E-Learning Tool for Backpropagation Neural Network Architecture

Reddy, G. N., Gurpreet Singh
Drayer Department of Electrical Engineering
Lamar University
Beaumont, Texas, USA
gnreddy@lamar.edu

ABSTRACT: *This paper presents an e-Learning tool for mastering the back-propagation neural network architecture. A short review of the existing tools is presented. It is developed using MS Visual C++. The tool's functionality can be summarized as: First, at its highest-level, it operates two basic modes: the training mode and the recall mode. Second, while it is in training, it has two sub-modes: the learning-mode and the application-mode. In learning mode, the software generates text-output traces corresponding to the top-down design steps of the NN-architecture. The generated numeric traces have dual-usage, either they can be used learning purposes or for generating class room tests. While in application-training mode, the tool displays only the input-output relations – the values before and after the training. In this mode the tool also generates a cumulative error-index to monitor the progress of the network training. Third, it enables the user to enter the network training termination criteria. Fourth, at the end of the network training, it is stores the trained network into a text-file. Fifth, in the test or recall mode, the trained network is retrieved from a stored-file, it then generates the network response corresponding to the entered test input. The e-Learning tool is tailored for mastering, class room teaching, and test generation of the BP-NN-architecture.*

Keywords: Neural Network Architectures, Back-propagation Nueral Network, Modeling and Simulation, e-Leaning Tool, Educational Software

Received: 18 May 2013, Revised 26 June 2013, Accepted 30 June 2013.

1. Introduction

Back propagation neural network architecture is complex and it requires a good e-learning tool to master its understanding [1]. Some of the commercial and open-source BP-NNA software packages include: 1. MathWorks Neural Network Toolbox [2] -- it has built-in features to view the intermediate results to explore the BP-NNA. One needs to consider its price-tag on licensing; 2. Back-propagation neural network software from soft112 [2] -- it is a matlab code specific application – face recognition; 3. BP-C#-program by McCaffrey [4] -- it has all features that you look for in e-Learning tool. It describes step-by-execution of the overall Bp-architecture and the corresponding C#-code. Only thing one might look for is the mathematical descriptions of each of these steps; 4. Neural network C#-libraries from codeproject [5] -- the applications are GUI-based and provide text and chart displays, the display system dynamic equations used in the individual steps will be nicer to have. The tool developed generates an output simulation trace similar to the Griiffith's-trace [6]. Wikipedia has excellent review on neural network software [7]. This is a partial list of tools one finds frequently in a web-search. It is hard to compare one tool with the other, as each has its own unique features. One needs to generate a figure of merit index, as the weighted sum all its features, to compare one tool with the other. Some of the features one looking for include: Numeric trace generation; context sensitive display of the system dynamic equations; GUI-interface; learning-mode and application development mode; if it is conducive for test generation; language used to develop the tool; and finally open source or licensed. The tool presented in this paper excels for learning and teaching. The following sections describe the BP-architecture and the simulation system.

2. Back-Propagation Neural Network Architecture

Figure 1 shows the architecture of the back-propagation neural network architecture with one hidden layer. One can have any number of hidden layers in the back-propagation network [1]. Typically there are two hidden layers, but one is sufficient for majority of the applications. In this software we have used one hidden layer to simplify overall network complexity. As shown, it is a multi-layer, fully-connected, feed-forward network. The three layers are: the input layer (*in*), the hidden-layer (*hl*), and the output-layer (*ol*).

The weight-matrices of the input-layers, hidden-layer, and the output-layer are correspondingly denoted as *Win*, *Whl*, and Wout. Initially all its weight matrices are initialized with small adaptive random weights (Ad-Rnd-Wts) between ± 0.1. A bias-elements *B* (the 0th-element) is added to the input and the hidden-layer. How information is processed within each of the neural elements is specified by their neurodynamics. The neurodynamics is a combination of a summation function SF followed by a transfer function *TF*. All of the layers use the weighted summation function (weighted-sum). The transfer function, however, can be different for each layer. Input and the hidden-layers can have sine or sigmoid or tanh transfer functions (*S*/*G*/*T*). Output layer can also have above three transfer functions; however, we have fixed it as sigmoid to simplify the complexity of the network training algorithm. One can have any number of elements in each layer (INmax, HLmax, OLmax). This educational version of the software number of elements is limited to 25. For each layer the internal activations are denoted by *I* or sum (*Iin*, *Ihl*, *Iout*) and the corresponding output activations denoted as *Y* or act (*Yin*, *Yhl*, *Yout*).

