NEURAL NETWORK FOR PLC

L. Körösi, J. Paulusová

Institute of Robotics and Cybernetics,

Slovak University of Technology, Faculty of Electrical Engineering and Information Technology

Abstract

The aim of the proposed paper is to present a neural network (NN) conversion between Schneider Electric PLC and Matlab, which represents joining the simulation tool with a real control system used in industry. The conversion of the NN structure is bidirectional and it simplifies the time consuming manual conversion. The conversion is realized via XML file. The XML file is generated in Matlab using GUI or in Unity Pro XL PLC programming software. The XML file contains the pre-allocated tags (number of input neurons, hidden neurons, output neurons, weights, biases, etc.) needed to build up the NN structure in PLC. The proposed solution has been validated on simulated data.

1 Introduction

The theory of neural networks is based on neurophysiological knowledge of the brain. Artificial neural networks are modeling the structure of the human nervous system. The applicability of neural networks is based on some basic properties of the neural network. One and the most important is the fact that the neural network is a universal function approximator. Neural networks are suitable in the fields of pattern recognition and classification, recognition time sequence in signals, mathematical statistics, optimization problems, natural language processing, data compression, etc. Programmable logic controllers (PLC) are industrial controllers with large number of instructions containing also the instructions for PID controllers. In several cases there are presented also the other control structures like Internal Model Control, etc. but there are missing neural networks for modeling and control of nonlinear processes. In this paper the perceptron neural network is used for nonlinear process modeling and prediction. The design of the neural network can be made directly in PLC and also in Matlab using Neural Network Toolbox.

2 NN structure in Matlab

Matlab neural network toolbox contains many commands for creating and training neural network structures. One of them is *newff* which creates a feed-forward backpropagation network. The syntax of this command is

net = *newff*(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF),

where PR is matrix of minimum and maximum values for all input elements, Si is the size of *i*-th layer, for Nl layers, TFi is the transfer function of *i*-th layer, TF is the backpropagation network training function, BLF is the backpropagation weight and bias learning function and finally PF is the performance function. We consider a standard tree layer NN. In this case the following parts of the NN structure are important:

net.IW - containing 1 input weight matrix

net.LW - containing 1 layer weight matrix

net.b - containing 2 bias vectors

The weights between input and hidden layer are stored in net.IW{1}. The weights between hidden and output layer are stored in net.LW{2}. Biases for the hidden layer are in net.b{1} and for the output layer in net.b{2}. Number of neurons in each layer is given by the size of the weight matrixes. Apart from the weights and biases the transfer functions should be known. The complete information about layers are stored in

net.layers - is a structure defining layers where net.layers{1} is the hidden layer and net.layers{2} is the output layer. The transfer functions are stored in net.layers{1}.transferFcn and net.layers{2}.transferFcn.

The training parameters (number of epochs, goal – error to reach, etc.) of the NN are stored in the net.trainParam structure. These are also needed for the import and export of the NN structure.

3 NN structure in PLC

The structure of the NN was designed in Unity Pro XL software. It's a configuration, programming and diagnostic software for Schneider Electric PLC's. The structure of the NN was designed as Derived Function Block (DFB). It's a defined function block by user with inputs, outputs, local variables, global variables and programs in different programming languages. The advantage of DFB is simplifying the programming and memory saving because the user code is stored only ones in the PLC's memory and it can be used several times with instances (activations). The graphical design of NN DBF in Ladder Diagram programming language is shown in Fig. 1.

