An Algorithm of Incremental Construction of Nonlinear Parametric Approximators

Jarosław Arabas and Andrzej Dydyński

Warsaw University of Technology, Institute of Electronic Systems, Warsaw, Poland,
email: {jarabas, a.dydynski}@elka.pw.edu.pl

Abstract. A novel algorithm of incremental construction of nonlinear parametric approximators is introduced that needs a relatively small number of parameters to achieve good accuracy. The approximator is defined as a linear combination of nonlinear base functions, and the method adds the base functions one by one, observing the correlation between the residue function and the candidate for the base function. The base functions do not need to be homogenous, i.e. to share the same formula. The method was experimentally verified and compared to neural networks, and the preliminary results were encouraging enough to develop the presented approach further.

1 Introduction

Nonlinear approximation is one of the fundamental tasks of many practical applications, such as forecasting, data compression, control, etc. There is a continuous interest in development of nonlinear models, such as polynomials, wavelets, fuzzy systems, neural networks and many others. The aforementioned approaches rise from a concept of modeling an unknown nonlinear function to be approximated as a linear combination of certain nonlinear functions (called base functions) of inputs. If the number of the base functions is fixed, and each base function comes from a certain class and is completely defined by a fixed number of parameters, then the whole model can be represented as a vector of parameters. The approximation task is formulated as minimization of error in the space of parameter vectors, therefore, models of such type are called *parametric*. The error takes in general many local minima in the space of parameter vectors, so global optimization methods are usually used to solve the nonlinear approximation task. In practice it is impossible to measure the error exactly, so its value is estimated using sets of samples. The minimization is performed with the error estimated from the training set, and the model accuracy is tested with the use of the test set; both sets have usually no common elements.

When using a parametric model, one has to choose the number of the base functions, i.e. the number of parameters to be optimized. It is well known that increasing the number of parameters results in general in getting lower error measure on the training set. With the test sets the dependence is nonmonotonic, the error first decreases, and after the number of parameters exceeds a certain threshold value, the error again grows up. When designing the parametric model, it is desirable to set the number of the base

functions near the threshold value; unfortunately, this value cannot be apriori predicted in general.

In this paper we present an approach to overcome this difficulty. We introduce a model that is built incrementally. We present an iterative algorithm that introduces a single base function in each iteration basing on nonlinear correlation measures between the base function and the approximation residue. We focus on models with the monotonic base functions, but contrary to the popular parametric models, the base functions can come from different classes, like polynomials or hyperbolic tangent.

One of the first incremental approximation algorithms was introduced by Ivakhnenko [1]. The presented algorithm of *group method of data handling* (GMDH) allows a complex decision hypersurface to be approximated by a set of polynomials. Thresholds are employed at each layer in the network to identify those polynomials which best fit into the desired hypersurface. Only the best combinations of inputs are allowed to pass to succeeding layers, where more complex combinations are formed.

In the field of neural networks, probably the most popular algorithm for incremental construction of a network is *cascade correlation* [4]. The approximator is a "cascade" of functions: each function observes the approximator input and the output from the previous function in a cascade. The functions are hyperbolic tangent.

Another approach is the projection of all training samples into one-dimensional space, then the separation of a hyperplane so that the relevant class is separated. Then the same operation for all other classes until complete separation is achieved. Minimization of the hyperplanes number is done by methods based on Boolean algebra. Representative for this approach are Mezard-Nadal algorithm [5], Marchand algorithm [5] or Li-Tufts method [6]. The aforementioned methods are unfortunately inefficient when the input vectors are large and they are not yet a alternative for the neural network reduction method like OBD [5].

2 Parametric approximation model

2.1 Model definition

Consider a function $f : R^n \to R$ to be approximated using a model $g : R^n \times R^N \times R^{m+1} \to R$ of a form

$$g(\mathbf{x}, \mathbf{p}, \mathbf{w}) = \sum_{i=0}^{m} w_i f_i(\mathbf{x}, \mathbf{p}_i) \tag{1}$$

where $\mathbf{p}$ and $\mathbf{w}$ are parameter vectors of size N and $m+1$, respectively, and $f_i(\mathbf{x}, \mathbf{p}_i)$ is the base function dependent on N_i parameters, i.e. $f_i : R^n \times R^{N_i} \to R$. To simplify the notation, we assume that there exists a function $f_0 : R^n \to 1$ with $N_0 = 0$ parameters. The vector $\mathbf{p}$ is obtained by putting the vectors $\mathbf{p}_i$ one after another, so $N = \sum_{i=1}^{m} N_i$.

