JKAU: Eng. Sci., **Vol. 16** No. 2, pp: 97-113 (2005 A.D. / 1426 A.H.)

Development of a Cross-Platform Artificial Neural Network Component for Intelligent Systems

Gültekin B. Çetiner[1] and H.M. Aburas[2]

Department of Industrial Engineering,
Faculty of Engineering, King Abdulaziz University,
P.O. Box 80204 Jeddah 21589, Saudi Arabia
[1]gultekin@drcetiner.org and[2] haburas@kau.edu.sa

Abstract. In recent years, cross-platform application development has become a major issue in the area of information systems technology. Modeling and design tools have been developed for multiple platforms due to their potential capability in many environments. Unified Modeling Language (UML), as an example, combines all stages of application development life cycle, and generates cross-platform codes using object-oriented methodology. Artificial Neural Networks (ANNs) have been adapted extensively for solving a wide range of problems efficiently. Nowadays, systems need to be developed within the shortest possible time because of the competence in the market. Therefore, one needs a quicker way of embedding ANNs into Software Development Life Cycles on multiple platforms. This approach requires a cross-platform ANN tool. This paper describes a component that is suitable for UML environment and discusses some of its features. Finally, two case studies are presented in order to illustrate the usefulness of the suggested component.

Keywords: Neural Networks, Component Based Intelligent System Development, Cross-Platform Application Development.

1. Introduction

Artificial Neural Networks (ANNs) have been used successfully for solving problems in many areas ranging from computer vision to business forecasting via the available data[1]. Nowadays, systems need to be developed within the

shortest possible time because of the competence in the market. Therefore, we should have a quicker way of embedding ANNs into our software development life cycle.

Neural networks are composed of a large number of interconnected units divided into input, output, and hidden nodes. A single processing unit merely sums up the weighted activation on its inputs, transforms this sum according to an activation function, and passes the resulting function to its output. Therefore, in general terms, information processing in neural networks consists of the units transforming their input into some output, which is then modulated by the weights of connections as inputs to other units. Learning in these systems is defined in terms of the total adjustments of the weights continuously so that the network's output tends toward the desired output without involving changes to the structure of the network. Neural networks are trained by specifying some constraints such as learning parameters, network structure, and training examples. When learning is complete, or stopped at an adequate level, knowledge is said to be represented by the optimized connection weights among processing units in the entire network.

Unlike traditional expert systems where a knowledge base and necessary rules have to be defined explicitly, neural networks do not need rules. They, instead, generate rules by learning from given examples. This makes ANNs general purpose classification tools to be used in pattern recognition and classification systems. Neural networks provide a closer approach to human perception and recognition than traditional computing. When inputs are noisy or incomplete, neural networks can still produce reasonable results. Neural networks are used successfully in many areas such as Natural Language Processing, Speech Processing, Data Compression, Computer Security, Image Recognition (Optical Character Recognition, Texture Classification, Handwriting Recognition, Target Classification, Industrial Inspection), Optimization problems, Signal processing (Prediction, System Modeling, Noise Filtering, etc.), Financial and Economic Modeling and Control Systems. There are other areas in which neural networks might be applied successfully. One of these areas includes intelligent e-commerce applications in which customer buying intentions are recognized by the vendors. Possibility and ease of the use in many areas make ANNs an ideal solution for many intelligent systems[2]. However, embedding Neural Networks into such systems in shorter time limits is also important for the current software development projects.

Such constraint requires the use of Rapid Application Development (RAD) environments based on modeling approaches (specifically object oriented ones) to analyze, design, and build systems. Computer Assisted Software Engineering (CASE) Tools help developers building a system in very short time. This

approach, also known as model driven development, makes use of a set of Object-oriented Modeling notations called Unified Modeling Language (UML) that supports many of the object-oriented languages[3]. Another method namely Fusion method developed by the Object-oriented Design Group at Hewlett Packard Laboratories, Bristol builds on existing, first generation, methods, and provides a direct route from a requirements definition through to a programming language implementation[4].

A software component has been described in this paper to embed ANN functionality into such applications. Architecture of the component is designed in an object-oriented fashion for the purpose of building such applications easily. In addition, the paper describes a component that is suitable for UML environment and discusses some of its features. Finally, two case studies are presented in order to illustrate the usefulness of the suggested component.

