ELSEVIER

Computer Standards & Interfaces 24 (2002) 311–322

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

Application of XML for neural network exchange

Denis V. Rubtsov[a,*], Sergei V. Butakov[b]

[a]Department of Mathematics, Altai State University, 61a Lenina Street, Barnaul 656099, Russia
[b]Altai State Technical University, 49-5 Lenina Street, Barnaul 656099, Russia

Received 14 January 2002; received in revised form 24 February 2002; accepted 20 April 2002

Abstract

This article introduces a framework for the interchange of trained neural network models. An XML-based language (Neural Network Markup Language) for the neural network model description is offered. It allows to write down all the components of neural network model which are necessary for its reproduction. We propose to use XML notation for the full description of neural models, including data dictionary, properties of training sample, preprocessing methods, details of network structure and parameters and methods for network output interpretation. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Artificial neural networks; XML; Markup language; Exchange format

1. Introduction

Within the last decade, artificial neural networks has become a very popular technique in the field of empirical data analysis. Now, they are not only an object of scientific research but also a useful tool in the practice of such domains as medicine, engineering, finance, economics, etc. The problems for which neural networks are intended vary from classification and prediction to data visualization and compression. There are well-known models such as Hopfield networks, multilayer perceptrons, radial basis function networks, support vector machines, self-organization maps and others. At that, the number of neural-like models and schemas is increasing permanently. There is also a great variety of the ways to implement neural models: as outstanding software simulators, problem-oriented languages, modeling packages and libraries for common programming environments, hardware devices, etc.

But in our opinion, there is a number of barriers, which makes the application of neural networks quite problematic. One of them is determined by the lack of common approach to the description and representation of the neural models. From the practical point of view, there is no convenient way of distribution of the neural models between the simulation systems or exchange them in order to share the neural network solution to the particular problem. This causes certain difficulties for the building of the multicomponent and distributed systems, and makes it impossible to develop a standard interface between the neural simulators and other components of the information systems.

On the other hand, apparently most of current neural network researches are devoted to the process of creation of neural networks. But since we believe that neural networks are suitable for the practical application, solid studies of the ways to apply neural models in

* Corresponding author. Tel.: +7-8-3852-381037.
E-mail addresses: rubtsov@math.dcn-asu.ru (D.V. Rubtsov), swb@agtu.secna.ru (S.V. Butakov).

PII: S0920-5489(02)00034-X

an effective way are absolutely necessary. A significant part of such an activity would be aimed at standard investigation for the neural networks in order to provide a successful implementation of neural techniques. The discussion of this question is beyond the scope of this article, but we feel that it makes a contribution.

As a step to solution for these problems, we consider the introduction of an interchange electronic format that allows representing any neural model in a unified way. In this work, we concentrate on exchange of neural solutions. Let us define the neural solution of the particular data analysis problem as the computational model, built with methods of neurocomputing, which allows to obtain the answer to the desired question on the basis of certain data. Neural solution is simply the result of the neural net technology application to the data analysis problem (see Fig. 1).

Note that the neural model does not consist of a trained neural network only. It necessarily includes a set of input and output variables, which represents the network environment, methods of its preprocessing and methods for sensible interpretation of the network output signals.

In the current work, we have made an attempt to bring together the most common ideas about neural network description and to develop rational and flexible framework for neural solution exchanging format. In essence, the framework is the language based on eXtensible Markup Language (XML) [4] and called Neural Network Markup Language (NNML). Our main goal is to develop a descriptive language that can provide a comprehensive way to write down any neural-like models with heterogeneous elements and arbitrary topology, as well as their environment. We believe that NNML could be concerned as a starting point to a true exchange format due to some of its features.

In particular, NNML is designed to represent all the information sufficient for an unambiguous reconstruction of the neural model and its proper application. In addition to the neural network, it allows to describe the problem that the neural model is intended for, as well as the model data dictionary and methods for the preliminary data transformations and the post-processing techniques. Therefore, we have a full and structured description of the neural model without any dependence on the particular features of the system that produced it.

Another important NNML property is that NNML, being an XML-based language, is a cross-platform format independent of the particular simulation system or programming language. It clears the way to distribute neural models between the applications based on the different operation systems. For example, the neural net created on the UNIX machine can be used by a Windows-based program. From another perspective, XML technology provides us for convenient means to enable openness and flexibility of the format. Developers and users can easily introduce their own elements and parameters of the neural net description.

