RICBIS: New Zealand Repository for Intelligent Connectionist-Based Information Systems

Da Deng, Irena Koprinska, Nikola Kasabov
Dept. of Information Science, Univ. of Otago, Dunedin, New Zealand
E-mail: ddeng, ikoprinska, nkasabov@infoscience.otago.ac.nz

Abstract

RICBIS stands for Repository for Intelligent Connectionist-Based Information Systems. It includes generic (standard and novel) information processing techniques and application oriented systems accessible via a platform independent graphical user interface implemented in Java. In this paper we present an introduction to the RICBIS algorithms and modules, systematic implementation and interface features. Usage of the repository is also demonstrated with examples.

1 Introduction

The goal of the Repository for Intelligent Connectionist-Based Information Systems (RICBIS) is to incorporate various traditional and novel intelligent information processing techniques, that can be applied to key areas, such as speech and language communication, environmental process control, climate and inflow prediction, bioinformatics (e.g., sheep DNA analysis), image and video processing, financial analysis and prediction, and robot control.

In this paper we discuss the first release of RICBIS that includes modules and systems based on artificial neural networks, fuzzy logic inference, evolutionary programming, chaos analysis, statistical methods, rule-based systems, image, speech and signal processing techniques. These modules and systems are easily accessible by a platform independent Java interface.

The organisation of the paper is as follows. In the next section the basic RICBIS modules and the algorithms behind them are presented. The objects and data flow of the Java interface are discussed in section 3. The usage of the repository is demonstrated with two examples - clustering of the Iris data and prediction of the gas-furnace dynamic time series. Finally, we conclude our paper with a discussion on future directions for RICBIS development.

2 RICBIS Modules

RICBIS modules can be divided into three groups: (i) generic methods, (ii) application oriented methods and (iii) application oriented systems. The methods and systems are implemented in different languages (Java, C++ and Matlab) as they serve different needs.

2.1 Generic Methods

2.1.1 Data analysis

Two methods for dynamic time series analysis are included in the current version of RICBIS and are implemented in Matlab: calculation of the first Lyapunov exponent of a time data series [18], and calculation of the fractal dimension of the graph of a time series. Several clustering techniques are also implemented.

2.1.2 Neural networks

A number of classical and novel neural approaches have been developed and implemented. The well known and widely used *multilayer perceptron* (MLP) trained with backpropagation algorithm [16] is implemented in Java and C++. A version with modified training - learning with forgetting is available as a Matlab implementation.

Another important and popular model of neural networks, namely *self-organizing map* (SOM) [15], has been included in RICBIS and implemented in Java and C++. SOM defines a nonlinear mapping from a high-dimensional input data onto a two-dimensional lattice. It places a number of reference (codebook) vectors into an input data space to approximate to its data sets in an ordered fashion. Local-order relations are

defined between the reference vectors and the relative values of these vectors are made to depend on each other as if their neighboring values would lie along an "elastic surface". By means of the self-organizing algorithm, this surface becomes defined as a kind of nonlinear regression of the reference vectors through the data points.

Elman's partially recurrent neural network [2] is implemented in C++. Similar to the MLPs, the connections in Elman networks are mainly feed-forward but also include a set of carefully chosen feedback connections that let the network remember cues from the recent past. The input layer is divided into two parts: the true input units and the context units that hold a copy of the activations of the hidden units from the previous time step. As the feedback connections are fixed, backpropagation can be used for training of the feed-forward connections. The network is able to recognize sequences and also to produce short continuations of known sequences.

Fuzzy Neural Networks (FuNNs) [5][6] are novel neural algorithms that use a MLP network and a modified backpropagation algorithm. They consist of 5 layers: input, condition, rule, action and output. Neurons in the input layer represent input variables. The condition element layer performs fuzzification where triangular membership functions are used. Each node in the rule layer corresponds to a complex fuzzy rule. Neurons in the action layer represent fuzzy labels. The output layer performs a modified center of gravity defuzzification. Both Matlab and C++ implementations are available.

Evolving Fuzzy Neural Networks (EFuNNs) [7][8] are fuzzy neural networks that feature fast, one pass training, good generalisation, robustness against forgetting and the ability to explain learned relationships via the extraction of fuzzy rules. Similar to FuNNs, EFuNNs have a five-layer structure: input neurons, fuzzy input neurons, rule nodes, fuzzy output nodes and output nodes. However, the rule nodes in EFuNNs evolve through incremental, supervised and unsupervised learning according to the ECOS principles [7]. Various EFuNN versions have been implemented and successfully applied for prediction and classification. They are available as Matlab, C++ and Java implementations.

Another evolving module included in RICBIS is *Evolving Self-Organizing Map* (ESOM), an ECOS extension of the SOM. ESOM works using learning rules similar to SOM, but its network structure is evolved dynamically from input data [1]. ESOM is written in Java.