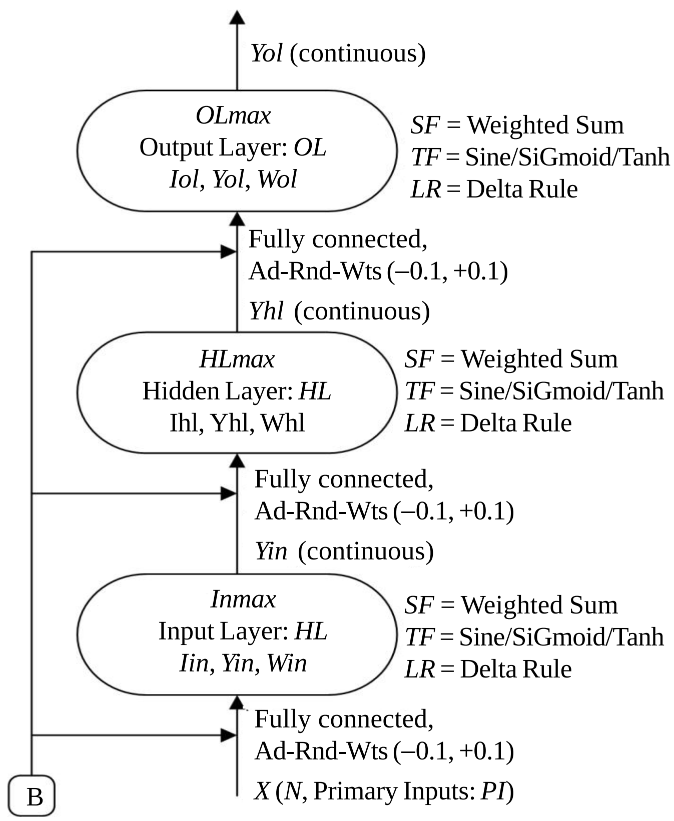

Figure 1. Back-propagation Neural Network Architecture

3. BP Network Training Procedure

The training of BP-network involves the following steps:

1a. **Initialize the network weights with small random weights:**

$$W_{ij}^{k} = r\text{and}\ (-0.1, +0.1) \tag{1}$$

where, k is the layer number: $k = 0$, is the primary input layer PI; $k = 1$, is the input-layer IL; $k = 2$, is the hidden layer HL; and $k = 3$, is the output layer OL. W_{ij}^{k}, is the weight from i-thelement in $(k-1)$-th layer to the j-th-element in k-th layer.

1b. **Set initial values:** Initialize the training cycle number to zero: $n = 0$; and tolerable error-level to a desirable value: $TssTh$ = typically 0.1.

2. **Set initial values for each epoch of training cycle:** Initialize pattern number to zero: $p = 0$; Global and local error-flags to zero: $flagG = 0$, $flagL = 0$. An error occurs when the computed value is different from the desired value.

3. **Do a forward-pass:** Apply the primary input vector Xp to the network and compute the corresponding output vector $Yout$. The generalized equations to compute internal activations Is and corresponding output activations Ys for any layer is given by:

$$I_j^{k} = \sum_{i=0}^{N_k} W_{ij}^{k} * X_i^{k-1} \tag{2}$$

$$Y_j^{k} = TF_k * I_j^{k} \tag{3}$$

Here, $X0$ represents primary input vector X; $TF3$ is the TF for the output layer which is fixed as sigmoid in this software; and $TF1$ and $TF2$ are the TFs for the input and the hidden layer TFs which can be any one of sine or sigmoid or tanh. Individual layer internal activations corresponding outputs are given by the following sets of equations.

Input layer sums Is and acts Ys are given by:

$$Iin_j = \sum_{i=0}^{PImax} Win_{ij} * X_{pi} \tag{4}$$

$$Yin_j = TF_1 * Iin_j \tag{5}$$

With $TF1$ = sigmoid, Yin is given by:

$$Yin_j = \frac{1}{1 + e^{-(Iin_j * G)}} \tag{6}$$

where G is the gain factor which usually vary between 1 and 10.