Obviously, most of the neural models now are implemented through numerical methods with the standard mathematical routines available in general-purpose programming environments and languages like Matlab, C++ or Fortran. To describe them in a more natural way, all the components of the neural models in NNML are described in an unified way with the common mathematical operators and functions

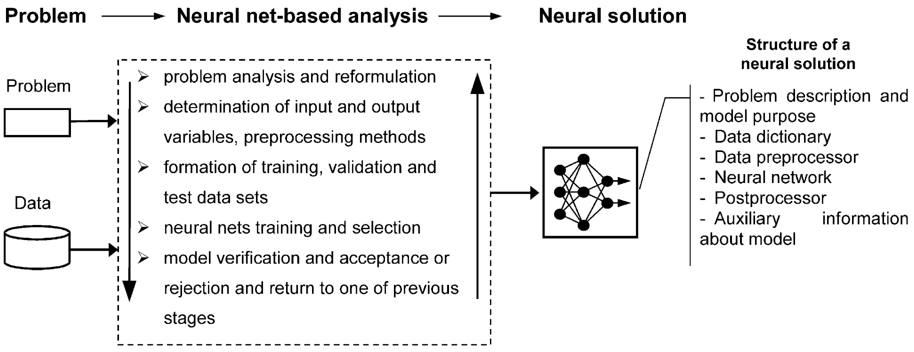
Problem
Neural net-based analysis
Neural solution
Problem
Data
problem analysis and reformulation
determination of input and output variables, preprocessing methods
formation of training, validation and test data sets
neural nets training and selection
model verification and acceptance or rejection and return to one of previous stages
Structure of a neural solution
- Problem description and model purpose
- Data dictionary
- Data preprocessor
- Neural network
- Postprocessor
- Auxiliary information about model

Fig. 1. Neural solution as a result of the neural net technology application to the data analysis problem.

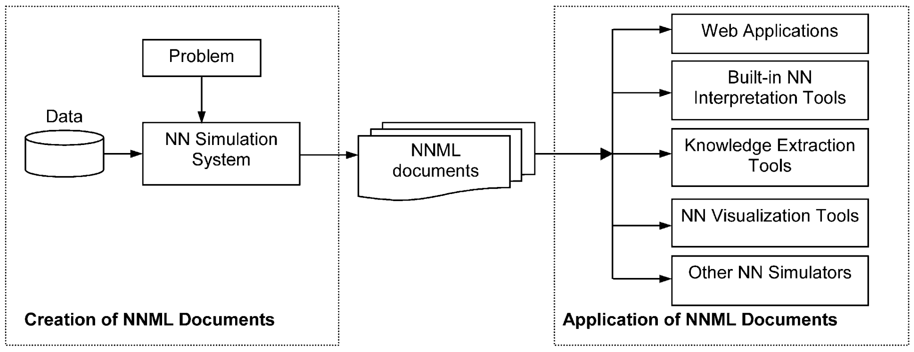
Problem
Data
NN Simulation System
NNML documents
Web Applications
Built-in NN Interpretation Tools
Knowledge Extraction Tools
NN Visualization Tools
Other NN Simulators
Creation of NNML Documents
Application of NNML Documents

Fig. 2. NNML-based interface between different systems concerned with the neural network applications.

instead of using the neural net specific terms like "synapse", "axon", etc. This provides a clear access to the essence of the model and allows to construct any desired functioning of the model components, as well as to introduce the new elements of the neural network.

Because of these features, NNML can be used as an interface format between the heterogeneous systems concerned with the neural network applications (see Fig. 2).

In further statement, the terms "format" and "language" are used as synonyms. In this article, the following convention is also accepted: the real names of NNML tags are printed with `courier` font.

The article is organized as follows. Section 2 concerns the brief analysis of the existing publications devoted to the neural network languages and related problems. In Section 3, an overview of the proposed format is given. Section 4 presents a simple example of neural network that helps us to illustrate the main features of NNML. In Section 5, we present a framework for handling of processing units. Section 6 is devoted to the contents of the main neural model component description. In Section 7, a framework for providing extensibility is described. Sections 8 and 9 touch upon questions of processing NNML documents and briefly discuss the possible applications.

2. Related works

The problem of interchange format development is closely connected to the neural network specification languages and the investigation of formalization and standards for various neural network model components. Some discussions of the electronic neural network description may be found in Refs. [9,16]. Several neural network specification languages have been proposed, such as Aspirin [10], PlaNet v.5 [12], AXON [6], CuPit [7], EpsiloNN [17], CONNECT [8] and Nspec [3]. In general, these languages are the specialized programming languages designed to facilitate the neural networks simulation process. They are based on notations, which are close to the high-level programming languages like C++, and use similar methodology for the neural system determination in terms of objects, classes, variables and functions. Such a description can be used within the specialized simulation environment, transformed into the code for common programming languages or implemented on the specialized neural simulators (possible with parallel processing capability).

