aNETka version 2.1

Operation Manual

Feed-forward, back-propagation Artificial Neural Network implemented in LabVIEW 5.1

Designed and created by

Dr Stan Zurek
zureks@cf.ac.uk → zureks@gmail.com
Wolfson Centre for Magnetics
School of Engineering
Cardiff University
Newport Road
Cardiff, CF24 3AA
United Kingdom

Cardiff, October 2005

List of contents:

1. Introduction

The idea of artificial neural networks (ANN) was first proposed many years ago [1], however the difficulties in "training" of such networks have delayed their practical use. Only after discovering successful methods of training the ANN has received a great deal of attention. The improvement in computer technology also allowed fast progress in ANN development. At present the ANN are used in numerous scientific, industrial and other areas, as a tool for analysis, prediction, control, identification, data processing, etc. [2]

The ANN is designed to work similar to neural tissue in the brain, where many independent neurons (processing units) are interconnected into one large data-processing network. In the ANN the "neurons" are represented as mathematical functions, which pass the data between themselves in some defined and organised manner. The way of organisation of the connections between the "neurons" defines the features of the ANN, and its capability to perform certain tasks.

One of the simplest ANN's is a feed-forward network, as shown in Fig. 1. (There are many types of neural network, but **aNETka** works only with this type, therefore other types of ANN will not be considered here.) Such a network consists of: one input layer, hidden layers (possibly more than one) and one output layer. In most cases all subsequent layers are connected in such way that all previous layer nodes are connected to all next layer nodes (see Fig. 1). The input layer works only as a buffer to the network – it does not "process" the data, and its task is only to transfer the input values to the first hidden layer.

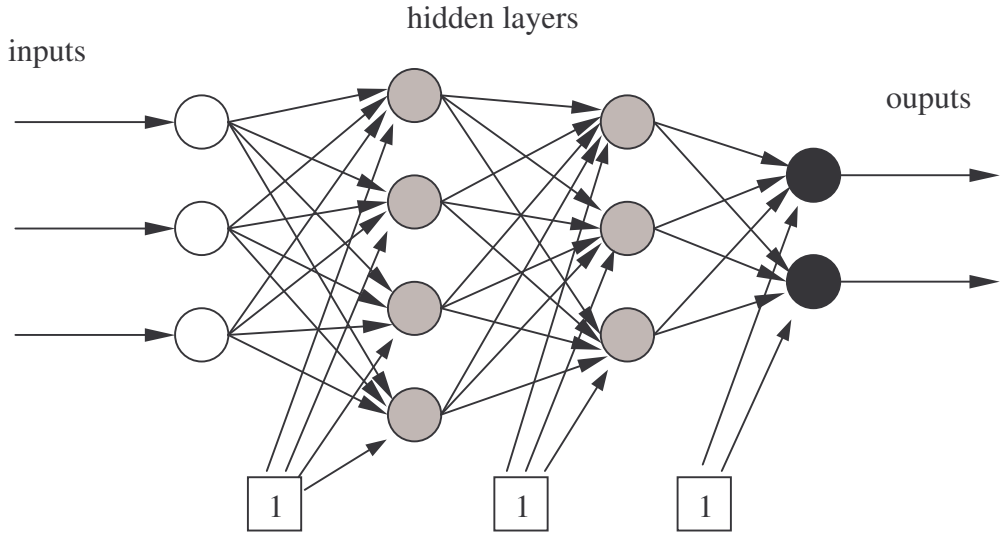

Fig. 1. An example of a multi-layer feed-forward ANN; white circles – input nodes, grey circles – hidden layer nodes, black circles – output nodes, squares – biases

The data are transferred in a form of "signals", i.e. values which are passed between the neurons (nodes). Each connection between the nodes has a special variable associated with it – the weight. Each neuron collects all the input values multiplied by the associated weights and processes them with activation function:

$$S_{x,i} = f\left(\sum_i S_{x-1,i} \cdot W_{x,i}\right) \tag{1}$$

where: f – activation function, $S_{x,i}$ – output signal of i^{th} neuron in x^{th} layer, $S_{x\text{-}1,i}$ – output of i^{th} neuron in x-1^{th} layer, $W_{x,i}$ – weights connected to neuron $S_{x,i}$

In general, the activation function should be continuous and differentiable. Two of the most frequently used functions are sigmoid (logistic) $y=1/(1+e^{x})$ and hyperbolic tangent $y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$ (but there are others [1]). It is desirable that the derivative of the activation function is simple, because that helps minimising the necessary calculations. (**aNETka** uses one of three functions: linear, sigmoid and hyperbolic tangent.)

A linear function is rarely used because the whole network can perform only linear transformation. Also in such case an increase in the number of hidden layers does not improve the performance of the ANN since the combination of linear functions will always give a linear function.

However, if the activation function is non-linear then such ANN can approximate almost any non-linear function. Of course the limitation is the number of neurons in the network and the processing time required for training. Nevertheless, non-linear networks can approximate very complex functions of many variables, as will be shown in section **Examples** (page 14).

Adding so-called bias can easily alter the working point of any neuron in the network. The bias is a signal always equal to +1, which is connected to each neuron in the ANN (see Fig. 1). By using appropriate weight the bias can enhance the working area of the neuron, thus allowing to obtain some solutions not accessible without bias. (The biases are implemented in **aNETka**.)

In order to use any ANN first the training procedure has to take place. The most widely known, simplest and most robust is a technique called back-propagation of errors. This training algorithm is implemented in **aNETka**, hence it will be described here.

2. Back-propagation

The training procedure modifies only the weights in the whole network – the weights store all the "wisdom" of any ANN.

In order to perform the back-propagation algorithm (BP) the target values need to be known. All the activation functions are set before training, and the weights (W) are set to some small random numbers (for example, in **aNETka** they are in a range from -0.1 to +0.1). When the first set of inputs is presented to the ANN the output values (Y) can be calculated. These values are different from the ideal (target values, T), hence the errors (d) can be calculated for each neuron in output layer. The errors for the neurons in hidden layers cannot be directly calculated because the target values are not known. However, it has been found that the errors from output layer can be propagated through the whole ANN back to the inputs. The errors propagated back to previous layer have to be multiplied with the weight of a given connection. The total error for a hidden neuron is found as the sum of all errors propagated back to that unit from the next layer (see Fig. 2).

