7	78.6±3.9(5) 2.6±1.2	77.3±4.8(6) 6.7±2.2	78.6±6.1(3) 2.4±1.0	**78.7±4.7**(4) 2.2±0.9	78.5±5.2(3) 2.3±1.0	78.3±4.8(5) 2.5±1.1	78.5±4.6(6) 2.9±1.3

Table 10. Ionosphere

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	88.4±5.3 5.8±1.0	91.2±3.9(5) 3.1±0.7	91.4±5.2(6) 1.1±0.3	90.1±5.1(6) 2.0±0.6	90.0±5.2(3) 3.2±0.8	90.6±5.1(7) 3.0±0.5	90.9±4.9(5) 3.2±0.6	**92.2±4.4**(5) 2.8±0.7
Pyramid	88.2±5.3 5.9±1.1	91.2±4.5(4) 3.3±0.6	**91.7±4.2**(6) 1.0±0.2	91.0±4.7(6) 2.0±0.6	90.5±4.9(3) 3.2±0.9	91.5±3.2(4) 3.3±0.6	91.0±4.0(3) 3.2±0.6	**91.7±4.1**(7) 3.0±0.7
Tiling	86.5±4.6 3.7±12.4	91.2±4.5(4) 11.4±8.1	**91.7±4.2**(6) 1.0±0.3	90.4±5.6(7) 2.0±1.0	91.1±5.1(3) 8.0±6.0	91.2±4.8(7) 10.6±5.6	90.9±4.4(5) 8.7±4.3	91.5±3.9(5) 11.4±6.1
PTI	86.4±5.1 1.0±0.0	86.4±5.8(3) 1.0±0.0	87.1±6.0(3) 1.0±0.0	86.7±5.4(3) 1.0±0.0	87.1±5.0(3) 1.5±3.3	**87.5±4.9**(3) 1.0±0.0	86.8±5.5(3) 1.0±0.0	86.6±5.4(3) 1.0±0.0
PC	90.7±5.3 5.0±1.5	89.3±5.3(3) 4.4±1.2	91.6±4.8(3) 4.8±1.4	90.2±3.7(3) 4.6±1.2	90.7±5.1(3) 4.6±1.5	**91.5±4.8**(3) 4.5±1.2	90.4±4.2(3) 4.4±0.9	91.2±5.0(3) 4.7±1.5
Shift	90.0±5.1 4.1±1.5	89.7±4.3(3) 3.6±1.1	89.2±4.8(3) 3.7±1.3	89.0±5.2(3) 3.7±1.1	88.4±4.4(3) 3.8±1.3	89.0±4.9(3) 3.7±1.1	89.7±5.1(3) 3.8±1.2	**90.1±5.3**(3) 3.8±1.4

Table 11. Credit (German)

CoNN	OG	PW	TR	CH1	CH2	FI	BO	LU
Tower	75.2±3.5	**75.8±4.2**(6)	74.0±3.5(6)	75.3±3.9(3)	75.5±5.4(3)	75.6±4.0(6)	75.6±4.3(4)	75.6±4.9(7)
	3.6±1.1	2.0±1.0	2.8±1.2	2.5±1.1	2.6±1.1	2.4±1.1	2.3±0.9	2.1±0.9
Pyramid	74.7±4.0	75.4±3.7(6)	74.7±4.2(7)	75.6±3.4(4)	75.6±4.4(3)	75.2±4.0(6)	75.4±3.6(6)	**75.8±4.2**(6)
	4.6±1.9	2.8±1.3	3.0±1.4	3.0±1.8	2.9±1.4	3.1±1.7	2.4±1.4	3.2±1.6
Tiling	73.0±4.0	**73.9±3.5**(3)	73.0±5.0(7)	74.5±4.7(7)	73.7±4.7(3)	73.6±4.5(7)	73.7±4.3(7)	73.8±4.4(5)
	124.2±109.0	92.3±98.1	65.0±81.3	60.8±80.4	85.4±82.3	69.1±72.1	69.6±75.4	64.0±77.5
PTI	74.6±4.5	74.8±3.1(5)	74.2±3.1(5)	75.0±3.4(3)	75.2±4.0(6)	**75.6±4.9**(4)	75.5±4.4(4)	74.7±4.2(6)
	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0	1.0±0.0
PC	74.3±5.1	75.3±4.5(5)	74.2±4.1(7)	**75.8±4.4**(3)	75.6±3.9(3)	75.3±4.5(3)	75.3±3.2(4)	75.2±4.8(3)
	2.4±1.7	2.1±1.4	2.4±1.2	2.3±1.6	2.0±1.0	2.6±1.5	2.1±1.2	2.1±1.3
Shift	**74.6±3.7**	74.6±4.1(3)	74.1±3.9(3)	74.3±4.1(3)	73.9±3.9(3)	74.3±4.9(3)	74.2±4.6(3)	74.3±3.9(3)
	24.4±8.2	23.6±7.3	25.0±7.4	24.9±7.3	23.5±8.2	23.0±8.6	24.2±5.9	22.8±7.6