These languages in their original forms cannot be considered as the exchange languages for neural networks due to the following reasons. Firstly, they are aimed at the description of the simulation process rather than at the exchange of networks. Secondly, they are supported only by their underlying simulation systems. Their usage is possible only within the appropriate environments and needs specialized compilers. Thirdly, most of these languages are intended for the neural network definition alone, without taking into account the specification of the data dictionary, pre- and postprocessing, etc.

Other types of language is presented in Ref. [2] as a part of the Predictive Model Markup Language (PMML). Description with PMML provides the extensive information about the project, the data dictionary and sample statistics. But it supports only the backpropagation neural networks with a few fixed types of neurons and uses a limited set of simple preprocessing techniques.

A few articles are devoted to the formalization and standardization of the neural systems. Fiesler and Caulfield [5] propose the hierarchical specification of known neural networks and introduces a mathematical notation for their description. Smith [16] and Atencia et al. [1] use more common mathematical constructions for the definition of the neural structures and dynamic. In Ref. [18], an extended analysis of the neural structures including the biological neural networks is given. At great length, all components of the neural network simulation system are described in Ref. [11]. But the analysis of those works has shown that the results they presented cannot be directly applied to the neural network model exchanging.

3. Overview on NNML

In this work, we treat the neural solution as a computational model, which is used to solve a prediction or classification problem. Mapping of input signals to output ones in such a model is constructed according to the methods of the neural network simulation. It is significant that a *trained* neural network model is treated as an object for exchange. A neural net is considered as a static object with the fixed structure and parameters. In other words, a complete computational scheme for obtaining the target (output) variables on the basis of the independent (input) variables is transferred. The main purpose of this format is to represent all relevant information essential for an unambiguous and exact reproduction of such a computational scheme by any neural simulator. That does not exclude further improving of this model by a simulator with its own training methods.

The development of NNML is based on the following assumptions. Firstly, we consider that the significant parts of each neural model description are:

- The problem and model purpose
- Data dictionary
- Data preprocessor
- Neural network
- Postprocessor
- Auxiliary information about the model.

The information about all these components is essential for the correct reconstruction of the computational model of the neural network. As shown in Ref. [11], a neural network can not be correctly reproduced without the information about the environment in which it was created. The description of the neural method for solving any practical problem, along with the definition of the neural network, must contain information about the structure of the data dictionary, the preprocessing methods for each input data field, the postprocessing methods for the network output signals and the extra information about the current model.

Secondly, we consider a neural network in broad sense, with minimum restrictions on topology and neuron's functionality. Suppose a neural network is a system of interconnected processing units, which may be represented as a directed graph. Nodes in such a graph designate neurons and branches mark connections between neurons. The directions of the branches define the signal transfer directions in the net. Each unit performs a particular mathematical transformation on its inputs and may have its own parameters. It maps the input vector of the real numbers to the output scalar real number. Weights of connections we consider as the parameters of the neuron function.

Further, we consider a preprocessor and a postprocessor also as sets of processing units, which have descriptions similar to neurons. The preprocessor receives input signals from an input data vector. Each preprocessing unit corresponds to the particular neural net input. Each postprocessor unit describes a single target parameter of the problem neural model is purposed to. Thus, the neural model has a fully modular structure where each object interacts with other objects only by passing its output signals to them (see Fig. 3).

In essence, NNML description is a sufficiently structured document formed by the hierarchy of the nested sections with the root element named `neural model`. Each of the high-level sections describes one basic aspect of the neural solution. The neural model description in NNML has the following main sections: Header, Data, Preprocessing, Neural Network, Postprocessing and Training Process Description.

In terms of XML, the section is an element, which is a basic unit of description. An element is formed by a couple of named tags and may contain any data or

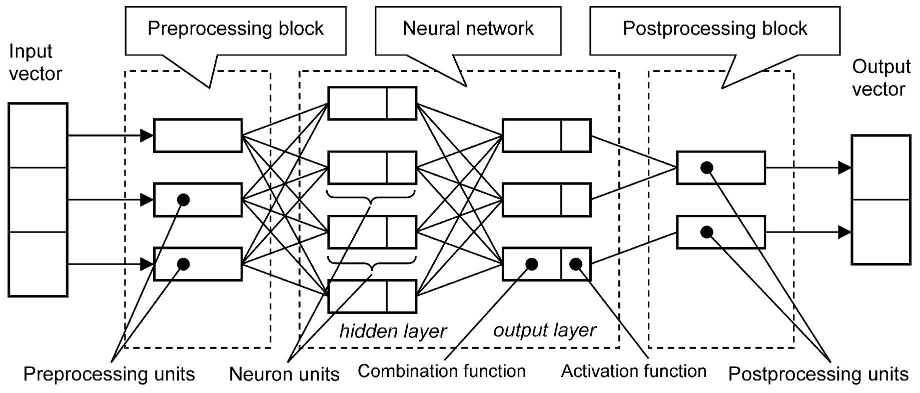

Fig. 3. Representation of neural model as a system of connected processing units.

child elements. The description of the element may be extended by a set of attributes.