2. Cross-Platform Applications Development

While Cross-platform means the ability of using a piece of software under different platforms (usually different operating systems), there are various ways to achieve this. There is a need of business analysis as a distinct stage during the Information Systems (IS) development process. A modeling methodology in order to be capable of modeling the business analysis stage should be able to provide sufficient level of abstraction and business analysis specific concepts that represent the organization independent of any design or implementation issues. The spiral model in Object-oriented software development is preferred to the waterfall model since there exists a lot of similarity in the modeling concepts used in business analysis in IS development and Object-oriented Analysis in Object-oriented software development[5]. Software integration is an important and useful approach to software reuse and cross-platform development. Most of software integration approaches use data-integration paradigm, *i.e.*, a common data format and (or) source code are required for integration. In a functional integration approach suggested[6], software is integrated by functionalities. No common data format or source codes of the integrated softwares are required in their approach. However, instead of a common data or source code, they had to develop a specification language[6]. Here a computer programming language (and a visual development environment) that is capable of working with multiple operating systems is employed for developing cross-platform information systems. This language has to provide extensive capabilities for cross-platform visual development. One way of achieving this might be to use a platform independent language such as Java or C. The resulting application developed by using a Cross-Platform Language may be run under both Windows and Linux Operating Systems. However, demanding requirements for

visual development suggest different alternatives since Java is slow as being a run-time interpreter and C is lacking standard libraries needed for cross-platform visual functions supporting relational databases in intelligent information systems. An ideal alternative is a development environment specifically designed for cross-platform application development with extensive graphical, relational database design capabilities. This alternative should also have a common object-oriented programming language with a standardized set of libraries common to both platforms. Delphi[1] and Kylix[2] are chosen for this purpose. Both Delphi and Kylix are object-oriented, visual programming environments for rapid development of highly efficient cross-platform applications with a minimum of manual coding. They provide a common comprehensive class library called the Borland Component Library for Cross Platform (CLX) and a suite of Rapid Application Development (RAD) design tools with extensive graphic and database capabilities. The Cross-Platform Artificial Neural Network (ANN) component described herein makes use of this CLX library components and objects in a truly object-oriented fashion. Figure 1 shows some of the main classes available in CLX library. This class hierarchy actually shows inheritances for Cross-Platform Artificial Neural Network component starting from *TObject.*

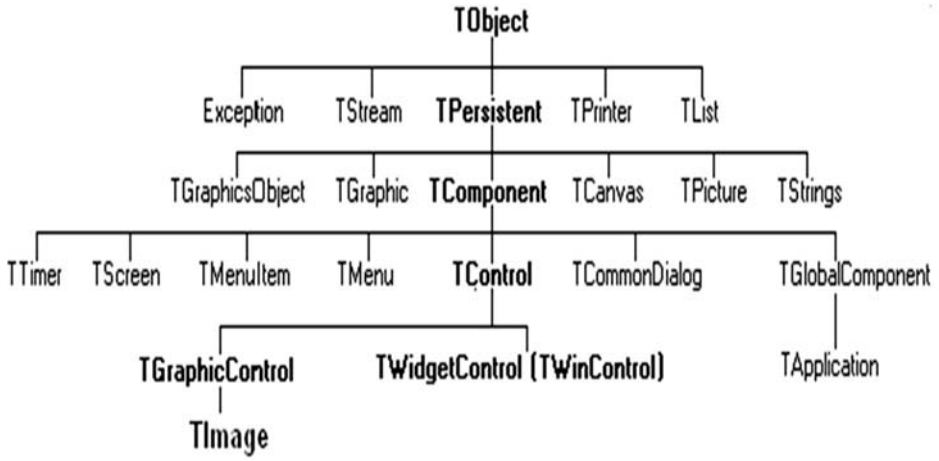


Fig. 1. General schema of CLX library used in artificial neural network component.

Some of the important classes used in development of cross-platform ANN component are as follows:

- *TObject* is the ultimate ancestor of all CLX objects and components and encapsulates fundamental behavior common to CLX objects by introducing

[1,2]Delphi and Kylix are both trademarks of Borland Corporation running under Windows and Linux platforms respectively. The main language for both is object-Pascal with extensive libraries for Rapid Application Development.

methods that create, maintain and destroy instances of the object by allocating, initializing, and freeing required memory and implementing other important functions.

- *TPersistent* is the ancestor for all objects that have assignment and streaming capabilities (reading from and writing to various kinds of storage media, such as disk files, dynamic memory, and so on).
- *TComponent* is the common ancestor of all components that can appear in the form designer and has the capabilities of appearing on Component palette and be manipulated in the form designer, owning and managing other components, and having enhanced streaming and filing capabilities.
- *TControl* is the base class for all components that are visible at runtime. The controls are visual components, meaning the user can see them and possibly interact with them at runtime. All controls have properties, methods, and events that describe aspects of their appearance, such as the position of the control, the cursor or hint associated with the control, methods to paint or move the control, and events that respond to user actions.
- *TGraphicControl* is the base class for all lightweight controls. It supports simple lightweight controls that do not need the ability to accept keyboard input or contain other controls. *TGraphicControl* provides a *Canvas* property for access to the control's drawing surface and a virtual *Paint* method called in response to paint requests received by the parent control.
- *TImage* displays a graphical image on a form. It is a base class for drawing, saving, and loading Artificial Neural Network structures by using properties and methods available in its TPicture property.

Figure 2 shows the object hierarchy in well-known object-oriented Unified Modeling Language (UML) notations. The seven objects are not shown in detail since they are available in manuals of Delphi, C++ Builder, and Kylix.

