Chapter 3

Generalized Regression Neural Networks with Application in Neutron Spectrometry

Ma. del Rosario Martinez-Blanco,
Víctor Hugo Castañeda-Miranda,
Gerardo Ornelas-Vargas,
Héctor Alonso Guerrero-Osuna,
Luis Octavio Solis-Sanchez,
Rodrigo Castañeda-Miranda,
José María Celaya-Padilla, Carlos Eric Galvan-Tejada,
Jorge Isaac Galvan-Tejada,
Héctor René Vega-Carrillo,
Margarita Martínez-Fierro, Idalia Garza-Veloz and
Jose Manuel Ortiz-Rodriguez

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/64047

Abstract

The aim of this research was to apply a generalized regression neural network (GRNN) to predict neutron spectrum using the rates count coming from a Bonner spheres system as the only piece of information. In the training and testing stages, a data set of 251 different types of neutron spectra, taken from the International Atomic Energy Agency compilation, were used. Fifty-one predicted spectra were analyzed at testing stage. Training and testing of GRNN were carried out in the MATLAB environment by means of a scientific and technological tool designed based on GRNN technology, which is capable of solving the neutron spectrometry problem with high performance and generalization capability. This computational tool automates the pre-processing of information, the training and testing stages, the statistical analysis, and the post-processing of the information. In this work, the performance of feed-forward backpropagation neural networks (FFBPNN) and GRNN was compared in the solution of the neutron spectrometry problem. From the results obtained, it can be observed that despite very similar results, GRNN performs better than FFBPNN because the former could be used as an alternative procedure in neutron spectrum unfolding methodologies with high performance and accuracy.

Keywords: artificial intelligence, statistical artificial neural networks, neutron spectrometry, unfolding codes, spectra unfolding

1. Introduction

Artificial Intelligence or AI is one of the newest fields of intellectual research that attempts to understand the intelligent entities [1]. Intelligence could be defined by the properties it exhibits: an ability to deal with new situations, to solve problems, to answer questions, to devise plans, and so on [2]. The phrase AI was coined by John McCarthy in the 1940s and to date evades a concise and formal definition [3]. A simple definition might be: AI is the study of systems that act in a way, that to any observer would appear to be intelligent, and involves using methods based on the intelligent behavior of humans and other animals to solve complex problems.

AI has been classified into three periods: the classical, the romantic, and the modern periods [1–4]. The major area of research covered under the classical period, in the 1950s, was intelligent search problems involved in game-playing and theorem proving. In the romantic period, from the mid-1960s until the mid-1970s, people were interested in making machines "understand," by which they usually meant the understanding of natural languages. The modern period started from the latter half of 1970s to the present day and includes research on both, theories and practical aspects of AI. This period is devoted to solving relatively simple or complex problems that are integral to more complex systems of practical interest.

The aim of the study of AI is to use algorithms, heuristics, and methodologies based on the ways in which the human brain solves problems. In the most recent decades, AI areas of particular importance include multi-agent systems; artificial life; computer vision; planning; playing games, chess in particular; and machine learning [5–6].

1.1. Machine learning and connectionism

Learning and intelligence are intimately related to each other. Learning is an inherent characteristic of human beings [3]. By virtue of this, people, while executing similar tasks, acquire the ability to improve their performance with the self-improvement of future behavior based on past experience. In most learning problems, the task is to learn to classify inputs according to a finite, or sometimes infinite, set of classifications [2]. Typically, a learning system is provided with a set of training data, which have been classified by hand. The system then attempts to learn from these training data how to classify the same data, usually a relative easy task, and also how to classify new data that are not seen [7].

The principles of learning can be applied to machines to improve their performance [8]. A system capable of learning is intelligent and is usually expected to be able to learn based on past experience. Such learning is usually referred to as "machine learning" (ML) which is an important part of AI and can be broadly classified into three categories: supervised, unsupervised, and reinforcement learning.

Supervised learning requires a trainer who supplies the input-output training instances. The learning system adapts its parameters using some algorithms to generate the desired output patterns from a given input pattern. In absence of trainers, the desired output of a given input instance is not known; consequently, the learner has to adapt its parameters autonomously. Such type of learning is termed unsupervised learning.

Reinforcement learning bridges the gap between the supervised and unsupervised categories. In reinforcement learning, the learner does not explicitly know the input-output instances, but it receives some form of feedback from its environment. The feedback signals help the learner to decide whether its action on the environment is rewarding or punishable. The learner thus adapts its parameters based on the states (rewarding/punishable) of its actions.

Recently, the connectionist approach for building intelligent machines with structured models like artificial neural networks (ANN) is receiving more attention [9]. Connectionist models are based on how computation occurs in biological neural networks. Connections play an essential role in connectionist models, hence the name *connectionism* [10]. The term connectionism was introduced by Donald Hebb in the 1940s, and it is a set of approaches in the fields of AI that models mental or behavioral phenomena as the emergent processes of interconnected networks of simple units [11]. The central connectionist principle is that mental phenomena can be described by interconnected networks of simple and uniform units.

Figure 1. The unit: the basic information processing structure of a connectionist model.

Units are to a connectionist model what neurons are to a biological neural network: the basic information processing structures. Since the flow of information in a network occurs through its connections, the link through which information flows from one member of the network to the next is known as synapses. Synapses are to neural networks what an Ethernet cable or telephone wire is to a computer network. Without synapses from other neurons, it would be impossible for a neuron to receive input and to send output from and to other neurons, respectively. Given the crucial role that connections play in a network of neurons, synapses in a biological neural network matter as much as the neurons themselves [12].

Most connectionist models are computer simulations executed on digital computers. In a connectionist computer model, units are usually represented by circles as shown in **Figure 1**. Because no unit by itself constitutes a network, connectionist models typically are composed of many units as illustrated in **Figure 2**. However, neural networks are organized in layers of neurons. For this reason, connectionist models are organized in layers of units as shown in **Figure 3**. **Figure 3** is still not a network because no group of objects qualifies as a network unless each member is connected to other members; it is the existence of connections that make a network, as illustrated in **Figure 4** [13].

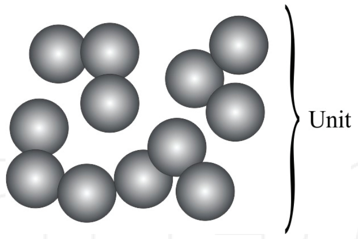

Figure 2. Connectionist model with 11 units.

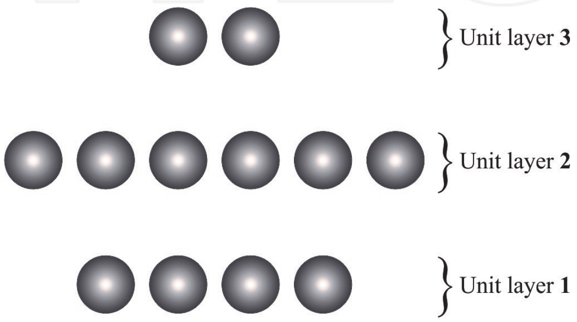

Figure 3. Connectionist model organized in layers.

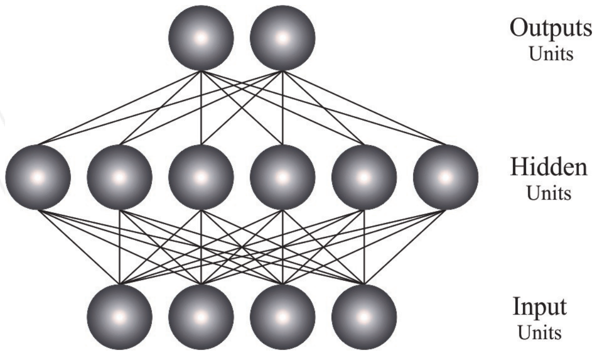

Figure 4. Network connectionist model.

In **Figure 4**, it can be seen that network connections are conduits through which information flows between the members of a network. In the absence of such connections, no group of objects qualifies as a network. There are two kinds of network connections: input and output. An input connection is a conduit through which a member of a network receives information. An output connection is a conduit through which a member of a network sends information. Although it is possible for a network connection to be both an input connection and an output connection, a unit does not qualify as a member of a network if it can neither receive information from other units nor send information to other units.

There are many forms of connectionism, but the most common forms use neural network models [14]. The form of the connections and the units can vary from model to model as shown in **Figures 5–9**, where it can be seen that any number of units may exist within each layer, and each unit of each layer is typically linked via a weighted connection to each node of the next layer. Data are supplied to the network through the input layer.

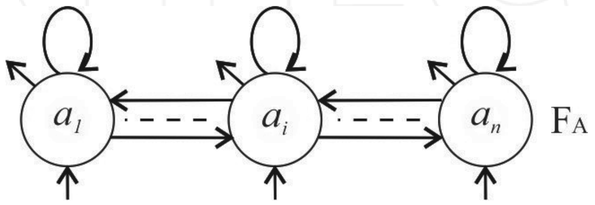

Figure 5. Single-layered recurrent net with lateral feedback structure.

Depending on the nature of the problems, neural network models are organized in different structural arrangements (architectures or topologies) [10]. The neural network architecture defines its structure including the number of hidden layers, number of hidden nodes, and number of nodes at the input and output layers. There are several types of ANN architectures. As illustrated in **Figures 5–9**, most of the widely used neural network models can be divided into two main categories: feed forward neural networks (FFNN) and feedback neural networks (FBNN) [10–14].

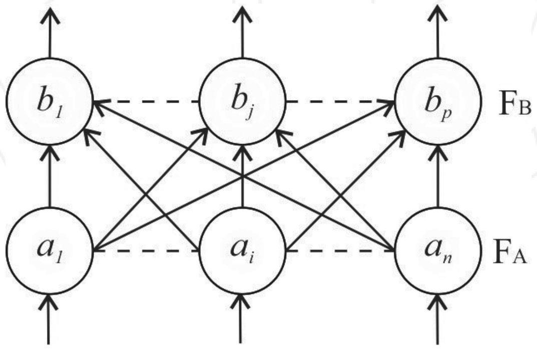

Figure 6. Two-layered feed-forward structure.

