6-2002

Implementation of Large Neural Networks Using Decomposition

Henry Selvaraj
University of Nevada, Las Vegas, henry.selvaraj@unlv.edu

H. Niewiadomski
Warsaw University of Technology

P. Buciak
Warsaw University of Technology

M. Pleban
Warsaw University of Technology

Piotr Sapiecha
Warsaw University of Technology

Implementation of Large Neural Networks using Decomposition

H. Selvaraj*, H. Niewiadomski, P. Buciak, M. Pleban, P. Sapiecha, T. Luba and V. Muthukumar*

** Department of Electrical and Computer Engineering,*
University of Nevada, Las Vegas,
Las Vegas, NV USA 89154
selvaraj@unlv.edu

Department of Electronics and Information Technology,
Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
luba@tele.pw.edu.pl

Abstract

The article presents methods of dealing with huge data in the domain of neural networks. The decomposition of neural networks is introduced and its efficiency is proved by the authors' experiments. The examinations of the effectiveness of argument reduction in the above filed, are presented. Authors indicate, that decomposition is capable of reducing the size and the complexity of the learned data, and thus it makes the learning process faster or, while dealing with large data, possible. According to the authors experiments, in some cases, argument reduction, makes the learning process harder.

Keywords

Neural networks, decomposition, back propagation, argument reduction, control systems.

1. Introduction

Often, in the area of neural networks, computers have to deal with unmanageable amount of information, stored in decision tables of many inputs and outputs [1,2]. The more complex such a function is, the more difficult it is to achieve satisfactory training results. So the objective is to reduce the number of inputs and outputs in such a way, that no information is lost. Hence, in such cases the application of decomposition seems to be a natural approach.

Generally speaking, decomposition appears as a fundamental problem of great importance for a wide spectrum of different scientific fields. The strong motivation for developing decomposition techniques comes from modern research areas such as pattern recognition (*PR*), knowledge discovery (*KD*) and machine learning (*ML*) in artificial intelligence and also from logic synthesis in computer-aided design of very large integrated circuits (*VLSI-CAD*). Simply speaking, decomposition means dividing a complex problem into several relatively easier and independent sub-problems. In this way we can divide one complex neural network into some smaller and simpler sub-networks. So decomposition is a powerful tool in neural network design.

In this paper, we will focus on decomposition of multiple-valued (*MV*) functions based on the graph coloring heuristics. The multiple-valued functions can be interpreted in different ways, depending on the application. Let us consider two different examples:

(1) *PR, KD* and *ML* [1,2,3]. Initial function is specified as a decision table. 'If-then-else' rules, typical for knowledge-based systems, can be compiled from such a table. When the table is large, there are many rules, some of them very complex, and the generalization power of the rules applied to new situations (don't care) may be weak. When the initial function is decomposed into several blocks, the *MV* rules compiled from the tables of smaller blocks are simpler, their number decreases, and their generalization power may increase because the decomposition process may detect some „hidden" properties of the data (all intermediate variables introduced in the process correspond to some new concepts). Finding automatically such hidden properties of data (or higher order features in terms of Pattern Recognition Theory) is perhaps the most exciting aspect of using decomposition in *ML*. When the rules are compiled from the decomposed blocks, their set becomes a kind of a program, so the evaluation of these rules is performed in software. There exist also innovative

computer architectures that allow to evaluate rules in hardware.

(2) *NEURAL NETWORKS* [4,5,6,7,8]. In gradient decent methods of supervised training, data is given as a table of inputs and associated outputs. The goal is to find the structure of a neural network suitable for given data, which is a serious problem. However, some research in this field has been done, but currently there is a lack of general method. Decomposition helps to structure neural network according to the given data.

Hence, we can claim that the decomposition techniques are universal and flexible in solving the above-mentioned scientific problems. The decomposition method of decision tables based on graph coloring heuristics is presented in this paper. It generalizes the decomposition of Ashenhurst, Curtis, Luba, Perkowski/Wang [9,10,11].

