Interval Basis Neural Networks

A. Horzyk[1]

[1]Department of Automatics, University of Science and Technology, Poland
E-mail: horzyk@agh.edu.pl

Abstract

The paper introduces a new type of ontogenic neural networks called Interval Basis Neural Networks (IBNNs). The IBNN configures the whole topology and computes all weights after *a priori* knowledge collected form training data. The training patterns are grouped together producing intervals separately for all input features for each class after statistical analyses of the training data. This IBNNs feature make possible to computed all network parameters without training. Moreover, the IBNN takes into account the distances between patterns of the same classes and builds the well-approximating model especially on the borders between the classes. Furthermore, the IBNNs are insensitive for quantity differences in patterns representation of classes. The IBNNs always correctly classify training data and very good generalize other data.

1 Introduction

Though neural networks are very modern computational tool they are not always easy to use because of many unknown configuration parameters. Many ontogenic neural networks [1,3-8] can help to solve problems of architecture construction but many times their generalization is poor. The generalization is a fundamental problem of ontogenic neural networks use.
This paper introduces a new type of ontogenic neural networks – called Interval Basis Neural Networks (IBNNs). The IBNNs are non-linear three-layers partially connected ontogenic neural networks constructed after the statistical analyses of the training data. The construction of IBNNs is deterministic. Training data are analyzed separately for each input feature searching for intervals characteristic for each class. Such separable intervals are completed with non-linear concave functions with controlled shape of descend that make possible to achieve good approximation especially on the borders between different classes. The shape of these concave functions (fig. 2) are computed after the quantity and distances of other training data of the same class from the hyperregions and sub-hyperregions borders. The hyperregions and sub-hyperregions are produced after the intervals computed for all training patterns of the same class.

The adaptation of RBF networks by Lowe [6] also enables to change the shape of radial basis functions. Lowe describes the importance of the adaptation of RBF function in view of generalization. The shape of slopes can be regulated also for bicentral functions [2].

2 IBNN Construction Process

For any given training data $U = \{(u^1, C^{m_1}), ..., (u^N, C^{m_N})\}$ consisting of the pairs: the input vector $u^n = [u_1^n, ..., u_K^n]$ ($u_k^n \in \Re$), the adequate class $C^m \in \{C^1, ..., C^M\}$ of this input vector $u^n \in C^{m_n}$ and input features *1, ..., K*, the intervals (for each feature) of patterns of the same class can be computed (fig. 1). Different intervals are characteristic for different groups of patterns of each class and can partially identify them. The model of each class can be defined by combining such intervals for all input features into hyperregions and sub-hyperregions in the input space. In order to find out such intervals effectively training data have to be sorted after each input feature.

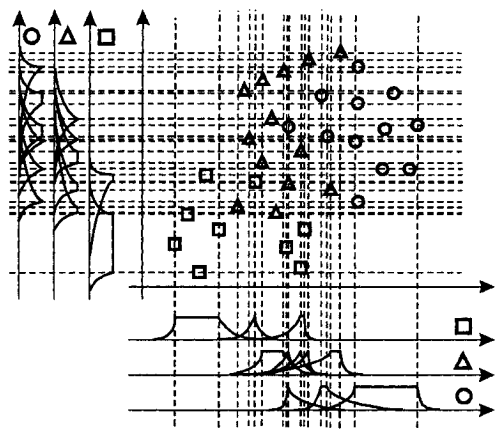

Fig. 1. Extraction of intervals and computation of transfer function for these intervals in view of hyperregions.

In order to generalize correctly especially outside the computed intervals the concave slopes (fig. 1.-2.) are added at the borders of such intervals. These concave slopes are functions that project the quantity and the distances of other input patterns of the same class from

the considered intervals. The training patterns are classified always as 100% similar to their classes. Each interval defines a single neuron in the 1-st layer of the constructed neural network (fig. 3.). Such a neuron is connected to the input related with feature of its interval. The transfer function f_T of the 1-st layer neurons is defined as follows:

$$f_T(x) = \begin{cases} 1 - tgh\dfrac{x_k - x_k^R}{1+\alpha_k^R} & if\ x_k > x_k^R \\ 1 & if\ x_k^L \le x_k \le x_k^R \\ 1 - tgh\dfrac{x_k^L - x_k}{1+\alpha_k^L} & if\ x_k < x_k^L \end{cases} \quad (1)$$