3. Cross-Platform Artificial Neural Network Component

A preliminary study has been made by the authors for the Artificial Neural Networks component described herein and a concise introduction has been given without any proper case study[7]. It also lacked some of the main features such as Cross-Platform ability addressed within this paper. This paper also describes the component in more detail by two case studies to illustrate its usefulness.

Artificial Neural Network component herein is a general purpose cross-platform component to be used under Delphi, C++ Builder and Kylix development environments to develop intelligent programs ranging from signal process-

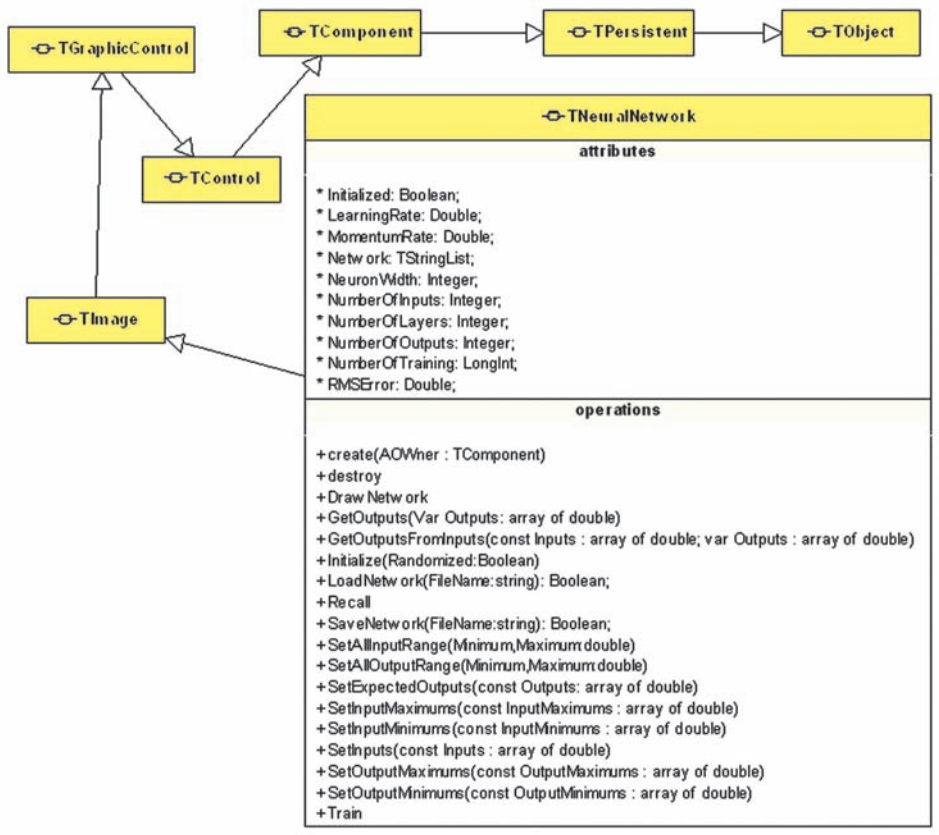


Fig. 2. UML description for *T Neural Network* component.

ing to intelligent internet applications. The component is based on a Multi-Layer Perceptron with Back Propagation Algorithm[8]. It supports other concepts to cope with some learning problems. It has a momentum coefficient to help the network escape from local minima problems and also a parameter of learning rate which regulates the speed of conversion during training. It also supports biases to all hidden layers and output layer. The transfer function used in each neuron of hidden layers, and output neurons is currently sigmoid function. But other functions such as hyperbolic tangent and others may be added to the component easily. It uses a mixture of Artificial Neural Network Technology to converge to correct solutions with shorter training times. It is quite flexible and accepts different ranges of inputs and outputs defined by the user. Network allows any number of hidden layers and the structure of the network can be visually displayed as in Fig. 3. When inserting 9 numbers to the network property at design-time, a seven hidden layer network is created automatically with the related number of neurons in each layer.

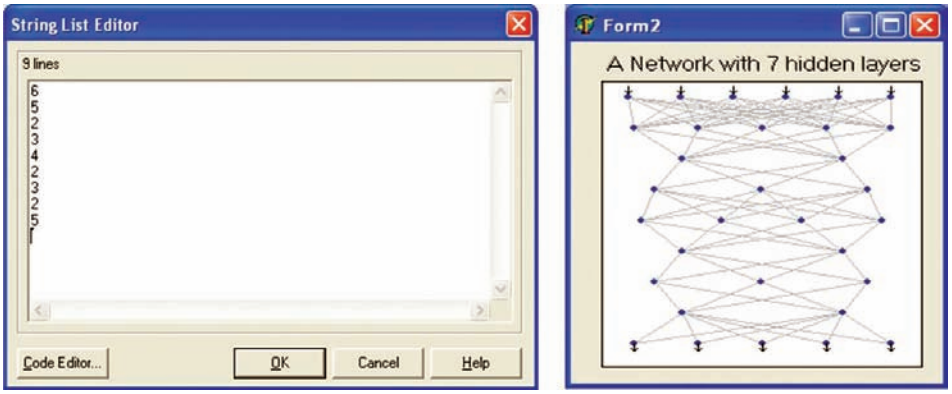


Fig. 3. Network after defining values in Network property on the left.