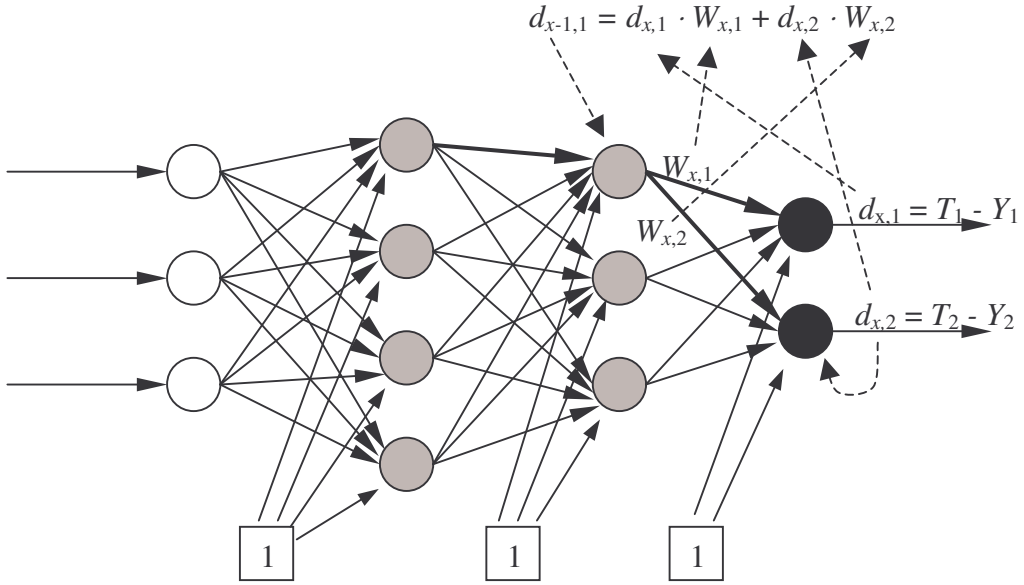

Fig. 2. The concept of back-propagation of errors: $d_{x,1}$ – error of first output, $d_{x,2}$ – error of second output, T – target value, Y – calculated value, x – number of last layer, x-1 – number of previous layer, W – weight

At the same time the correction of the weights (dW) can be found for a given connection. This can be calculated as:

$$dW_{x,j} = LR \cdot d_{x,\mathrm{i}} \cdot f'(S_{x,i}) \cdot Y_{x,i} \text{ (for the weights connected to output neurons)} \quad (3a)$$

$$dW_{x\text{-}1,j} = LR \cdot d_{x\text{-}1,\mathrm{i}} \cdot f'(S_{x,i}) \cdot Y_{x,i} \text{ (for the weights in hidden layers)} \quad (3b)$$

where: LR – learning rate (defines how quickly the ANN will learn), S – value calculate from equation (1) for a given neuron, $f'(S)$ – derivative of the activation function for value S, Y – output value of the neuron, x – layer number, i – neuron number, j – weight number.

Therefore, the weights can be corrected for the next iteration:

$$W_{n+1} = W_n + dW_n \tag{4}$$

where: n – iteration number, W – weight, dW – weight correction calculated from equation (3a) or (3b), respectively.

Frequently, the training data contain several input/output cases (sets of data). These cases are presented sequentially to the ANN, and the weights are corrected after each case. When all the training cases are presented to the network it is said that one epoch is completed, and the whole process is repeated from the first case until the total output error is decreased to the desired value. Usually, a large number of epochs is required, in some cases even up to several hundred thousands.

Usually a simple term called momentum (M) is added to the ANN:

$$W_{n+1} = W_n + dW_n + M \cdot (W_n - W_{n-1}) \tag{5}$$

This simple modification stabilizes some small oscillations of the outputs of the ANN and also speeds up the convergence times and overcomes some local minima of the total error – it is desired to find the global minimum. The value of M should is usually set between 0.1 and 1.0. (The momentum is implemented in **aNETka**.)

3. Learning rate

The value of learning rate (LR) from equation (3) determines the convergence time of the ANN. If the LR is too low, then the learning time can be very long, and in some cases the learning can get stuck in a deep local minimum. On the other hand, if the LR is too large, then there might be very large oscillations of the output values of the ANN, which can lead to divergence of the whole ANN. Therefore the LR should be set to some "medium" value. However, it is very difficult to decide what is "medium" for different ANN. The rule of thumb is that the greater the number of neurons the smaller the value of LR. For very small ANN (e.g. 5 neurons) the LR can be as large as 2, but for large ANN (e.g. 45 neurons) the "best" LR can be as low as 0.001. The easiest way of determining the maximum LR is simply to run the ANN – if there are large oscillations or the ANN diverge, then the LR should be decreased (for example by half).

Another solution of the problem is automatic change of the LR during the learning process. It is difficult to decide numerically when the LR should increase or decrease. The method implemented in **aNETka** is following:

a) wait for certain number of iterations before turning on the automatic LR
b) after turning on automatic mode, increase slowly the LR (up to defined maximum value) when the ANN error decreases, or
c) decrease slowly the LR (down to defined minimum value)) when the ANN error increases.

This algorithm is not very sophisticated, but can be very effective in unsupervised training. In any case the user should test the constant LR and the automatic LR in order to investigate which performs better. In **aNETka** the automatic LR can be disabled or enabled at any point.

4. Overtraining

The ANN have great capability to approximate various linear and non-linear functions. Most frequently, the approximation is far more useful than exact interpolation of the data. In the former case the ANN learns the general trend, in the latter case the ANN memorizes all the training cases. So in the first case if there is a set of new data presented to the network the output will be still quite close to the expected value. But if the memorization takes place, then the set of new input data may result in erroneous results. When the memorization begins to develop the network is said to be overtrained and further training may actually damage the prediction capabilities.

In order to prevent the overtraining the input data is split into two parts. First part (usually 90% to 50% of all cases) is used for normal training, and the second part (10% to 50% of all cases) is never used for training, but only for recall in order to test the generalization capabilities. If the training error and testing errors decrease then the ANN is still learning (generalization). But if the training error decreases and te test error begins to increase then the ANN is beginning to memorize (overtraining). Of course there might be small fluctuations of the change in errors (local increase and decrease) – the overtraining happens only if after many iterations the test error still increases.

The weights of the ANN are randomly selected at the beginning of training. For that reason each training procedure should be repeated at least 3 times to see possible variations of the training process.

In **aNETka** the user can select any ratio of training to test cases.

5. Description of aNETka

5.1. Software

LabVIEW is programming software utilizing graphical programming language – “G” [3]. Any program in LabVIEW is "written" by connecting icons (subprograms, controls and indicators) with special "wires" (see Fig. 3). If necessary LabVIEW can call other source codes (for example from “C” language).

The LabVIEW interface consists of a front panel (communication with user) and block diagram (code), as shown in Fig. 3.