where

x_L - is a left limit of the concerned interval,

x_R - is a right limit of the concerned interval,

α_L - is a left slope parameter of the transfer function,

α_R - is a right slope parameter of the transfer function,

The slope parameters of the transfer function (fig. 2) are defined as follows:

$$\alpha_k^L = q^m \sum_{i \in L} \frac{1}{1 + \left(x_k^L - u_k^i\right)^2} \quad (2)$$

$$\alpha_k^R = q^m \sum_{i \in R} \frac{1}{1 + \left(x_k^R - u_k^i\right)^2} \quad (3)$$

where

$$L = \left\{u^n : u^n \in U \cap C^m \ \&\ u_k^n < x_k^L\right\} \quad (4)$$

$$R = \left\{u^n : u^n \in U \cap C^m \ \&\ u_k^n > x_k^R\right\} \quad (5)$$

$$q^m = \frac{N}{K \cdot N_m} \quad (6)$$

N – is a number of training data,

N_m – is a number of training data that define class C^m,

K – is a number of classes defined in the training data.

The slope parameters (2) and (3) define the slopes of the transfer function (1). They depend on other training patterns of the same class. The more training patterns in the left (L) or in the right (R) direction of the interval are the less sharp the suitable slope is. This feature of the IBNN is very important in view of generalization. The specifically computed slopes of transfer functions contain information about other training patterns that define the same classes. Such intervals of each class define models of classes for each feature separately.

Coefficient q^m makes the computations of slope parameters (fig. 2) insensitive for differences in patterns quantity representating classes, for quantity of training data and for quantity of classes defined in the training data.

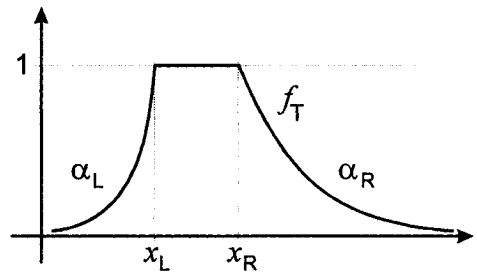

Fig. 2. The transfer function for the 1-th layer neurons.

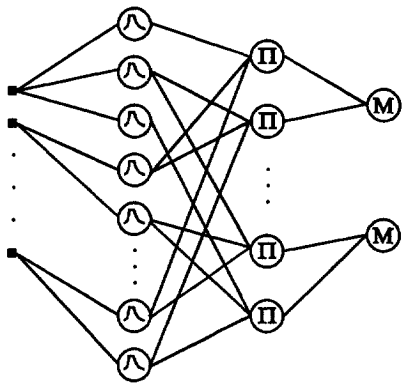

Fig. 3. The 3-layer topology of the IBNN for hyperregions.

Each neuron of 1-st layer is always connected to a single input corresponding with a certain input feature. The weights of 1-st layer neurons are always equal 1. The 2-nd layer contains neurons (fig. 3) that compute (limited – for sub-hyperregions) products of some combinations of output values of the 1-st layer neurons producing hyperregions or sub-hyperregions. Only these combinations given by training data that are related to intervals of different existing input features are transformed to connections. Each 2-nd layer neuron represents a group of training patterns of the same class. Such neurons can correctly approximate the values of classification inside the intervals (hyperregions or sub-hyperregions) and outside them after the specific information came from the computed value of slope parameters. Moreover, the advanced analyses of the sorted input data for all features lead to find out the separate sub-hyperregions for each class. The sub-hyperregions in data subspace can not include patterns of other classes. The sub-hyperregions are easy to find because they are convex. The sub-hyperregions are finally grouped together creating different sub-

hyperfigures as shown in the figures 4-5. Theoretically, there could exist many possible sub-hyperfigures in the hyperspace that consist of the different sub-hyperregions covering the training data in different ways as shown in the figure 6. Each training pattern has to be closed in at least one sub-hyperregion. The smallest sub-hyperregions can consist of single point in the input space representing single pattern. In order to avoid the problem of ambiguity of sub-hyperfigures there are used two criteria:

1. Criterion of the density of the patterns of the same class: the longer distances between patterns the lower density in the space and the lower probability to create the sub-hyperregion.
2. Criterion of maximum quantity of training samples closed in sub-hyperregions taking into account the quantity of patterns representing different classes. The sub-hyperregions for different classes are compared regarding the quantity of patterns they contain divided by the quantity of patterns representing appropriate classes they represent in order to make the IBNN insensitive for differences in quantity representation of different classes.