Hidden layer sums and acts are given by:

$$Ihl_j = \sum_{i=0}^{ILmax} Whl_{ij} * Yin_i \tag{7}$$

$$Yhl_j = TF_2 * Ihl_j \tag{8}$$

Output layer sums and acts are given by:

$$Iol_j = \sum_{i=0}^{HLmax} Wol_{ij} * Yhl_i \tag{9}$$

$$Yol_j = TF_3 * Iol_j \tag{10}$$

In this software TF_3 is sigmoid.

4. **Compute Tss_p:** find the mean square error of the current pattern:

$$Tss_p = \sqrt{\frac{1}{N_3}(D^3_{pj} - Y_j^{\,3})^2} \quad (11)$$

Here, *Dp* and *Y* are desired and the correspondingly computed values at the output layer.
If $(D^3_{pj} - Y^3_j) > TssTh\,x$, for any $j = 1,..N3$
then set *flagL* = *flagG* = 1;
else Go to step 6.

5a. **Find error functions,** weight-changes; and new weights If *flagL* = 1, compute error functions ds for each element; the weight-changes *DWs*; and the new weights *W 's*. The error functions are needed to find weight-changes to the network. Error functions at the output-layer, with sigmoid *TF*, are given by:

$$\delta^3_j = Y^{\,3}_j(1 - Y^3_j) * (D^3_j - Y^{\,3}_3) \quad (12)$$

Here, 3 is the output-layer number. The error function is a product of gradient * error. In (12) the gradient is $Y(1 - Y)$, and the error is $(D - Y)$. The gradient for different TFs is different, for sigmoid it is $Y(1 - Y)$ [1]. The error in known at the output-layer, as the desired value *D* and correspondingly computed value *Y* are known. For other lower-level layers *Ys* are known but not the desired values *Ds*. The generic error functions for the other-layers are computed as:

$$\delta^k_j = Y^k_j(1 - Y^k_j) * \sum_{m=1}^{N_{k+1}} W^{k+1}_{jm}\,\delta^{k+1}_m \quad (13)$$

That is, error-functions of the lower-level layers are computed from the upper-level layers.

The error functions for the hidden layer elements are computed as:

$$\delta hl_j = Yhl_j(1 - Yhl_j) * \sum_{m=1}^{OLmax} Wol_{jm}\,\delta ol_m \quad (14)$$

Here, the error of a hidden layer element is computed as the weighted summation of the output-layer error-functions.

The error functions for the input layer elements are computed as:

$$\delta in_j = Yin_j(1 - Yin_j) * \sum_{m=1}^{HLmax} Whl_{jm}\,\delta hl_m \quad (15)$$

Here, the error of an input layer element is computed as the weighted summation of the hidden-layer error-functions.

5b. **Find weight changes:** Find weight changes from primary inputs to the input-layer elements, *DWin*:

$$DWin_{ij} = \alpha * \delta in_j * X_{pi} \quad (16)$$

Here, α is the training coefficient ranging from 0.1 to 1.0; $i = 0,.., PImax$; and $j = 1,.. ILmax$.

Find weight changes from input-layer-elements to the hiddenlayer elements, *DWhl*:

$$DWhl_{ij} = \alpha * \delta hl_j * Yin_i \quad (17)$$

Here, $i = 0,.., ILmax$; and $j = 1,.. HLmax$.

Find weight changes from hidden-layer-elements to the output-layer elements, *DWol*:

$$DWol_{ij} = \alpha * \delta ol_j * Yhl_i \quad (18)$$

Here, $i = 0,.., HLmax$; and $j = 1,..OLmax$.

5c. **Find new weights *W* (*n* + 1)**:
New weights $W(n + 1)$ are computed as the old weights $W(n)$ plus the weight-changes $DW(n)$, *n* being the previous cycle and $n + 1$ is the current cycle:

$$Win\,(n+1)_{ij} = Win\,(n)_{ij} + DWin\,(n)_{ij} \tag{19}$$

$$Whl\,(n+1)_{ij} = Whl\,(n)_{ij} + DWhl\,(n)_{ij} \tag{20}$$

$$Wol\,(n+1)_{ij} = Wol\,(n)_{ij} + DWol\,(n)_{ij} \tag{21}$$

6. **Go to next pattern to train:**
Set $p = p + 1$; if ($p < pmax$) go to Step 3.