satisfy the faithfulness criterion and this may led to more robust NN. However, sometimes the same bias may lead to overfitting, by growing oversized NNs with poor generalization performance, as happened in the Artificial, Pima and Credit domains. The PTI algorithm also depends on a faithfulness criterion to build the NN; however its faithfulness criteria is frequently harder to be fulfilled than the one required by the Tiling. It is important to point out that there is no single CoNN learning algorithm that has the best results in all domains; but there is always one algorithm that has higher accuracy using functionally expanded data than when using the original data.

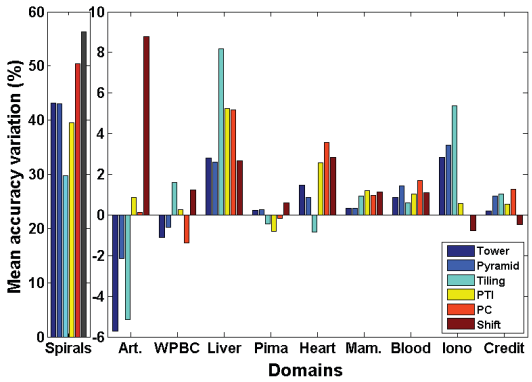

Figure 4. Mean accuracy variation between the results obtained when considering functional expansions and the results obtained with the original data. The results are grouped by CoNN learning algorithms and functional expansions

Finally in Figure 5 the results for the real data domains are grouped by CoNN and function expansion, aiming to uncover which CoNN algorithms presented the best results and which are the best functional expansion for each of them. Notice that, Tiling, again, was the most favored algorithm by the use of a functional expansion, again due to the satisfiability of the faithfulness criterion. The Trigonometric expansion has given good results, especially for Tiling and PC; however, compared to the other expansions, it did not seem a good option for the PTI. The Fibonacci and Boubaker functional expansions had the most consistent results. When using either of them, the results on average accuracy increase in 1%, when compared with results obtained with the original data.

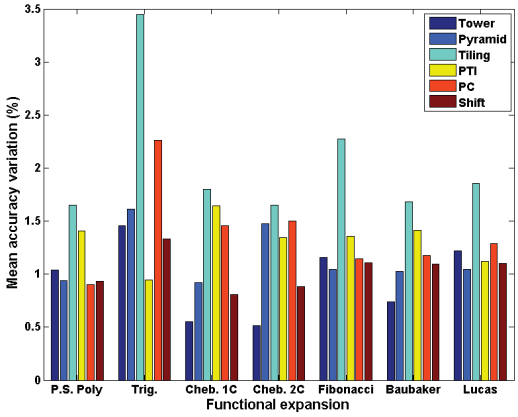

Figure 5. Mean accuracy variation between the results obtained when considering functional expansions and the results obtained with the original data. The results are grouped by CoNN learning algorithms and functional expansions