The described criteria are used to define the k-th sub-hyperregion coefficient computed as follows:

$$RC_m^k = \frac{Q^k}{V \cdot N_m} \tag{7}$$

where

Q – quantity of patterns included in the given sub-hyperregion,

V – volume of the sub-hyperregion.

where k-th sub-hyperregion consists of patterns of m-th class.

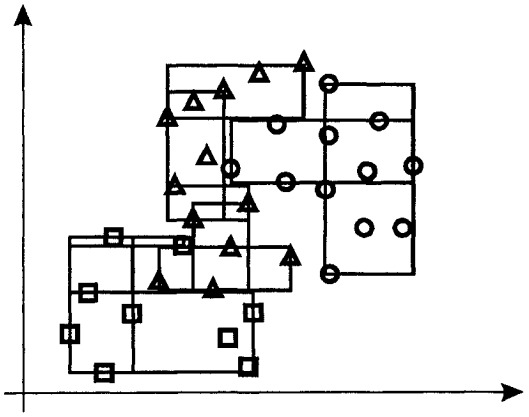

Fig. 4. Producing multiple sub-hyperregions after the intervals.

In order to IBNN classify properly using concave functions at boarders of sub-hyperregions forming sub-hyperfigures representing classes two important conditions have to be true:

1. The sub-hyperregions for the patterns of the same class should overlap if only possible.
2. The sub-hyperregions for the patterns of different classes have to be always separate.

Finally, the overlapped sub-hyperregions of different classes are compared together using coefficients computed for them after the described criteria in order to choose this one which has the maximal value of this coefficient. Other overlapping sub-hyperregions should be omitted or cut off by the winning sub-hyperregion.

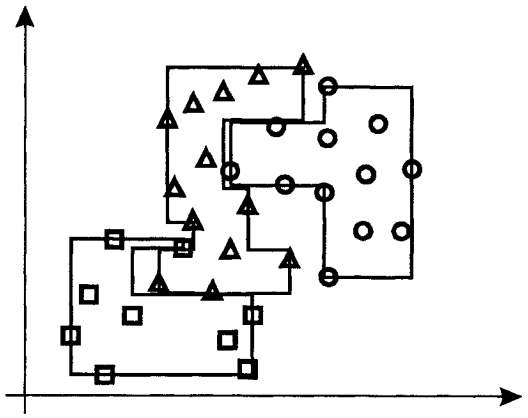

Fig. 5. Connected sub-hyperregions producing sub-hyperfigures for the presented classes.

The transfer function of the 2-nd layer neurons is defined as follows:

$$f_{\Pi}(y) = \prod_{k=1}^{K} y_k \tag{8}$$

The weights of 2-nd layer neurons are always set to 1.

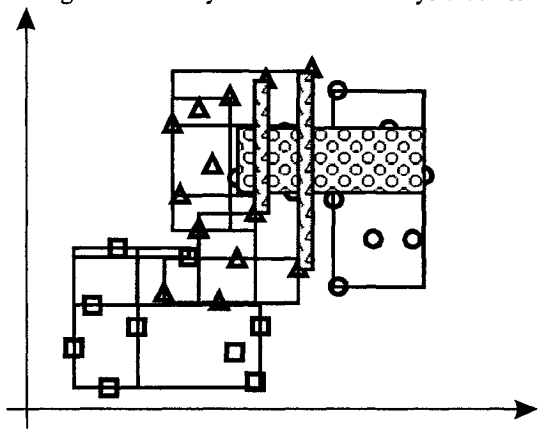

Fig. 6. The use of the criterion (7) prevent the possible collisions of sub-hyperregions.