7a. **Compute TssC**:
Normalized cumulative error, in cycle n, of all patterns is given as:

$$TssC_{(n)} = \frac{1}{pmax} \sum_{i=0}^{pmax-1} Tssp_i \tag{22}$$

7b. **Go to next epoch-training:**
If $flagG = 0$; then Go to Step 8.

Else Set $n = n + 1$; then Go to Step 2.

That is, repeat Steps 2 through 7 until all patterns are trained with acceptable error.

8. Write trained network to a file; Write cumulative network error *TssC* to a file; End network training.

4. BP Recall Procedure

In recall or test mode, for given test input X, the network response *Yout* is estimated. This is by successively computing activation vectors *Yin*, *Yhl*, and *Yout*. The response will be nearest output-match corresponding to the entered input. You can enter into the recall mode only after the network is trained. In this mode, first the trained network is read from bp-ckt.txt file which is generated at the end of training mode. For a test input X it finds the corresponding output *Yout*. The activation vectors *Yin*, *Yhl*, and *Yout* are computed as:

$$Yin_j = Y_j^{\,1} = TF_1 * Iin_j = TF_1 * \sum_{i=0}^{PImax} Win_{ij} * X_i \tag{23}$$

$$Yhl_j = Y_j^{\,2} = TF_2 * Ihl_j = TF_2 * \sum_{i=0}^{ILmax} Whl_{ij} * Yin_i \tag{24}$$

$$Yout_j = Y_j^{\,3} = TF_3 * Iol_j = TF_3 * \sum_{i=0}^{HLmax} Wol_{ij} * Yhl_i \tag{25}$$

In (23), $j = 1,..\ ILmax$; in (24), $j = 1,..\ JLmax$; and in (25), $j = 1,..\ OLmax$.

5. The BP-Software Architecture

Figure 2 shows the overall architecture of the BP-software.

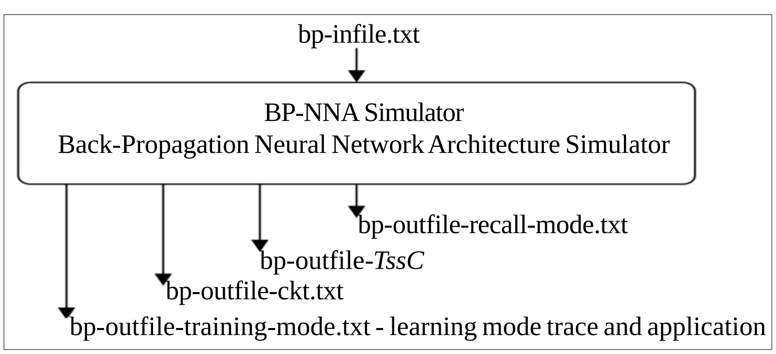

Figure 2. The BP-Software Architecture

In Figure 2, bp-infile.txt is the input data text file; bp-outfiletraining-mode.txt is the output simulation trace generated during the network training; bp-outfile-ckt.txt is the output file that contains the trained bp-network; bp-outfile-recallmode. txt is the output simulation trace generated during the network testing.

6. The BP-Simulator: Output Simulation Trace: Training Mode

Tables 1 through 7 are the input data files or the generated output files in different BP-simulator modes of operation. Table 1 is the input data file to run the network in learning mode. The input data file contains: network specification – number of elements in each layer of the BP-network; their transfer functions; level of weight changes to make in each successive cycle of training; training termination criteria; and the patterns to train. Table 2 is the corresponding output simulation trace while network is in learning-training-mode; this trace is useful for learning about the BP-network; it can also be used for test generation – formulation of numeric problems on BP-NNA. Major phases of training include: 1. the forward-pass – where activations of each element of the network are computed for a given input vector *X*; 2. find the error functions for each element of the network; 3. Find weight-changes to the network weights; 4. find new weights of the network; and 5. find cumulative network error *TssC*. Table 3 contains the trained BP-network. The network specification include: the number elements in each layer; each-layer's transfer functions; and the trained network weights. Table 4 gives the cumulative RMS-error in successive cycles of training. This is also shown in a chart-form in Figure 3. The network is continues to be trained until the network's cumulative error *TssC* is less than the set threshold error *TssTh*. For the trained network shown in Tables 2; it took 61 cycles to train with an initial error of 0.51. Table 5 contains an input-data file to use BP-network in application-development mode. Table 6 is the corresponding output simulation while the patters are trained. Here the details of training are disabled; the emphasis is placed on the application development. Table 7 contains the output simulation while BP is in recall mode or test mode. Various phases of network recall include: 1. Read and print the trained network; 2. For a given input vector *X*, do the forward-pass to find *Yout*; and 3. Prompt the way to terminate recall session.