The three application development environments mentioned above use Rapid Application Development (RAD) strategies for rapid development of applications for deployment on Windows and Linux Operating Systems. They are based on well defined component libraries for this purpose. These class libraries are made up of objects, some of which are components or controls. Components are subset of objects and can be manipulated at design time without writing code. Components are objects in the true object-oriented programming (OOP) sense since they

- *encapsulate* a set of data and data access functions
- *inherit* data and behavior from the objects they are derived from
- operate interchangeably with other objects derived from a common ancestor, through a concept called *polymorphism.*

Building intelligent applications using databases (single-tier or multi-tier), internet and intranet, and graphics is easy by using ANN component with other components. Furthermore, embedment of Computer Aided Software Engineering (CASE) tools make this process much easier by automating source code generation from class and object definitions. The class and object definitions may be categorized and reused by design patterns specific to some problem areas[9].

Cross-platform Artificial Neural Network component is a general purpose component and adding the functionality to an application is as easy as dropping an instance of component onto the design form as in Fig. 4. Using object-oriented modeling tools, this component could be added to the Unified Modeling Language (UML) diagrams of the system such as class diagrams just like other objects and the whole process can then be automated to obtain the source at the end.

The properties and methods in *TNeuralNetwork* component are described throughout the following implementation phases:

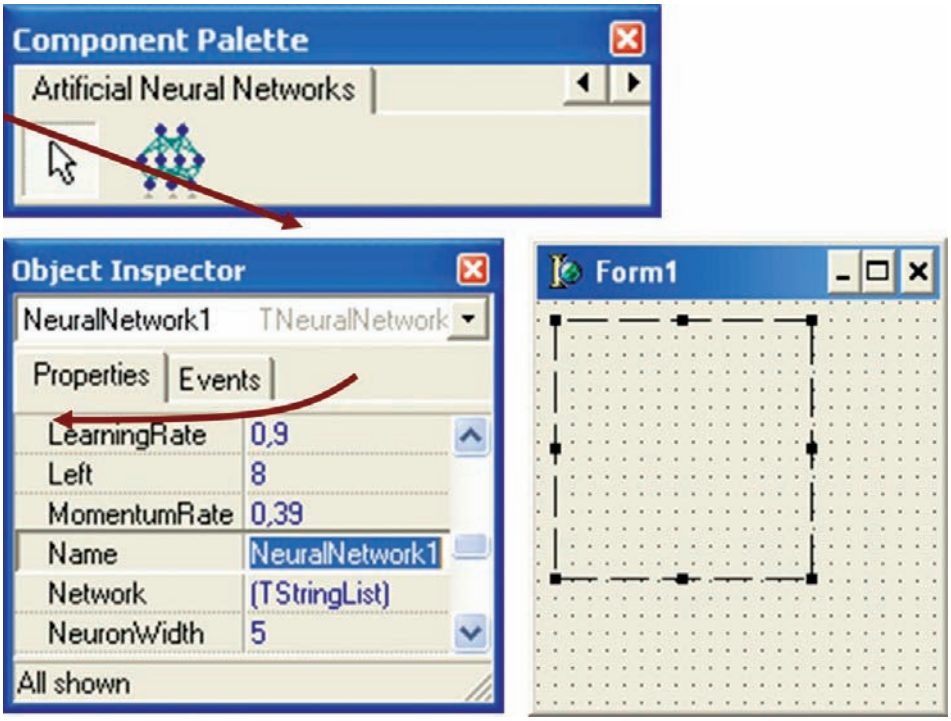


Fig. 4. Adding an instance of ANN component onto design form.

- Defining the Network Structure.
- Initialization.
- Specifying minimum and maximum values for inputs and outputs.
- Training.
- After Training.
- Using the Trained Network.