Consider the residue function $r : R^n \times R^N \times R^{m+1} \to R$ defined as

$$r(\mathbf{x}, \mathbf{p}, \mathbf{w}) = g(\mathbf{x}, \mathbf{p}, \mathbf{w}) - f(\mathbf{x}) \tag{2}$$

We can define a function $e : R^N \times R^{m+1} \to R$ being the error measure. In this paper we concentrate on the squared error

$$e(\mathbf{p}, \mathbf{w}) = \int_D [r(\mathbf{x}, \mathbf{p}, \mathbf{w})]^2 d\mathbf{x} \tag{3}$$

where D is a set of $\mathbf{x}$ values. Typically D can be a box, i.e. for each i there exist l_i, u_i values such that for each $\mathbf{x} \in D$ we get $l_i \leq x_i < u_i$. In practice it is impossible to compute the error (3) since this would require the computation of the integral value and this is possible only for functions with a known analytical form. Instead, the error (3) is estimated using a set of samples — pairs $\{\mathbf{x}, f(\mathbf{x})\}$, so the error measure is

$$e_A(\mathbf{p}, \mathbf{w}) = \sum_{\{\mathbf{x}, f(\mathbf{x})\} \in A} [g(\mathbf{x}, \mathbf{p}, \mathbf{w}) - f(\mathbf{x})]^2 \tag{4}$$

where A is the set of samples.

In this paper we make another simplification of the model (1) — we assume that each base function f_i takes a form

$$f_i(\mathbf{x}, \mathbf{p}_i) = h_i\left((\mathbf{p}_i)^T \mathbf{z}\right) \tag{5}$$

where $\mathbf{z}$ is an $n+1$ dimensional vector

$$\forall j = 1, ..., n \quad z_j = x_j \text{ and } z_{n+1} = 1 \tag{6}$$

The function $h_i : R \to R$ is called *primitive*. In our paper we assume that primitive functions are monotonic, continuous, and $\lim_{x \to -\infty} h_i(x) = 0$, $\lim_{x \to \infty} h_i(x) = 1$.

The approximation task consists in finding values of $\mathbf{p}$ and $\mathbf{w}$ such that the error measure $e_T(\mathbf{p}, \mathbf{w})$ is minimized for a certain training set T. Note that the minimization takes place in $(nm + m + 1)$-dimensional space, and if functions h_i are nonlinear, there may exist more than a single minimum of the error measure, so when solving the approximation task one gets a global optimization task. The model structure (1) allows for decomposition of this task into subproblems of:

1. searching for the primitive functions h_i and the parameters $\mathbf{p}_i$,
2. error minimization with respect to the vector $\mathbf{w}$.

Note that the residue function (2) is linear with respect to the vector $\mathbf{w}$, so the error measure (3) takes a unique minimum with respect to $\mathbf{w}$ if $\mathbf{p}$ is kept constant. This makes the problem 2. relatively easy to solve numerically. Problem 1 however is more complicated, and the main challenge is to formulate the quality criteria for choosing the functions h_i and the parameters $\mathbf{p}_i$.

2.2 Correlations and the decomposed model

Linearity of the model (1) with respect to $\mathbf{w}$ provides the opportunity to compute the contribution of the f_i function to the residue function (2). Note that if the values of $\mathbf{x}$ are driven from a random variable, then the model output $g(\mathbf{x}, \mathbf{p}, \mathbf{w})$ and the value of each base function $f_i(\mathbf{x}, \mathbf{p}_i)$ are random variables and it is possible to compute their covariance

$$\text{cov}\{f_i(\mathbf{x}, \mathbf{p}_i), g(\mathbf{x}, \mathbf{p}, \mathbf{w})\} = \sum_{j=1}^{m} w_j \text{cov}\{f_i(\mathbf{x}, \mathbf{p}_i), f_j(\mathbf{x}, \mathbf{p}_j)\} \tag{7}$$