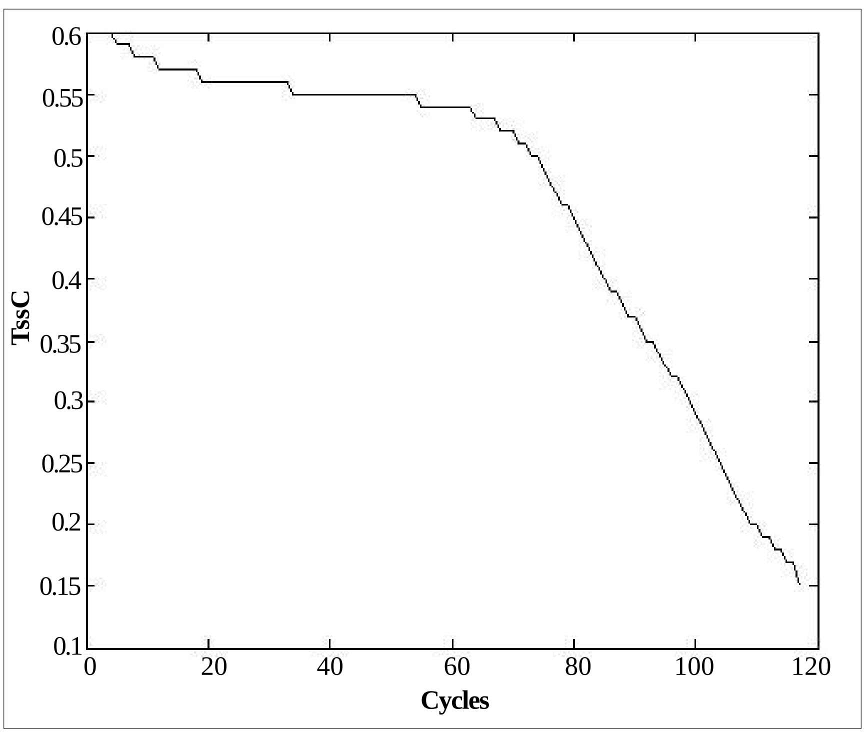

Figure 3. Cumulative network RMS-error in successive training cycles: TssC

7. Conclusions

This paper presents an e-Learning tool that can aid in depth understanding of the Back-Propagation Neural Network Architecture. The tool is developed for class teaching and test generation.

Table 1. Input Data File (bp-infile.txt)

```
bp-infile.txt
4          : PImax, Number of PIs
3          : ILmax, Nuber of elemnets in IL
3          : HLmax, Nuber of elemnets in HL
3          : OLmax, Nuber of elemnets in OL
0.1        : alpha, Training coefficient
G          : iltf & hltf: S/T/G: sime/tan/sigmoid
G          : oltf, TF for OL
2.0        : Gain for all TFs: I' = I * Gain
0.2        : Tssth, Threshold error: 0.3 => 30%
1          : Pmax & the Pattern associations: Yi, Xi
1 0 0 1
1 1 0
```