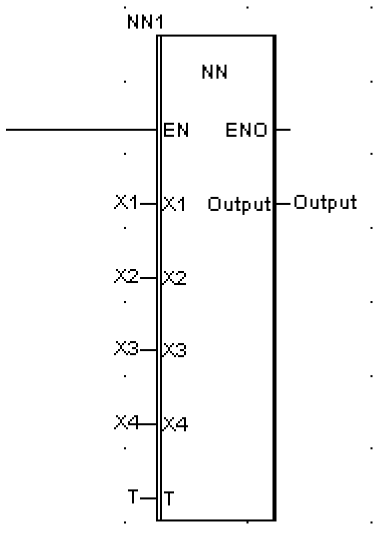
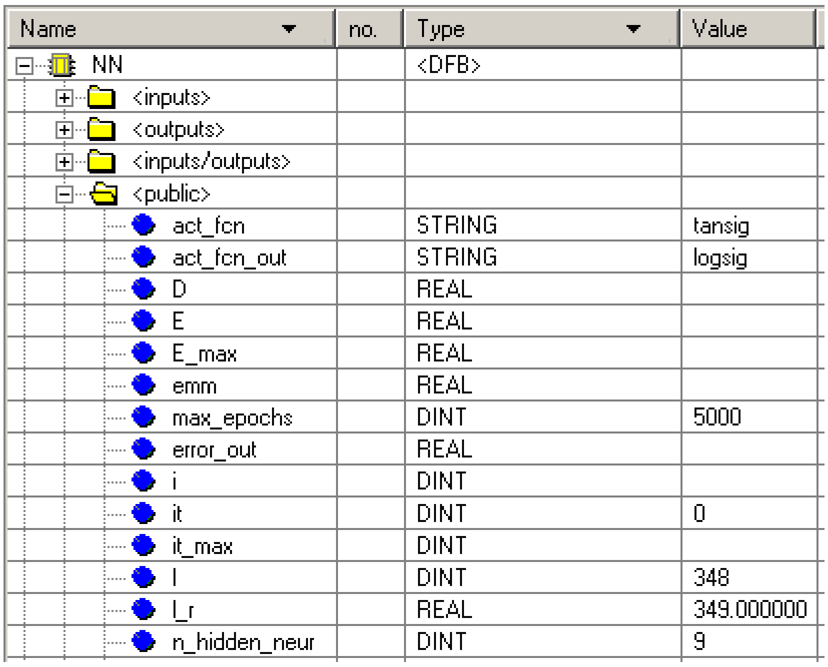


Figure 1: Example of NN DBF

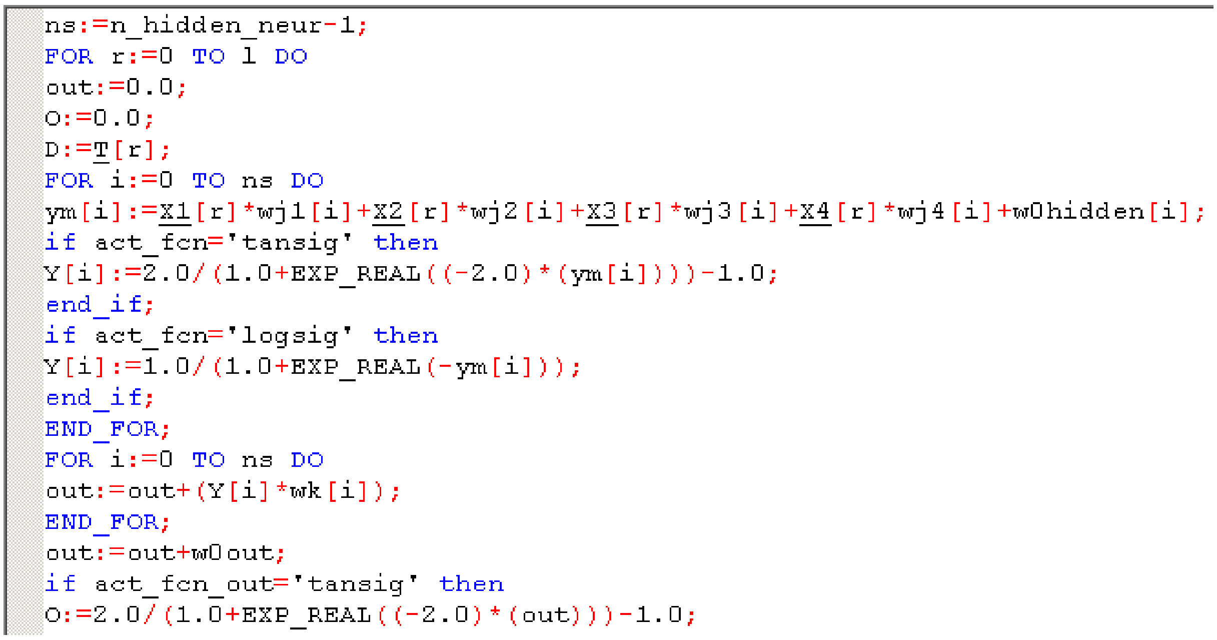
The NN DFB has 5 inputs and 1 output. The first four inputs (X1, X2, X3, X4) are linear neurons and the last one (T) is the sample time for inner computation. The DFB has 1 output neuron (Output). The learning method with parameters, the number of hidden neurons, activation function types, weights, biases etc. is stored in global (public) variables. Part of the variables is shown in Fig. 2.