3.1 Defining the Network Structure

Network property of the component is the most important property and it defines the structure of the network. By setting the values of this property we can define the number of inputs, number of hidden layers, neurons in each hidden layer, and the number of outputs. If we want to construct a network as shown in Fig. 3, we can specify 2, 3, 2 and 1 values using object inspector during design time. To define the structure of the network run-time, just add the following codes:

```
nn1.Network.clear;
nn1.Network.Add('2');
```

```
nn1.Network.Add('3');
nn1.Network.Add('2');
nn1.Network.Add('1');
```

3.2 Initialization

The individual properties set by *Network* property can be accessed at run-time using read-only *NumberOfInputs*, *NumberOfLayers*, *NumberOf-Outputs* properties. We can display the graphical structure of the network by calling the *DrawNetwork* method. The following code displays the network structure and saves the image as bitmap file on the hard disk. Due to object-oriented design, the picture property inherited from *TImage* makes it very easy to save or load network at any time.

```
nn1.DrawNetwork;
nn1.Picture.SaveToFile('c:\network.bmp');
```

NeuronWidth property can be used to change the size of the neurons in neural net picture. After the network structure is defined, the network has to be initialized using the initialize method. This can be done by just writing the following code:

```
nn1.initialize(true);
```

Each initialization step verifies the structure of the network and assigns random values. After initialization, the learning process has to be repeated if the network is not saved. Therefore, before initialization, the network needs to be stored using *SaveNetwork* method. *True* parameter in heading initializes the built-in random number generator with a random value (obtained from the system clock) so that each initialization assigns completely different random values to connection weights. *False* parameter starts initialization with the same set of random values. *LearningRate* and *MomentumRate* are two important properties which affect learning curve. Since they are design and run-time properties, they can be changed during run-time or they can be fixed at the beginning of learning.

Initialized is run-time and read-only property to query whether the network is initialized. This is important because each network structure has to be initialized before training. We can ensure the network is initialized before training by implementing the following code:

```
if not nn1.Initialized then
nn1.Initialize(True);
```

3.3 Specifying Minimum and Maximum Values for Inputs and Outputs

The minimums and maximums have to be supplied for both inputs and outputs before starting the training phase. The methods available for this purpose are:

SetInputMinimums, SetInputMaximums, SetOutputMinimums, Set-OutputMaximums, SetAllOutputRange and *SetAllInput Range.*

SetAllOutputRange and *SetAllInputRange* methods are the easiest way to perform this. As an example, the following code specifies all input-output minimum values to 5, and maximum values to 10.

```
nn1.SetAllInputRange(5,10);
nn1.SetAllOutputRange(5,10);
```

For specifying the minimum and maximum values for each individual input or output following methods might be used.

SetInputMinimums, SetInputMaximums, SetOutputMinimums and *Set-OutputMaximums.*

3.4 Training of Neural Network

Training of the network is an iterative process. For each iteration, the network has to be supplied with inputs and their associated outputs using *Set-Inputs* and *SetExpectedOutputs* methods, respectively. The ranges of the inputs and outputs have to conform the ones specified by *SetInput-Minimums, Set-InputMaximums*, *SetOutputMinimums*, *SetOutputMax-imums*, *SetAllOutputRange* or *SetAllInputRange* methods. The following code sets inputs and desired outputs for this input set:

```
nn1.SetInputs(input);
nn1.SetExpectedOutputs(Desired);
```

After setting inputs and outputs, the *Train* method can be used to train network for one cycle as follows:

```
nn1.Train;
```

Following the *Train* method, *RMSError* (Rooted Mean Square Error) property is calculated automatically. This read-only value can be used to draw an error graph to track the learning process. A different set of input and corresponding output values are presented to the network during each training iteration. The iteration may be stopped after a satisfied *RMSError* level or a predefined maximum iteration number is reached. It is suggested that the examples should be presented to the network in a completely random fashion.

3.5 After Training

When the learning is completed, or at least a satisfactory RMSError level is reached, the network must be saved using *SaveNetwork* method. *SaveNetwork* method saves the weights and input-output minimums and maximums of the network to use for recognition and other purposes. The next section describes the recognition phase.

3.6 Using the Trained Network

The network should be saved after it learns all patterns. As soon as the network is trained or loaded from a file it can be used for recognition purposes. To present an unknown input pattern to the network and to get the answer from it, the following steps are needed:

1. Use *SetInputs* to specify the input neurons.
2. Use *Recall* method to query the network.
3. Use *GetOutputs* method to get the answer of the network regarding the outputs for the inputs at Step 1.

RecallOutputs can also be used to specify the input pattern and to get the outputs. In this case the first three steps are not needed.

4. Case Studies

Two case studies are described in this section to illustrate the ease of use of the suggested Artificial Neural Networks Component. The following two sections define the problems in both case studies together with solutions.

4.1 Using Multiple Neural Networks in Parallel to Solve any Classification Problem

Within this case study, the problem was to solve by the use of Neural Network, any classification problem with sufficient inputs and corresponding outputs. A generic application was developed for this purpose as shown in Fig. 5. The application uses 5 different instances of ANN component each having a different number of hidden neurons in hidden layer. The number of hidden neurons is same as number of inputs defined in the first step. The development of this application took only one day. Many different linear problems such as 'logical and' and 'logical or' sets have been trained within a few minutes and tested with high accuracy. However, the most important problems were the non-linear ones. The screen shot shows the steps of solving a simple but yet a difficult non-linear problem known as 'logical xor'. Figure 6 compares a linear problem with 'logical or' and a non-linear problem with 'logical xor'. As seen from

the figure, there is always misclassification with a linear separation line in 'logical xor' problem since the problem is a non-linear one whereas all the data in 'logical or' may be classified correctly.

