Trakia Journal of Sciences, Vol. 3, Suppl. 3, pp160-163, 2005

Available online at:
http://www.uni-sz.bg

ISSN 1312-1723

RELIABILITY OF MAPPING APPROXIMATION BY NEURAL NETWORKS

N. Petrov

Trakian University – St. Zagora, Yambol, Bulgaria, nikipetrov@lycos.com

ABSTRACT

In this paper is researched reliability mapping approximation with neural networks of the ***risk technical systems*** (RTS). The Reliability Mapping Regresor (RMR) is an incremental self-organizing neural network with adaptive chains among neurons. These chains yield supplementary information to the network. RMR is capable to approximate every function or relation and, simultaneously its inverse function, if it exists, on the inverse relation. It also yields all the solutions (even infinite), their corresponding mapping branches and, if the case, the equilevel surfaces. The basic principle is the transformation of the function approximation problem into a pattern recognition problem under an unsupervised framework. Some examples conclude the paper.

Key Words: Reliability mapping approximation; Neural network

INTRODUCTION

1.1. The Function Approximate Problem

In 1957 Kolmogorov proposed a theorem which states that a continuous multivariate function on a compact set can be expressed in terms of sums and compositions of a finite number of single variable functions, which are continuous monotonically increasing [5]. It has been suggested [2], [3] that this theorem provides theoretical support for the function approximation by neural networks. Rigorous proofs for the universality of feedforward neural networks employing continuous sigmoidal activation functions have been given (see, e.g. [1]). The key result is that a single hidden layer of feedforward neural networks is capable of approximating uniformly any continuous multivariate function to any degree of accuracy. In [4] it was shown that these networks can approximate not only unknown functions, but also their derivatives and that even non differentiable (in the classic sense) functions which possess a generalized derivative. An interesting survey of these theoretical results can be found in [6].

1.2. A Simplified Progressive Learning Network (PLN)

A particular self-organizing neural network is proposed in the following sections. It's basically a simplification of the Progressive Learning Network (PLN), which is derived from the Counterpropagation network [2]. PLN is made up of three layers of processing elements (PE's). The input and output layers, consisting of PE's with linear transfer function, have the task of normalizing and are fully connected to the hidden layer, which is composed of competitive PE's and has no predefined dimensions. The number of its PE's is increased of decreased automatically according to the amount of data acquired progressively by the ANN and to the required mapping accuracy. Its training requires only one epoch.

This PLN variant, called EXIN Segmentation Neural Network (EXIN SNN) only works as an unsupervised network and so doesn't need the weights from the hidden layer to the output layer. The network parameters of the modified unsupervised PLN are: K, dimension of the hidden layer (variable); w_k, weidht vector connecting the input layer to the k-th PE of the hidden layer; s_k, state variable for the neuron k, equal to the number of samples identified by the k-th PE of the hidden layer; ρ_x, input space vigilance threshold (positive scalar). The

number K of PE's of the hidden layer is initially set to zero. The index i stands for the i-th presented sample. The learning is given by the following procedure:

1. Present the first vector of the ***training set*** (TS) x_1, let $K=1$ and assign the vector representing the input as weight of the first PE: $w_1 = x_1 . Let\ i = 2$.

2. Present the i-th sample x_i and compute the K Euclidian distances δ_k between the vector x_i and the weights w_k of the K PE's of the hidden layer.

3. Sort out the δ_k's in increasing order and select the closest (winning) neuron, whose weight is w_w.

4. If $\delta_w > \rho_x$ go to 6.

5. Updata the weights and the state of the w-th PE according to:

$$w_w = \frac{s_w . w_w + x_i}{s_w + 1}, \quad s_w = s_w + 1, \tag{1}$$

and go to 7.

6. Create a new PE setting:

$$K = K + 1, w_k = x_i, s_k = 1. \tag{2}$$

7. Let $i = i + 1$ and go to 2.

After learning (one epoch), the modified PLN has clustered the input data according to the given attribute vectors.

RELIABILITY MAPPING REGRESSOR (RMR)

The Reliability Mapping Regressor (RMR) is an incremental self-organizing competitive neural network with adaptive chains among neurons with diagnosis ***risk technical systems*** (RTS). Both neurons and chains carry information, i.e. the neuron weights and the linking among neurons are used for the computation of the output of the network (production or recall phase). RMR is able to approximate every possible mapping (function or relation) in both senses, i.e. $M(x,y): x \in \Re^m \quad y \in \Re^n$. It's also able to find all solutions to a given input, determine for each solution, he corresponding branch of the mapping and generalize the input to every possible collection of components of x and y and estimate the remaining components. For these features, RMR can be defined as a *global mapper*.

The RMR approach requires the unsupervised learning of the given mapping. In order to do so, for the i-th example (x_i, y_i), build the augmented vector $z_i = \left[x_i^T, y_i^T \right]^T \in \Re^{m+n}$. RMR performs a vector quantization of this augmented subspace. This choice is justified by the fact that in the augmented subspace the mapping branches can be considered as clusters.