As shown in **Figures 6** and **8**, FFNNs allow signals to travel one way only; data enters the inputs and passes through the network, layer by layer, until it arrives at the output. There is no feedback or loops between layers. These networks are extensively used in pattern recognition and classification. FBNN can have signals traveling in both directions by introducing loops in the network as shown in **Figures 5**, **7**, and **9**. FBNNs are dynamic; their state changes continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.

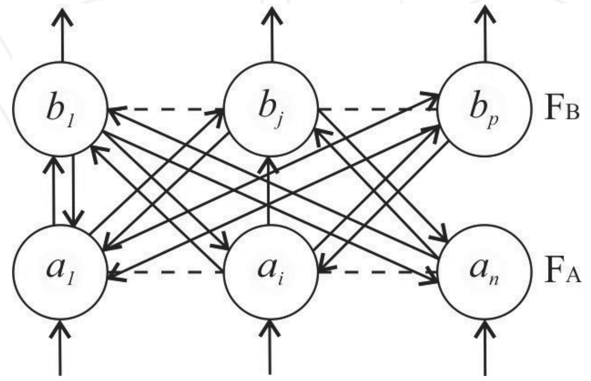

Figure 7. Two-layered feedback structure.

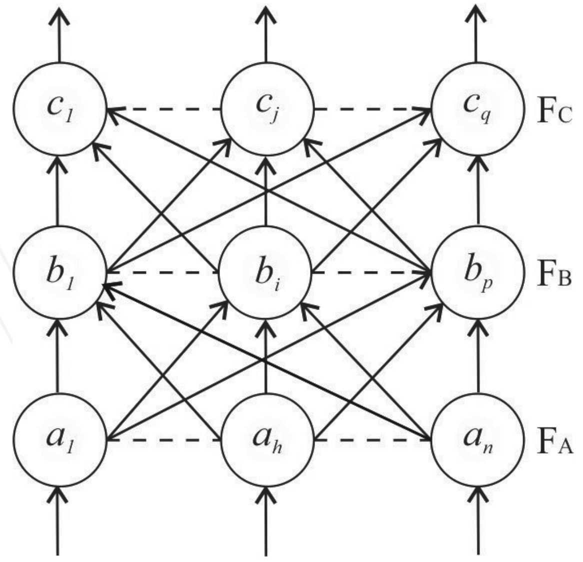

Figure 8. Three-layered feed-forward structure.

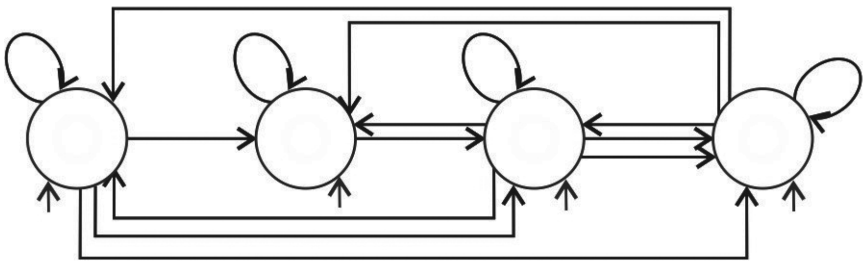

Figure 9. Single-layered recurrent structure.

In most connectionist models, units are organized into three layers: an input layer, one or more "hidden" layers, and an output layer [10–14]. **Figures 4** and **8** show a 3-layered FFNN consisting of 3 layers of units, where each unit is connected to each unit above it, and where information flows "forward" from the network's input units, through its "hidden" units, to its output units. The nodes of the hidden layer process input data they receive as the sum of the weighted outputs of the input layer. Nodes of the output layer process input data they receive as the sum of the weighted output of the units within the hidden layers, and supply the system output.

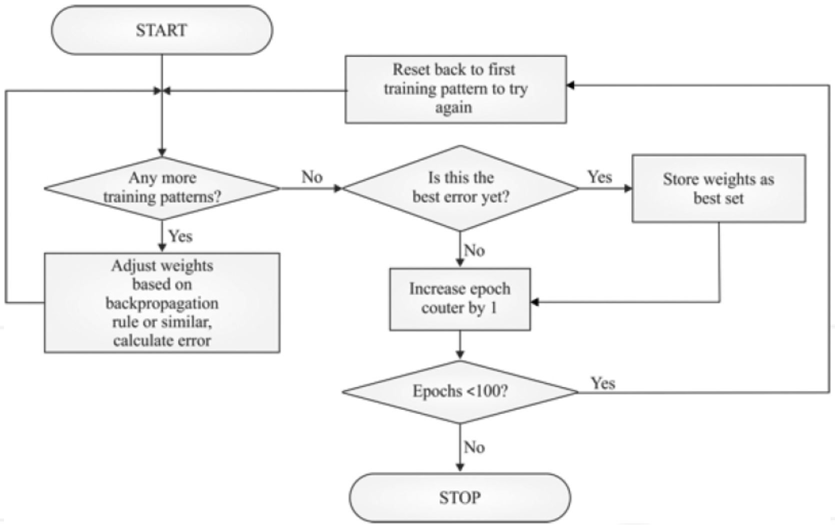
START
Reset back to first training pattern to try again
Any more training patterns?
No
Is this the best error yet?
Yes
Store weights as best set
Yes
Adjust weights based on backpropagation rule or similar, calculate error
No
Increase epoch couter by 1
Epochs <100?
Yes
No
STOP

Figure 10. Supervised learning of ANN.

As mentioned earlier, the principles of learning can be applied to machines to improve their performance [15]. In FFNN, network learning is a very important process. The learning situation can be divided into two major categories: supervised and unsupervised. With supervised learning, the ANN must be trained before it becomes useful. Training consists of presenting input and output data to the network. **Figure 10** shows the distinguishing nature of supervised neural network, which incorporates an external trainer in which input and output are known, and its objective is to discover a relationship between the two. In this mode, the actual output of ANN is compared to the desired output.

An important issue concerning supervised learning is the problem of error convergence: the minimization of error between the desired and computed values. The performance of the network is evaluated based on the comparison between the computed (predicted) output and actual (desired) output value [10–15]. There are several types of measurements of prediction accuracy; the most common measurements used are as follows:

1. Coefficient of determination (R^2)

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \tag{1}$$

2. Mean Square Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(Y_i - \hat{Y}_i)^2 \tag{2}$$

3. Root Mean Square Error (RMSE)

$$RMSE = \left[\frac{1}{n}\sum_{i=0}^{n}(Y_i - \hat{Y}_i)^2\right]^{1/2} \tag{3}$$

4. Mean Absolute Percentage Error (MAPE)

$$MAPE\% = \frac{1}{n}\sum_{i=0}^{n}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right| \times 100 \tag{4}$$

where Y_i is the actual value of output, $\hat{Y}_i$ is the predicted value, and (n) is the number of observations.

Unlike supervised learning, unsupervised neural network uses no external feedback and it is based upon only local information. As can be seen from **Figure 11**, in unsupervised learning only the input is known and the goal is to uncover patterns in the features of the input data. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Unsupervised learning's goal is to have the computer learn how to do something that we do not tell it how to do. The common applications of unsupervised learning are classification, data mining, and self-organizing maps (SOM), also called Kohonen Neural Network (KNN).

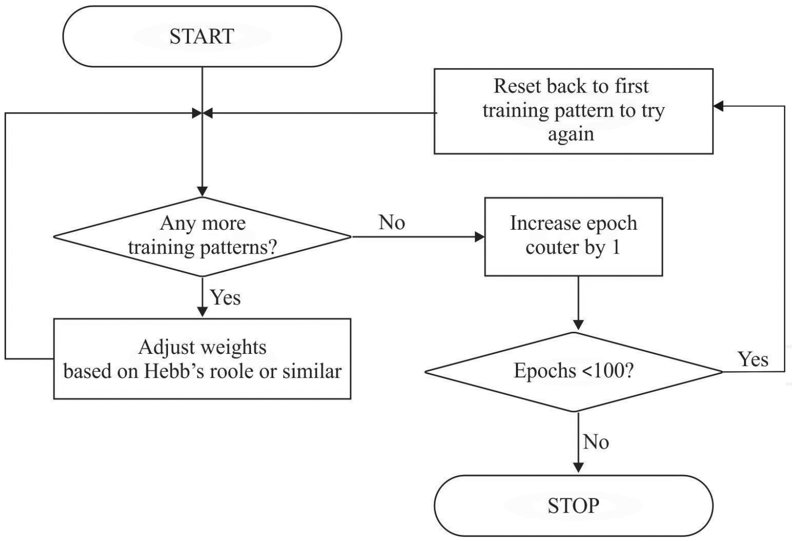
START
Reset back to first training pattern to try again
Any more training patterns?
No
Increase epoch couter by 1
Yes
Adjust weights based on Hebb's roole or similar
Epochs <100?
Yes
No
STOP

Figure 11. Unsupervised learning of ANN.

In FFNN with supervised training, two very different types of neural networks exist: FFNN trained with Backpropagation (BP) algorithm (FFBPNN) and Statistical Neural Networks (SNN) [10, 11, 14]. FFBPNNs use equations that are connected using weighting factors [11]. The selection of the weighting factors makes these neural nets very powerful. The multilayer perceptron (MLP) is the most common and successful neural network architecture with FFNN topologies, while the most common supervised learning technique used for training artificial neural networks is the multilayer backpropagation (BP) algorithm [10–15].

BP is a systematic method for training multilayer FFNN as shown in **Figure 8**. Since it is a supervised training algorithm, both the input and the target patterns are given (**Figure 10**). For a given input pattern, the output vector is estimated though a forward pass on the network. After the forward pass is over, the error vector at the output layer is estimated by taking the component-wise difference of the target pattern and the generated output vector. A function of errors of the output layered nodes is then propagated back through the network to each layer for adjustment of weights in that layer. The weight adaptation policy in BP algorithm is derived following the principle of steepest descent approach of finding minima of a multi-valued function.

BPFFNNs consist of neurons organized into one input layer and one output layer and several hidden layers of neurons as shown in **Figure 8**. Neurons perform some kind of calculation using inputs to compute an output that represents the system. The outputs are given on to the next neuron. An edge indicates to which neurons the output is given. These arcs carry weights.