Assuming that

$$\forall i, j = 1, ..., m, i \neq j \quad \text{cov}\{f_i(\mathbf{x}, \mathbf{p}_i), f_j(\mathbf{x}, \mathbf{p}_j)\} = 0 \tag{8}$$

we get

$$\mathrm{cov}\{f_i(\mathbf{x}, \mathbf{p}_i), g(\mathbf{x}, \mathbf{p}, \mathbf{w})\} = w_i \mathrm{var}\{f_i(\mathbf{x}, \mathbf{p}_i)\} \tag{9}$$

In other words, if base functions are orthogonal in a sense of (8) then the covariance of the model output and each base function f_i depends on the weight w_i and the variance of the base function values observed when inputting random values of $\mathbf{x}$.

If primitive functions are monotonic, it is possible to observe another correlation property. Namely we can observe for each primitive function that

$$c_f\{h_i(x), x\} = 1 \tag{10}$$

where c_f is a *fractile correlation*.

The fractile correlation [8] whose sample analog is called Spearman or rank correlation, assumes that the two variables are measured on at least interval scales and the value of correlation (i.e., correlation coefficient) determines the extent to which values of the two random variables are "proportional" to each other. The fractile correlation c_f of two random variables X and Y is defined as

$$c_f(X, Y) = 12E\{F(X)F(Y)\} - 3 \tag{11}$$

where E is the symbol of the expected value, and $F(X)$ is the cumulative distribution function of X. Value of $c_f(X, Y)$ is invariant to strictly increasing transformations of X and Y, so it holds

$$c_f(X, Y) = c_f(f_1(X), f_2(Y)) \tag{12}$$

for any strictly increasing functions f_1, f_2.

Observe that the following equation is true

$$c_f\{f_i(\mathbf{x}, \mathbf{p}_i), (\mathbf{p}_i)^T\mathbf{z}\} = 1 \tag{13}$$

Moreover, for arbitrary vector $\mathbf{q}$ the value $c_f\{h_i(\mathbf{x}, \mathbf{p}_i), a(\mathbf{q})^T\mathbf{z} + b\}$ takes its maximum at $\mathbf{q} = \mathbf{p}_i$ for any pair of a, b values (provided that $a \neq 0$).

3 Approximator construction algorithm

We can use the above observations to propose an algorithm for incremental construction of the approximator (ICA). The algorithm is outlined in Fig. 1. The method starts with an empty set of base functions, and the residue and the approximated functions are equal. After that, the algorithm iterates the main loop. In each iteration, a fractile correlation c_f is computed between the residue function $r^{(i)}(\mathbf{x})$ and the scalar product $\mathbf{d}^T\mathbf{x}$ (i is the iteration index). Vector $\mathbf{d}^*$ is computed that maximizes the correlation. Next the base function is determined by selecting a primitive function that gives the best linear correlation with the residue function. It is necessary to compute a_j and b_j coefficients for each function separately since the fractile correlation of the weighted input $(\mathbf{d}^*)^T\mathbf{x}$ and the residue function does not determine the bias b_j. Having determined the base function f_i one has to recompute the weights vector $\mathbf{w}^{(i)}$.

The algorithm implies performing a series of optimization tasks. If the method is stopped after m iterations then the approximator is comprised of m base functions. For

$\mathbf{p}^{(0)} = \emptyset, \mathbf{w}^{(0)} = \emptyset, r^{(0)}(\mathbf{x}) = f(\mathbf{x})$
repeat
  compute $\mathbf{d}^* = \arg\max_{\mathbf{d}\in R^n} \left| c_f\{\mathbf{d}^T\mathbf{x}, r^{(i)}(\mathbf{x})\}\right|$
  for each primitive function h_j
    compute $\{a_j, b_j\} = \arg\max_{\{a,b\}\in R^2} \left| c_l\{h_j(a(\mathbf{d}^*)^T\mathbf{x} + b), r^{(i)}(\mathbf{x})\}\right|$
  use the function g_{j*} as a base function
    $f_i(\mathbf{x}, \mathbf{p}_i) = h_{j*}(a_j(\mathbf{d}^*)^T\mathbf{x} + b_j)$
    where $j^* = \arg\max_j \left| c_l\{h_j(a_j\mathbf{d}^T\mathbf{x} + b_j), r^{(i)}(\mathbf{x})\}\right|$
  $\mathbf{p}_i = [b_{j*}, a_{j*}\mathbf{d}]$,
  $\mathbf{p}^{(i)} = [\mathbf{p}^{(i-1)}, \mathbf{p}_i]$,
  compute $\mathbf{w}^{(i)} = \arg\min_{\mathbf{w}\in R^{i+1}} e^{(i)}(\mathbf{p}, \mathbf{w})$
until a stop criterion is satisfied