Table 2. BP-Simulator Output simulation trace – in learning mode (bp-outfile-traning-mode).

```
**************************************************
** Back-Propagation Neural Network Simulator    **
** Traning mode: Output Simulation trcae        **
**************************************************
Reading input data from: bp-infile.txt
BP NETWORK - TRAINING MODE:
# of PEs in PI/IN/HL/OL:          4  3  3  3
LR for IN, HL, and OL:            Delta Rule
Training Coefficient (alpha):     0.10
TF Input/Hidden Layers (hltf):    Sigmoid
TF of Output Layer (oltf):        Sigmoid
Gain factor for the TFs:          2.00
Error Threshold (TssTh):          0.20
# of pattern associations (pmax): 1
              0         1         2         3         4
X[ 0]      1.00      1.00      0.00      0.00      1.00
D[ 0]                1.00      1.00      0.00
Network training starts here:
CYCLE:  1
Weight Matrices of the Network:
          PI: 0     PI: 1     PI: 2     PI: 3     PI: 4
IL: 1 -0.018000  0.034000 -0.032000 -0.100000  0.038000
IL: 2 -0.052000  0.056000  0.016000  0.024000  0.028000
IL: 3 -0.090000 -0.010000  0.062000 -0.046000  0.022000

          IL: 0     IL: 1     IL: 2     IL: 3
HL: 1  0.082000  0.090000 -0.016000 -0.046000
HL: 2 -0.028000  0.082000 -0.092000 -0.096000
HL: 3  0.006000  0.084000  0.064000 -0.058000

          HL: 0     HL: 1     HL: 2     HL: 3
OL: 1 -0.068000 -0.064000  0.090000 -0.006000
OL: 2 -0.048000  0.042000 -0.024000  0.038000
OL: 3 -0.076000  0.034000  0.098000 -0.030000

Forward-pass: Activations of each PE in the Network:
           IL: 0     IL: 1     IL: 2     IL: 3
sum        0.000     0.054     0.032    -0.078
act        1.000     0.527     0.516     0.461

           HL: 0     HL: 1     HL: 2     HL: 3
sum        0.000     0.100    -0.077     0.057
act        1.000     0.550     0.462     0.528

           OL: 1     OL: 2     OL: 3
sum       -0.065    -0.016    -0.028
act        0.468     0.492     0.486
Tssp: rms errors: p0...pmax:  0.51
TssC[cycle]: Cumulative-Tssp rms error(before)-: 0.51

CYCLE:  1 Pattern: 0
Delta-Fn for each PE in the Network:
        IL: 1        IL: 2         IL: 3
   -6.31e-005    2.20e-004    4.40e-005
        HL: 1        HL: 2         HL: 3
   -1.80e-003   -7.50e-004    1.91e-003
        OL: 1        OL: 2         OL: 3
    1.33e-001    1.27e-001   -1.21e-001
Delta Weight Matrices of the Network:
          PI: 0     PI: 1     PI: 2     PI: 3     PI: 4
IL: 1 -0.000006 -0.000006 -0.000000 -0.000000 -0.000006
IL: 2  0.000022  0.000022  0.000000  0.000000  0.000022
IL: 3  0.000004  0.000004  0.000000  0.000000  0.000004

          IL: 0     IL: 1     IL: 2     IL: 3
HL: 1 -0.000180 -0.000095 -0.000093 -0.000083
HL: 2 -0.000075 -0.000040 -0.000039 -0.000035
HL: 3  0.000191  0.000101  0.000099  0.000088

          HL: 0     HL: 1     HL: 2     HL: 3
OL: 1  0.013253  0.007287  0.006120  0.007001
OL: 2  0.012696  0.006980  0.005863  0.006706
OL: 3 -0.012142 -0.006676 -0.005607 -0.006414
CYCLE:  2

Weight Matrices of the Network:
          PI: 0     PI: 1     PI: 2     PI: 3     PI: 4
IL: 1 -0.018006  0.033994 -0.032000 -0.100000  0.037994
IL: 2 -0.051978  0.056022  0.016000  0.024000  0.028022
IL: 3 -0.089996 -0.009996  0.062000 -0.046000  0.022004

          IL: 0     IL: 1     IL: 2     IL: 3
HL: 1  0.081820  0.089905 -0.016093 -0.046083
HL: 2 -0.028075  0.081960 -0.092039 -0.096035
HL: 3  0.006191  0.084101  0.064099 -0.057912

          HL: 0     HL: 1     HL: 2     HL: 3
OL: 1 -0.054747 -0.056713  0.096120  0.001001
OL: 2 -0.035304  0.048980 -0.018137  0.044706
OL: 3 -0.088142  0.027324  0.092393 -0.036414

Forward-pass: Activations of each PE in the Network:
           IL: 0     IL: 1     IL: 2     IL: 3
sum        0.000     0.054     0.032    -0.078
act        1.000     0.527     0.516     0.461

           HL: 0     HL: 1     HL: 2     HL: 3
sum        0.000     0.100    -0.077     0.057
act        1.000     0.550     0.462     0.528

           OL: 1     OL: 2     OL: 3
sum       -0.041     0.007    -0.050
act        0.480     0.503     0.475
Tssp: rms errors: p0...pmax:  0.50
TssC[cycle]: Cumulative-Tssp rms error(before)-: 0.50
…
CYCLE: 61
Tssp: rms errors: p0...pmax:  0.20
TssC[cycle]: Cumulative-Tssp rms error(before)-: 0.20
```