After the first phase diagnosis RTS, the RMR architecture is composed of a pool of p neurons. The weight vector for neuron $j\ (j = 1,..., p)$ is $w_j \in \Re^{m+n}$. The overall TS is represented by the matrix $Z \in \Re^{N \times (m+n)}$ where N is the number of input data and $Z = \left[z_1^T, z_2^T, ..., z_N^T \right]^T$. Define the *linking matrix* $V \in \Re^{P \times P}$ whose element $V_{\alpha\beta}$ represents the strength of the connection between neuron α and neuron β. This matrix has zero diagonal and is symmetric. Its initialization is given by the null matrix. At each neuron i is associated with a state variable r_i which is called the domain state variable and represents, as it will be clear soon, the "radius" of the domain of the neuron in the weight space. Define the vector $r = \left[r_1, ..., r_p \right]^T$ as the ***domain vector***. This vector is initialized by setting its components all equal to a predefined constant $\rho_{\min}$ which is equal to $k_{\min} . \rho_2$ with $k_{\min}$ less than one. For each input z_i, the following operations are performed, mainly in order to determine a neuron to be linked to the winning neuron.

Compute all the Euclidian distances between the neuron weights and the input. Classify them according to the increasing distance and label the neurons according to their position. Hence, w_1 represents the weight vector of the nearest neuron. Define $d_j = s_j - w_1$ for $j = 1,..,k$ where $s_j = w_j$ for $j > 1$, $s_1 = z_i$ and k is given by the greatest index such that:

(3) $\|z_i - w_k\|_2 \le \delta.\|d_1\|_2$,

being δ is predefined constant greater than. Hence, the d_j's represents, in the weight space, with regard to the winning neuron, the position vector of the input of the non winning neurons situated inside a hyper sphere centered in the input and of radius proportional to the distance from the input to the winning neuron. Test (3) eliminates neurons which are too far from the winning neuron and so are not allowed to be linked to the winning neuron.

Compute, $\forall j = 2,...,k$,

(4) $p_j = d_1.d_j / \|d_1\|_2.\|d_j\|_2$,

and the integer h such that:

(5) $p_h = \max_j p_j, \max_j |p_j|$,

if $\exists p_j > 0$ for at least one value of j otherwise.

The neuron whose weight has index h, viz. w_h, is connected to the winning neuron. Equation (5) means that the selected unit is the neuron j whose relative weight vector, viz. d_j, is the nearest to the straight line parallel to d_1 (*linking direction*), among all neurons whose relative weights are situated in the half space bordered by the hyper plane containing the wining neuron weight vector and normal to the linking direction.

Once the previous steps are completed, RMR is able to yield the required solution. The input can be generalized to vector of dimension not necessarily equal to m, whose components belong either to the input space or to the output space. For sake of simplicity, call the general input x as before and consider it as m - dimensional. Define the subspace of the input as X. Associate to the neuron i, for $i = 1,..., p$, two state variables: l_i which is an integer representing the *level* of the neuron and b_i representing the *branch* of the mapping containing the neuron. This phase is composed of the following steps:

1. Feed x to RMR.

2. Compute the norm of the projections on the space $X \in \Re^m$ of the vectors from x to all RMR weights. Classify these *retricted* distances and the corresponding neurons in increasing order (o_i is the position of the neuron i in the classification).

3. Define the winning neuron as w, i.e. $o_w = 1$. Consider the associated component of r, viz. r_w. Cheek equation:

(6) $\|x - w_w\| < r_w$,

that is if the input is in the domain of the winning neuron.

4. For each neuron j linked (by matrix V) to neuron w, chek the level. If neuron j is level zero, viz. $l_j = 0$, then $l_j = 2$ and $b_j = w$ (neuron j belongs to the branch of the winner). Resuming, the connected level zero neuron j becomes a *level two* neuron belonging to the winner branch.

5. As anticipated in the previous step, consider now as winner w the neuron i such that $o_i = 2$ and the level one test is satisfied. If these conditions are true, set l_w to one (even if w was already labeled as level two) and b_w to w. Then repeat the previous step for the labeling of the branches and the determination of the level two neurons. Once this labeling is finished, this step is repeated for increasing values of o_i until the neuron i has the input out of its domain.

CONCLUSIONS

A novel neural network, which is incremental self-organizing and uses adaptive chains (linking) among neurons has been presented. It transforms the function approximation problem into a pattern recognition problem. RMR is an universal approximator for *risk technical systems* (RTS) which is also able to yield infinite solutions by using the links. The originality and force of RMR originates from its open architecture, in the sense that different neural modules can be used for the same task. Indeed, RMR is above all, a strategy which exploits neural features. In this RTS is justified by the use of a *neural grammar*. This grammar has some fundamental ingredients which are the basic prototypes of the neural networks (neural modules) and a syntax given by the way these modules interchange their information. An open neural architecture proposes a novel strategy

based on this grammar.

REFERENCES

1. Comon P. Classification Supervise Par Reseaux Multicouches. Traitement Du Signal, 1991.
2. Hecht-Nielsen R. Neurocomputing. Addison-Wesley. 1990.
3. Hornik K., M. Stinchkombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. 1989.
4. Hornik K., M. Stinchkombe, and H. White. Universal Approximation of an Unknown Mapping and its Derivatives using Multilayer Feedforward Networks. Neural Networks, 1990.
5. Kolmogorov A.N. On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition. Doklady Akademii Nauk USSR. 1957.
6. Sprecher D.A. A Universal Mapping for Kolmogorov's superposition Theorem. Neural Networks. 1993.