2. Decomposition of MVL Functions

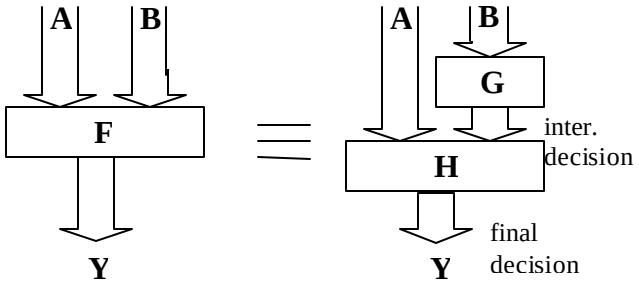

Figure 1. Decomposition scheme

This section contains the formulation of several basic ideas related to functional decomposition [2,12,13]. Let C_i and D be finite sets for all $i \in \{1, 2, ..., n\}$. *The multiple-valued function F* is defined as a mapping $F(X_1, ..., X_n)$: $C_1 \times ... \times C_n \rightarrow D$. Every element of the domain $C_1 \times ... \times C_n$ is called *a minterm* (*sample* or *object*). A listing of minterms with the value of the function is called *a decision table* (or *truth table* in binary case). Decision tables do not include minterms for which the function values are not specified. For the sake of clarity this unspecified set of minterms will be called *Don't Care Set* (*DC*). There, the functions with non-empty *DC*-set will be called *the partial functions*. Let F be a multiple-valued function representing functional dependency $Y = F(X)$, where X is the set of *input variables* and Y is the set of *output variables*. Let the partition of input variables into two disjoint sets: *the free set A* and *the bound set B* be given. We say that there exists *a functional decomposition* of F if $F(X) = H(A, G(B))$. More precisely speaking, G and H denote functional dependencies $Y = H(A, Z)$ and $Z = G(B)$. Function $H(A, G(B))$ is called a simple disjoint decomposition of function F. The structure of decomposed decision function is shown in Figure 1. It is necessary to emphasize that the function after decomposition is equivalent to one before decomposition, as decomposition is a method which helps to structure a decision table.

The interpretation of the presented scheme is as follows. At the beginning, the intermediate decision is taken on the basis of attributes from subset B. Then consequently, the final decision is taken with respect to both intermediate decision and attributes from subset A. The fundamental problem is then, given a function F and the partition of the set of input variables into sets A and B, to find the functions G and H, such that $F = H(A, G(B))$. To solve this problem, let us first introduce some useful concepts.

The decomposition chart of the function F consists of a two dimensional matrix with columns indexed using the values of the bound set variables and rows indexed using the values of the free set of variables. Elements of the matrix $m_{i,j}$ are the values assumed by the function F for the vectors constructed from i-th row and j-th column. *The column multiplicity* of the matrix is denoted by $v(B|A)$.

Theorem 1 *Let* $F : [m]^n \rightarrow [m]^k$, $F = \{f_i\}_{i \in [k]}$ *be a group of finite functions, where* $f_i : [m]^n \rightarrow [m]$. *Let A and B be a pair of disjoint subsets of variables Var*(*F*). *Then* :

$F(A, B) = H(A, G(B)) \Leftrightarrow v(B|A) \leq d^j$,

where:

$G = \{g_k(B)\}_{k = 1, ..., j}$ *for* $g_k : [m]^{Card(B)} \rightarrow [d]$ *and*

$H = \{h_l(G(B), A)\}_{l = 1, ..., n}$ *for* $h_l : [u]^{j + Card(A)} \rightarrow [m]$,

$u = \max\{d, m\}$

Proof See [12,13].

This theorem suggests that the fundamental problem of a partial finite function decomposition is nothing but finding an expansion of the function for which $v(B|A)$ has the least value, and this problem can be reduced to a graph coloring problem [13,14]. Its objective is to color all the vertices of the graph with a minimal number of colors, in such a way, that no two adjacent vertices (in our case columns, that are contradictory), are assigned the same color. Even though the above-mentioned problem is in general NP-hard, many heuristics have been invented (e.g. Seq. Algorithm, Seq. Algorithm with Ordering, Maximal Independent Sets Algorithm), which solve it in polynomial time [15].

Theorem 2 *The problem of k-decomposition of partially defined finite functions can be reduced to the problem of k-coloring of the graph (in PTIME).*

Proof See [3].

On the basis of the above explanation we propose the following heuristic for decomposing finite functions (Algorithm 1).

Algorithm 1 Decomposition heuristic

Require: Finite group of finite functions F and a partition of the input variables into a pair of subsets A and B.