The results presented in this paper were obtained through a consistent and standard process aiming to establish a fair comparison among different functional expansions. It is important to mention that the adopted conditions may not be the best

for all of the presented situations. For example, if more iterations of the PRM algorithm had been used, not all the results would be enhanced due to overfitting in some cases or, if a different stopping condition was employed for the CoNN learning algorithms, it would also affect differently each of them. Therefore, in practical situations, all these details need to be taken into account when selecting a definitive NN model. Nonetheless, this work corroborates and completes the work presented in [3] by providing more empirical evidence for stating that the use of functionally expanded input data, when learning with CoNN, enhances the quality of the induced NN.

6 Conclusion

This paper aimed to verify whether, and up to what extent, CoNN accuracy performance can be enhanced by functionally extending the input data, prior to training and classification. For the experiments, seven different functions were chosen to implement the functional expansions and a comparison among them was conducted. The experiments also considered six CoNN learning algorithms and ten data domains. The obtained results confirm the previous work conclusions, showing that the performance of CoNN algorithms can be enhanced through functional expansion. However, the extent of the possible enhancement is domain dependent and some particular function may work better than others on different data domains.

References

[1] E. Amaldi and B. Guenin, Two constructive methods for designing compact feedfoward networks of threshold units, International Journal of Neural System, vol. 8, no. 5-6, 1997, pp. 629-645.

[2] J. K. Anlauf, M. Bieh, The AdaTron: an adaptive perceptron algorithm, Europhysics Letters, vol. 10,1989, pp. 687-692.

[3] J. R. Bertini Jr. and M. C. Nicoletti, Refining constructive neural networks using functionally expanded input data, Proc. Int. Joint Conference on Neural Networks, 2015, pp. 1-8.

[4] J. R. Bertini Jr. and M. C. Nicoletti, A constructive neural network algorithm based on the geometric concept of barycenter of convex hull, In: Computational Intelligence: Methods and Applications, IEEE Comp. Intelligence Society, Poland, 2008, pp. 1-12.

[5] N. Burgess, A constructive algorithm that converges for real-valued input patterns, International Journal of Neural Systems, vol. 5, no. 1, 1994, pp. 59-66.

[6] S. Fahlman and C. Lebiere, The cascade correlation architecture, in Advances in Neural Information Processing Systems, vol. 2, 1990, pp. 524-532.

[7] S. E. Fahlman, Faster-learning variations on backpropagation: an empirical study, In: Proc. of the 1988 Connectionist Models Summer School, D. S. Touretzky, G. E. Hinton and T. J. Sejnowski (Eds.), Morgan Kaufmann, San Mateo, CA, 1988, pp. 38-51.

[8] S. Fahlman and C. Lebiere, The cascade correlation architecture, in Advances in Neural Information Processing Systems, vol. 2, 1990, pp. 524-532.

[9] L. Franco, D. A. Elizondo and J. M. Jerez, Constructive Neural Networks, Studies in Comp. Intelligence Series, v. 258, Springer, 2010.

[10] M. Frean, A thermal perceptron learning rule, Neural Computation, vol. 4, 1992, pp. 946-957.

[11] M. Frean, The upstart algorithm: a method for constructing and training feedforward neural networks, Neural Computation, vol. 2, pp. 198-209, 1990.

[12] S. I. Gallant, Perceptron-based learning algorithms, IEEE Transactions on Neural Networks, vol. 1, no. 2, 1990, pp. 179-191.

[13] S. I. Gallant, Neural Network Learning and Expert Systems, The MIT Press, London, England, 1994.

[14] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, In: Proc. AISTATS, 2010, pp. 249-256.

[15] T. Hrycej, Modular Learning in Neural Networks, A. Wiley, N. York, 1992.

[16] Y-C. Hu, Functional-link nets with genetic-algorithm-based learning for robust nonlinear interval regression analysis, Neurocomputing, vol. 72, 2009, pp. 1808-1816.