Generally, BP learning consists of two passes: a forward pass and a backward pass. In the forward pass, an activity pattern is applied to the sensory nodes of the network. It is at last that a set of outputs is produced as the actual responses of the network. During this path, the synaptic weights are fixed. During backward pass, the synaptic weights are adjusted in accordance with an error correction rule.

BPFFNNs have the desirable characteristic of being very flexible. They can be used for pattern recognition as well as for decision-making problems. Another advantage is that like for every other neural network, the process is highly parallel and therefore the use of parallel processors is possible and cuts down the necessary time for calculations. However, BPNNs have negative characteristics. The training of the network can need a substantial amount of time [16]. The size of the training data for BPFFNN has to be very large. In some instances, it is almost impossible to provide enough training.

On the other hand, SNNs use statistical methods to select the equations within the structure and do not weigh these functions differently [17].

1.2. Statistical neural networks

SNNs are an important and very popular type of neural networks that mainly depend on statistical methods and probability theory [18]. Three of the most important types of these networks are Radial Basis Function Neural Network (RBFNNs), Probabilistic Neural Network (PNNs), and General Regression Neural Network (GRNNs) [19].

1.2.1. Radial basis function neural network

RBFNN was introduced by Broomhead and Lowe in 1988 and is a popular alternative to FFBPNN [20]. The behavior of the network depends on the weights and the activation of a transfer function F, specified for the units [21]. Activation functions are mathematical formulas that determine the output of a processing node [22]. The activation function maps the sum of weighted values passed to them by applying F into the output value, which is then "fired" on to the next layer.

There are several kinds of transfer or activation functions, typically falling into four common categories: Linear function (LF), Threshold function (TF), Sigmoid Function (SF), and Radial Basis Function (RBF) [23]. RBFs are a special class of activation functions which form a set of basis functions, one for each data set. The general form of RBF is:

$$G\left(\|X-\mu\|\right) \tag{5}$$

where G(.) is a positive nonlinear symmetric radial function (kernel); X is the input pattern and μ is the center of the function. Another important property of RBF is that its output is symmetric around the associated center μ. Thus, $f(X_i)$ can be taken to be a linear combination of the outputs of all the basis functions:

$$f(x) = \sum_{i=1}^{n} w_i G(X - \mu) \tag{6}$$

There are several common types of radial basis functions represented in **Table 1** [19–23]:

Function name	Mathematical form
Thin plate spline	$G(x) = (x - \mu)^2 \log(x - \mu)$
Multi-quadratic	$G(x) = \sqrt{(x-\mu)^2 + \sigma^2}$
Inverse multi-quadratic	$G(x) = \frac{1}{\sqrt{(x-\mu)^2 + \sigma^2}}$
Gaussian	$G(x) = exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$

Table 1. Types of radial basis functions.

where these function parameters are the center (μ) and the radius (σ^2). A Gaussian function, also called "bell shaped curve" or normal distribution, is the most common applicable type of RBF. It is suitable not only in generalizing a global mapping but also in refining local features. The Gaussian function tends to be local in its response and is biologically more acceptable than other functions. RBF is unique, because unlike the others, it monotonically decreases with distance from the center, and forms the classic bell shaped curve which maps high values into low ones, and maps mid-range values into high ones. A plot of a Gaussian function is represented in **Figure 12**.

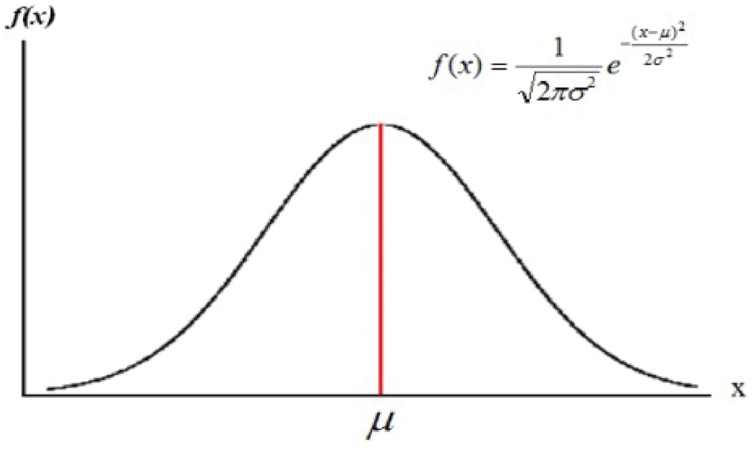

Figure 12. Plot representing a Gaussian function.

The mathematical form of this function for the case of a single variable is given by:

$$f(x)=\frac{1}{\sqrt{2\pi^2}}e^{\frac{(\chi-\mu)^2}{2\sigma^2}} \tag{7}$$

where

$$\mu=E(X)=\int_{-\infty}^{\infty}\chi.f(x)dx \tag{8}$$

$$\sigma^2=E(X-\mu)^2=\int_{-\infty}^{\infty}(\chi-\mu)^2.f(x)dx \tag{9}$$

μ: is the mean (center) of the distribution.

σ^2: is the variance (width or radius) of distribution.

Extending the formula (7) to multiple dimensions, we can get the general Gaussian probability density:

$$f(x)=\frac{1}{(2\pi)^{p/2}|\Sigma|^{p/2}}exp\left(-\frac{1}{2}(\underline{x}-\underline{\mu})^T\Sigma^{-1}(\underline{x}-\underline{\mu})\right) \tag{10}$$

where p is the number of dimensions, μ is the mean p-dimensional vector and Σ is the covariance p x p matrix.

RBFNNs are useful in solving problems where the input data are corrupted with additive noise and can be used for approximating functions and recognizing patterns [24]. As shown in **Figure 13**, the RBFNN has a feed forward architecture, and it is composed of many interconnected processing units or neurons organized in three successive layers. The first layer is the input layer. There is one neuron in the input layer for each predictor variable. The second layer is the hidden layer. This layer has a variable number of neurons. Each neuron consists of a RBF centered on a point with as many dimensions as there are predictor variables.

The standard euclidean distance is used to measure how far an input vector is located from the center. The value coming out from the neuron in the hidden layer is multiplied by a weight (W_i) associated with the neuron, also a bias value that is multiplied by a weight (W_o), is passed to the summation layer which adds up the weighted values and presents this sum as the network outputs.

The training of RBFNNs is radically different from the training of FFNNs [19–24]. RBFNN training may be done in two stages: First, calculating the RBF parameters, including centers and the scaling parameter. Various parameters, such as the number of neurons in the hidden layer, the coordinates of the center of each hidden layer function, the radius (width) of each function in each hidden unit, and the weights between the hidden and output units, are determined by the training process; second, estimating the weights between the hidden and output layers. Opposed to BPFFNN, in RBFNN training, there is no changing of the weights with the use of the gradient method for function minimization. In RBFNNs, training resolves itself into selecting the centers and calculating the weights of the output neuron.

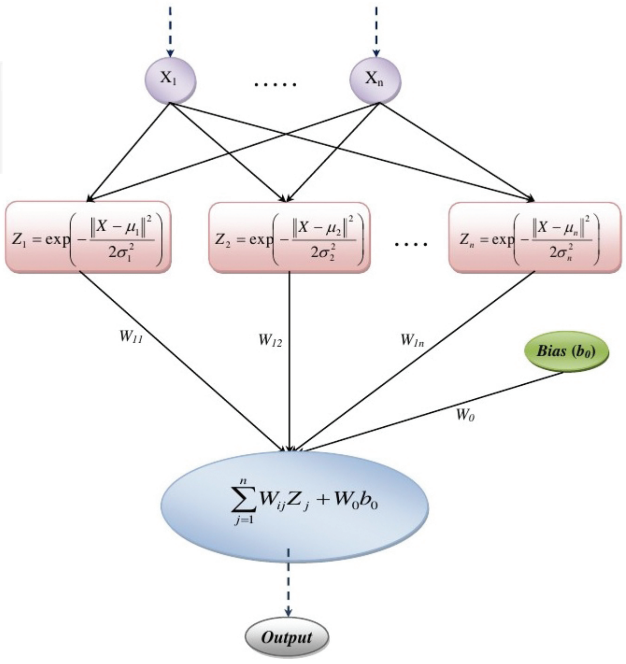
X_1
.....
X_n
$Z_1 = \exp\left(-\frac{\|X-\mu_1\|^2}{2\sigma_1^2}\right)$
$Z_2 = \exp\left(-\frac{\|X-\mu_2\|^2}{2\sigma_2^2}\right)$
....
$Z_n = \exp\left(-\frac{\|X-\mu_n\|^2}{2\sigma_n^2}\right)$
W_{11}
W_{12}
W_{1n}
Bias (b_0)
W_0
$\sum_{j=1}^{n} W_{ij} Z_j + W_0 b_0$
Output

Figure 13. Network architecture of RBFNN.

The center (μ) and width radius (σ) of the radial function and final weights are the parameters of the model. Many algorithms have been designed to determine these parameters by minimizing the error between the target and actual output. Determination of centers is important for the success of the RBFNN and there are several methods to choose suitable centers for network, such as random selection from data set, randomly fixed, and clustering approach.

Determination of the width is very important for the success of the RBFNN. If the width values are large, the model will not be able to closely fit the function; on the other hand, a large width parameter would give better generalization but poorer output. A small width parameter gives good recall of the training patterns but poor generalization, and the model will over fit the data because each training point will have too much influence. There are several methods to determine the width. Two of the common methods for width selection are fixed method and distance averaging.

The number of hidden units is very important and plays a major role in RBFNN performance. It is very difficult to find a suitable number of hidden units. If the number of hidden units is too low, the network cannot reach a desired level of performance because of an insufficient number of hidden neurons. Many researchers assumed that the number of hidden units is fixed and is chose a priori.

There are several types of learning that can be used in RBFNNs, such as General Regression Neural Network (GRNN), Orthogonal Least Squares, K-Means Clustering, and P-Nearest Neighbour.

1.2.2. Probabilistic neural network

Specht first introduced the probabilistic neural network (PNN) in the 1990s. It closely related to "the Bayes Strategy for Pattern Classification" rule and Parzen nonparametric probability density function estimation theory. It performs classification where the target variable is categorical [19–24].