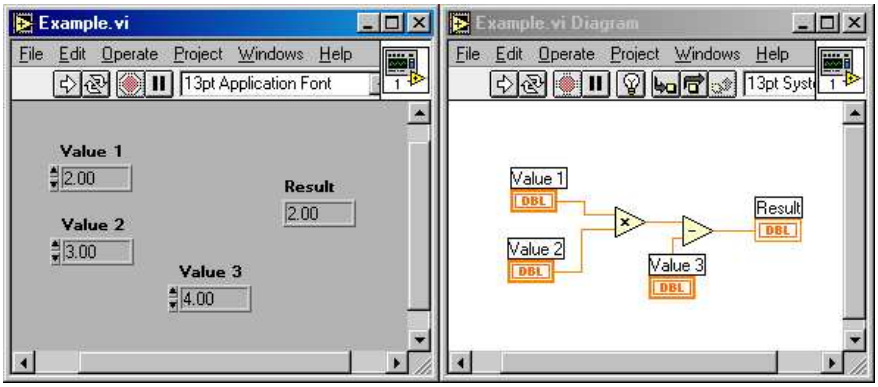

Fig. 3. An example of LabVIEW program

LabVIEW has excellent capabilities of managing very complex data structures, signal acquisition and processing. Because it fully integrates with data acquisition cards from National Instruments [4] it is often used for signal acquisition. The author of **aNETka** used programs written in LabVIEW for acquisition and measurement so it was a natural choice to implement the ANN also in LabVIEW.

There are many commercial software packages, which allow the use of ANN (MATLAB, QNET, etc), but their main disadvantage is that they are "commercial". If some institution already owns LabVIEW, then **aNETka** can be freely used without any additional cost. Also the source files can be modified as required. In the compiled version, **aNETka** is freely available to anyone (LabVIEW package is not required), which is major advantage over other commercial packages. However, in the compiled version the modification of the source code is not possible.

5.2. Files and folders

aNETka consists of 3 main programs, and the library of subprograms. All the files should be kept in the original folder (the whole folder can be freely moved, no installation required).

Each of the 3 programs contains some LabVIEW coding as well as subprograms from the *Library* folder. The most complex part of **aNETka** is the ANN "core" and the back-propagation algorithm. Those two parts are kept separate in the code, but it is very difficult to simplify the diagram any further. Moreover, the core and BP could not be enclosed within subprograms because they are executed hundreds of times in each iteration and that would slow down the execution.

The ANN core and BP look very complicated at the first sight, but mathematically they perform exactly the ANN calculating and training as described at the beginning of this manual.

The parts of code, which are not critical to the execution (they are outside of the main loop) were made as subprograms in order to simplify the diagram.

All data handling is performed automatically during the execution, and the user is only asked to intervene if absolutely necessary.

5.3. Training mode – program *Train aNETka.vi*

Before any prediction or approximation can take place **aNETka** has to be trained. This is achieved by running the file: *Train aNETka.vi* The program will ask about the following configurations:

a) **ANN configuration** (see Fig. 4).

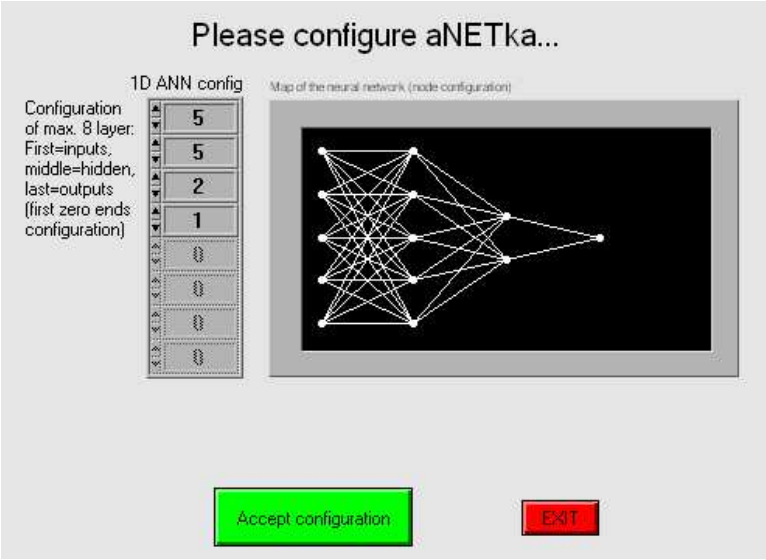

Fig. 4. ANN configuration

This is a one-dimensional array. The top field means the number of inputs, the middle fields are hidden layers and the bottom field (does not have to be the last in the array) is the number of outputs. There have to be at least 3 layers. First zero in the array disables all the inputs below. For convenience the graphical representation of chosen configuration is displayed on the right. If larger number of nodes in any of the layers is selected (for example greater than 10) then the drawing of the configuration may be slow. In such cases the inscription "Wait..." appears on the graph. After selecting the required configuration the green button should be pressed for acceptance. Pressing the red button will close the dialog box and also terminate the main program.

b) Activation function (see Fig. 5). One of 3 options can be selected (click on the name of function): Linear, Sigmoid and Tangent hyp.

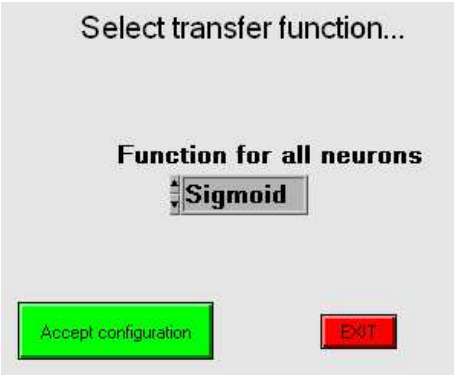

Fig. 5. Activation function

c) Input data (Fig. 6). For the training mode the input data must be saved in a suitable format: ASCII text spreadsheet file, tab delimited, with the first row containing captions for each column (it can be left blank, but is extremely useful to have appropriate captions). The Microsoft Excel format or similar is unsuitable – the file has to be exported to a text file. Also, the file name has to have a 3-letter extension, for example in a form "*.txt*" The program will ask for the input file. After opening the content will be displayed in the dialog box.

The configuration of ANN in the top right corner reminds about the number of required inputs and outputs. The largest table in the window contain all the data from the input file: the first row contains all the indices for the data in order to simplify the selection, the second row contains all the captions and further rows contain the actual input and target data. It is also necessary for all the input data to be in adjacent columns. For instance, if there are 5 inputs then the data have to be in columns from 0 to 4, or from 7 to 11, and it cannot be columns 1, 2, 3, 6 and 9 (they are not adjacent). The same applies to target data. The two indices above the largest table signify the beginning of the input and target columns. The choice of inputs and targets can be verified by looking at the two small tables at the bottom of the screen. If the data table is really large, and does not fit on the screen, then the screen can be scrolled and/or the table can be scrolled, by clicking the index *Columns* on the left of the largest table.