1. Construct a decomposition chart of function F using given bound and free sets;
2. Find the graph $G_{F(A, B)}$ of incompatible columns in the decomposition chart;
3. Find the minimal coloring for the graph $G_{F(A, B)}$;
4. Find the decomposition H(A, G(B)) for the function F from the results obtained in step 2;

Theorem 3 *The problem of k-decomposition for k = 1, 2 for partially defined Boolean functions (and also for partial finite functions) is in PTIME, and for k ≥ 3 is NP – complete*.
Proof See [3]

Example 1.
Let us decompose function F given in Table 1 in conformity with Algorithm 1.

Table 1. Decision table of function F

B			A		F
x_1	x_2	x_3	x_4	x_5	
0	0	0	0	0	0
2	0	2	2	0	2
1	1	1	0	0	0
1	1	1	1	1	1
0	0	0	1	1	1
2	2	2	2	0	2
1	1	2	1	1	1
1	0	1	0	0	1
1	1	1	2	0	1
1	1	2	0	0	2

Table 2. Decomposition chart

A \ B	000	101	111	112	202	222
00	0	1	0	2	-	-
11	1	-	1	1	-	-
20	-	-	1	-	2	2

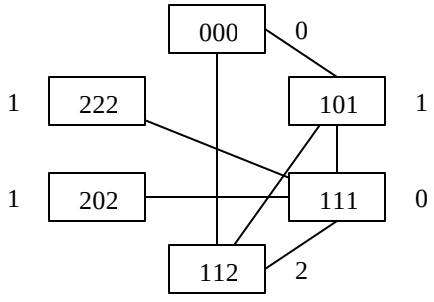

Figure 2. Incompatibility graph

The decomposition chart of function F is constructed in Table 2.
Afterwards the graph $G_{F(A,B)}$ (Figure 2) of incompatible columns is created and coloring algorithm is executed. For instance columns 000 and 101, in contrast to columns 112 and 202, are incompatible and consequently they are colored with different numbers.
Finally decision tables for functions G and H (respectively Table 3 and 4) are derived from previously colored graph $G_{F(A,B)}$.

Table 3. Function G

B			G
x_1	x_2	x_3	
0	0	0	0
1	0	1	1
1	1	1	0
1	1	2	2
2	0	2	1
2	2	2	1

Table 4. Function H

A		G	H
x_4	x_5		
0	0	0	0
0	0	1	1
0	0	2	2
1	1	0	1
1	1	2	1
2	0	0	1
2	0	1	2

4. Argument reduction

A *multi-valued function*, often named an *information system* or a *decision table*, is a pair $S = (U, A)$, where U is a nonempty, finite set called the *universe*, and A is a nonempty, finite set of attributes, i.e. $a : U ? V_a$ for $a \in A$, where V_a is called the *value set* of a [25]. The set of attributes A is partitioned into two nonempty disjoint sets C and D, called *condition* and *decision* attributes respectively. These names are based on one of the interpretation of such defined information system, in which it describes an expert system operating using *if-then* rules, making decisions based on whether certain conditions are met or not. Decision tables are widely used in artificial intelligence, machine learning, logic synthesis and other applications. An information system can be represented as a table, in which the rows correspond to single objects from the universe, and columns correspond to attributes. An example of such presented information system is given in Table 5.

Table 5. Example information system

U	a	b	c	d	e	x
1	0	1	1	1	1	1
2	1	1	0	1	0	1
3	1	0	0	1	1	0
4	1	0	0	1	0	0
5	1	0	0	0	0	1
6	1	1	0	1	1	0

We will now define the notions of a discernibility matrix and a discernibility function [26]. Suppose we

have an information system $S = (U, A)$, in which $A = C \cup D$ and $U = \{x_1, x_2, \ldots x_n\}$. A *discernibility* matrix of S is a matrix $(m_{ij})_{i,j \in \{1, 2, \ldots, n\}}$ with elements defined in the following way: $m_{ij} = \{c \in C : c(x_i) \neq c(x_j)\}$ if $\exists d \in D\ d(x_i) \neq d(x_j)$, otherwise $m_{ij} = \emptyset$. Intuitively, we write in this matrix the sets of condition attributes, which can be used to discern the pairs of objects from U for which different decisions need to be made. We will call this matrix $M(S)$. For simplicity, we often write $M(S)$ by listing only its nonempty elements.

A *discernibility function* f_S is a boolean function with boolean variables v_i corresponding to attributes a_i, and defines as follows: $f_S(v_1, v2, \ldots, v_m) = \wedge\{\vee(m_{ij}) : 1 \leq j < i \leq n, m_{ij} \neq \emptyset\}$, where $\vee(m_{ij})$ is the disjunction of all variables v_k such that $c_k \in m_{ij}$. When writing discernibility functions, we will give the variables the same names as the attributes from C when no confusion will arise.