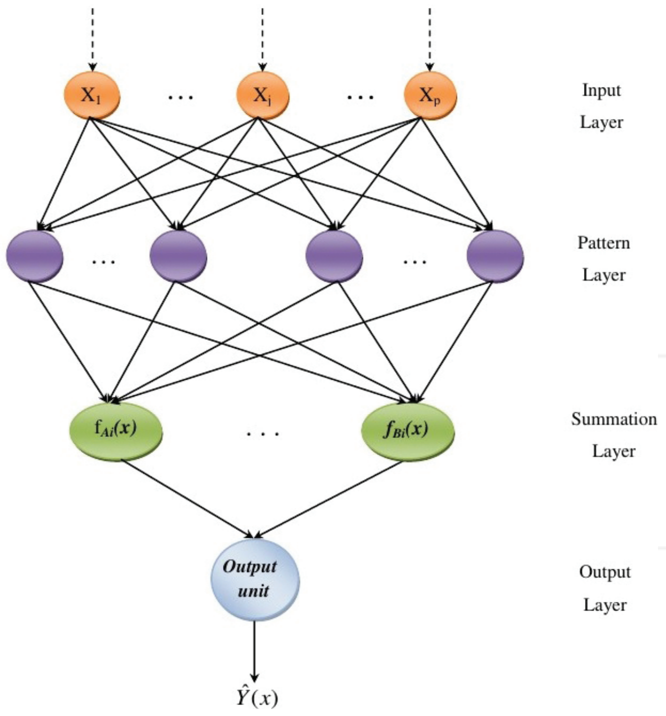

Figure 14. Block diagram of a probabilistic neural network (PNN).

PNNs are often more accurate than FFNNs and it is usually much faster to train PNNs than FFNNs. The greatest advantages of PNNs are the fact that the output is probabilistic which makes interpretation of output easy, and the training speed. Training a PNN is very fast because it requires that each pattern be presented to the network only once during training, unlike BPFFNNs, which require feedback of errors and adjusting weights and many presentations of training patterns. These PNNs, with variation can be used for mapping, classification, and associative memory. The greatest disadvantage is the network size since PNNs require more memory space to store the model.

The general structure of PNN, which is presented in **Figure 14**, consists of four layers. The first layer is the input layer. The input unit nodes do not perform any computation and simply distribute the input to the neurons in the first hidden layer (pattern layer). There is one neuron in the input layer for each predictor variable.

The second layer is the pattern layer. Each pattern unit represents information on one training sample. Each pattern unit calculates the probability of how well the input vector fits into the pattern unit. The neurons of the pattern layer are divided into K groups, one for each category. The *i*-th pattern neuron in the *k*-th group computes its output using a Gaussian kernel with the form:

$$f_{Ai}(X) = \frac{1}{(2\pi\sigma^2)^{p/2}} exp\left[-\frac{(X - X_{Ai})^T - (X - X_{Ai})}{2\sigma^2}\right] \tag{11}$$

where:

i: is the pattern number.

p: denotes the dimension of the pattern vector x.

σ: is the smoothing parameter of the Gaussian Kernel.

X_{Ai}: is the center of the kernel.

The third layer is the summation layer. In the summation layer, there is one pattern neuron for each category of the target variable. The neurons of this layer compute the approximation of the conditional class probability function through a combination of the previously computed densities as the following equation:

$$f_A(X) = \frac{1}{(2\pi\sigma^2)^{p/2}} \frac{1}{M} \sum_{i=1}^{M} exp\left[-\frac{(X - X_{Ai})^T - (X - X_{Ai})}{2\sigma^2}\right] \tag{12}$$

The fourth layer is the output layer (also called decision layer). At the output layer, we have a hard-limiting threshold: (+1) whenever an input pattern X belongs to category (A), and (-1) if it is from category (B).

The use of PNN is especially advantageous due to its ability to converge to the underlying function of the data with only few training samples available. The additional knowledge needed to get the fit in a satisfying way is relatively small and can be done without additional input by the user. GRNN falls into the category of PNN. This neural network, like other SNNs, needs only a fraction of the training samples a BPFFNN would need, mainly because the data available from measurements of an instance is generally never enough for a BPFFNN. This makes GRNN a very useful tool to perform predictions and comparisons of system performance in practice.

The invention of GRNN was a great turn in the history of neural networks. Researchers from many fields including medicine, engineering, commerce, physics, chemistry, geology, statistics, etc., benefited from this technique for their research.

1.2.3. Generalized regression neural network

GRNN is a type of supervised FFNN and is one of the most popular neural networks. Donald F. Specht first introduced it in 1991. Specht's GRNN is related to his probabilistic neural network (PNN) classifier. Like PNN networks, GRNNs are known for their ability to train quickly on sparse data sets. Rather than categorizing data like PNN, GRNN applications are able to produce continuous valued outputs. An important by-product of the GRNN network is Bayesian posterior probabilities. The training of GRNN networks is very fast because the data only needs to propagate forward once, unlike most other BPNNs, where data may be propagated forward and backward many times until an acceptable error is found [19–24].

GRNNs work well on interpolation problems. However, because they are function approximators, they tend to trade accuracy for speed. The GRNN is used for estimation of continuous variables, as in standard regression techniques. It uses a single common radial basis function kernel bandwidth (σ) that is tuned to achieve optimal learning.

The regression performed by GRNN is in fact the conditional expectation of Y, given X = x. In other words, it outputs the most probable scalar Y given specified input vector x. Let f(x, y) be the joint continuous probability density function of a vector random variable, X, and a scalar random variable, Y. Let x be a particular measured value of the random X. The regression of Y given x (also called conditional mean of Y given x) is given by:

$$E[Y/x] = \int_{-\infty}^{\infty} Y.f(Y/x)dy = \frac{\int_{-\infty}^{\infty} Y.f(x,\, Y)dy}{\int_{-\infty}^{\infty} (x,Y)dy} \tag{13}$$

If the relationship between independent (X) and dependent (Y) variables is expressed in a functional form with parameters, then the regression will be parametric. Without any real knowledge of the functional form between the x and y, nonparametric estimation method will be used. For a nonparametric estimate of f(x, y), we will use one of the consistent estimators that is a Gaussian function. This estimator is a good choice for estimating the probability density function, *f*, if it can be assumed that the underlying density is continuous and that the first partial derivatives of the function evaluated at any *x* are small. The good choice for probability estimator $\widehat{f}(x,y)$ is based on sample values x_i and y_i of the random variables *X* and *Y* is given by:

$$\widehat{f}(x,y) = \frac{1}{(2\pi)^{(p+1)/2.\sigma^{p+1}}} . \frac{1}{n} \sum\nolimits_{i=1}^{n} \left\{ exp\left[-\frac{(X-X_i)^T (X-X_i)}{2\sigma^2} \right] exp\left[\frac{(Y-Y_i)^2}{2\sigma^2} \right] \right\} \tag{14}$$

p: is the dimension of the vector variable.

n: is the number of training pairs ($x_i \rightarrow y_i$).

σ: is the single learning or smoothing parameter chosen during network training.

Y_i: is desired scalar output given the observed input x_i.

The topology of GRNN presented in **Figure 15** consists of four layers: The first layer is the input layer that is fully connected to the second layer. The input units are merely distribution units, which provide all of the (scaled) measurement variables *X* to all of the neurons on the second layer, the pattern units. The second layer is the first hidden layer (also called the pattern layer). This layer consists of *N* processing elements or nodes, where *N* is the number of sample within a training data set and each node represents the input vector, *Xi*, associated with the vector assigned with the *j*th sample in training data. In each node, each input vector is subtracted from the vector assigned to the node, *Xj*. This difference is then squared by the node. The result is fed into a nonlinear kernel, which is usually an exponential function. The pattern unit outputs are passed on to the summation units.

Note that the second hidden layer always has exactly one more node than the output layer. When you need a multidimensional (vector) output, the only change to the network is to add one additional node to the second hidden layer, plus an additional node in the output layer for each element of the output vector.

The third layer is the second hidden layer (Summation layer) which has two nodes. The input to the first node is the sum of the first hidden layer outputs, each weighted by the observed output *yj* corresponding to *Xj*. The input of the second node is the summation of the first hidden layer activations.

The fourth layer is the output layer. It receives the two outputs from the hidden layer and divides them to yield an estimate for *y* (or to provide the prediction result).

In the GRNN architecture, unlike other network architectures as in BP, there are no training parameters such as learning rate and momentum, but there is a smoothing factor (σ) that is applied after the network is trained. The choice of smoothing factor (parameter) of the kernel σ is very important. It has the effect of smoothing the training examples. Small values of σ tend to make each training point distinct, whereas large values force a greater degree of interpolation between the training observations. For GRNNs, the smoothing factor must be greater than 0 and can usually range from 0.01 to 1 with good results. We need to experiment in order to determine which smoothing factor is most appropriate for our data.