Name	no.	Type	Value
NN		<DFB>	
<inputs>			
<outputs>			
<inputs/outputs>			
<public>			
act_fcn		STRING	tansig
act_fcn_out		STRING	logsig
D		REAL	
E		REAL	
E_max		REAL	
emm		REAL	
max_epochs		DINT	5000
error_out		REAL	
i		DINT	
it		DINT	0
it_max		DINT	
l		DINT	348
l_r		REAL	349.000000
n_hidden_neur		DINT	9

Figure 2: Example of NN DFB global variables

The NN output calculation and learning algorithm were designed in Structured Text programming language. This language is suitable for mathematical equations, program parts that are repeated (commands like for; repeat-until; do-while; etc.). The preprocessing of input signals to the hidden layer is shown in Fig. 3.

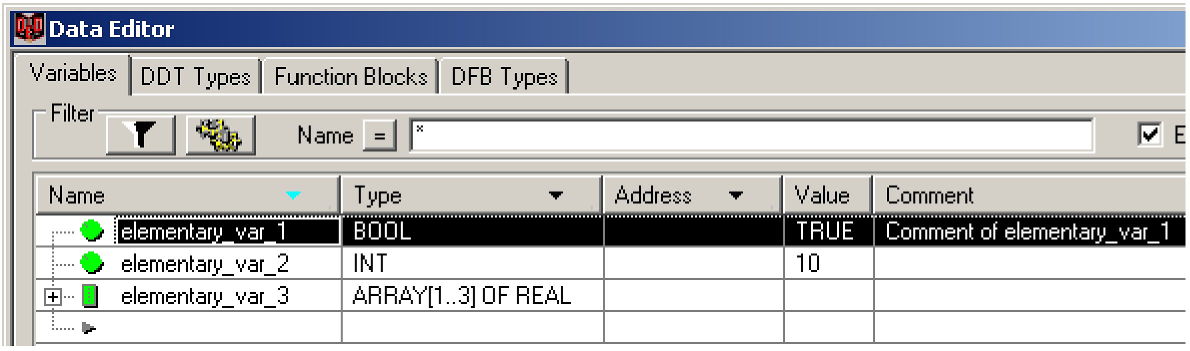


```
ns:=n_hidden_neur-1;
FOR r:=0 TO l DO
out:=0.0;
O:=0.0;
D:=T[r];
FOR i:=0 TO ns DO
ym[i]:=X1[r]*wj1[i]+X2[r]*wj2[i]+X3[r]*wj3[i]+X4[r]*wj4[i]+w0hidden[i];
if act_fcn='tansig' then
Y[i]:=2.0/(1.0+EXP_REAL((-2.0)*(ym[i])))-1.0;
end_if;
if act_fcn='logsig' then
Y[i]:=1.0/(1.0+EXP_REAL(-ym[i]));
end_if;
END_FOR;
FOR i:=0 TO ns DO
out:=out+(Y[i]*wk[i]);
END_FOR;
out:=out+w0out;
if act_fcn_out='tansig' then
O:=2.0/(1.0+EXP_REAL((-2.0)*(out)))-1.0;
```

Figure 3: Part of source code of NN output computations

4 Structure of the XML file

XML file was used to exchange the information between PLC and Matlab. In Unity Pro XL the export of DFB activation was needed while in Matlab GUI was designed. An example of elementary variables created in Unity Pro XL is shown in Fig. 4.



Data Editor

Variables | DDT Types | Function Blocks | DFB Types

Filter Name = *

Name	Type	Address	Value	Comment
elementary_var_1	BOOL		TRUE	Comment of elementary_var_1
elementary_var_2	INT		10	
elementary_var_3	ARRAY[1..3] OF REAL			

Figure 4: Example of elementary variables

The XML file for the example above is the following:

First line of the file contains information about the version and the encoding of the XML file defined between <?xml ?> tags.
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

Body of the XML file is defined with paired tag <VariablesExchangeFile> </VariablesExchangeFile>. Inside this tag is the detailed definition of the project structure.
<VariablesExchangeFile>

The file header contains information about company, date, time, etc., but the most important is the document type definition (DTD) version.

```
<fileHeader company="Schneider Automation" product="Unity Pro XL V8.0 - 131118"
dateTime="date_and_time#2014-10-13-10:44:48" content="Variable source file"
DTDVersion="41"></fileHeader>
```

In the content header is saved the basic information about the project like project name and version.

```
<contentHeader name="Project" version="0.0.000"></contentHeader>
```

Between <dataBlock> paired tag are defined the variables with their properties.

```
<dataBlock>
```

Symbolic variable is uniquely defined with its name and data type:

```
<variables name="elementary_var_1" typeName="BOOL">
```