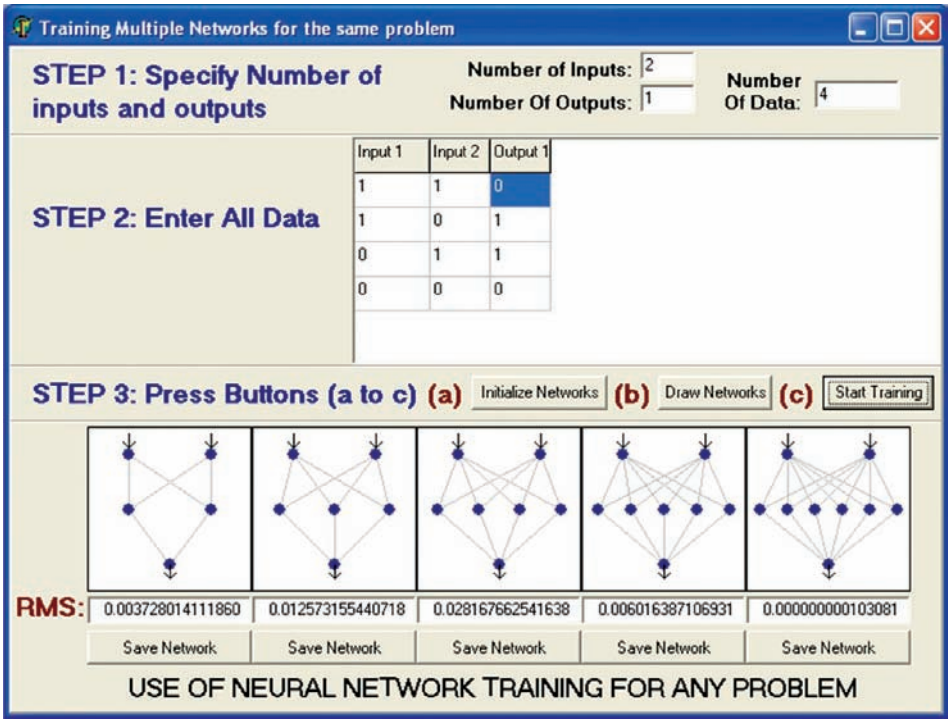
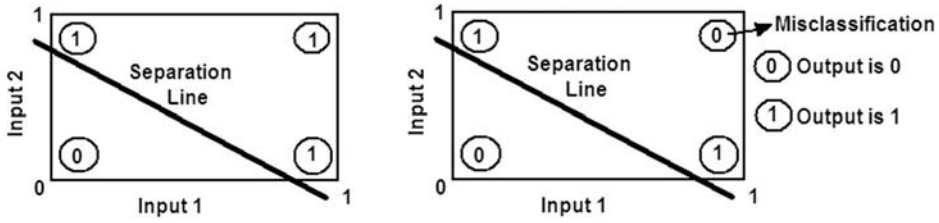


Fig. 5. A sample program for solving any problem using any number of instances for ANN component.



(a) Logical or' problem **(a) 'Logical xor' problem**

Fig. 6. Comparison of classifications from 'Logical or' and 'Logical xor'.

The numbers of hidden neurons in networks were 2, 3, 4, 5, and 6. All of the networks converged to solutions. However, the last network with 6 hidden neurons gave the best result. The steps for classifying any set of data by means of this sample application is as follows:

Step 1: Specification of number of inputs and outputs. The numbers of inputs and outputs of the network and total number of data patterns are entered. These parameters are used to create a blank grid to hold the required data. In case of 2 inputs, and one output, and 4 as number of data, a grid with 4 rows and 3 columns is created.

Step 2: Entering all data. After a grid with appropriate number of cells is created, the data in cells may be entered one-by-one, or may be copied from MS Excel sheet and pasted into the grid created in the first step. In this step, the program may be rewritten to accept inputs from a database table or flat file.

Step 3: (a) Initializing Networks. All the networks are initialized with different number of hidden neurons in hidden layer. All the weights between the neurons of the input-to-hidden layers, hidden-to-hidden layers, and the last hidden-to-output layers are assigned randomly. (b) Drawing Networks. This step is optional and it is used to draw networks. (c) Starting Training. The application starts the training of all networks simultaneously. The object-oriented nature of the component makes the creation of many instances of ANN object very easy. Each network has different number of neurons in its hidden layer. Only one hidden layer has been employed for all networks since one hidden layer was found to be sufficient for all problems tried herein. If desired, it is very easy to set any number of hidden layers and neurons in each layer.

After the RMS error of any network decreases to a sufficient level, the network may be saved with all required parameters to be recalled later and employed for classification of new data. In the final step, the best network is chosen with the least RMS error.