The structure and contents for each of sections have been developed on the basis of the hierarchical neural model decomposition with taking into account the existing works in this field (Refs. [3,5,11,15], in particular).

Further, we describe the main NNML features with a simple example of a neural network.

4. A simple example of a neural solution

Here, we describe a simple example of a neural network, which will help us to illustrate the features of NNML. We consider the task of classification into two classes. The underlying function is logical and presents a particular case of a parity problem for two variables, known as "exclusive or" or XOR problem. Its true table is given in Fig. 4. It will also serve as a training data set for the neural network. The data dictionary of this task contains three variables: x_1 and x_2 as the input variables, and y as an output one. Let x_1 and x_2 be real numbers, varying in the range [0,1] and y a discrete one and may be in two states {1,2}, each of which designates a particular class. The input and output variables are scaled into [– 1,1] interval before they are submitted to the network. The output of the network is interpreted as follows: if the signal of an output neuron is in [– 1, – 0.2], there is a first class, or if it is in [0.2,1], there is a second class, and if it is in [– 0.2,0.2], the class is not defined.

The structure of the neural network and its parameters are shown in Fig. 4. The actual values of the weights are indicated close by the corresponding connections.

5. A framework for description of processing units

The basic components of model—preprocessor, neural network, postprocessor—are described as sys-

The XOR problem

x_1	x_2	y
1	1	2
1	0	1
0	1	1
0	0	2

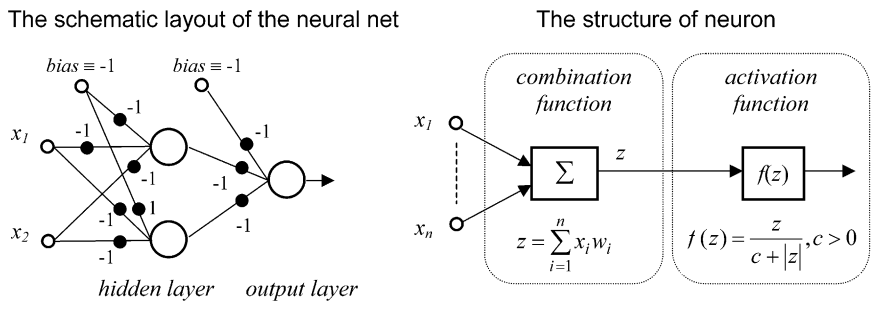

Fig. 4. An example of feedforward neural network.

tems of interconnected processing units, and representation of their underlying mathematical transformations is the key for NNML. It is necessary to note that the development of the rational framework for a mathematical component definition is not a trivial task. On the one hand, the representation must be very flexible and allow to describe as much mathematical functions as possible, even if they are not actually used in the neurocomputing nowadays. On the other hand, there are particular sets of functions, which are the most popular in the field of the neural network simulation and it would be useful to simplify their application.

A proposed framework for the representation of processing units consists of the following. There are basic (elementary) operands of two types: scalar number (constant) and reference (connection) to the output of one of the processing units. All transformations are applied to those operands.

(1) The number is set by number tag, which has an attribute name to designate the argument semantic, for example “weight” or “bias”. In general, this tag contains a real number. All numerical parameters of the model—weights of neurons, parameters of activation functions, parameters of pre- and postprocessing blocks—are written down by this tag. An example of the NNML code for description of weight of connection between neuron N_{11} and input x_1 (see Fig. 4) is:

```
<number name = “weight” >
– 1 < /number>
```

(2) The connection is set by an empty connect tag. It has the following attributes: object_type which contains the name of the signal source type (a name of one of next tags: preprocessing_field, neuron, data_field), object_id which is the identification number of the particular object of the specified type, nn_layer which is the id of layer for object of neuron type. connect tag serves for the description of the connections between the processing units. The fact that the neuron N_{21} connected with the neuron N_{11} would be described as:

```
<connect object_type = “neuron”
object_id = “1” nn_layer = “1”/>
```

Really, connection also represents a number, which is the result of functioning of the particular processing unit. But at design time, we do not know the values of outputs of the processing units because they depend on the specific values of the input parameters. Therefore, connections are the parameters, computed in run time, when numbers are the parameters determined in design time.