The 3-rd layer neurons choose the maximum value of the 2-nd layer neurons of the same class. The output values

of the 3-rd layer neurons measure the similarities to the classes defined in the training data set. The transfer function of the 3-rd layer neurons is defined as follows:

$$f_M^m(z) = \max_{j \in J} \{z_j^i : z_j^i \in C^m\} \quad (9)$$

The result classification is defined by the function:

$$f_C(s) = \arg \max_{m=1,\ldots,M} \{s_m : s_m = f_M^m(z)\} \quad (10)$$

This function can be optionally transformed to the neuron of 4-th layer of IBNN. If there is no maximum value for 3-rd layer neurons the output can be defined as zero. The zero output suggest that the network can not univocally qualify the input vector to any of the trained classes.

3 Comparison to other methods

The IBNNs are easy to use ontogenic neural networks. They don't need to be trained because all network parameters are computed in the configuration process automatically. They work similarly to RBF networks (KNN), bicentral based networks and PNN [1, 4, 10, 15] creating a specific hyperregions in the input data space. In comparison to well-known RBF networks which are partially configured and partially trained the IBNNs are only configured after *a priori* knowledge about training data. Only some statistical analyses are needed to find out specific intervals for each class and each feature. For big training data sets computation of IBNN could be sometimes time consuming because the training data have to be sorted for each input feature in order to separate intervals. Intervals as well as RBF neurons group together input data of the same class. In MLP networks the hard-limiter neuron the boundary is a hyperplane, in RBF networks the boundary is the circumference of a hypervolume (hypersphere with Euclidean distance) centered around class samples, while in IBNNs the boundary is hyperregion outspreaded on some subset of class samples. The hyperregions are completed with specific slope functions that are necessary to approximate and generalize outside the IBNN hyperregions and sub-hyperregions. The IBNN is also a good alternative for automatically configured ontogenic SONN [4-5] because SONNs demand binary data for configuration of the networks and computation of weights parameters. IBNN as well as SONN are deterministic and fully automatic. They do not demand any configuration parameters.

4 Conclusions

The described IBNNs can be used as an alternative method to other kernel neural networks (KNN). The deterministic IBNNs configuration process is based on some specific statistical analyses of the training data. The introduced method finds out intervals specific for each training class and constructs specific sub-hyperregions for them. Each hyperregion and sub-hyperregion define the data of one of the trained class and can be used to approximate and generalize data in-between the boundaries of a considered hyperregion. There are defined new specific slope functions that help to generalize outside the hyperregions. The training data are always correctly classified. Moreover, the IBNNs are not sensitive for differences in patterns quantity representation of classes, for quantity of classes. While the IBNNs need to sort training data for each input feature the computations can be time-consuming for big training data sets. On the other hand, big data sets can be better optimized and the IBNN model can be better created.

Acknowledgements: Support from research funds of Polish Committee for Scientific Research is gratefully acknowledged.

References

[1] Duch, W., Korbicz, J., Rutkowski, L., Tadeusiewicz, R. (eds) (2000) Biocybernetics and biomedical engineering, Neural networks, vol. 6, EXIT, Warsaw.

[2] Duch, W., Jankowski N. (1997) New neural transfer functions, Journal of Applied Mathematics and Computer Science, 7(3): 639-658.

[3] Fiesler, E., Beale, R. (eds) (1997) Handbook of neural computation, IOP Publishing Ltd and Oxford University Press, Bristol & New York.

[4] Horzyk, A., Tadeusiewicz, R., (2004) Self-Optimizing Neural Networks, Advances in Neural Networks – ISNN 2004, Proc. of International Symposium on Neural Networks, Dalian, China, Springer-Verlag, Berlin - Heidelberg, pp. 150-155.

[5] Horzyk, A. (2003) Introduction to Self-Optimising Neural Networks for Classification Tasks, Advanced Computer Systems, Sołdek J., Drobiazbiewicz L. (eds), Szczecin, INFORMA, pp.127-135.

[6] Lowe, D. (1989) Adaptive radial basis function nonlinearities and the problem of generalization, 1st IEEE International Conference on Artificial Neural Networks, London, UK, pp. 171-175.

[7] Specht, D. F. (1990) Probabilistic neural networks. Neural Networks, 3: 109-118.

[8] Tadeusiewicz, R., Mikrut, Z., (1998) Neural-Based Object Recognition Support - From Classical Preprocessing to Space-Variant Sensing. Invited paper. In M. Heiss (ed.): Proceedings of the International ICSC/IFAC Symposium on Neural Computation NC '98, Vienna University of Technology, ICSC Academic Press, Canada/Switzerland, pp. 463-468.