This simple application illustrates the easiness of the component based development. It may be used to solve any classification problem using Artificial Neural Network component. It should also be noted that use of multiple neural network instances with different parameters in parallel makes training time much less than using one neural network. Any number of ANN component may be created in design or run-time with different number of hidden layers and processing neurons, and learning parameters. Learning parameters are either defined once at the beginning, or increased or decreased during the execution of program. The best network is then chosen based on the RMS error and saved to be used for recalling.

The component may be used easily for developing web based and database applications. Due to component based development, a number of components may be dropped onto a form with different learning parameters and structures. The best network architecture is then chosen based on RMS and saved to be used later. The component currently saves the results on a flat file. However, the results can be assigned to another standard component or stored in a database file by making small changes.

4.2 Using ANN Component to Solve Optical Character Recognition (OCR) Problem

The second small application was developed to solve Optical Character Recognition Problem by employing Haralick features which were successfully used in classifying textured images[10]. The screenshot of application is shown in Fig. 7. It uses database functionality. In the first step of program a table in a database is chosen. This table consists of inputs and outputs which are computed from digital images of letters. Four Haralick features namely energy, entropy, contrast, and correlation have been computed from co-occurrence matrices obtained from the images of letters in English alphabet. The images were firstly converted into gray levels. Then the total number of grey levels in the images was reduced to 16 gray levels by histogram equalization method to compute the co-occurrence matrices in shorter times since the computation time for each co-occurrence matrix depends on the maximum grey level in each image. The 4 Haralick features were used as inputs to the Neural Network. The total number of data was 653, the 279 of which have been used for training and the 374 of which have been used for testing respectively. The output for each letter was converted into sequential numbers since neural networks accept only numerical values. Since number of data was high the training took nearly one hour.

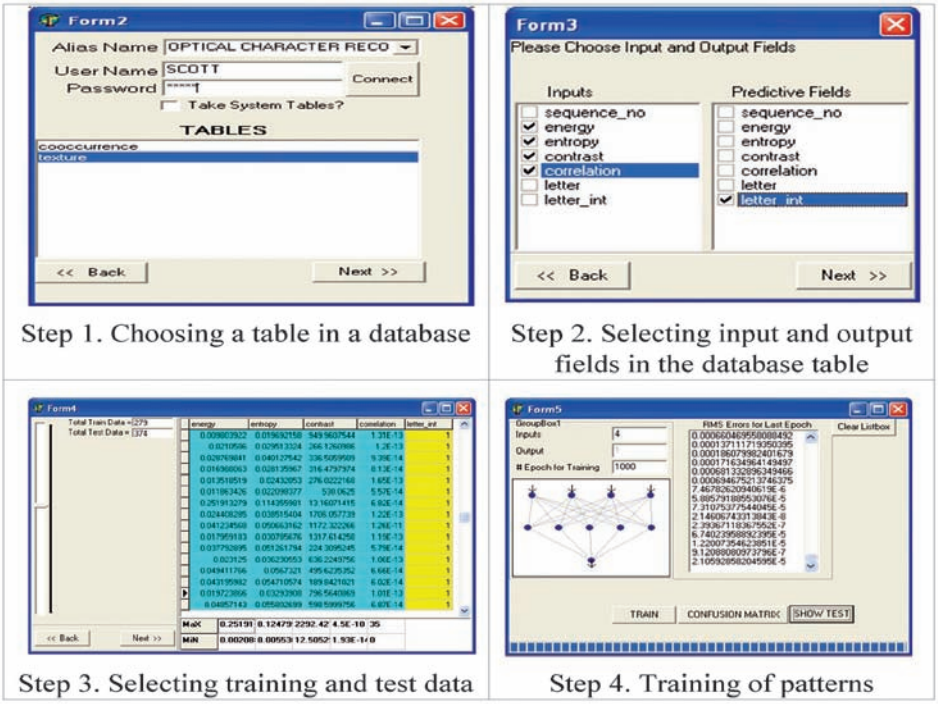


Fig. 7. A sample program for Optical Character Recognition (OCR) using ANN component.

Again the development time for application took only two days. However, this does not include the development time for the image processing including gray level conversion, and the grey level reduction using histogram equalization, the computation of co-occurrence matrices from images, and Haralick features from co-occurrence matrices.

The CLX library components for database development make it very easy to utilize database functionalities. Database management systems such as InterBase, Oracle, IBM/DB2 are supported natively and any other Database Management systems having Open Database Connectivity may be adopted within the same application without any change. This is also true for multiple operating systems, *i.e.* Windows and Linux. In the second step, all available table columns are shown to be selected as input and output fields.

The values for input fields are computed beforehand from the characters. After training and testing patterns are selected, the maximum and minimum values for each column are calculated to be used for the definition of Neural Network component. The classification performance after the tests reached 99%. This high performance may be due to one type of font used and clean images obtained from computer screenshots. The performance is expected to be lower in case of real images obtained through scanner and also the variation of the fonts with different sizes. In that case, some other features rather than texture features may be employed to increase the performance.