There are three levels of function specification:

• Using the basic functions. The set of the elementary functions is introduced to define various mathematical transformations. It includes unary and n-ary arithmetic operations (sum, multiplication, raising to power, division, capture of absolute value, etc.), trigonometric operations, logarithmic operations (logarithms, exponent), relations (more, less, equally, etc.), if–then condition, matrix operations (addition, scalar product, etc.) and piecewise function. These operations may be performed on the basic operands, as well as on the results of basic functions. It means that the tags of the basic functions can contain other basic functions or elementary operands. The analysis of such expressions is carried out by the construction of a computational tree. Terminal nodes of this tree always should contain one of elementary operands. The hierarchy of nesting tags determines the priority of operations: all “internal” functions are calculated first without taking into account their type. To denote the above type of function definition, we use explicit_record tag. For example, the combination function of the neuron N_{21} may be described as following piece of code:

```
<explicit_record> < sum>
< mult>
< connect object_type = “neuron”
object_id = “1” nn_layer = “1”/>
< number name = “weight”>
– 1 < /number>
< /mult>
< mult>
< connect object_type = “neuron”
object_id = “2” nn_layer = “1”/>
< number name = “weight”>
– 1 < /number>
< /mult>
< mult>
< number name = “bias”>
– 1 < /number>
```

```
<number name="weight">
−1</number>
</mult>
</sum></explicit_record>
```

• Using the predefined functions. The set of the predefined functions is intended to decrease the length of the description. It includes the most popular functions in the neural simulation, in particular: preprocessing functions (normalization, coding), neuron combination functions (adaptive summator, square summator, Euclidean distance, etc.), activation functions (various kinds of sigmoids, the threshold function, linear and piecewise-linear function, exponent, etc.), methods of postprocessing (denormalization, "winner takes all"). Description of all the predefined functions may be found on the NNML site [13]. For example, the activation function of the neuron given in Fig. 4 may be described with the predefined function rational_sigmoid:

```
<rational_sigmoid>
<connect object_type="combination function"/>
<sigmoid_slope>0,1</sigmoid_slope>
</rational_sigmoid>
```

• Functions defined by the user. The user can introduce his own functions. In the body of the model description, the user_function tag is used with the parameters: name—the name of user function, source—the Internet address, which provides the information about the contents of the transformation. Within this tag, only special tags arg is allowed. arg tags have attribute id and can contain the basic operand tags as a number or connect. By default, semantics of transformations given by the user is not determined in any way.

6. Contents of the main NNML sections

6.1. Header

Element header mostly contains human-intended information. It includes three subsections: problem, creator and project. Subsection problem describes the problem, which the solving current neural model is intended for. The description fields are: name of domain, name of the specific task and the task description with explanation of essential detail. Subsection creator contains the information about the neural simulator, which produced current model (name and version), author's name, contact information and copyright. Subsection project includes the name of the current model, identification field, date and time of model creation and comments of the author. For neural net presented above, this section would be like this:

```
<header>
<problem>
<task_name>XOR problem
</task_name>
<domain>Logic computation</domain>
<task_description>Simple illustration of
nonlinear neural net mapping
</task_description>
</problem>
<creator>
<simulator name="NNET"
version="0.2"/>
</creator>
<project>
<datestamp>25.04.2001</datestamp>
<model_id>1</model_id>
<model_name>xor2_01
</model_name>
<model_comment>multilayer
perceptron for separation into two classes
based on xor(x1,x2) function
</model_comment>
</project>
</header>
```

6.2. Data

Section data contains a description of the model's data dictionary. It is divided into data fields. A single data field is determined with the following attributes:

- id—unique identifier for each data field,
- name of the data field that can be displayed in the dialog with the user,
- type of underlying variable: continuous, nominal, ordered,

- way of data field usage in the current model: as an input, an output or an auxiliary field.