Figure 1. Algorithm for approximator construction

each base function, it is necessary to perform a single optimization in R^n (fractile correlation is maximized with respect to $\mathbf{d}$) and a series of k optimizations in R^2 (linear correlation maximization with respect to a, b), where k is the number of primitive functions h_j. In both cases it appears that the maximization task may not have a unique maximum. In addition, after selecting a base function, the error is minimized with respect to the parameters $\mathbf{w}$, but fortunately in this case a unique minimum exists. Consequently, one has to solve m global optimizations tasks in R^n, mk global optimization tasks in R^2, and m convex optimization tasks in R^2 up to R^{m+1}.

4 Numerical example

Test data We verified experimentally the presented algorithm. using the PROBEN1 benchmark originally published by Prechelt [2]. PROBEN1 contains 12 different datasets, including 9 datasets for the classification task, and 3 datasets for the approximation. A quick overview of the approximation problems datasets is given in Table 1. Each dataset has its name, and the benchmark provides a few permutations of each dataset (e.g. 'building2' denotes a certain permutation of the dataset 'building').

Algorithm details We assumed the following set of the primitive functions: hyperbolic tangent $h(x) = (\tanh(x) + 1)/2$, sigmoid function $h(x) = 1/(1 + \exp(-x))$ and the series of power functions

$$h(x) = \begin{cases} 0 & x < -1 \\ 0.5(1 - |x|^k) & -1 \le x < 0 \\ 0.5(x^k + 1) & 0 \le x < 1 \\ 1 & x \ge 1 \end{cases} \tag{14}$$

where $k = 1, 2, 3, 4, 5$. All optimization steps in the algorithm, i.e. maximization of correlations and error minimization with respect to $\mathbf{w}$ were performed using the downhill

Table 1. Attribute structure of approximation problems. Number of binary and continuous network inputs, number of network inputs used to represent missing values, number of outputs, and number of examples. (Continuous means more than two different ordered values).

Problem	Input values				Outputs	Examples
	b	c	m	tot.	c	
building	8	6	0	14	3	4208
hearta	18	6	11	35	1	920
heartac	18	6	11	35	1	303

simplex method by Nelder and Mead [3]. The algorithm was terminated after completing the fifth iteration, so the number of base functions was $m = 5$.

Test results Each of the datasets was divided into two disjoint, equal size sets: the training set and the test set. The training set was used to build the approximator, and the test set — to evaluate the approximator's quality.

We compared the ICA method to neural networks (NN) with a single hidden layer. Since the number of hidden neurons strongly influences the NN performance, we tested the nets with 5 hidden neurons (which makes the number of parameters equal to the number used by the ICA) and we analyzed the NN structures with the number of hidden neurons varying from 1 up to 10. Hidden neurons' activation function was the hyperbolic tangent, and the Levenberg-Marquardt minimization method was used to minimize the NN error. All tests were performed in the MATLAB environment.

Prechelt published the results of applying neural networks to his benchmark datasets [7]. He reported the relative mean squared error defined by

$$E(\mathbf{p}, \mathbf{w}) = 100 * (f_{max} - f_{min}) e_V(\mathbf{p}, \mathbf{w}) \tag{15}$$

where f_{min} and f_{max} are the minimum and maximum values of the $f(\mathbf{x})$ values, and V is the test set.

We performed 60 independent runs of ICA and NN for each dataset in the testbed. Table 2 gives the mean error and its standard deviation values for all compared approximation algorithms.

Prechelt used RPROP [2] for training neural networks. RPROP is a fast backpropagation algorithm which is very efficient for medium data sets such of those at PROBEN1. As a baseline for further comparison, Prechelt made a number of runs using multilayer networks with sigmoidal hidden nodes. For each problem, 12 different network topologies were used: one-hidden-layer networks with 2, 4, 8, 16, 24, or 32 hidden nodes and two-hidden-layer networks with 2+2, 4+2, 4+4, 8+4, 8+8, and 16+8 hidden nodes on the first and second hidden layer, respectively. All of these networks had all possible feed forward connections. For each dataset, results for the best NN were reported.