5. Conclusion

In this paper, an Artificial Neural Network software component is introduced to use intelligence based on Artificial Neural Network functionality in a cross-platform application development. This functionality can be embedded into any software development project with very little code. The paper introduces two case studies solved with this component in a very short time, the first as being a general classifier and the second as a simple OCR learner. The main goals of the study may be summarized as follows:

a) A General Framework for Description and Development of a Cross-Platform Artificial Neural Network component: The component described herein may be run under Windows and Linux platforms.

b) Easiness in embedding Artificial Neural Networks functionality in new applications by setting only some properties with minimal coding.

c) Use of Multiple networks simultaneously: One of the disadvantages of ANNs is the trial-error basis setting of the number of hidden neurons, the learning parameters etc. This is overcome by creating a number of instances of Neural Networks with different parameters and finding the best one with small-

est Rooted Mean Square (RMS) Error. The first case study shows such an example.

d) Run-time Creation of Neural Network Component: The instance of the component developed herein may be created not only in design-time by setting properties but also in run-time. This makes the component very useful to be embedded in web applications.

Acknowledgements

The authors thank the referees for their invaluable comments.

References

[1] **Pham, D.T.** and **Liu, X.,** *Neural Networks for Identification, Prediction and Control*, Springer-Verlag (1997).

[2] **Hornik, K., Stinchcombe, M.** and **White, H.,** Multilayer feedforward networks are universal approximators, *Neural Networks*, **2** (5): 359-366 (1989).

[3] **Jacobson, I., Booch, G.** and **Rumbaugh, J.,** The unified process, *IEEE Software*, **16** (3): 96-102 (1999).

[4] **Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F.** and **Jeremaes, P.,** *Object-Oriented Development: The Fusion Method*, Prentice Hall, Englewood Cliffs, NJ, Object-Oriented Series edition (1994).

[5] **Glykas, M., Wilhelmij, P.** and **Holden, T.,** Object orientation in enterprise modeling and information system design, *IEE Colloquium on Object Oriented Development*, **8** (1): 819 (1993).

[6] **Lin, J.M.,** Cross-Platform Software Reuse by Functional Integration Approach, *COMPSAC '97-21st International Computer Software and Applications Conference:* 402-408 (1997).

[7] **Aburas, H.M.** and **Çetiner, B.G.,** Component Based Intelligent Systems Development using Artificial Neural Networks, *International Twelfth Turkish Symposium on Artificial Intelligence and Neural Networks* (2003).

[8] **Lippman, R.P.,** An Introduction to computing with neural nets, *IEEE ASSP Msg.*, **4:** 4-22 (1987).

[9] **Budinsky, F.J., Finnie, M.A., Vlissides, J.M.** and **Yu, P.S.,** Automatic Code Generation from Design Patterns, *IBM Systems Journal*, **35** (2): 151-172 (1996).

[10] **Haralick, R.M., Shanmugam, K.** and **Dinstein, I.,** Textural Features for Image Classification, *IEEE Trans. Systems, Man and Cybernetics*, **3** (6): 610-621 (1973).

تطـوير مركب شبـكات عصبيـة اصطناعيـة متعـددة المنصـات لنظم ذكيـة

سيتنر. بي. جولتكن، و هاني محمد أبو راس
قسم الهندسة الصناعية، كلية الهندسة، جامعة الملك عبد العزيز
جـــدة – المملكة العربية السعودية

المستخلص. أصبح تطوير التطبيقات متعددة المنصات في السنوات الأخيرة من الأمور المهمة في مجال تكنولوجيا نظم المعلومات، ونظرا للقدرات الكامنة للبرامج متعددة المنصات في البيئات المختلفة فقد استحدثت وطورت طرق وأدوات للتخطيط والتصميم لها. فعلى سبيل المثال نجد أن لغة التخطيط الموحد تضم كل مراحل دورة تطوير التطبيق، كما استحدثت الشفرات متعددة المنصات باستخدام البرمجة الكائنية التوجه. ولقد تم كذلك تهيئة الشبكات العصبية بشكل كبير لاستخدامها في حل مجال واسع من المسائل وبشكل فاعل. ونظرا للمنافسة الشديدة، فقد أصبح ضروريا في الوقت الراهن تطوير الأنظمة في فترة زمنية قصيرة مما أدى إلى الحاجة الماسة إلى طريقة سريعة يتم من خلالها تبني الشبكات العصبية الاصطناعية في دورة تطوير برمجيات المنصات المتعددة وبالتالي فإن هذا التوجه يتطلب أدوات شبكات عصبية اصطناعية متعددة المنصات. لذلك فإننا نقدم في هذه الورقة العلمية مركبًا يناسب بيئة لغة التخطيط الموحد، ونناقش بعض خصائصه قبل توضيح فوائد المركب المقترح من خلال استعراض بعض الأمثلة.