Example 2
For the information system defined in Table 5, we obtain the following discernibility matrix (Table 6) and discernibility function:

Table 6. The discernibility matrix for example information system

	1	2	5
3	*abc*	*be*	*de*
4	*abce*	*b*	*d*
6	*ac*	*e*	*bde*

$$f_S(a, b, c, d, e) = b \wedge d \wedge e \wedge (a \vee c) \wedge (b \vee e) \wedge (d \vee e) \wedge (a \vee b \vee c) \wedge (b \vee d \vee e) \wedge (a \vee b \vee c \vee e)$$

We define a discernibility relation with $B \subseteq A$ being a subset of attributes, in the following way: $DIS(B) = \{(x, y) : x, y \in U \wedge \exists\, b \in B\ b(x) \neq b(y)\}$ [26]. Intuitively, two elements from the universe U are in this relation if they can be discerned by one or more attributes from B, i.e. the values of this attribute for these elements are different. We can define this relation for condition as well as decision attributes.

A *reduct* is a set $B \subseteq A$ such that $DIS(B) = DIS(D)$, i.e. the values of attributes from B are sufficient to discern all the objects from universe U for which different decisions need to be made. It is obvious that $DIS(C) = DIS(D)$. By $RED(S)$ we will call the set of all nontrivial reducts of an information system S, i.e. reducts that do not contain other reducts.

Theorem 4 Let $S = (U, A)$ be an information system, and $\emptyset \neq B \subseteq A$. The following conditions are equivalent:

1. $B \in RED(S)$;
2. $f_S(v_B(a_1), v_B(a_2), \ldots, v_B(a_m)) = 1$, where $v_B(a_i) = 1$ if $a_i \in B$, otherwise $v_B(a_i) = 0$;
3. $\forall\, i, j\ c_{ij} \neq \emptyset \Rightarrow c_{ij} \cap B \neq \emptyset$;

Proof Can be found in [26].

From theorem 4 it follows that to find the $RED(S)$ it is sufficient to minimize the function f_S. For example defined in Table 5, we have:

$$MIN(f_S) = b \wedge d \wedge e \wedge (a \vee c)$$

Thus $RED(S) = \{\{A, B, D, E\}, \{B, C, D, E\}\}$.

The *minimal reduct problem* can be defined in the following way: given an information system $S = (U, A)$, find a reduct $R \in RED(S)$ such that R is minimal, i.e. its cardinality is not bigger than any other reduct from $RED(S)$.

Theorem 5 The minimal reduct problem is NP-hard.

Proof (sketch). The well-known *NP*-complete *k-test-collection* problem can be reduced to the minimal reduct problem in polynomial number of steps.

According to theorem 4, we can reduce the minimal reduct problem to the minimal transversal problem, and therefore apply any of the well-known minimal transversal algorithms. We can do this in a following way: for a given information system $S = (U, A)$ we construct a hypergraph in which the vertices correspond to the variables from A, and edges correspond to all sets from the discernibility matrix $M(S)$. We remove all the edges that contain any other edge, in order to make the hypergraph a simple one. Then we find a minimal transversal for this hypergraph, which correspond to a set $B \subseteq A$ of variables. From theorem 5 it follows directly that B is a minimal reduct of S.

Therefore we can summarize this section by giving the complete algorithm for argument reduction of a multi-valued function (algorithm 2). Note that this algorithm requires $O(|U|^2 \cdot |A|)$ steps for graph construction, since in the for loop it needs to check all possible pairs of objects and for each pair, the values of all variables need to be examined.

Algorithm 2 Argument reduction for multi-valued functions

Require: A multi-values function with the sets C of condition variables and D of decision variables;
Construct a hypergraph with $m = |C|$ vertices, of which each vertex corresponds to one variable from the set C;
for all pair (u_1, u_2) such that $u_1 \in U$, $u_2 \in U$ and $\exists d \in D\ d(u_1) \neq d(u_2)$ **do**
Construct an edge E joining all and only those vertices, that correspond to variables from C which have different values for u_1 and u_2;
end for
Apply any algorithm finding a minimal transversal T in constructed hypergraph, for example the greedy approximation algorithm;
Return the set of condition attributes that correspond to vertices in T;
steps for graph construction, since in the for loop it needs to check all possible pairs of objects

3. Serial decomposition of neural networks

The time required to train the weights of a neural network depends on complexity of the training patterns and their number. Therefore, we expect that this time could be reduced if these patterns (which can be viewed as multiple-valued functions) were decomposed into several smaller sub-functions. Then, instead of one large network, one would use a number of smaller ones, connected hierarchically, similarly to Figure 1.