In addition to attributes, the data field contains data_field_properties tag. This tag describes the underlying variable properties, for example, the permissible range of variable's values. For the discrete (nominal or ordered) variables, there is a description for each of its discrete states: id and the name of a discrete state, the interval of numeric values assigned to the discrete state. For input variable x_1 from our simple example, we can use this piece of code:

```
<data_field id="1" name="x1" type="real" usage="input">
<comments>continuous input variable
</comments>
<datafield_properties>
<value_range min="0" max="1" brackets="11"/>
</datafield_properties>
</data_field>
```

6.3. Preprocessing

Applying of preliminary transformations to the data sample is a common practice for the neural net training. Section preprocessing is intended to the definition of methods, which are applied to the raw input data to produce the neural network input signals. Most of used preprocessing techniques are simple transformations of the type "scalar-to-scalar" (various normalization and scaling transformations) or "scalar-to-vector" (various types of coding). In the latter case, several preprocessing output signals are corresponded to a single input data field. However, more sophisticated preprocessing techniques may be used [11].

To provide flexibility of the preprocessing description, we specify a separate processing unit for each signal that preprocessing block submits to a neural network. Each preprocessing unit receives signals from one or more data fields and generates an output signal, which is submitted to the neural network. To describe linear transformation for variable x_1 from the example, we should use the predefined function scaling:

```
<preprocessing>
<preproc_field id="1">
<scaling>
<source_min>
<number>0</number>
</source_min>
<source_max>
<number>1</number>
</source_max>
<source_current>
<connect object_type="data_field"
object_id="1"/>
</source_current>
<target_min>
<number>-1</number>
</target_min>
<target_max>
<number>1</number>
</target_max>
</scaling>
...
</preprocessing>
```

6.4. Neural network

Section neural_network contains the description of the neural network and its properties. The main purpose of this description is to simplify the automatic recovering of underlying mathematical procedures. Because of the lack of the standard terminology (and the standard ways of decomposition) for the neural networks, we have decided not to use terms like "synapse" or "axon". On the contrary, the network structure is recorded with the common mathematical operations.

The neural network is defined as a scheme consisting of the connected processing units or neurons, which exchange signals via links. There is a tag layer, which groups neurons (a fully connected network has one layer). Each neuron is represented as a couple of serially connected functions. The first function combines the inputs of neuron with the weights to yield a single value to which the activation function is applied. This is called a neuron combination function (in terms of Sarle [15]). The second function represents a neuron activation function and usually performs a nonlinear transformation of the combination function output. They are specified in the tags combination_function and activation_function, respectively.

There are no special constraints on the structure of these functions. This means that a neuron may have any functioning, which may be defined with the combination of arithmetic operations or/and elementary classical functions. Varying these functions we can get different types of neurons. For example, using the scalar product of inputs and weights as a combination function and varying an activation function, we can describe some important types of the neuron units, used in multilayer perceptrons, e.g. the neurons with various sigmoid functions, with threshold, linear and piecewise-linear transfer functions. The application of various types of combination functions permits to describe such kinds of neurons, as radial basis functions (Euclidean distance), neurons of high degrees (high-order polynomial), etc.

An output signal of the combination function is automatically submitted to the activation function of the same neuron. Each neuron may be connected with other neurons (and with itself), the preprocessing units and the data fields. Weights and connections of neuron are set as the arguments of the combination function. Thus, the network structure is determined implicitly by the references in the combination function descriptions. This approach lets to describe the networks with any topology. The neuron functioning may be specified with the predefined NNML functions or the user defined functions (see above). This clears the way to the specification of the heterogeneous neural networks.

The length of the network description may be reduced. If all neurons in a layer have the same type of combination or activation function, its description may be placed as the first tag within layer tag.

Along with the neural network structure, tag neural_network contains the rules for the network dynamic.

Subsection neural_network_properties includes the common information about the current network: the type of the structure (feedforward, feedback), the neural paradigm (e.g. multilayer perceptron, the radial basis function network) and the comments of the creator. Example of the neural net structure description for the neural net presented in Fig. 4 is:

```
<neural_network>
<nn_structure>
<layer id="1">
<neuron id="1"
<combination_function>
<explicit_record>
...
</explicit_record>
</combination_function>
<activation_function>
<rational_sigmoid>
...
</rational_sigmoid>
</activation_function>
</neuron>
<neuron id="2">
...
</neuron>
</layer>
<layer id="2">
<neuron id="1">
...
</neuron></layer></nn_structure>
```

6.5. Postprocessing

Typically, a neural net produces a set of real numbers, which needs to be correctly interpreted. The postprocessing section is intended to provide the information about the methods of the substantial answer delivering on the basis of the network target signals. The methods of the network output signal transformation are defined in the same way as the preprocessing methods. Usually, denormalization (for regression), or the rule "winner takes all" (for the classification) are used, but more complex methods are also allowable.