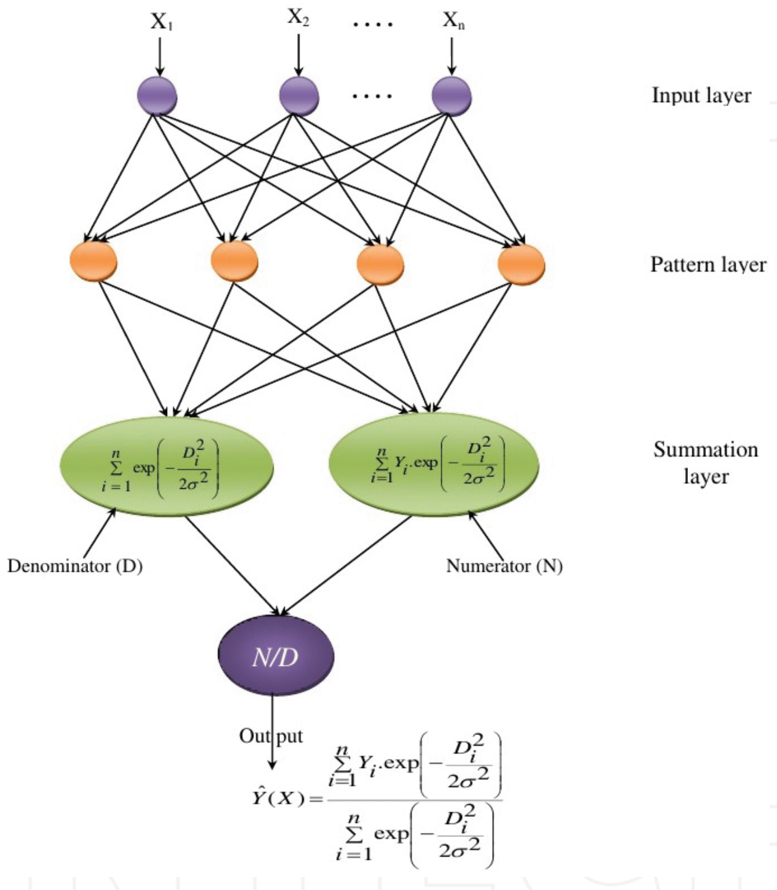
X_1
X_2
X_n
Input layer
Pattern layer
Summation layer
$\sum_{i=1}^{n} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)$
$\sum_{i=1}^{n} Y_i.\exp\left(-\frac{D_i^2}{2\sigma^2}\right)$
Denominator (D)
Numerator (N)
N/D
Out put
$\hat{Y}(X)=\frac{\sum_{i=1}^{n} Y_i.\exp\left(-\frac{D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}$

Figure 15. The basic GRNN architecture.

A useful method of selecting an appropriate σ is the Holdout method. For a particular value of σ, the Holdout method consists in removing one sample at a time and constructing a network based on all of the other samples. The network is then used to estimate Y for the removed sample. By repeating this process for each sample and storing each estimate, the mean square error can be measured between the actual sample values Yi and the estimates. The value of σ giving the smallest error should be used in the final network.

Fortunately, in most applications there is a unique σ which produces the minimum MSE between the network output and the desired output for the testing set that can be found quickly by trial and error.

2. Neutron spectrometry by means of generalized regression neural networks

2.1. Neutron spectrometry

In general, neutrons are more difficult to detect than gamma rays because of their weak interaction with matter and their large dynamic range in energy [25]. Neutrons have mass but no electrical charge [26]. Because of this, they cannot directly produce ionization in a detector, and therefore cannot be directly detected. This means that neutron detectors must rely upon a conversion process where an incident neutron interacts with a nucleus to produce a secondary charged particle [27]. These charged particles are then directly detected and from them the presence of neutrons is deduced.

The derivation of the spectral information is not simple because the unknown is not given directly as a result of measurements [28]. The spectral information is derived through the discrete version of the Fredholm integral-differential equation of first type [29]. Normally, researchers solve a discrete version of this equation, which gives an ill-conditioned system of equations which have no explicit solution, may have no unique solution, and are referred to as ill-posed [30].

Since the 1960s, the Bonner Sphere Spectrometer (BSS) has been the most used method for radiological protection purposes [28]. The isotropy of the response, the wide energy range (from thermal to GeV neutrons), and the easy operation make these systems still applicable. BSS consists of a thermal neutron detector located at the center of several high-density polyethylene spheres of different diameters [29]. By measuring the count rates with each sphere individually, an unfolding process can, in principle, provide some information about the energy distribution of the incident neutrons.

The most delicate part of neutron spectrometry based on BSS is the unfolding process [30]. The unfolding spectrum of the neutrons measured consists of establishing the rate of energy distribution of fluency, known as response matrix, and the group of carried-out measures. Because the number of unknowns overcomes the number of equations, this ill-conditioned system has an infinite number of solutions. The process of selecting the solution that has meaning for the problem is part of the unfolding process.

To solve the system of equations for BSS unfolding, several approaches have been used [29]: iterative procedures, Monte Carlo, regularization, and maximum entropy methods. The drawbacks associated with these traditional unfolding procedures have motivated the need for complementary approaches. Novel methods based on AI have been suggested. In neutron spectrometry, the theory of ANN has offered a promising alternative to the classic calculations with traditional methods. Previous researches indicate that BPFFNNs perform well and have been the most popular networks used in neutron spectrometry [30–35].

BPFFNN have the characteristic of being very flexible; the process is highly parallel and can be used to solve diverse problems; however, this neural network topology has some drawbacks: the structural and learning parameters of the network are often determined using the trial-and-error technique [36]. This produces networks with poor performance and generalization capabilities which affect its application in real problems. Training can require a substantial amount of time to gradually approach good values of the weights. The size of the training data has to be very large and often it is almost impossible to provide enough training samples as in the case of the neutron spectrometry problem.

Another drawback is that adding new information requires retraining the network and this is computationally very expensive for BPFFNN, but not for GRNN which belongs to SNNs. GRNNs use a statistical approach in their prediction algorithm given the bases in the Bayes strategy for pattern recognition. To be able to use the Bayes strategy, it is necessary to estimate the probability density function accurately. The only available information to estimate the density functions is the training samples. These strategies can be applied to problems containing any number of categories as in the case of the neutron spectrometry problem.

2.2. Neutron spectrometry by means of generalized regression neural networks

A GRNN has certain differences compared to BPFFNN approach [24]. The learning of BPFFNN can be described as trial and error. This is no longer the case of the GRNNs because they use a statistical approach in their prediction algorithm which is capable of working with only few training samples. The experience is learned not by trial but by experience others made for the neural network. GRNNs are very flexible and new information can be added immediately with almost no retraining. The biggest advantage is the fact that the probabilistic approach of GRNN works with one-step-only learning.

A further big difference that exists between BPFFNN and GRNN is the difference in the process inside the neurons. A GRNN uses functions that are based on knowledge resulting from the Bayes strategy for pattern classification. The structure of the calculations for the probabilistic density function in GRNN has striking similarities to a BPFFNN. The strength of a GRNN lies in the function that is used inside the neuron.

It would be desirable to approach the parameters in one-step-only approach. The Bayes strategy for pattern classification extracts characteristics from the training samples to come to knowledge about underlying function.

In this work, both BPFFNN and GRNN architectures were trained in order to solve the neutron spectrometry problem using customized technological tools designed with this purpose. A comparison of the performance obtained using both architectures was performed. Results obtained show that the two architectures solve the neutron spectrometry problem well, with high performance and generalization capabilities; however, the results obtained with GRNN are better than those obtained with BPFFNN, mainly because GRNN does not produce negative values and oscillations around the target value.

As mentioned, a GRNN is a BPFFNN based on non-linear regression. It is suited to function approximation tasks such as system modeling and prediction. While the neurons in the first three layers are fully connected, each output neuron is connected only to some processing units in the summation layer. The function of the pattern layers of the GRNN is a Radial Basis Function (RBF), typically the Gaussian kernel function.

In this work, a neutron spectrum unfolding computer tool based on neural nets technology was designed to train a GRNN capable of solving the neutron spectrum unfolding problem with high performance and generalization capabilities. The code automates the pre-processing, training, testing, validation, and post-processing stages of the information regarded with GRNN. The code is capable of training, testing, and validating GRNN. After training and testing the neural net, the code analyzes, graphs, and stores the results obtained.

2.3. Methods

The use of GRNN to unfold the neutron spectra from the count rates measured with the BSS is a promising alternative procedure; however, one of the main drawbacks is the lack of scientific and technological tools based on this technology. In consequence, a scientific computational tool was designed to train, to test, to analyze, and to validate GRNN in this research domain.

Statistical methods tend to put more emphasis on the structure of the data. For neural network methods, the structure of the data is secondary. Therefore, the amount of data needed for statistical methods is a lot smaller than the amount of data needed for ANN approaches. GRNNs are frequently used to classify patterns based on learning from examples. PNNs base the algorithm on the Bayes strategy for pattern recognition.

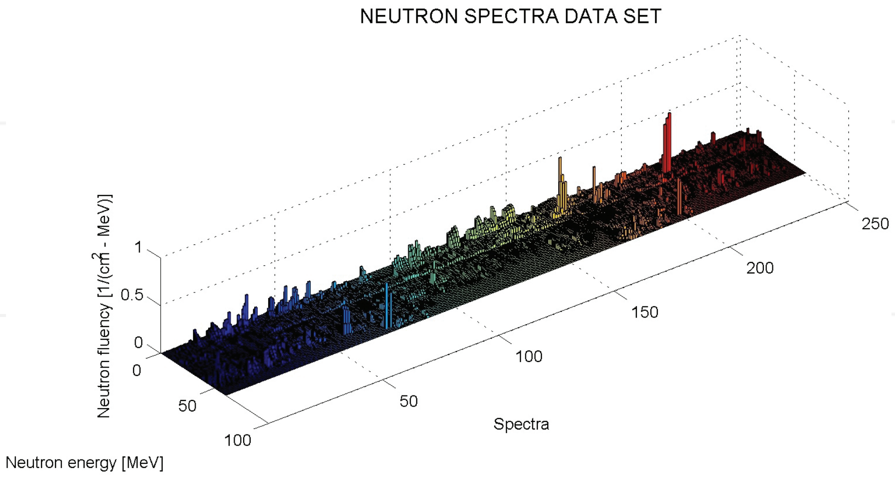

Figure 16. Neutron spectra data set expressed in energy units, used to train the GRNN.

In order to train both BPFFNNs and GRNNs, the only available information is a neutron spectra compilation of the International Atomic Energy Agency (IAEA) which contains a collection of 251 different neutron spectra [37]. This compendium was made with the aim to provide specific technical information that could be used by radiation protection specialists for proper selection of dosimeters and survey instruments, and for interpretation of data obtained with these detectors.