4. Experimental results

At The Telecommunication Institute of Warsaw University of Technology, taking advantage of the above-presented algorithms, the decomposition and reduction system *HOSEA* was created [1,3,14]. This system runs both under Linux and Win32 operating systems. For neural networks simulations, the Stuttgart Neural Network Simulator (*SNNS*) was used [19]. *SNNS* is a joint effort of a number of people at the Institute for Parallel and Distributed High Performance Systems (IPVR) at The University of Stuttgart, the Wilhelm Schickard Institute for Computer Science at the University of Tübingen, and the European Particle Research Lab CERN in Geneva. Because of the fact, that almost all important algorithms and architectures are implemented in this program in a very effective way, *SNNS* is an efficient and flexible simulation software for research and application of neural nets [7]. In our experiments we used various algorithms based on well-known error back-propagation approach.

The main objective of our experiments was the analysis of the influence of decomposition and argument reduction on neural network training process.
The data used for our experiments were acquired as a result of air pollution measurements of a power plant (unfortunately, the name and location can not be disclosed due to official restrictions).

The structure of the data used in process of training is described below.

Inputs of neural network: (parameters specifying the state of the power plant and number of corresponding 4-valued inputs)	
- oil pressure	6
- amount of anti soot substance	6
- oil temperature	6
- power of the generator	6
- for each of 24 oil burners: oxygen valve open	12
- for each of 24 oil burners: gas valve open	12
- for each of 24 oil burners: oil valve open	12

Outputs of neural network: (parameters measuring the air pollution and number of corresponding 4-valued outputs)	
- air opacity	6
- concentration of NO_x	6
- concentration of CO in eastern direction	6
- concentration of CO in western direction	6

The data set was normalized and digitalized into 4-valued set, and finally became a decision table of 60 inputs, 24 outputs and 17688 disjoint patterns. In order to find out how the results of our approach depend on the size of a pattern set, smaller training sets were prepared, using selection with equal probability from original pattern set.

As authors wanted to examine the influence of argument reduction and decomposition on the process of learning of neural networks, the training was carried out 4 times: with sets of not processed data, with decomposed data, with reduced data and with reduced and afterwards decomposed data.

The decision tables (and therefore neural networks) were decomposed into five sub-networks connected in a way presented in Figure 3 (the number of neurons is limited because of clarity). In Figure 3a a simple three-layer network, which is equivalent to completely specified function $F(X)$, is shown. Let $X = \{X_i\}_{i = 1, 2, ..., 12}$ and $P\{X\} = \{\{1, 2, 3\},\{4, 5, 6\},\{7, 8, 9\},\{10, 11, 12\}\}$. After computing the decomposition $F = H(G_1(X_1, X_2, X_3), G_2(X_3, X_4, X_5), G_3(X_6, X_7, X_8), G_4(X_9, X_{10}, X_{11}))$, network in Figure 3b consists of five sub-nets, which correspond to functions

G_1, G_2, G_3, G_4 and H. It is noteworthy, that there are many independent ways to design each sub-net. We tried to train two architectures of neural networks: the first one, which consisted of single neural network and the second – composed of five neural networks connected in a serial way.

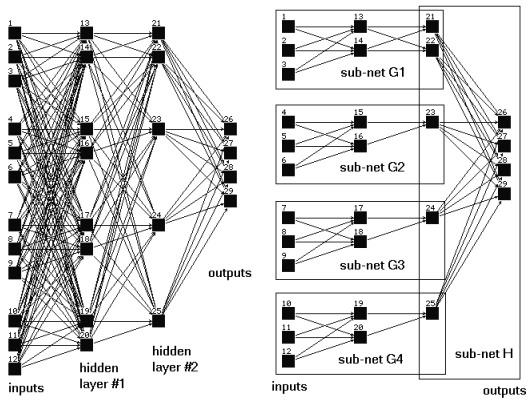

Figure 3a, 3b. The network before and after decomposition

The application of argument reduction eliminated the redundant input variables, what decreased the size of training data and corresponding neural network.