Table 3. Trained network (bp-outfile-ckt.txt)

```
4  3  3  3  G  G 2

 -0.010052  0.041949 -0.032000 -0.100000  0.045949
 -0.051681  0.056319  0.016000  0.024000  0.028319
 -0.095006 -0.015006  0.062000 -0.046000  0.016994

  0.107394  0.103536 -0.002890 -0.034394
 -0.003866  0.094858 -0.079541 -0.084966
  0.042027  0.103176  0.082598 -0.041517

  0.344648  0.165523  0.283310  0.216568
  0.344183  0.260153  0.159735  0.249546
 -0.450879 -0.174549 -0.077649 -0.232241
```

Table 4. Cumulative error TssC (bp-outfile-TssC.txt)

```
Tssc: Cumulative error at each cycle
Cycle:  TssC:
1       0.51
2       0.50
3       0.49
...
61      0.20
```

```
Table 5. Input Data File (bp-infile.txt): application mode

bp-infile.txt
4          : PImax, Number of PIs
3          : ILmax, Nuber of elemnets in IL
3          : HLmax, Nuber of elemnets in HL
3          : OLmax, Nuber of elemnets in OL
0.6        : alpha, Training coefficient
G          : iltf & heltf: S/T/G: sime/tan/sigmoid
G          : oltf, TF for OL
5.0        : Gain for the TF: I' = I * Gain
0.25       : Tssth, Threshold error: 0.3 => 30%
4          : Pmax & the Pattern associations: Yi, Xi
1 0 0 1
0 1 0
0 1 1 0
1 0 1
1 1 1 0
1 0 0
0 1 1 1
0 0 1
```

```
CYCLE:574
YOL[ 0]-:   0.10   0.87   0.00
YOL[ 1]-:   0.81   0.00   1.00
YOL[ 2]-:   0.79   0.07   0.05
YOL[ 3]-:   0.23   0.01   0.84
Tssp: rms errors: p0...pmax:  0.09 0.11 0.13 0.16
TssC[cycle]: Cumulative-Tssp rms error(before)-: 0.12

Network Training Complete: Cycles: 574

Writing Network into the File bp-ckt.txt...

Writing TssC into the File bp-outfile-TssC.txt...

END BACK-PROPAGATION SIMULATION: TRAINING SESSION
```

```
Table 6. BP-Simulator Output simulation trace - in
         application mode (bp-outfile-training-mode.txt)

**************************************************
** Back-Propagation Neural Network Simulator    **
** Traning mode: Output Simulation trcae        **
**************************************************
Reading input data from: bp-infile.txt
BP NETWORK - TRAINING MODE:
# of PEs in PI/IN/HL/OL:           4  3  3  3
LR for IN, HL, and OL:             Delta Rule
Training Coefficient (alpha):      0.60
TF Input/Hidden Layers (hltf):     Sigmoid
TF of Output Layer (oltf):         Sigmoid
Gain factor for the TFs:           5.00
Error Threshold (TssTh):           0.25
# of pattern associations (pmax): 4
              0          1          2          3          4
X[ 0]      1.00       1.00       0.00       0.00       1.00
D[ 0]                 0.00       1.00       0.00
X[ 1]      1.00       0.00       1.00       1.00       0.00
D[ 1]                 1.00       0.00       1.00
X[ 2]      1.00       1.00       1.00       1.00       0.00
D[ 2]                 1.00       0.00       0.00
X[ 3]      1.00       0.00       1.00       1.00       1.00
D[ 3]                 0.00       0.00       1.00
Network training starts here:
Weight Matrices of the Network:
          PI: 0      PI: 1      PI: 2      PI: 3      PI: 4
IL: 1 -0.018000  0.034000 -0.032000 -0.100000  0.038000
IL: 2 -0.052000  0.056000  0.016000  0.024000  0.028000
IL: 3 -0.090000 -0.010000  0.062000 -0.046000  0.022000
          IL: 0      IL: 1      IL: 2      IL: 3