The neural network could return several answers. Therefore, methods of postprocessing are grouped by postproc_field tags, each of which corresponds to one of the model target parameters (not to one of the network output neurons). The tag postproc_field contains the reference to the appropriate target data field described in the data section. As input signals for postprocessing blocks, the signals of the network output neurons are used.

Each target field may include tag verbalization. It is intended for the association text information with certain values (groups of values) of target parameter, for example: if the target parameter is in [2.5,3.5], then the network answer is "satisfactorily".

Postprocessing for our example may be written out with the piecewise function:

```
<postprocessing>
<postproc_field id="1"
datafield_id="3">
<postproc_function>
<explicit_record>
<piecewise>
<case> <case_value>1</case_value>
<case_arg><connect
object_type="neuron"
object_id="1" nn_layer="2"/>
</case_arg>
<case_interval min="-1" max="-0,2"
brackets="11"/>
</case>
<case> <case_value>not defined
</case_value>
<case_arg>
<connect object_type="neuron"
object_id="1" nn_layer="2"/>
</case_arg>
<case_interval min="-0,2" max="0,2"
brackets="00"/>
</case>
<case> <case_value>2</case_value>
<case_arg><connect object_type="neuron"
object_id="1" nn_layer="2"/></case_arg>
<case_interval min="0,2" max="1"
brackets="11"/>
</case></piecewise>
</explicit_record>
</postproc_function>
</postproc_field>
</postprocessing>
```

7. Providing of NNML extensibility

Providing of the openness and extensibility of the language is a vital issue. At the present stage of development, the following features for NNML extension are supported:

- Using of the functions determined by the user (above mentioned).
- Introducing new elements of the description. For introducing of the user-defined elements serves insertion tag. It has attributes: type, name, author and source. The type can be equal to "parameter" or to "construct" (by default). "Parameter" value specifies that the insertion tag designates the element of the description with the type of pair "name of parameter–value of parameter". Then the name attribute is interpreted as the name of the parameter, and the contents of tag—as its value. For example, if one wants to extend the description of the data field with the value of its arithmetic mean, he may use following code:

```
<insertion type="parameter"
name="mean">0,5</insertion>
```

If the type is specified as "construct", then the tag insertion can have any contents, for example, the structure of nested tags. The rules of such tag analysis are not determined and depend on the features of the particular simulation systems.

8. Processing of NNML documents

As NNML is XML-based language, the rules for NNML files (documents) formation are set as Document Type Definition (DTD). DTD for the current version NNML can be found in Ref. [13]. DTD is also used for NNML document validation.

Processing of NNML documents is carried out by standard software modules—XML parsers. The creation of the NNML document contains the following steps:

- The neural network is generated and trained by means of the neural network simulator.
- The hierarchical structure of the model description is created by the interface module on the basis of internal representation.
- Methods of any XML parser are called, object tree of the model are constructed and the NNML file is generated.

For loading of the ready NNML file, actions are performed in the reverse order.

Thus, the main task for the software developers is the creation of the appropriate interface modules between the internal representation of neural networks in different simulation systems and XML-oriented libraries or modules. The fact that most up-to-date software developing platforms are supplied with XML-friendly tools makes this task simpler.

9. Applications of NNML

Mainly, NNML could be considered as an interface between various software systems concerning neural networks. We suppose that NNML allows the specialization of the neural network simulation programs: separation of neural networks generators, interpreters, tools for visualization and knowledge extraction (see Fig. 2). This makes NNML particularly suited for exchanging neural solutions within the large information systems [14]. It could facilitate the effective documenting and storing models, as well as makes the introduction of the neural network facilities into the existing information systems easier.

NNML can also be applied to the distribution of the neural network models. It is possible to implement the use of remote machines for producing of the neural network models "by request", maintaining the global archives to share the neural network solutions for various problems.

As the XML-based language, NNML could be easily applied for the introduction of the neural network features in the World Wide Web applications or the development of the neural-based Web services. With the help of NNML, the developers could integrate the powerful simulation systems like Matlab with Web interface, where the NNML interpreter could be implemented, for example, with the eXtensible Stylesheet Language (XSL) technology.

10. Conclusions

In this article, we have presented an XML-based language for the description of the neural network models–NNML. It allows to describe the neural network solution completely, including the data dictionary, pre- and postprocessing, details of the structure and parameters of the neural network and the auxiliary information. The proposed framework provides features to write down a wide range of pre- and post-processing procedures, and describe the neural network of any topology with the heterogeneous elements. Most widespread neural networks (such as multilayer perceptrons and radial basis function networks) can be described in a compact way. A user has an ability to use any functions for designing of neurons. NNML provides an open and extensible description of the neural models. It is designed to simplify the exchange of the neural solutions among the simulation systems. In addition, it could be a base for discussions among the researches about the particular neural models.