Additional information is the variable comment, initial value or others:

```
<comment>Comment of elementary_var_1</comment>
<variableInit value="TRUE"></variableInit>
```

The variable declaration ends with paired tag.

```
</variables>
```

In the example below we can see that the emelentary_car_2 has no comment definition (the paired tag <comment> is missing):

```
<variables name="elementary_var_2" typeName="INT">
<variableInit value="10"></variableInit>
</variables>
```

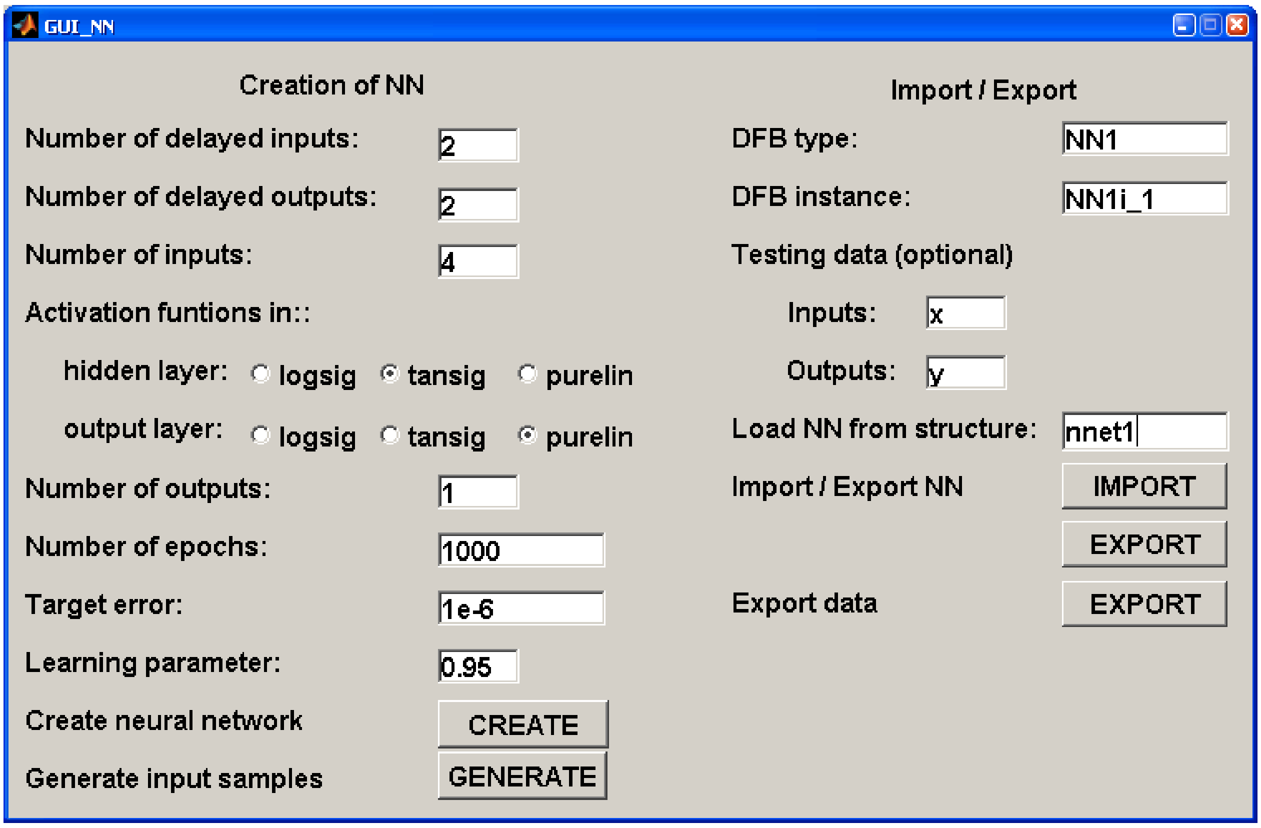
Array variables with initial values are described with <instanceElementDesc> and <value> paired tags. The name inside this paired tags are the indexes of the array variable.

```
<variables name="elementary_var_3" typeName="ARRAY[1..3] OF REAL">
<instanceElementDesc name="[1]">
<value>12.2</value>
</instanceElementDesc>
<instanceElementDesc name="[2]">
<value>13.3</value>
</instanceElementDesc>
<instanceElementDesc name="[3]">
<value>14.4</value>
</instanceElementDesc>
```

Finaly the rest of the paired tags end the XML structures of the example above.

```
</variables>
</dataBlock>
</VariablesExchangeFile>
```

The graphic user interface (GUI) design in Matlab is shown in Fig. 5. With this GUI it's possible to import and export 3 layer NN structures created with *newff* command, from and to PLC.


