Optimal transfer function neural networks

Norbert Jankowski and Włodzisław Duch

Department of Computer Methods Nicholas Copernicus University
ul. Grudziądzka 5, 87–100 Toruń, Poland, e-mail:{norbert,duch}@phys.uni.torun.pl
http://www.phys.uni.torun.pl/kmk

Abstract. Neural networks use neurons of the same type in each layer but such architecture cannot lead to data models of optimal complexity and accuracy. Networks with architectures (number of neurons, connections and **type of neurons**) optimized for a given problem are described here. Each neuron may implement transfer function of different type. Complexity of such networks is controlled by statistical criteria and by adding penalty terms to the error function. Results of numerical experiments on artificial data are reported.

1 Introduction

Artificial neural networks approximate unknown mappings F^* between pairs $\langle \mathbf{x}_i, y_i \rangle$, for $i = 1, \ldots, n$ for set of observations S. For this set $F(\mathbf{x}_i) = y_i$, where $F(\cdot)$ is an output of a neural network (in general a vector). The performance of the trained network depends on the learning algorithm, number of layers, neurons, connections and on the type of transfer functions computed by each neuron. To avoid over- and underfitting of the data the *bias–variance* [1] should be balanced by matching the complexity of the network to the complexity of the data [6, 3].

Complexity of the model may be controlled by Bayesian regularization methods [8, 1, 9], using ontogenic networks that grow and/or shrink [4, 7] and judicious choice of the transfer functions [3, 5]. All these methods are used in optimal transfer function neural networks (OTF-NN) described in the next section. Some experiments on artificial data are presented in the third section and a few conclusions are given at the end of this paper.

2 Optimal transfer function neural networks

Accuracy of MLP and RBF networks on the same data may significantly differ [3]. Some dataset are approximated in a better way by combinations of sigmoidal function $\sigma(I) = \frac{1}{1+e^{-I}}$, where the activation $I(\mathbf{x};\mathbf{w}) = w^t x + w_0$, while other datasets are represented in an easier way using gaussian functions $G(D,b) = e^{-D^2/b^2}$ with distance

Support by the Polish Committee for Scientific Research, grant 8 T11C 006 19, is gratefully acknowledged.

function $D(\mathbf{x};\mathbf{t}) = \left[\sum_{i=1}^{d}(x_i - t_i)^2\right]^{\frac{1}{2}}$. More flexible transfer functions may solve this problem. In [3] bicentral transfer functions that use $3N$ parameters per neuron were described (gaussian functions use $2N$ or $N+1$, and sigmoidal function use $N+1$ parameters). Here a constructive network optimizing the type of transfer functions used in each node is described. The OTF neural model is defined by:

$$F(\mathbf{x}) = o\left(\sum_i w_i h_i[A_i(\mathbf{x};\mathbf{p}_i)]\right) \tag{1}$$

where $h_i(A_i(\cdot)) \in H$ (H is the set of basis functions) is transfer function ($h_i(\cdot)$ is the output function, $A_i(\cdot)$ is activation function), and p_i is the vector of adaptive parameters for neuron i. An identity or a sigmoidal function is used for $o(\cdot)$ output function of whole network. Sigmoidal outputs are useful for estimation of probabilities but may significantly slow down the training process.

The network defined by Eq. 1 may use arbitrary transfer function $h_i(\cdot)$. In the next section gaussian output function with scalar product activation $G^I(\mathbf{x},\mathbf{w}) = e^{\mathbf{x}^T\mathbf{w}}$ is used together with gaussian and sigmoidal transfer functions in one network. The gradient descend algorithm was used to adapt the parameters. Network architecture may be controlled during learning by the criteria proposed below.