The developed code based on GRNNs technology utilizes these 251 neutron spectra and both, the response matrixes from IAEA's compilation and those that could be introduced by the user. The designed technological tool automates the following activities:

- Read the neutron spectra data set coming from IAEA's compendium, which are expressed in 60 energy bins.
- Read a response matrix used to train the neural network.
- Because the neutron spectra coming from IAEA's compendium are expressed in lethargy units, the code converts these spectra in energy units.
- The neutrons expressed in energy units are multiplied by the selected response matrix in order to calculate the count rates.
- To train the GRNN, the code uses the 251 calculated count rates as entrance data, and their corresponding neutron spectra are expressed in energy units as the output data as shown in **Figure 16**.
- The code randomly generates the training data set, 80% of the whole data, and the testing data set, remaining 20%, as shown in **Figure 17**.
- Using the earlier calculated information, the following stage is to determine the spread constant value. To calculate this value, the computer tool trains several neural networks varying this value from 0 in increments of 0.01 through 2 and compares the mean square error (MSE), which is used to determine the performance of the network. The minimum value obtained is selected as the spread constant value (**Figure 18**).
- After the developed code selects the spread constant value, a final GRNN is trained.
- After training, a testing stage is performed in order to analyze the performance and generalization capabilities of the trained network. In this stage only the input is proportionated to the network. Fifty neutron spectra are randomly selected by the code to test the performance and generalization capabilities of the trained network. In order to analyze the performance of the trained network, chi square and correlation tests are performed.
- Finally, the code plots and stores the generated information.

In this work, a comparison of the performance obtained in the solution of the neutron spectrometry problem using two different neural network architectures, BPFFNN and GRNN, is presented. Both BPFFNN and GRNN were trained and tested using the same information: 251 neutron spectra, extracted from IAEA's compilation. Eighty percent of the whole data set, randomly selected, was used at training stage and remaining 20% at testing stage. Fifty neutron spectra were used as testing data set.

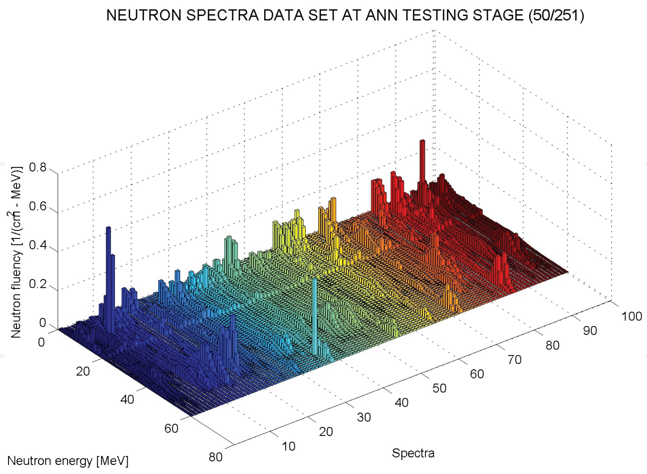

Figure 17. Neutron spectra data set used at testing stage, compared with target spectra.

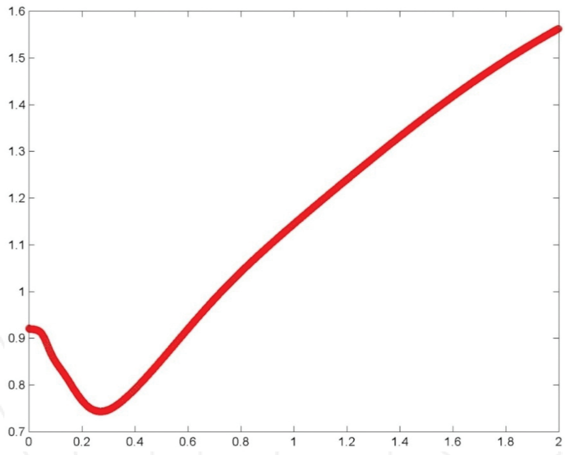

Figure 18. Optimum spread constant value, sigma, and determination.

The architectural and learning parameters of BPFFNN were optimized using a statistical methodology known as Robust Design of Artificial Neural Networks Methodology (RDANNM) [36]. In GRNN, the only parameter determined was the spread constant value, known as sigma. For both architectures, BPFFNN and GRNN, customized scientific computational tools were used for the training, testing, analysis, and storage of the information generated in the whole process of both network architectures. It can be observed from the results obtained that although the two network architectures present very similar performance and generalization capabilities, GRNN performs better than BPFFNN in the solution of the neutron spectrometry problem. BPFFNNs produce negative values and high oscillations around the target values, which makes this type of network unusable in the solution of the problem mentioned.

2.4. Results

In this work, by using two different technological tools, two different artificial neural networks architectures, BPFFNN and GRRN, were trained and tested using the same information. The performance of the networks was compared. From the results obtained, it can be observed that GRNN performs better than BPFFNN in the solution of the neutron spectrometry problem.

Network parameters	**BPNN (trial and error)**	**BPNN (RDANNM)**	**GRNN**
Networks tested before training	Undetermined	50 in 150 minutes	2000 in 154 seconds
Hidden layers	Undetermined	1	Fixed architecture
Neurons in hidden layer	Undetermined	10	According input
Training algorithm	Undetermined	Trainscg	Statistical methods
Learning rate	Undetermined	0.1	–
Momentum	Undetermined	0.01	–
Spread constant	–	–	0.2711
Performance (MSE)	Undetermined	2.12E-4	2.48E-4
Training time (seconds)	Several hours	170.40	0.058
Epochs	Often millions	50E3	1
Best chi-square test BPNN	–	2.3525	0.049
Statistical margin 34.7			
Best correlation test BPNN	–	0.9928	0.99571
Statistical margin 1			
Worst chi-square test BPNN	–	0.44704	0.3223
Worst correlation test BPNN	–	0.2926	0.46023

Table 2. Comparison between BPFFNN and GRNN values in neutron spectrometry.

By using the RDANNM, around 50 different network architectures were trained in 150 minutes average, before the selection of the near-optimum architecture. By testing different network architectures according to RDANNM, each network was trained in 50E3 epochs and 180 seconds average, stopping the training when the network reached the established mean square error (MSE) equal to 1E-4, the value used to measure the network performance. After selecting the near-optimum architectural and learning parameters of the BPFFNN, the network was trained and tested using the values shown in **Table 2**: one hidden layer with 10 neurons, a trainscg training algorithm, and a learning rate and momentum equal to 0.1 and 0.01, respectively.

As can be seen in **Table 2**, contrary to BPFFNN the spread constant or sigma was the only value determined in GRNN. Using the same training and testing data sets used for BPFFNN, around 2000 neural networks were trained in 154 seconds average in order to determine the spread constant equal to 0.2711. Each GRNN was trained in 0.058 seconds average in only one-step-only learning. Further, a final GRNN was trained and tested in 0.058 seconds average in only one epoch.

Table 2 shows the values obtained after training the two network architectures compared in this work. As can be seen, when the trial-and-error technique is used, it is very difficult to determine if the performance of the network is good or bad, mainly because a scientific and systematic methodology is not used for determining the near-optimum learning and architectural values as when RDANNM is used.

As can be appreciated in **Table 2**, after training both network architectures, BPFFNN was optimized using RDANNM and GRNN, the performance, MSE, reached by the two networks is very close to each other. In BPFFNNs, the MSE is a value optimized by the network designer using RDANNM; in GRNN network the value was automatically obtained by the network based on the training information used by the automated code. The anterior demonstrates the powerful RDANNM in the optimization of the near-optimum values of BPFFNN architectures.

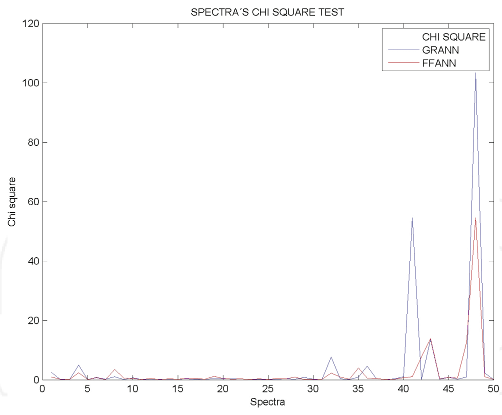

Figure 19. Chi-square test comparison for BPFFNN and GRNN.

Figures 19 and **20** show that at testing stage, the chi square and correlation tests are very close in both BPFFNN and GRNN network architectures. The same 50 neutron spectra were used for testing the two network architectures. At testing stage, only the count rates were proportionated to the trained networks. The output produced by the networks was compared with the expected neutron spectrum taken from IAEA's compilation by means of chi square and correlation tests. In the trained networks, two spectra are above the statistical margin of the chi-square test. In correlation tests, two values are below 0.5. This shows the high performance of the networks.

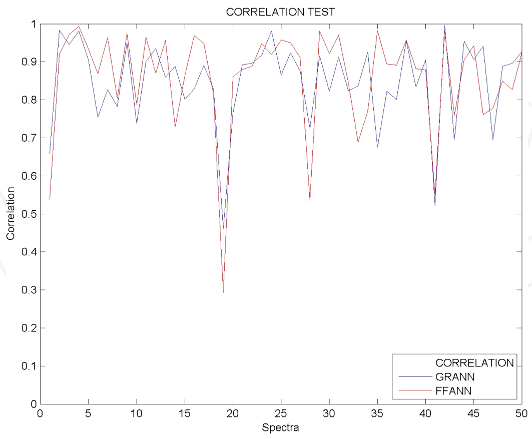

Figure 20. Correlation test comparison for BPFFNN and GRNN.

As can be seen from **Figures 19** and **20**, the 50 chi-square and correlation tests of trained networks are very similar. In both cases, the average value is around 0 and 0.8 respectively, which is near the optimum values equal to 0 and 1. This means that BPFFNN and GRNN have high performance and generalization capabilities and demonstrates the effectiveness of the RDANNM in the design of near-optimum architectures of BPFFNN.

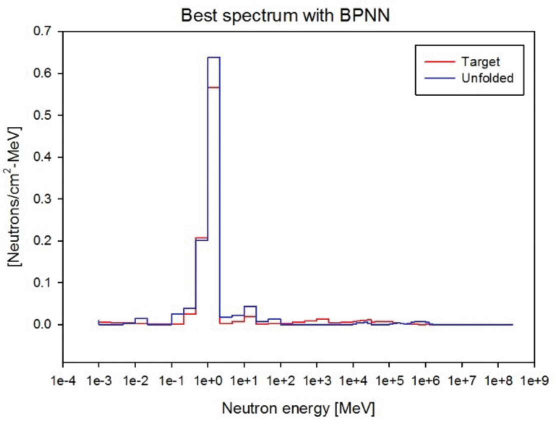

Figure 21. Best spectrum obtained with BPFFNN.