[17] W. Krauth and M. Mézard, Learning algorithms with optimal stability in neural networks, Journal of Physics A, vol. 20, pp. 745-752, 1987.

[18] M. Lichman, UCI Machine Learning Repository. [Online]. Available: http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science, 2013.

[19] D. Martinez and D. Estève, The offset algorithm: building and learning method for multilayer neural networks, Europhysics Letters, vol. 18, no. 2, 1992, pp. 95-100.

[20] M. Mézard and J.-P. Nadal, Learning in feedfoward networks: the tiling algorithm, Journal of Physics A: Mathematical and General, vol. 22, 1989, pp. 2191-2203.

[21] M. Muselli, Sequential constructive techniques, Neural Network Systems Techniques and Applications, C. Leondes (Ed.), San Diego, CA: Academic, vol. 2, 1998, pp. 81-144.

[22] M. C. Nicoletti and J. R. Bertini Jr., An empirical evaluation of constructive neural network algorithms in classification tasks, International Journal of Innovative Computing and Applications, vol. 1, 2007, pp. 2-13.

[23] M. C. Nicoletti, J. R. Bertini Jr., D. Elizondo, L. Franco and J. M. Jerez, Constructive neural network algorithms for feedforward architectures suitable for classification tasks, In: Constructive Neural Networks, Studies in Comp. Intelligence, D. Elizondo, L. Franco and J.M. Jerez, Springer, 2010, pp. 1-23.

[24] Y. H. Pao, Adaptive pattern recognition and neural networks, Addison-Wesley, Reading, MA, 1989.

[25] Y. H. Pao and Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer, vol. 25, no. 5, 1992, pp. 76-79.

[26] R. G. Parekh, J. Yang and V. Honavar, Constructive neural-network learning algorithms for pattern classification, IEEE Transactions on Neural Networks, vol. 11, no. 2, 2000, pp. 436-451.

[27] H. Poulard, Barycentric correction procedure - fast method of learning threshold unit, In: Proc. of WCNN 95, vol. 1, 1995, pp. 710-713.

[28] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, Nature, vol. 323, no. 6088, 1986, pp. 533-536.

[29] M. R. Spiegel, Mathematical Handbook of Formulas and Tables, Schaum's outline series, McGraw-Hill Inc., USA, 1968.

[30] L. Toth and T. Grosz, A comparison of deep neural network training methods for large vocabulary speech recognition, In: TSD 2013, I. Habernal and V. Matousek (Eds.), LNAI 8082, Springer, 2013, pp. 36-43.

[31] A. Wendmuth, Learning the unlearnable, Journal of Physics A: Mathematical and General, vol. 28, 1995, pp. 5423-5436.

[32] J. Yosinski, J. Clune, Y. Bengio and H. Lipson, How transferable are features in deep neural networks?, Advances in Neural Information Processing Systems 27, NIPS Foundation, 2014, pp. 3320-3328

Joao R. Bertini Jr. received his B.S. and M.Sc. degrees both in Computer Science from the Federal University of Sao Carlos, Brazil, in 2004 and 2006, respectively. In 2011 he received his Ph.D degree in Computer Science and Computational Mathematics from University of Sao Paulo, Brazil. Currently he is Post-doctorate associated to the Federal University of Sao Carlos. He has experience in Computer Science, acting on the following topics: machine learning, supervised and semi-supervised learning, neural networks and learning from data stream.

Maria do Carmo Nicoletti holds a M.Sc. by research in Computer Science from Oxford University, England (1980) and a Ph.D in Computer Science from IFSC-USP-S. Paulo, Brazil (1994). She did her post-doctoral studies at the University of New South Wales, Australia, during 1999 and 2000. Currently she supervises CS research work at FACCAMP (C. L. Paulista), Brazil. Her main areas of interest are machine learning, knowledge representation and uncertainty.