Pruning: in the first version of MBFN the weight elimination method proposed by Weigend (at. al.) [9] is used:

$$E_{we}(F,\mathbf{w}) = E_0(F) + \lambda\sum_{i=1}^{M}\frac{w_i^2/w_0^2}{1+w_i^2/w_0^2} = \sum_{i=1}^{M}[F(\mathbf{x}_i) - y_i]^2 + \lambda\sum_{i=1}^{M}\frac{w_i^2/w_0^2}{1+w_i^2/w_0^2} \tag{2}$$

where w_0 factor is usually around 1, and λ is either a constant or is controlled by the learning algorithm described in [9]. M is the number of neurons.

Statistical pruning is based on a statistical criterion leading to inequality P

$$P:\ \frac{L}{Var[F(\mathbf{x};\mathbf{p}_n)]} < \chi^2_{1,\vartheta} \qquad\qquad L = \min_i \frac{w_i^2}{\sigma_{w_i}} \tag{3}$$

where $\chi^2_{n,\vartheta}$ is ϑ% confidence on χ^2 distribution for one degree of freedom, and σ_{w_i} denotes the variance of w_i. Neurons are pruned if the saliency L is too small and/or the uncertainty of the network output R_y is too big.

Variance σ_{w_i} may be computed iteratively:

$$\sigma^n_{w_i} = \frac{N-1}{N}\sigma^{n-1}_{w_i} + \frac{1}{N}\left[\Delta w_i{}^n - \overline{\Delta w_i{}^n}\right]^2 \tag{4}$$

$$\overline{\Delta w_i{}^n} = \frac{N-1}{N}\overline{\Delta w_i{}^{n-1}} + \frac{1}{N}\Delta w_i^n \tag{5}$$

where n defines iteration, and $\Delta w_i^n = w_i^n - w_i^{n-1}$. N defines the *tail* length.

A criterion for network growth is based on a hypothesis for the statistical inference of model sufficiency, defined as follows [6]:

$$H_0:\quad \frac{e^2}{Var[F(\mathbf{x};\mathbf{p}) + \eta]} < \chi^2_{M,\theta} \tag{6}$$

where $\chi^2_{n,\theta}$ is θ% confidence on χ^2 distribution for n degree of freedom, $e = y - f(\mathbf{x};\mathbf{p})$ is the error and $\eta]$ is variance of data. The variance is computed one time per epoch using the formula:

$$Var[F(\mathbf{x};\mathbf{p}_n)] = \frac{1}{N-1}\sum_i \left[\Delta F(\mathbf{x}_i;\mathbf{p}_n) - \overline{F(\mathbf{x}_j;\mathbf{p}_n)}\right]^2 \tag{7}$$

or an iterative formula:

$$Var[F(\mathbf{x};\mathbf{p}_n)] = \frac{N-1}{N} Var[F(\mathbf{x};\mathbf{p}_{n-1})] \; + \; \frac{1}{N}\left[\Delta F(\mathbf{x}_i;\mathbf{p}_n) - \overline{F(\mathbf{x}_j;\mathbf{p}_n)}\right]^2 \tag{8}$$

where $\Delta F(\mathbf{x}_i;\mathbf{p}_n) = F(\mathbf{x}_i;\mathbf{p}_n) - F(\mathbf{x}_i;\mathbf{p}_{n-1})$. N, as before, defines the *tail* length.

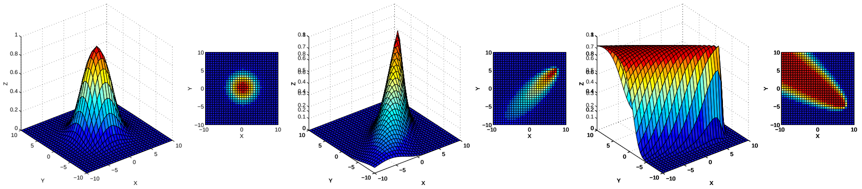


Figure 1: Extended conic functions

OTF-NN version II In this version of the OTF-NN model an extension of conic activation functions (Fig. 1) introduced by Dorffner [2] is used:

$$A_C(\mathbf{x};\mathbf{w},\mathbf{t},\mathbf{b},\alpha,\beta) = -[\alpha I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \beta D(\mathbf{x},\mathbf{t},\mathbf{b})] \tag{9}$$