Table 7. The training results (data without argument reduction)

# of pat-terns	Original network			Decomposed network (into 5 blocks)			
	Mean Square Error	# of error pat.	# of error bits	Sub-net's name	Mean Square Error	# of error pat.	# of error bits
1102	1.650	1102	11618	G1	0.00446	23	23
				G2	0.00124	4	4
				G3	0.00051	0	0
				G4	0.00153	1	1
				H	0.11273	119	154
2222	2.033	2222	26214	G1	0.00771	66	66
				G2	0.00963	71	71
				G3	0.00081	0	0
				G4	0.00360	3	4
				H	0.07861	100	128
4369	2.355	4369	82289	G1	0.01279	167	180
				G2	0.01428	183	196
				G3	0.00173	0	0
				G4	0.00616	3	5
				H	0.23967	310	715
8850	3.965	8850	153801	G1	0.00519	6	6
				G2	0.01137	11	15
				G3	0.00064	0	0
				G4	0.00733	8	12
				H	0.19936	284	532
17688	6.089	17688	312889	G1	0.02868	636	716
				G2	0.08750	1911	4431
				G3	0.17426	20	20
				G4	0.10471	82	87
				H	1.61480	2350	23892

Table 8. The training results (data after argument reduction)

# of pat-terns	Original network			Decomposed network (into 5 blocks)			
	Mean Square Error	# of error pat.	# of error bits	Sub-net's name	Mean Square Error	# of error pat.	# of error bits
1091	21.81	1091	26184	G1	0.1408	6	6
				G2	1.5159	41	49
				G3	2.4068	55	76
				G4	0.4818	10	10
				H	22.882	1091	12437
2194	22.77	2194	52656	G1	0.4852	16	16
				G2	1.7034	44	55
				G3	2.4068	51	73
				G4	0.6797	16	18
				H	23.909	2194	27199
4275	24.51	4275	102600	G1	0.8514	100	119
				G2	0.8754	50	67
				G3	3.0904	232	407
				G4	0.4804	6	6
				H	24.595	4274	56820
8471	25.32	8471	203304	G1	1.3092	135	170
				G2	0.7300	52	69
				G3	2.8522	221	379
				G4	0.9480	45	52
				H	24.599	8468	112910
16356	25.23	16356	392544	G1	1.7960	861	1361
				G2	1.3103	1006	1389
				G3	1.3765	158	213
				G4	1.1670	54	64
				H	25.390	16318	239554

Weights were initialized with random values. Different learning methods were applied: standard back-propagation, SCG, RPROP and other. Generally, all the results were similar. The effects of training with Scaled Conjugate Gradient algorithm are presented in Table 7,8. The time of training was limited to 3600 seconds.

It is necessary to emphasize, that networks before and after decomposition are equivalent in information sense, as the decomposition methodology, what has been already stated, does not lose information. Therefore, training error depends only on a training algorithm and sizes of neural network and training data, which are both reduced in suggested approach.

It turned out (especially while processing not reduced data – Table 7), that it is impossible to train such huge neural network in conventional way, and in order to get smaller, manageable sub-networks, decomposition techniques described above had to be applied.

We can see that, after applying the decomposition, the mean square error as well as the number of erroneous patterns and bits decreased considerably. It is important to point out that before the application of the decomposition the data were too complex to be trained effectively. However, after the decomposition of the data, we achieved a reasonable training results. As it is shown in Table 7, the most difficult problem was to properly learn the H sub-

network. The reason of that is the difficulty of the intermediate decision encoding.

Another interesting conclusion is the fact, that argument reduction decreased significantly the effectiveness of learning. At first sight it seems to be counterintuitive, but it may be explained by the form of the data after reduction. In our case the number of neural network's inputs was decreased, but the size of the data (compare the first column of Tables 8 and 7) remained almost the same, what had a negative impact an training process. According to our results, the efficiency of training may be improved after argument reduction only if the degree of freedom of neural network and the size of data are diminished in comparative degree.

5. Conclusions

The article presents the method of dealing with the huge data, which is based on decomposition of artificial neural networks into smaller sub-nets. This method allows processing of data which, in the original form, are not suitable for data mining, because of its huge size. Our novel strategy was successfully used when processing information concerning air pollution, measured around the power plant. Not only was it possible to deal with such large amounts of information, but it also proved to be very effective, as the number of mistakes was kept at a reasonable level. The influence of argument reduction strategy was also examined. The experiments proved, that it is not a general approach and in some cases it can even make training impossible.