GUI_NN

Creation of NN

Number of delayed inputs: 2

Number of delayed outputs: 2

Number of inputs: 4

Activation funtions in::

hidden layer: logsig [✓ tansig] purelin

output layer: logsig tansig [✓ purelin]

Number of outputs: 1

Number of epochs: 1000

Target error: 1e-6

Learning parameter: 0.95

Create neural network CREATE

Generate input samples GENERATE

Import / Export

DFB type: NN1

DFB instance: NN1i_1

Testing data (optional)

Inputs: x

Outputs: y

Load NN from structure: nnet1

Import / Export NN IMPORT EXPORT

Export data EXPORT

Figure 5: Guide for NN conversion (from/to PLC) in Matlab

By choosing the NN Export, the XML file will be created containing the NN DBF structure (inputs, outputs, global variables and also the program code in structured text). Example of the export subroutine is shown below:

```
fprintf(fid,'%s \n','<variables name="n_hidden_neur" typeName="DINT">');
fprintf(fid,'%s','<variableInit value="');
n_skryt_neur=net.layers{1}.size;
fprintf(fid,'%d',n_skryt_neur);
fprintf(fid,'%s \n','"></variableInit>');
fprintf(fid,'%s \n','</variables>');
```

5 Conclusion

The proposed paper presented a bidirectional NN structure conversion between Matlab and Schneider Electric PLC using XML. The conversion was tested and verified on simulated data and it can be concluded, that the presented approach speeds up the NN development for PLC systems implemented in Unity Pro XL programming software. Other advantage of the proposed approach is using IEC XML file structure for ensuring compatibility between other PLC supporting this standard.

ACKNOWLEDGMENT

This paper was supported by the Slovak Scientific Grant Agency VEGA, Grant no. 1/2256/12.

References

[1] BEŇUŠKOVÁ, Ľ. *Umelé neurónové siete*. [online], 2001, Available on the Internet:

http://ii.fmph.uniba.sk/~benus/books/UNS_revised.pdf

[2] TUCKOVÁ, J. *Úvod do teorie a aplikací umelých neuronových sítí*. Praha: ČVUT, 2003. 103p. ISBN 80-01-02800-3.

[3] KVASNIČKA, V. - BEŇUŠKOVÁ, Ľ. - POSPÍCHAL, J. - FARKAŠ, I. – TIŇO, P. - KRÁĽ , A. *Úvod do teórie neurónových sietí* [online], 2010. Available on the Internet:

http://ics.upjs.sk/~novotnyr/home/skola/neuronove_siete/nn_kvasnicka/Uvod%20do%20NS.pdf

[4] VOLNÁ, E. *Neurónové sítě 1* [online].Ostrava, 2008, Available on the Internet:

http://www1.osu.cz/~volna/Neuronove_site_1_informace.pdf

[5] Schneider Electric: *Unity Pro I/O Management Block Library*, 2012, Available on the Internet:

http://www.global-download.schneider-electric.com/mainRepository/Index.nsf/DisplayProductDocumentation?OpenAgent&L=EN&App=Schneider&~EN~OT&p=548%26c=43,201,203,204,304

[6] Schneider Electric: *Unity Pro Operating Modes*, 2012, Available on the Internet:

http://www.global-download.schneider-electric.com/mainRepository/Index.nsf/DisplayProductDocumentation?OpenAgent&L=EN&App=Schneider&~EN~OT&p=548%26c=43,201,203,204,304

[7] Schneider Electric: *Unity Pro Standard Block Library*, 2012, Available on the Internet:

http://www.global-download.schneider-electric.com/mainRepository/Index.nsf/DisplayProductDocumentation?OpenAgent&L=EN&App=Schneider&~EN~OT&p=548%26c=43,201,203,204,304

[8] Schneider Electric: *Unity Pro Program Languages and Structure*, 2012, Available on the Internet:

http://www.global-download.schneider-electric.com/mainRepository/Index.nsf/DisplayProductDocumentation?OpenAgent&L=EN&App=Schneider&~EN~OT&p=548%26c=43,201,203,204,304

[9] MathWorks: *Neural Network Toolbox*, Available on the Internet:

http://www.mathworks.com/

Ladislav Körösi, Jana Paulusová

Institute of Robotics and Cybernetics,
Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology,
Ilkovičova 3,
812 19 Bratislava, Slovak Republic
e-mail: ladislav.korosi@stuba.sk, jana.paulusova@stuba.sk