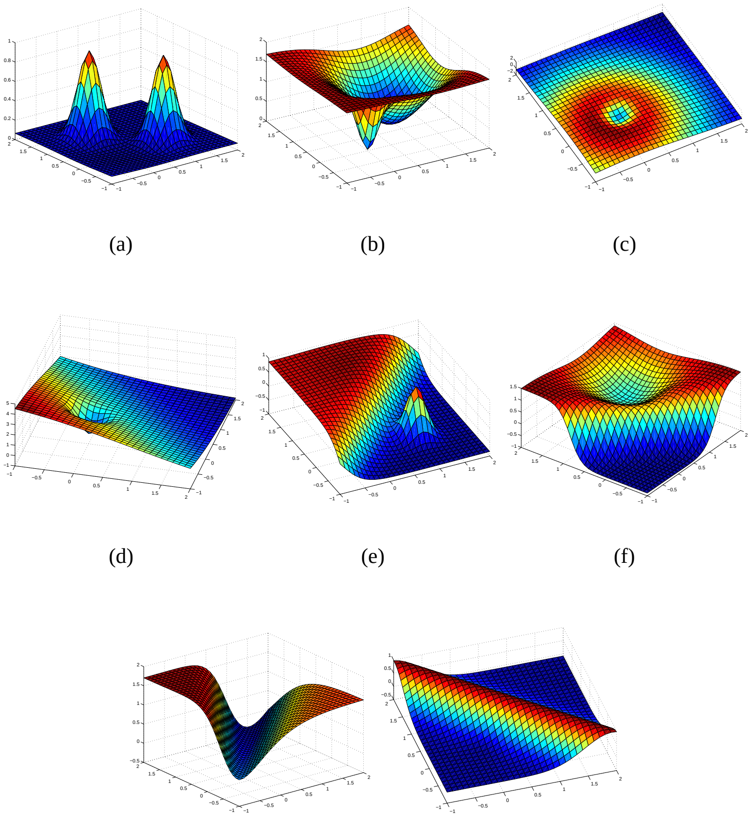
The output function is sigmoidal. Such functions change smoothly from gaussianto sigmoidal. A new penalty term is added to the error function:

$$E_{we}(F,\mathbf{w}) = E_0(F) + \lambda \sum_{i=1}^{M} \left[\frac{\alpha_i^2/\alpha_0^2}{1+\alpha_i^2/\alpha_0^2} \cdot \frac{\beta_i^2/\beta_0^2}{1+\beta_i^2/\beta_0^2}\right] \tag{10}$$

allowing the learning algorithm to simplify the conic activation leaving sigmoidal or gaussian function.

3 Results

XOR XOR is the most famous test problem. How do different OTF solutions look like? OTF-NN network with 4 nodes, 2 of sigmoidal and 2 of gaussian character has been initialized with random values (between -0.5 and 0.5) for weights and centers. After some learning period the λ parameter of 2 has been increased to obtain simple structure of the network. Weight elimination has been especially effective for weights between hidden and output layers.



(a) (b) (c)

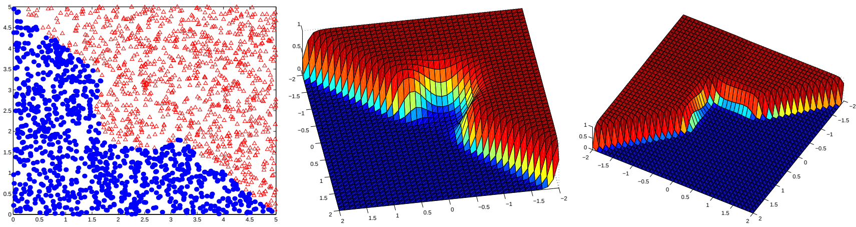
(d) (e) (f)

(g) (h)

Figure 2: Various solutions for the XOR problem

Using such network the training process (taking 2 000 – 10 000 iterations) may finish with different correct solutions. Subfigures a)–b) of Fig. 2 present solutions found by OTF network. Some of them use combinations of gaussian functions (a), b) and c)), other combinations of sigmoidal and guassian functions; a combination of two sigmoidal functions is very hard to find if any gaussian nodes are present. Subfigure h) in Fig. 2 presents the simplest solution using a single neuron (!) in the hidden layer, constructed from gaussian output function with inner product activation function. Each network which had just one such neuron removes all others as spurious.