As mentioned earlier, 50 neutron spectra were randomly selected at the testing stage. The same training and testing data sets were used to train and to test the performance and generalization capability of the networks. The best and the worst cases for both BPFFNN and GRNN are showed in **Figures 21–28**. **Figures 21–22** and **23–24** show the best cases observed at testing stage for BPFFNN and GRNN, respectively. From these figures, it can be observed that the chi-square and correlations tests for both BPFFNN and GRNN are near 0 and 1, respectively, which means that the compared neutron spectra are very similar.

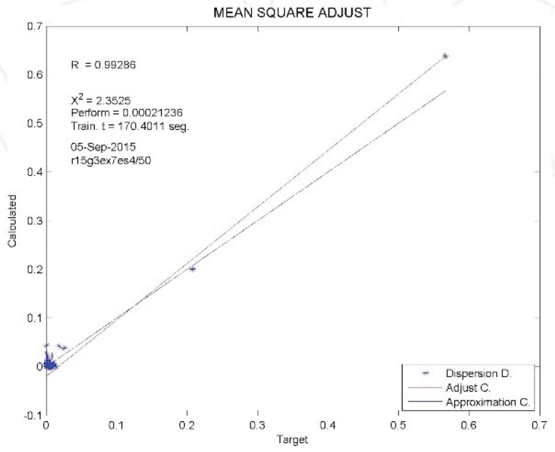

Figure 22. Best chi-square and correlation tests for spectrum obtained with BPNN.

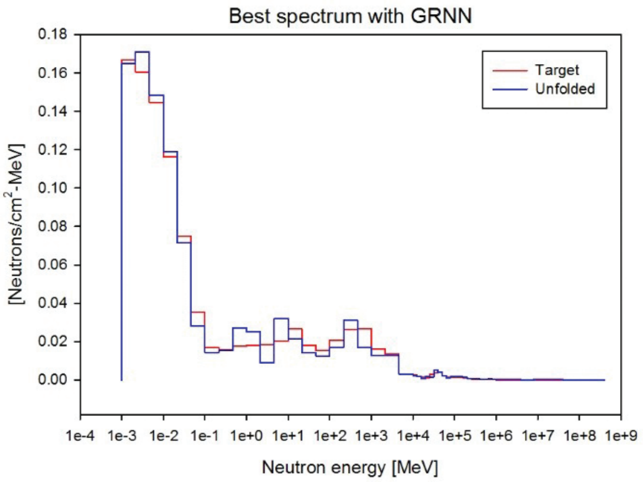

Figure 23. Best spectrum obtained with GRFFNN.

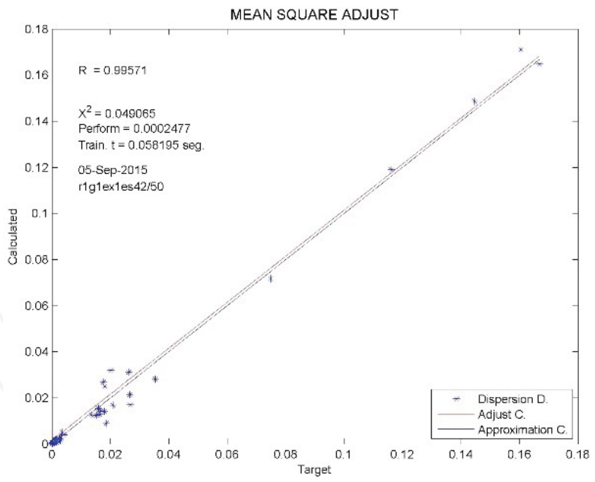

Figure 24. Best chi-square and correlation tests for spectrum obtained with GRNN.

As can be appreciated in **Figures 21**–**28**, despite the good results obtained with BPFFNN, one drawback is that the calculated neutron spectra produce negative values which have no meaning in real problems. These negative values are eliminated from the output produced by the network; however, when the BPFFNN is applied in real workplaces, because the training received, the network tends to produce negative values and oscillations around the target value. GRNN networks do not produce these negative values and oscillations and therefore the performance is better than BPFFNN in the solution of the neutron spectrometry problem.

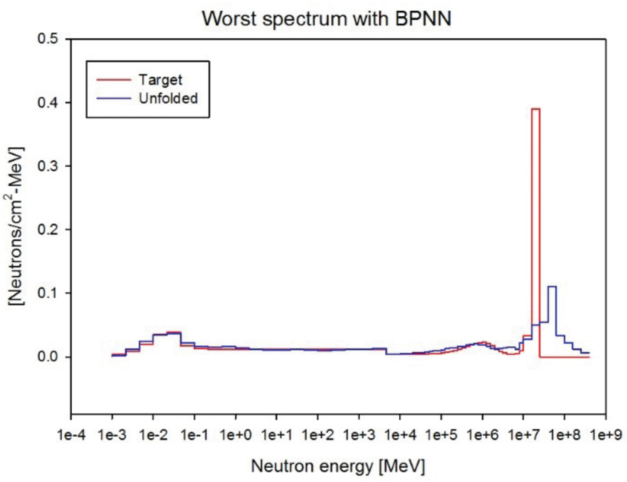

Figure 25. Worst spectrum obtained with BPFFNN.

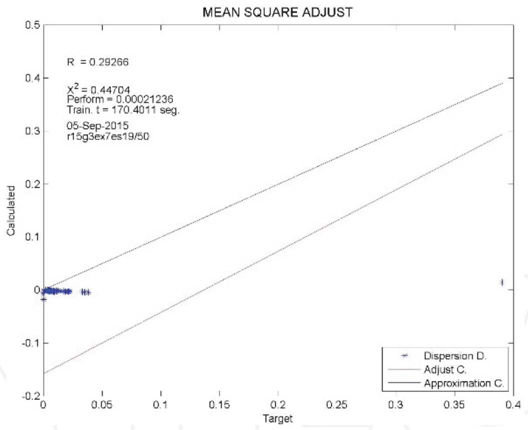

Figure 26. Worst chi-square and correlation tests for spectrum obtained with BPNN.

Figures 25–28 show the worst case observed at the testing stage for BPFFNN and GRNN networks, respectively. As can be seen from these figures, both BPFFNN and GRNN selected the same neutron spectra as the worst. This could be because of the 50 energy bins that the neural networks calculated; 49 values are very similar and only one value is far from the expected target value, which causes that the chi-square and correlation tests to produce low values. From **Figures 25–28**, it can be observed that in the GRNN architecture, the output is closer to the target values of the neutron spectra if compared with BPFFNN. This shows that in the worst case, GRNNs have better performance than BPFFNN.

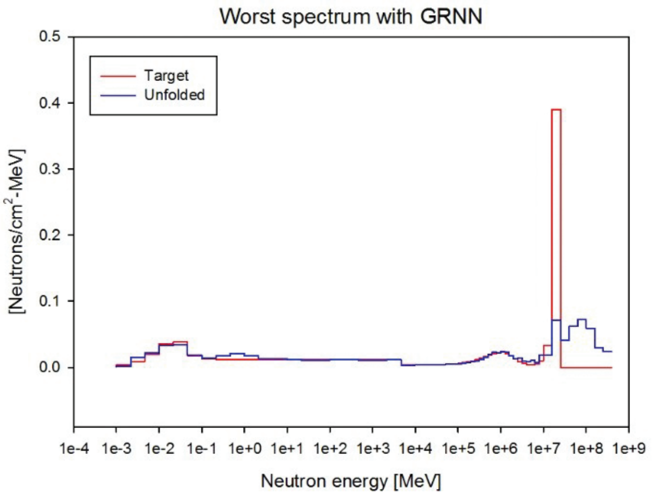

Figure 27. Worst spectrum obtained with BPFFNN.

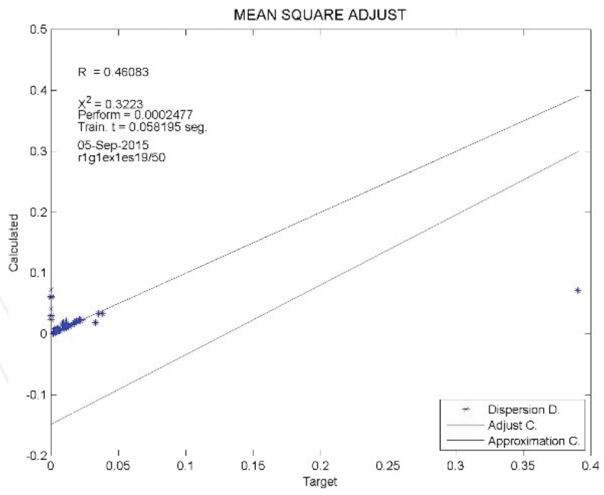

Figure 28. Worst chi-square and correlation tests for spectrum obtained with GRNN.

The results observed in this work indicate that GRNN is able to predict the unknown neutron spectrum presented to the network with good accuracy. As can be seen from **Figures 21–24**, due to proper selection of the spread constant value, the GRNN calculated values, each one of the 60 energy bins of the spectrum, are around the target value (the spectrum from IAEA's compendium). As opposed to BPFFNN, non-negative values and oscillations around the target value are generated when GRNNs are used.

Since there is only one parameter in GRNN, this type of ANN is also called a nonparametric model. It stores the training data as the parameter, rather than calculating and modifying the weights and bias in each hidden layer as the input data imported into the model. When the query comes, the model will calculate the value by summing the values of the other points weighted by the RBF function. Therefore, unlike parametric models such as BP, there are no weights and bias information produced to characterize the trained model.

3. Discussion and conclusions

Different approaches exist to model a system with available data. Each one of them has its own qualities and therefore advantages. GRNN falls into the category of PNN. This neural network, like other PNNs, needs only a fraction of the training samples a BPNN would need. The data available from measurements of an instance is generally never enough for a BPNN. Therefore, the use of GRNN is especially advantageous due to its ability to converge to the underlying function of the data with only few training samples available. The additional knowledge needed to get the fit in a satisfying way is relatively small and can be done without additional input by the user.