It is possible to find out the last version of NNML Document Type Definition and other related materials on the NNML site [13].

Acknowledgements

This research was partially supported by the Russian Foundation for Basic Research, grants 01-01-01046 and 02-01-06427.

References

[1] M.A. Atencia, G. Joya, F. Sandoval, A formal model for definition and simulation of generic neural networks, Neural Processing Letters, vol. 11, Kluwer Academic Publishers, Dordrecht, 2000, pp. 87–105.

[2] The Data Mining Group, PMML 1.1, Neural Network, http://www.dmg.org/html/neuralnetwork.html.

[3] G. Dorffner, H. Wiklicky, E. Prem, Formal neural network specification and its implications on standardization, Technical report OFAI TR-93-24, Austrian Research Institute for Artificial Intelligence, 1993.

[4] Extensible Markup Language (XML) 1.0, W3C Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210.

[5] E. Fiesler, H.J. Caulfield, Neural network formalization, Computer Standards and Interfaces 16 (3) (1994) 231–239.

[6] R. Hecht-Nilsen, Neurocomputing, Addison-Wesley, Reading, MA, 1990.

[7] H. Hopp, L. Prechelt, CuPit-2: a portable parallel programming language for artificial neural networks, in: A. Sydow (Ed.), Proc. 15th IMACS World Congress on Scientific Computation, Modelling, and Applied Mathematics, vol. 6, Wissenschaft and Technik Verlag, Berlin, 1997, pp. 493–498.

[8] G. Kock, N.B. Serbedzija, Simulation of artificial neural networks, Systems Analysis–Modelling–Simulation (SAMS), vol. 27 (1), Gordon & Breach Science Publishers, Reading, UK, 1996, pp. 15–59.

[9] G. Kock, N.B. Serbedzija, Artificial neural networks: from

compact descriptions to C++, in: M. Marinaro, P.G. Morasso (Eds.), Proc. of the International Conference on Artificial Neural Networks, Springer-Verlag, Berlin, 1994, pp. 1372–1375.
[10] R.R. Leighton, The Aspirin/MIGRANES Neural Network Software, Users Manual, MITRE, Reston, VA, 1992.
[11] E.M. Mirkes, Neurocomputer, Project of Standard, Nauka, Novosibirsk, 1998 (in Russian).
[12] Y. Miyata, A User's Guide to PlaNet Version 5.6—A Tool for Constructing, Running, and Looking in to a PDP Network, Computer Science Department, University of Colorado, Boulder, 1991.
[13] Neural Network Markup Language Home Page, http://www.nnml.alt.ru.
[14] D.V. Rubtsov, Development of technology of artificial neural networks application in applied information systems, PhD thesis, Department of Mathematics, Altai State University, Barnaul, 2000 (in Russian).
[15] W. Sarle, Frequent asked question on neural network, ftp.sas.com/pub/neural/FAQ.html.
[16] L. Smith, Using a framework to specify a network of temporal neurons, Technical report, University of Stirling, 1996.
[17] A. Strey, EpsiloNN—a tool for the abstract specification and parallel simulation of neural networks, Systems Analysis–Modelling–Simulation (SAMS), vol. 34, Gordon & Breach Science Publishers, Reading, UK, 1999, p. 4.
[18] A. Strey, A unified model for the simulation of artificial and biology-oriented neural networks, Proc. of the International Workshop on Artificial Neural Networks, vol. 2, Springer-Verlag, Berlin, 1999, pp. 1–10.

Denis Rubtsov is a Professor in the Department of Mathematics at the Altai State University. He received his Master's degree from the Computer-Aided Design Faculty of the Altai State Technical University, Russia, in 1996, and his PhD in Computer Science from the Altai State University, Russia, in 2000. His PhD thesis is devoted to the development of the technology of artificial neural networks application in the applied information systems. His current research interests include topics of artificial neural networks, knowledge-based systems and applications of XML.

He also has been working at the Altai State Center for Real Estate Registration as a main specialist at the Computer Department since 2000.

Sergey Butakov is an Associate Professor in the Department of Information Systems at the Altai State Technical University, 49-5 Lenina St., Barnaul 656099, Russia. His e-mail address is swb@agtu.secna.ru. He received his Master's degree from the Altai State Technical University, Russia, and his PhD in Computer Science from Altai State University, Russia, in 2000.

He has been working at the Altai State Technical University since 2000. His current research activities include applied intelligent systems and hybrid expert systems. He is author and coauthor of more then 20 papers in these fields.