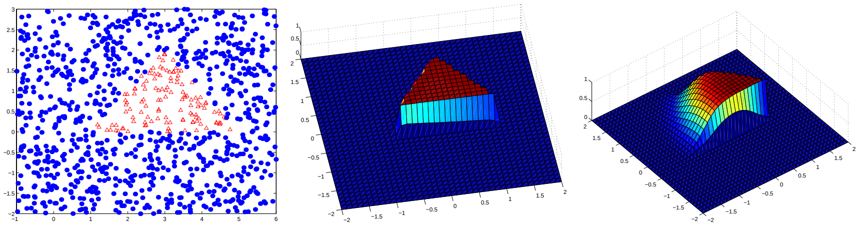
Half-sphere + half-subspace. The 2000 data points were created as shown in Fig. 3. The initial OTF network has 3 gaussian and 3 sigmoidal neurons. The simplest and optimal solution consists of one gaussian and one sigmoidal node (Fig. 3b), although 3 sigmoids give also an acceptable solution, Fig. **??**c. The number of learning



(a) (b) (c)

Figure 3: Half-sphere + half-subspace

epochs was 500 and the final accuracy was around 97.5–99%. Similar test made in 10-dimensional input space gave 97.5–98% correct answers. The final networks had 2 or 3 neurons, depending on the pruning strength.



(a) (b) (c)

Figure 4: Triangle

Triangle. 1000 points were generated as shown in Fig. 4. The OTF-NN started with 3 gaussian and 3 sigmoidal neurons. The optimal solution for this problem is obtained by 3 sigmoidal functions. The problem is hard because gaussians quickly cover the inner part of the triangle (Fig. **??**c), nevertheless our network has found the optimal solution, Fig. **??**b. The problem cannot be solved with identity output function, sigmoidal output function must be used. The number of the learning epoch was 250 and the final accuracy between 98–99%.

4 Conclusions

First experiments with Optimal Transfer Function networks were presented here. Pruning techniques based on statistical criteria allow to optimize not only the parameters but also the type of functions used by the network. Results on artificial data are very encouraging. Trained OTF networks select appropriate functions for a given problem creating architectures that are well-matched for a given data. Small networks may not only be more accurate but also should allow to analyze and understand the structure of the data in a better way. OTF networks will now be tested on real data.

References

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] G. Dorffner. A unified framework for MLPs and RBFNs: Introducing conic section function networks. *Cybernetics and Systems*, 25(4):511–554, 1994.

[3] W. Duch and N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–212, 1999.

[4] E. Fiesler. Comparative bibliography of ontogenic neural networks. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 793–796, 1994.

[5] N. Jankowski. Approximation with RBF-type neural networks using flexible local and semi-local transfer functions. In *4th Conference on Neural Networks and Their Applications*, pages 77–82, Zakopane, Poland, May 1999.

[6] N. Jankowski and V. Kadirkamanathan. Statistical control of RBF-like networks for classification. In *7th International Conference on Artificial Neural Networks*, pages 385–390, Lausanne, Switzerland, October 1997. Springer-Verlag.

[7] N. Jankowski and V. Kadirkamanathan. Statistical control of growing and pruning in RBF-like neural networks. In *Third Conference on Neural Networks and Their Applications*, pages 663–670, Kule, Poland, October 1997.

[8] T. Poggio and F. Girosi. Network for approximation and learning. *Proceedings of the IEEE*, 78:1481–1497, 1990.

[9] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight elimination with application to forecasting. In R. P. Lipmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882, San Mateo, CA, 1991. Morgan Kaufmann.