Statistical methods tend to put more emphasis on the structure of the data. For neural network methods, the structure of the data is secondary. Therefore, the amount of data needed for statistical methods is a lot smaller than the amount of data needed for ANN approaches.

Most methods are asymptotically good but most of them have severe drawbacks as well. BPNNs need a very large number of training samples and need a lot of time to gradually approach good values of the weights. Addition of new information requires retraining and this is computationally very expensive for BPNN but not for PNN. PNNs have the big advantage that the prediction algorithm works with only few training samples. Other big advantage is that they are very flexible and new information can be added immediately with almost no retraining.

PNNs use a statistical approach in their prediction algorithm. The bases for the statistical approach are given in the Bayes strategy for pattern recognition. These strategies can be applied to problems containing any number of categories as in the case of the neutron spectrometry problem. To be able to use the Bayes strategy, it is necessary to estimate the probability density function accurately. The only available information to estimate the density functions is the training samples.

The structure of the calculations for the probabilistic density function has striking similarities to a backpropagation feed-forward neural network. PNNs are frequently used to classify patterns based on learning from examples. PNNs base the algorithm on the Bayes strategy for pattern classification. Different rules determine patterns statistics from the training samples. BPNN uses methods that are not based on statistical methods and need a long time and many iterations and feedback until it gradually approaches the underlying function. It would be desirable to approach the parameters in one-step-only approach. The Bayes strategy for pattern classification extracts characteristics from the training samples to come to knowledge about underlying function.

In this work, two different artificial neural networks architectures, BPNN and GRRN, were trained and tested using the same information. The performance of the networks was compared. From the results obtained, it can be observed that GRNN performs better than BPNN in the solution of the neutron spectrometry problem.

PNNs have a very simple structure and are therefore very stable procedures. PNNs perform very well for only few available training samples and the quality increases as the number of training samples increases. This makes GRNN a very useful tool to perform predictions and comparisons of system performance in practice. GRNN is a promising technological tool that can be applied to solve with high efficiency the problems related to neutron spectrometry.

Acknowledgements

This work was partially supported by Fondo Sectorial de Investigación para la Eduación under contract 241771, Fondos Mixtos SEP-CONACYT under contract ZAC-C03-2015-26357-4, and PROSOFT under contract 201513723. The first and second authors want to thank the Doctorate scholarships, with scholarship holder numbers 23386 and 23385, respectively, received by Fondo Sectorial de Investigación para la Eduación under contract 241771. The third and fourth authors want to thank the Doctorate scholarships received by Fondos Mixtos SEP-CONACYT under contract ZAC-C03-2015-26357-4. The seventh author want to thank conacyt for the post-doctoral scholarship number 24296. The authors want to thank the active and determined participation and collaboration on several activities on this research project of the undergraduate students: Ana Isabel Ortiz Hernández, Miguel Ángel Acosta García, Fabian García Vázquez, Edgar Viveros Llamas, and Rogelio Osbaldo Reyes Vargas.

Author details

Ma. del Rosario Martinez-Blanco[1,3], Víctor Hugo Castañeda-Miranda[1,3], Gerardo Ornelas-Vargas[1,3], Héctor Alonso Guerrero-Osuna[1,3], Luis Octavio Solis-Sanchez[1,3], Rodrigo Castañeda-Miranda[1,3], José María Celaya-Padilla[1,3], Carlos Eric Galvan-Tejada[3], Jorge Isaac Galvan-Tejada[3], Héctor René Vega-Carrillo[4], Margarita Martínez-Fierro[1,5], Idalia Garza-Veloz[1,5] and Jose Manuel Ortiz-Rodriguez[1,3,2*]

*Address all correspondence to: morvymm@yahoo.com.mx

1 Centro de Investigación e Innovación Tecnológica Industrial (CIITI), Universidad Autónoma de Zacatecas, Zacatecas, México

2 Laboratorio de Innovación y Desarrollo Tecnológico en Inteligencia Artificial (LIDTIA), Universidad Autónoma de Zacatecas, Zacatecas, México

3 Unidad Académica de Ingeniería Eléctrica (UAIE), Universidad Autónoma de Zacatecas, Zacatecas, México

4 Unidad Académica de Estudios Nucleares (UAEN), Universidad Autónoma de Zacatecas, Zacatecas, México

5 Laboratorio de Medicina Molecular, Unidad académica de Medicina Humana y Ciencias de la Salud (UAMHCS), Universidad Autónoma de Zacatecas, Zacatecas, México

References

[1] Fritzsche P. Tools in artificial intelligence. Viena, Austria: InTech; 2008.

[2] Negnevitsky M. Artificial intelligence, a guide to intelligent systems. Reading, MA, USA: Addison Wesley; 2005.

[3] Coppin B. Artificial intelligence illuminated. Burlington, MA, USA: Jones and Barttlet Publishers; 2004.

[4] Russell S.J., Norvig P. Artificial intelligence a modern approach. Mexico: Prentice Hall; 2004.

[5] Luger G.F. Artificial intelligence structures and strategies for complex problem solving. Reading, MA, USA: Addison-Wesley; 2005.

[6] Baldi P., Brunak S. Bioinformatics, the machine learning approach. Cambridge, MA, USA: Mit Press; 2001.

[7] Yu W. Recent advances in intelligent control systems. London: Springer-Verlag; 2009.

[8] Munakata T. Fundamentals of the new artificial intelligence, neural, evolutionary, fuzzy and more. London: Springer; 2008.

[9] Chennakesava R.A. Fuzzy logic and neural networks, basic concepts and applications. New Delhi,India: New Age International Publishers; 2008.

[10] Arbib M.A. Brain theory and neural networks. Cambridge, MA, USA: The Mit Press; 2003.

[11] Haykin S. Neural networks: a comprehensive foundation. Mexico: Prentice Hall; 1999.

[12] Zupan J. Introduction to artificial neural network methods: what they are and how to use them. Acta Chimica Slovenica. 1994;41(3):327–352.

[13] Jain A.K., Mao J., Mohiuddin K.M. Artificial neural networks: a tutorial. IEEE: Computer. 1996;29(3):31–44.

[14] Lippmann R. An introduction to computing with neural nets. IEEE ASSP Magazine. 1987;4(2):4–22.

[15] Floreano F., Mattiussi C. Bio-inspired artificial intelligence, theories, methods and technologies. Cambridge, MA, USA: The MIT Press; 2008.

[16] Gupta M., Jin L., Homma N. Static and dynamic neural networks: from fundamentals to advanced theory. New Jersey, USA: John Wiley Sons; 2003.

[17] Huang D.S. Radial basis probabilistic neural networks: model and applications. International Journal of Pattern Recognition and Artificial Intelligence. 1999;13(7):1083–1101.

[18] Mao K., Tan K., Ser W. Probabilistic neural network structure determination for pattern classification. IEEE Transactions on Neural Networks. 2000;11(4):1009–1016.

[19] Spetch D.F. Probabilistic neural networks for classification, mapping or associative memory. IEEE International Conference on Neural Networks. 1998;1:525–532.

[20] Spetch D.F. Probabilistic neural networks. Neural Networks. 1990;3(1):109–118.

[21] Spetch D.F. Enhancements to probabilistic neural networks. International Joint Conference on Neural Networks. 1992;1:761–768.

[22] Taylor J.G. Mathematical approaches to neural networks. Holland: North-Holland Mathematical library; 1993.

[23] Spetch D.F., Romsdhal H. Experience with adaptive probabilistic neural networks and adaptive general regression neural networks. IEEE International Conference on Neural Networks. 1994;2:1203–1208.

[24] Spetch D.F., Shapiro P. Generalization accuracy of probabilistic neural networks compared with backpropagation networks. IJCNN-91-Seattle International Joint Conference on Neural Networks. 1991;1:887–892.

[25] Attix F.H. Introduction to radiological physics and radiation dosimetry. New Jersey, USA: Wiley-VCH; 2004.

[26] Lilley J. Nuclear physics, principles and applications. New Jersey, USA: John Wiley & Sons, Ltd.; 2001.

[27] Bromley D.A. Detectors in nuclear science. Nuclear Instruments and Methods. 1979;162:431–476.

[28] Bramblett R.L., Ewing R.I., Bonner T.W. A new type of neutron spectrometer. Nuclear Instruments and Methods. 1960;9:1–12.

[29] Thomas D.J. Neutron spectrometry for radiation protection. Radiation Protection Dosimetry. 2004;110(1–4):141–149.

[30] Matzke M. Unfolding procedures. Radiation Protection Dosimetry. 2003;107(1–3):155–174.

[31] Braga C.C., Dias M.S. Application of neural networks for unfolding neutron spectra measured by means of Bonner spheres. Nuclear Instruments and Methods in Physics Research Section A. 2002;476(1–2):252–255.

[32] Kardan M.R., Setayeshi S., Koohi-Fayegh R., Ghiassi-Nejad M. Neutron spectra unfolding in Bonner spheres spectrometry using neural networks. Radiation Protection Dosimetry. 2003;104(1):27–30.

[33] Kardan M.R., Koohi-Fayegh R., Setayeshi S., Ghiassi-Nejad M. Fast neutron spectra determination by threshold activation detectors using neural networks. Radiation Measurements. 2004;38:185–191.

[34] Vega-Carrillo H.R., et al. Neutron spectrometry using artificial neural networks. Radiation Measurements. 2006;41:425–431.

[35] Vega-Carrillo H.R., Martinez Blanco M.R., Hernandez Davila V.M., Ortiz Rodriguez J.M. Ann in spectroscopy and neutron dosimetry. Journal of Radioanalytical and Nuclear Chemistry. 2009;281(3):615–618.

[36] Ortiz-Rodriguez J.M., Martinez-Blanco H.R., Vega-Carrillo H.R. Robust design of artificial neural networks applying the Taguchi methodology and DoE. Proceedings of the Electronics, Robotics and Automotive Mechanics Conference (CERMA'06), IEEE Computer Society. 2006;1:1–6.

[37] IAEA. Compendium of neutron spectra and detector responses for radiation protection purposes. Technical Report 403; Vienna, Austria: International Atomic Energy Agency (IAEA); 2001.