HL: 1  0.082000  0.090000 -0.016000 -0.046000
HL: 2 -0.028000  0.082000 -0.092000 -0.096000
HL: 3  0.006000  0.084000  0.064000 -0.058000
          HL: 0      HL: 1      HL: 2      HL: 3
OL: 1 -0.068000 -0.064000  0.090000 -0.006000
OL: 2 -0.048000  0.042000 -0.024000  0.038000
OL: 3 -0.076000  0.034000  0.098000 -0.030000
CYCLE:  1
YOL[ 0]-:   0.41   0.49   0.46
YOL[ 0]+:   0.28   0.66   0.31
DOL[ 0] :   0.00   1.00   0.00
YOL[ 1]-:   0.28   0.66   0.31
YOL[ 1]+:   0.47   0.46   0.50
DOL[ 1] :   1.00   0.00   1.00
YOL[ 2]-:   0.47   0.46   0.50
YOL[ 2]+:   0.64   0.31   0.34
DOL[ 2] :   1.00   0.00   0.00
YOL[ 3]-:   0.64   0.31   0.34
YOL[ 3]+:   0.45   0.24   0.53
DOL[ 3] :   0.00   0.00   1.00
Tssp: rms errors: p0...pmax:  0.46 0.69 0.50 0.56
TssC[cycle]: Cumulative-Tssp rms error(before)-: 0.55
```

```
Table 7. BP-Simulator Output simulation trace - in Recall
mode (bp-outfile-recall-mode.txt)

BACK-PROPAGATION NETWORK - RECALL MODE:
Reading Network Weights from: bp-outfile-ckt.txt
Trained BP Network:
Number of Elements: PI/IL/HL/OL:  4  3  3  3
Transfer Function for the IN, HL:      G
Transfer Function for the OL:          G
Gain factor for Sine/siGmiod/Tanh TFs:  5.0

Weight Matrix WIL:
         0          1          2          3          4
  1 -0.117929  0.373708 -0.169350 -0.237350 -0.004736
  2 -0.290731  1.331819 -0.518565 -0.510565  0.278290
  3 -0.305431  0.594064 -0.425380 -0.533380  1.221188
Weight Matrix WHL:
         0          1          2          3
  1  0.095929 -0.120610 -0.397326 -0.594480
  2  0.319776 -0.298146 -0.848796 -0.394399
  3 -0.021513  0.066980  0.139034 -0.572369
Weight Matrix WOL:
         0          1          2          3
  1 -0.588324  0.765280 -0.185429  1.184759
  2  0.436281 -1.127587 -1.576824 -0.380790
  3 -1.586459  1.415305  2.336450  0.023212

BACK-PROPAGATION NETWORK: Reacll Mode:
Back-Propagation Network: Results of Testing:
Enter 9 9 9 9 9 to Terminate Testing:
Enter a Test Input: x1..x4
        0      1      2      3      4
 PI:  1.000  1.000  0.000  0.000  1.000
IIL:         0.251  1.319  1.510
YIL:  1.000  0.778  0.999  0.999
IHL:        -0.989 -1.154 -0.403
YHL:  1.000  0.007  0.003  0.118
IOL:        -0.444  0.379 -1.566
YOL:         0.098  0.869  0.000

Enter a Test Input: x1..x4
        0      1      2      3      4
 PI:  1.000  0.000  1.000  1.000  0.000
IIL:        -0.525 -1.320 -1.264
YIL:  1.000  0.068  0.001  0.002
IHL:         0.086  0.298 -0.018
YHL:  1.000  0.606  0.816  0.478
IOL:         0.290 -1.716  1.189
YOL:         0.810  0.000  0.997
…
Enter a Test Input: x1..x4
END BACK-PROPAGATION SIMULATION: RECALL SESSION
```

References

[1] Stephen T. Welstead. (1994). Neural Network and Fuzzy Logic Applications in C/C++, Wiley.

[2] MathWorks.com, Neural Network Toolbox, Novi, Michigan, (2013).

[3] Soft112.com, Backpropagation neural network, (2013).

[4] James McCaffrey. (2013). Neural Network Back-Propagation Using C#.

[5] Codeproject.com, Andrew Kirillov, Neural Networks on C#, November, (2006).

[6] Niall Griffith. (2013). Backpropagation algorthm, MIT Computer Scienec and Artificial Intelligence, CIS, Tutorial 10.

[7] Wikipedia.org, Neural network software, July, (2013). Update 2.5, 11/1/13