6. References

[1] T. Luba, H. Niewiadomski, H. Pleban, H. Selvaraj, P. Sapiecha, Functional decomposition and its application in design of digital circuits and machine learning, *IASTED Inter. Conf. On Applied Informatics*, Innsbruck, Austria, 2001.

[2] B. Zupan, M. Bohanec, J. Demšar, I. Bratko, Feature transformation by function decomposition, IEEE Expert, Special Issue on Feature Transformation. 1997.

[3] P. Sapiecha, H. Selvaraj, M. Pleban, Decomposition of Boolean Relations and Functions in Logic Synthesis and Data Analysis, *The Second International Conference on Rough Sets and Current Trends in Computing*, Banff, Canada, 2000.

[4] S. Y. Kung, *Digital Neural Networks*, PTR Prentice Hall, 1993.

[5] T. Masters, *Neural Networks in Practice*, Prentice-Hall, 1996.

[6] T. Masters, Pract*ical Neural Networks Recipes in C++*, Academic Press, 1993.

[7] M. Riedmiller, Untersuchungen zu konvergenz und generalisierungsverhalten überwachter lernverfahren mit dem SNNS, *Proc. of the SNNS 1993 Workshop*, 1993.

[8] M. Riedmiller, H. Braun, A direct adaptive method for faster back-propagation learning: The RPROP algorithm, *Proc. of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, 1993.

[9] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, Princeton, N.J., Van Nostrand, 1962.

[10] R. L. Ashenhurst, The Decomposition of Switching Functions, *Proc. Int. Symp. of Theory. of Switching*, 1957.

[11] C. Yang, V. Singhal, M. Ciesielski, BDD decomposition for efficient logic synthesis, International Conference on Computer Design, 1999.

[12] T. Luba, Decomposition of multiple-valued functions, *Proc. IEEE-ISMVL*, USA, 1995.

[13] W. Wan, M. Perkowski, A New Approach to the Decomposition of Incompletely Specified Multi-Output Function Based on Graph Coloring and Local Transformations and Its Application to FPGA Mapping, Proc. Euro-DAC'92.,1992.

[14] H. Selvaraj, H. Niewiadomski, M. Pleban, P. Sapiecha, Decomposition of Digital Circuits and Neural Networks, Proc. SCI 2001, US, 2001.

[15] M. Perkowski, R. Malvi, S. Grygiel, M. Burns, A. Mishchenko, Graph coloring algorithms for fast evaluation of Curtis decompositions, *DAC*, 1999.

[16] D. de Werra, Heuristics for Graph Coloring, Computational Graph Theory, Tinhofer, Mayr, Noltenmeier, Syslo (Eds.), Springer-Verlag, New York, 1990.

[17] J. M. Zurada, Introduction to Artificial Neural Systems, West Publishing Company, St. Paul, 1992.

[18] Martin, Fodslette, Moller, A scaled conjugate algorithm for fast supervised training, *Neural Networks 6*, 1993.

[19] http://www-ra.informatik.untuebingen.de/SNNS/

[20] S. Chang, M. Marek-Sadowska, T. Hwang, Technology mapping for TLU FPGA's based on decomposition of Binary Decision Diagrams, *IEEE Transaction on Computers*, vol. 15, No. 10, 1996.

[21] Y. Lai, K. Pan, M. Pedram, OBDD-based functional decomposition: algorithms and implementation, *IEEE Transaction on Computers*, vol. 15, No. 8, 1996.

[22] Y. Lai, M. Pedram, S. Vrudhula, EVOBDD-based algorithms for integer linear programming, spectral transformation, and functional decomposition, *IEEE Transaction on Computers*, vol. 13, No. 8, 1994.

[23] T. Luba, C. Moraga, S. Yanushkevich, M. Opoka, V. Shmerko Evolutionary Multi-Level Network Synthesis in Given Design Style, *Proc. IEEE-ISMVL*, Portland, Oregon, USA, 2000.

[24] J. Roth, R. Karp, Minimization over boolean graph, IBM J. Res. and Develop, 1962.

[25] Z. Pawlak, Rough Sets: Theoretical Aspects of Reasoning About Data, Kluwer, 1991

[26] A. Skowron, C. Rauszer, The discernibility matrices and functions in information systems, in: R. Slowinski (Ed.), Intelligent Decision Support. Handbook of Applications and Advances on the Rough Set Theory, Kluwer, 1992