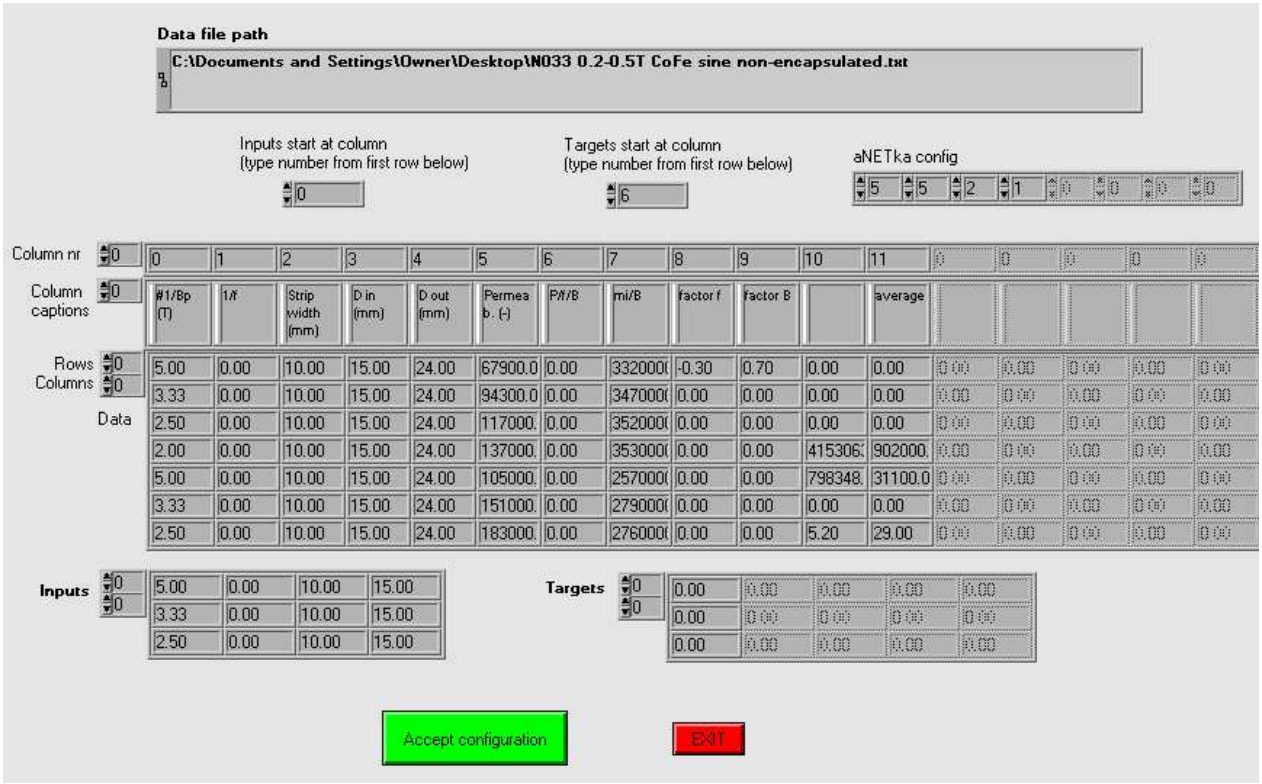

Fig. 6. Input data

d) **Test cases** (Fig. 7). The number of test cases (see also section **Overtraining**) can be selected to any number not smaller than 0 and not greater than the total number of cases.

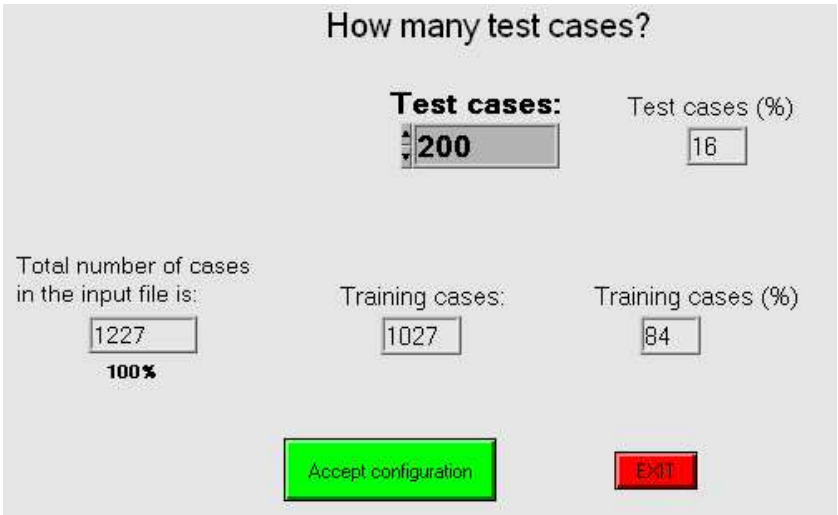

Fig. 7. Test cases

e) **Iterations and other values** (Fig. 8). In this dialog box there are 4 values to set: total **number of iterations** (epochs) to execute, **total RMS percentage error** (combined for all ANN outputs) at which the training should stop (the RMS error decreases gradually during the training; if a certain error is satisfactory then there is no need for further training; only the training set is taken into account here), **momentum** (see description above) and **screen update** (the screen will be refreshed after the number of

iterations specified; small values may slow down the execution, large values will result in no control over the training process).

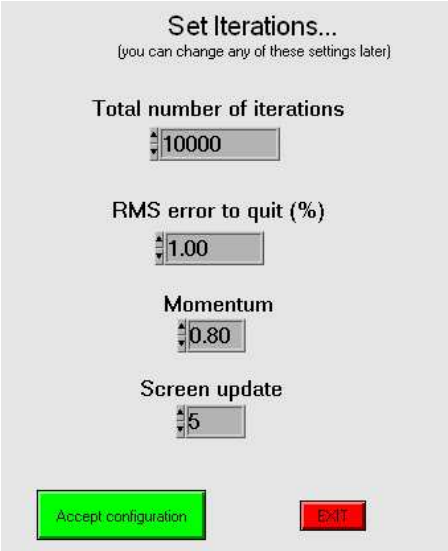

Fig. 8. Iterations and other values

After pressing the green button in the dialog box *Set iterations* the training will start immediately. The main screen of the training program contains several indicators (see Fig. 9). Apart from the values chosen during the configuration process described above the following are displayed:

- Waveform graph. This graph presents all the outputs of the ANN as a single waveform graph. It provides quick orientation how far the trained ANN is from the target values. Any part of the graph can be easily zoomed (in our out) by using the tools provided at the top of the graph.
- XY Graph. Basically it presents the same data as the Waveform graph, but in a different way. The user can see not only all the data, but also how far the training cases (blue dots) and the test cases (yellow spots) are from the target values (red straight line).
- Function. Activation function for all neurons.
- **aNETka** configuration. Graphical representation of the chosen ANN configuration.
- RMS history. This is a running graph (maximum number of displayed iterations is 500), which presents the history of RMS % errors for the training set (green curve) and the test set (red curve). This graph allows the user deciding if the ANN is being overtrained (see the definition of overtraining above).
- Progress bar. The bar displays the completed iterations as a green field, and the iterations to be completed as a red field. The percentage to be completed is displayed just below the bar.
- Correlation (train and test). The correlation gives information about the degree of "agreement" with the target data. Correlation = 1.0 denotes perfect match. **aNETka** calculates the correlation as defined in [5].
- RMS % error (train and test). The RMS percentage error is calculated for the whole set of data (i.e. all cases are included, see the explanation above). The RMS % error is calculated as: $RMS\% = \sqrt{\left(\sum\left[\left(\frac{T-O}{T}\right)^2\right]/N\right)} \cdot 100\%$, where: T – targets, O – ANN outputs, N – number of values.

- Total RMS % error. Defined as above, but the training data and test data are combined together.
- Red LED next to Iterations. If the total number of iterations is reached, the program stops and this LED is turned on.
- Red LED next to RMS to quit. As above but it indicates that the program was terminated because of RMS condition.
- STOP button. The program can be stopped at any time by pressing this button (or key Escape from keyboard). During normal iteration this button is green. If the program stops it turns red.

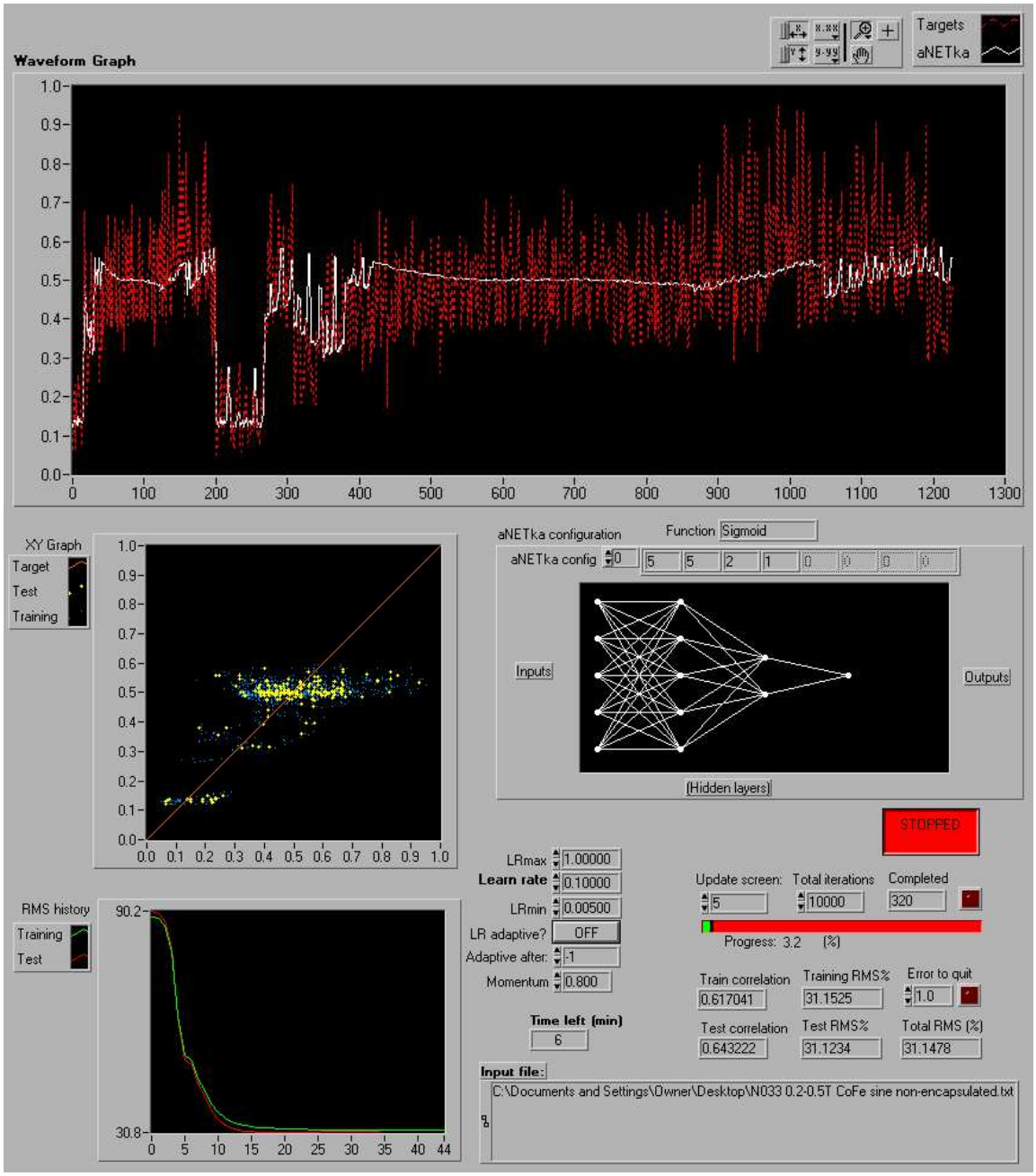

Fig. 9. The main screen of training program

The main program can stop for three reasons: total number of iterations reached, RMS error reached or the STOP button pressed. In any of these cases the program terminates immediately and asks the user for the place and name of the output file to save the trained

ANN. The dialog box is a standard Windows dialog – at the top a disk/folder can be selected, and at the bottom the name of the files can be specified. It is advised that the name of the output file has an extension "*txt*" – such file can be easily opened in order to check the configuration of the ANN (number of neurons, input file, the captions of input and output data used for training, etc).

If the user does not want to save the trained network, then the button *Cancel* should be pressed at this point. An error box will appear, as shown in Fig. 10. If the button *Continue* is pressed the program will repeat the saving procedure (asking for the name of the file). If the button *Stop* is pressed the file is not saved and the program terminates immediately.