Table 2. Results of tests for different approximators. 'NN eq.' stands for the NN with 5 hidden neurons, and 'NN best' is the network structure for which the smallest test set error was observed when changing the number of hidden neurons from 1 to 10. 'Prechelt' stands for the NN results reported in [2]

	ICA				NN eq.			
dataset	train	stdv	test	stdv	train	stdv	test	stdv
building1	0.18	0.17	1.15	0.37	0.70	0.25	2.49	0.59
building2	0.54	0.64	0.58	0.65	0.99	0.45	1.01	0.29
building3	0.72	0.01	0.71	0.01	0.88	0.01	0.90	0.03
hearta1	2.96	0.15	3.67	0.13	2.40	0.72	9.35	0.246
hearta2	3.16	0.10	3.56	0.21	2.70	0.84	6.47	0.031
hearta3	3.17	0.09	3.48	0.12	2.49	0.56	6.12	0.016
heartac1	6.75	0.53	10.66	1.17	2.08	1.28	12.92	1.37
heartac2	6.42	0.34	8.15	0.90	1.43	1.09	9.23	0.95
heartac3	6.06	0.39	8.04	1.01	1.41	0.80	8.67	1.33

	NN best				Prechelt			
dataset	train	stdv	test	stdv	train	stdv	test	stdv
building1	0.41	0.25	1.86	0.51	0.47	0.28	1.36	0.63
building2	0.44	0.80	0.46	0.68	0.24	0.15	0.28	0.20
building3	0.40	0.01	0.41	0.01	0.22	0.01	0.26	0.01
hearta1	3.19	0.38	5.32	0.32	3.55	0.53	4.55	0.41
hearta2	3.33	0.51	5.46	0.62	3.45	0.56	4.33	0.15
hearta3	2.75	0.46	5.73	0.74	3.74	0.72	4.89	0.91
heartac1	2.80	0.35	4.46	0.56	3.59	0.24	2.47	0.38
heartac2	1.64	0.59	6.79	0.74	2.58	0.42	4.41	0.56
heartac3	1.57	0.84	8.65	1.79	2.45	0.46	5.55	0.52

5 Summary and conclusions

A new approach to incremental construction of approximators was introduced. The approach is competitive to standard methods such as neural networks. For all datasets ICA turned out to be better than NN with the same number of parameters. ICA is still a winner for certain datasets even when compared to more complicated NN structures with higher number of hidden neurons and of hidden layers.

In the analysis of the approximator (1) we assumed no correlation between the base functions, but this assumption was not verified in the ICA algorithm. This issue still needs further investigation, since if such correlations appear then the approximator quality may decrease. We also plan to generalize the ICA method to cover the spherically symmetric base functions (RBF).

Bibliography

[1] Ivakhnenko A.G.: Polynomial Theory of Complex Systems. *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-1, no.4, October 1971.

[2] Prechelt, L.: PROBEN1: A set of benchmarks and benchmarking rules for neural network training algorithms. *Technical Report 21/94*, Fakultät für Informatik, Universität Karlsruhe, Germany, 1994.

[3] Nelder J.A., Mead R.: *Computer Journal*, vol. 7, 1965, pp. 308–313.

[4] Fahlman S.E.: Faster learning variations on backpropagation: an empirical study. *Proc. Connectionist Models Summer School*, Morgan Kaufmann, 1988, pp.38–51.

[5] Hertz J., Krogh A., Palmer R.G.: *Introduction to the Theory of Neural Computing.* MIT Press, 1992.

[6] Li Q., Tufts D.: Synthesizing neural networks by sequential addition of hidden modes. *Proc. IEEE Int. Conf. on Neural Networks*, 1994, pp.708–713.

[7] Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The PROP algorithm. *Proc. IEEE Int. Conf. on Neural Networks*, 1993.

[8] Biller, B., Ghosh, S.: Dependence modeling for stochastic simulation, *Proc. 2004 Winter Simulation Conf.*, 2004.