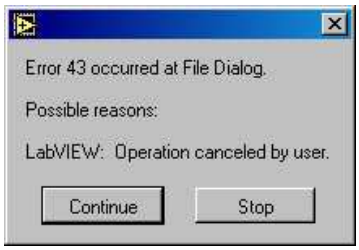

Fig. 10. Error box

If the data was saved in the output file then the user can see all the configurations and weights saved to that file. These data will be later needed for recall program, as described below.

5.4. Batch recall mode – program *Recall aNETKa (batch).vi*

As in the training mode, in batch recall mode the input data has to be saved in the ASCII text file, tab delimited, with captions in the first row, and 3 letters of file name extension. The dialog boxes are very similar to the training mode, however the user has to select two files at the beginning – the trained **aNETka** file, and the input data file. The outputs are calculated immediately for the whole data set (all cases are presented to the network only once, hence it is very fast). The targets are not presented to the network, and they are not even selected in the configuration process. In order to simplify the data processing, the input file is copied to the same location, the name of the new file is modified to contain the string "*(aNETka)*", and the calculated outputs are added as a last column in the file, for convenience. The recall is sensible only if **aNETka** was previously trained. In such case the captions of appropriate targets are saved in the trained file, and they are added to the output file in the recall mode.

The screen of the program contains a waveform graph, configuration graph, configuration array and files paths, as shown in Fig. 11.

It is extremely important that the data with which the trained network is recalled is within the limits used for the training. For example, the trained ANN had configuration 3-5-1, and the input values varied: first input from –10 to +10, second input from 0 to +100 and third input from +10.5 to +11.3. Then in the recall mode none of the input data can be outside those ranges. Otherwise the ANN can return a number, which is completely out of any range. This is an intrinsic problem of any ANN.

Of course, the number of input values has to match the number of input values used in training. A greater number of inputs will be truncated down to the required value, but a lower number of inputs will result in completely incorrect outputs from the ANN. (The software does not check for errors here.)

Important!

The user has to be very careful when using the recall mode (and also training), because **aNETka** does not check for any errors – it is the user's responsibility to supply the data exactly as required without the smallest mistake. If some data was incorrect then some of the indicators can show *NaN* instead of a number. *NaN* means "not a number" and its appearance indicates critical error during the execution of the program. In such case the program should be restarted.

If the *NaN* appears during the training then the program should be stopped immediately and training repeated. (It is not dangerous to run the program further, but completely pointless since the training cannot return to correct procedure.) The appearance of *NaN* can also mean the divergence of the ANN – in such case lower setting of the learning rate or momentum (or both) should be chosen in the next run.

The only case when error *NaN* is not critical is when the number of the test cases is set to zero. In the calculation of *RMS error* and *Correlation* there is a division by the number of test cases – therefore division by zero will return *NaN*. Only the values related to the test cases are affected – the rest of the training procedure performs as usual.

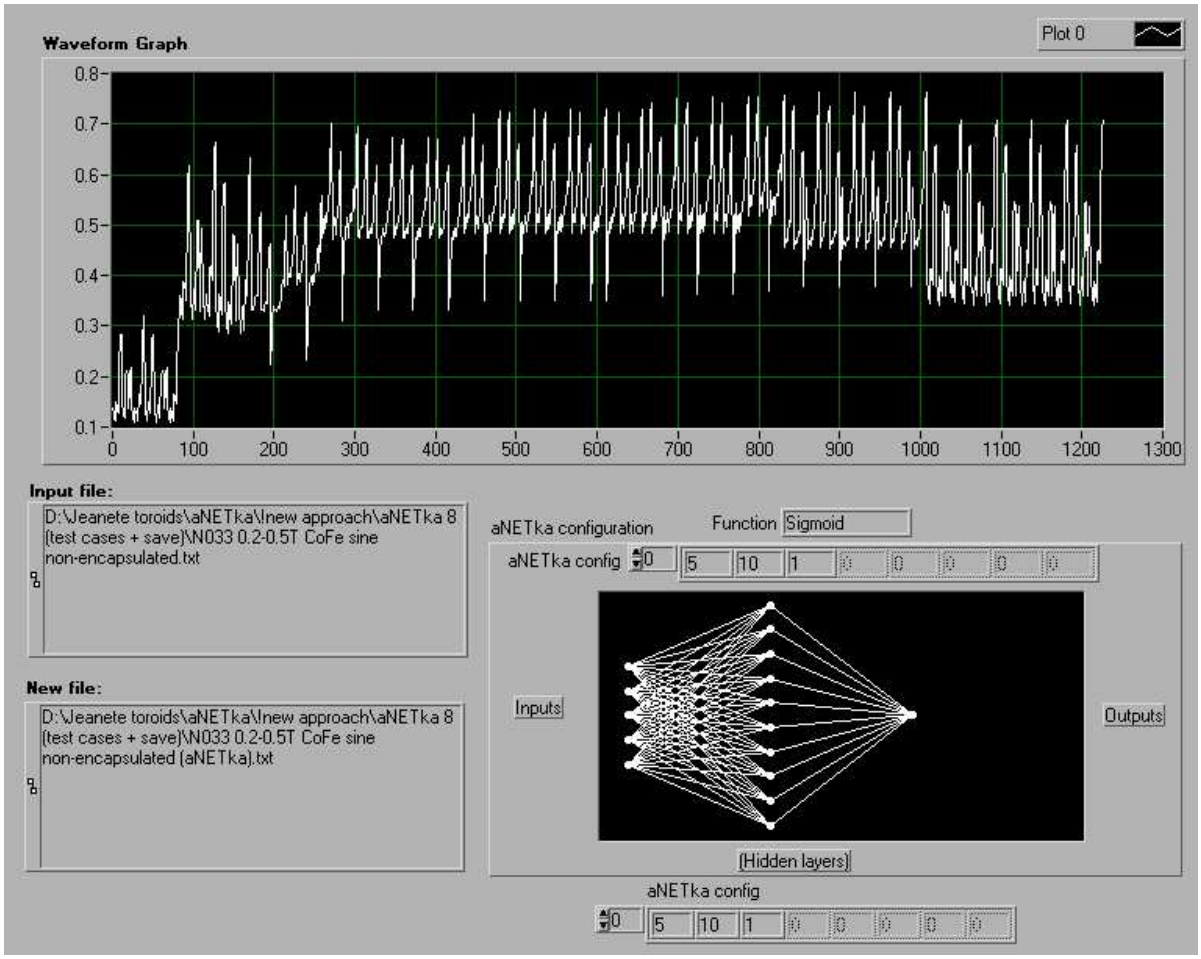

Fig. 11. Screenshot from batch recall program

5.5. Single recall mode – program *Recall aNETka (single).vi*

Single recall mode is very similar to batch recall. However, the input data cannot be saved in the file, but the user has to type in all the required input data after running the program. Before running, the program will ask as many times for the input value, as there were input values used for training. If the captions were provided in the training mode they will be shown for each input (to make the selection of values easier). Also, the ranges of training data for each input will be given (even if the captions were not provided), see Fig. 12.

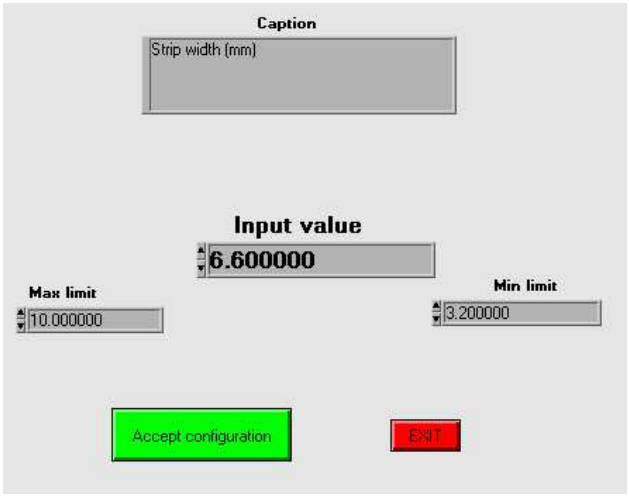

Fig. 12. An example of a dialog box asking for subsequent input data

After all the inputs are provided, the main screen is shown and the program immediately calculates the outputs. The table of calculated outputs is shown at the right of the main screen (see the green table in Fig. 13).

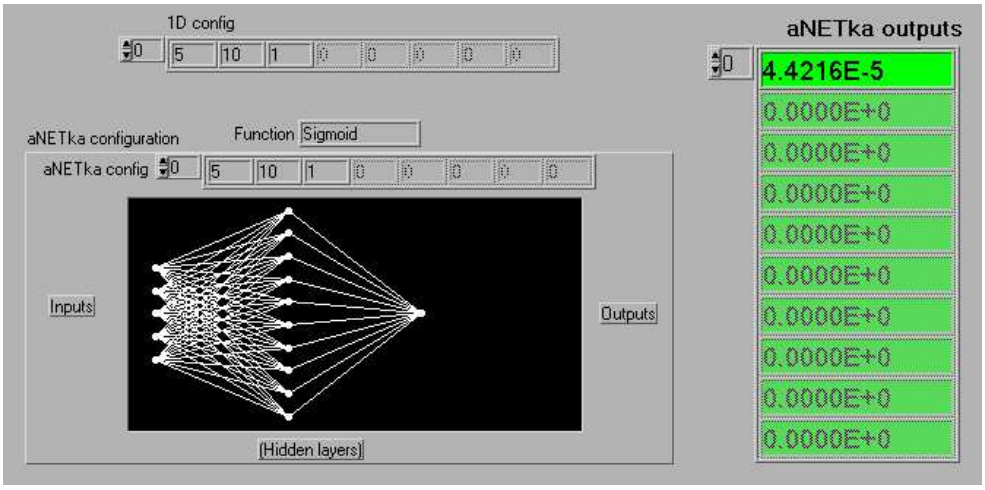

Fig. 13. The main screen of the single recall program

The format of the output data is similar to that used in Microsoft Excel. For example, the number shown in the table in Fig. 13 is 4.4216E-5, which should be understood as $4.4216 \cdot 10^{-5}$. The table shows only 10 outputs, but larger numbers of outputs are accessible by scrolling the index of the table – if the number "0" in at the top left corner of the table is changed to any higher number, then all the values will be scrolled up, hence the values positioned lower in the table can be accessed. The output values calculated by **aNETka** are not saved to any file. Such mode is very useful if there is a need of quick and frequent "look up" into the trained ANN.

Again, **aNETka** does not check for any errors in this mode, so it is the user responsibility to provide correct data.

6. Examples

aNETka was initially designed to predict the power loss and magnetic permeability of magnetic toroidal cores. A range of cores was selected (28 cores), and the power loss and permeability were measured at some frequencies (11 frequencies) and at various flux densities (4 values). Therefore, there were 27·11·4 = 1232 data sets to be used by the ANN. One of the aims of the project was to predict the properties of the cores for various geometries. Thus, 5 inputs were selected: flux density (varied for each core), frequency (varied for each core), inner diameter (varied for most of the cores), outer diameter (varied for most of the cores) and width of the strip (varied for most of the cores). (The last parameter is also shown in Fig. 12.)

Although **aNETka** scales the input data to make use of the full range of ANN calculations, in some cases additional data processing might be required before starting the training procedure. In the case of magnetic cores, the power loss varies five orders of magnitude (from 10^{-3} to 10^{2} W/kg). It is very difficult for any ANN to accurately approximate such changes scaled down to range from 0.05 to 0.95. Therefore, the span of the input values should be reduced if possible. For example, the power loss of magnetic cores strongly depends on the frequency, thus instead of supplying just the power loss value to **aNETka**, the power loss was divided by frequency (frequency varied from 20 Hz to 20 kHz, that is 3 orders of magnitude). The range of the "new" outputs was therefore reduced by around 1000 times, which means that it was 1000 times easier for the ANN to model the small changes. The great trends in data are usually very easy to model, but if they are much greater than the small ones, then they will overshadow the training process. However, if the large trends can be linked directly to one of the inputs, then there is no need to try to model them with the ANN (they can be feasibly found by simple division or multiplication). It is better then to let the ANN do the hard work. Otherwise, the very small values cannot be predicted with sufficient accuracy, in some cases resulting in non-physical negative values!

The waveforms depicted in Fig. 9 show how close (or far) the trained data (white curve) are from the target data (red curve). If the test cases are chosen to be included in the training procedure, then their agreement is displayed at the beginning of the graph (although their presence is not accentuated in any way).

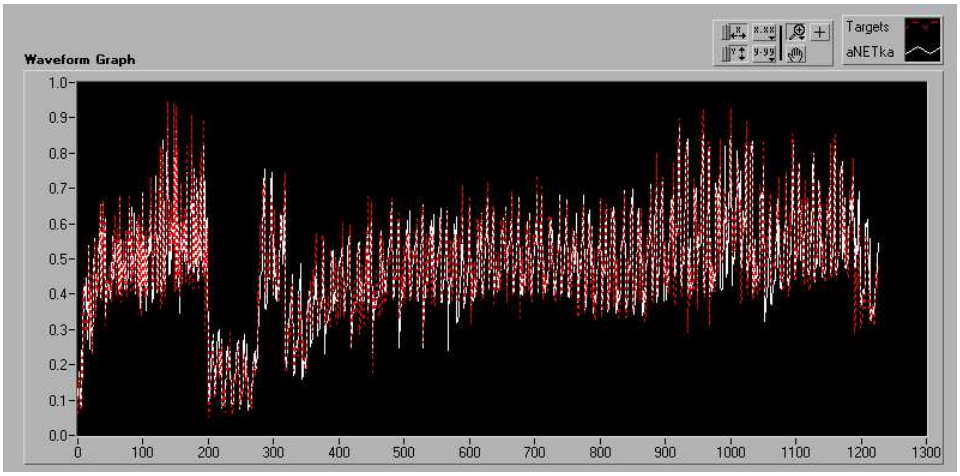

Fig. 14. The same training data as in Fig. 9, but **aNETka** was trained for 10,000 iterations

After 320 iterations the main trend has been found, but the sharp fluctuations of the data were not (compare white and red curves). If the ANN was trained for 10,000 iterations then the mapping of the training data was much better, as shown in Fig. 14. For the training

shown in Fig. 9 the total RMS error was 31%, whilst for the data shown in Fig 14. the RMS error was decreased to 11%.

The value of 11% may not seem to infer very accurate prediction, but for this particular case the result was much better than any other approach covering that wide spectrum of measurements.

To prove that **aNETka** can approximate with much greater accuracy, an example of a simpler non-linear function is shown (see Fig. 15). The functions were arbitrary chosen to be non-linear, as: $Input\_1_{i+1} = Input\_1_i \cdot 2$; $Input\_2_{i+1} = Input\_2_i + Input\_2_i \cdot 0.05$; and Output_1 = (In1+In2)/(In2-In1) – the corresponding data is shown in Table 1. For convenience the values from Table 1 are stored in a file: " *example 2 inputs, 1 output.txt* ".

Table 1. Example of training data (file: " *example 2 inputs, 1 output.txt* ")

Input_1	Input_2	Output_1
1	20.00	1.11
2	21.00	1.21
4	22.05	1.44
8	23.15	2.06
16	24.31	4.85
32	25.53	-8.89
64	26.80	-2.44
128	28.14	-1.56
256	29.55	-1.26

The configuration of the ANN was 2-5-1 (2 inputs, 5 hidden neurons, 1 output). The targets are shown as red squares, and trained values as white line. As can be seen, the approximation is accurate. The RMS error was below 1% (that is the RMS percentage error for all the training points). Also, it should be noted, that the inputs from Table 1 reach quite high values (sever hundreds), whilst some of the output values are negative. All these ranges are handled automatically so that arbitrary values can be predicted by the ANN. (The output of the sigmoid function can change only between 0 and 1.) This process is transparent to the user, but it should be remembered that input/output data from any range will be scaled down to range from 0.05 to 0.95. For example, it is not good practice to use data, which varies from 0.0001 to 10000 – in such case the lowest values will be very poorly trained. The user has to find a way of narrowing this range, for example by relating to other input values (dividing by one of the inputs, etc).

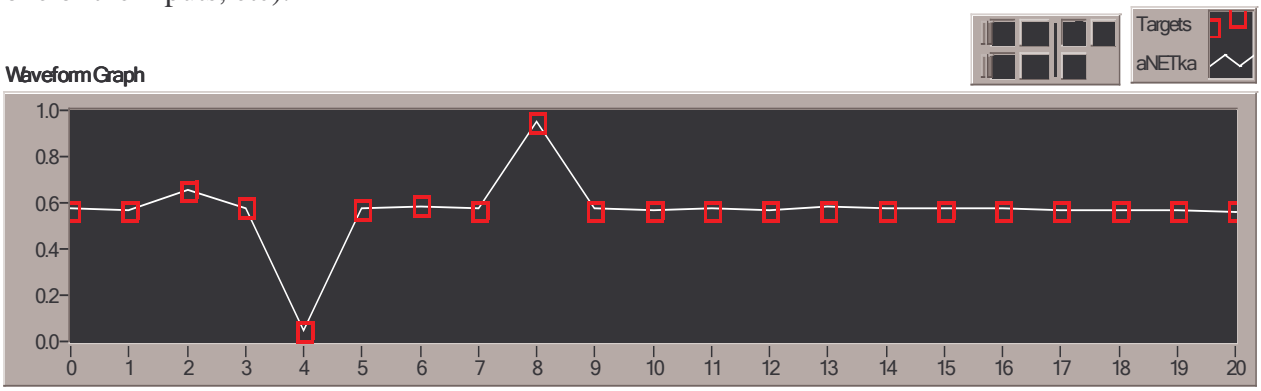

Fig. 15. Approximation of some non-linear function

The trained network from example shown in Table 1 and Fig. 15 is saved to a file: " *trained aNETka example.txt* ". The batch-recalled outputs are stored (together with the original data) in a file: " *example 2 inputs, 1 output (aNETka).txt* ".

7. Summary of aNETka's features:

Maximum number of layers – 8
Maximum number of neurons per layer – 256, but also limited by the memory of the PC, and practically by the training time for extremely large training data.
Activation functions - Linear, Sigmoid, Tangent hyp.
Type of ANN - fully interconnected, feed-forward, back-propagation, min. one input, hidden and output layer.
Data reading, processing and saving – automatic.
Data normalizing – linear from 0.05 to 0.95 (automatic, transparent to user).
Weights randomization – from -0.1 to 0.1.
Modes of operation - Training, Batch recall, Single recall.
Programming software – LabVIEW 5.1.
Format of input/output data – ASCII text, tab delimited, first row captions, compatible with most spreadsheet software.

8. License

9. References

[1] Internet: http://en.wikipedia.org/wiki/Artificial_neural_network
[2] Book: D.T. Pham, X. Liu, Neural networks for identification, prediction and control, Springer-Verlag London Ltd., UK, 1999
[3] Internet: http://www.ni.com/labview
[4] Internet: http://www.ni.com
[5] Internet: http://en.wikipedia.org/wiki/Correlation

10. Acknowledgements

The author would like to acknowledge the following sources, which helped invaluably in understanding of the back-propagation algorithm of **aNETka** (in the order of relevance):

1. Internet: http://www.google.com
2. Internet: http://galaxy.agh.edu.pl/~vlsi/AI
3. Internet: http://nrn.prv.pl (in Polish)
4. Book: Phil Picton, Neural Networks, PALGRAVE, Basingstoke, 2000
5. Book: Stephen T. Welstead, Neural network and fuzzy logic applications in **C/C++**, WILEY, New York, 1994
6. Internet: http://www.qnetv2k.com