As the main drawback of neural networks is their inability to explain why a given decision was reached, deriving a symbolic description which closely mimics the behaviour of the network in a concise, accurate and comprehensible form is highly desirable. Algorithms for *rule extraction* from trained FuNNs and EFuNNs have been developed [6][10][13] and implemented in C++ and Matlab.

2.2 Application oriented methods

TouchUp [12] is an *image enhancement and feature extraction* software package implemented in C++.

Two modules for *speech analysis and processing* have been developed and implemented in Matlab. The first one calculates the so called *Mel-scale coefficients* that are frequency representation of the speech data. After Fourier analysis, the frequencies are broken into several 'bins'. The bin spacing is based on models of the ear, being linearly spaced to 1kHz, and logarithmically spaced thereafter. The purpose of the second module is *noise reduction*. It is based on adaptive noise cancellation [4]. Two microphones are used: the primary one obtains the noise-corrupted speech while the reference one receives only a correlated component of the noise present in the primary microphone. The noise in the reference microphone is processed by an adaptive filter in order to generate a replica of the noise component in the primary input.

A method for *adaptive filtering* is developed and implemented in Matlab as well. One of the most typical applications of adaptive filtering is the equalization of frequency dependent channel attenuation in data communications. It is applied to identify and track the channel characteristics since the frequency response of the most data channels varies slowly with time but the response is not known in advance.

2.3 Application oriented systems

The current version of RICBIS contains the following application oriented systems: HySpeech [11], English-Maori-English Word Translator [20], Insect Pest Identification System [19], Kiwi Fruit Classification System [14], System for Financial Risk Analysis and Prediction [9], and a robot control system. FuzzyCOPE 3 is a generic environment for building connectionist-based systems

and is also a part of RICBIS [17].

3 The RICBIS Java Interface

RICBIS Java Interface is a Java applet which can be accessed via WWW. With specially designed architecture, it aims to provide a powerful computing framework that is capable of easy integration of information processing modules, provides an easy-to-follow user interface, and allows for upgrade into a distributive agent-based architecture.

3.1 Objects

Being object-oriented, Java enables data and data manipulation methods to be encapsulated and communicate with each other in an effective way. Main objects in the RICBIS Java environment are data sets and entries, computing modules, a data parser, and user interface.

- Data sets and data entries

  Data sets are the major data objects to be processed in RICBIS. A *DataSet* class is a class to hold a self-contained data set, including raw data, data structure information (e.g., input and output numbers, class number, number of total entries etc.), and textual labelling information if available.

  A data entry can be retrieved from a DataSet as a *Signal* or *DataInstance* for carrying out online learning. A Signal has no output information and is used usually for unsupervised learning. A DataInstance inherits data structure from its parent DataSet and is used for online supervised training.

- Computing modules

  A computing module is a working process which operates on the current data set. RICBIS modules fall into three categories:

  1. Data manipulation modules, in which data sets go through some preprocessing procedures such as normalisation and standardisation, feature extraction algorithms such as PCA.
  2. Unsupervised modules, including clustering algorithms such as the SOM, K-Means, etc.
  3. Supervised modules used for classification or approximation purposes, e.g., EFuNN, MLP, etc.

  In order to provide a unified manipulation mechanism for various computing modules, a virtual computing module (VCM) is defined as an abstract class with learning and testing methods declared. Each computing module is implemented as an inheritance from VCM, defining its own learning and testing methods. Thus the main applet class can start, or stop any process of a computing module, by taking it as an instance of VCM. Process control in the Java interface is simplified in this way.

  On the other hand, similar computing modules, modules which handle clustering e.g., can be organised into a Java package where some abstract prototypes can also be defined. This allows for flexible implementation of computing modules with hierarchical diversity and overall unification.

- Data parser

  As data sets should be self-contained information entities, they should be represented by certain markup format and a parser is also needed to allow data information being transferred to and from data sources. A *DataSetParser* is a class to parse textual data accessible from a Java reader class. This means that data set can be read and parsed from local files, web URLs, or even from a text string. This adds flexibility of data access in the RICBIS environment. User can even paste data set text from clipboard and then make use of it.

  Data set parsing and generation in DataParser is hand-coded using a SGML markup format. This is supposed to be upgraded to be fully XML compatible by using interpreter and encoder of industrial standard as XML is expected to dominate data exchange on the Internet [3].

- User interface.

  RICBIS interface is developed by up-to-date Java graphical user interface techniques using JFC/Swing lightweight components, including frames, panels, menus, buttons, combo boxes, dialogs etc. A task driven approach is applied into the user interface so as to provide a guide for novice users to solve their problems by selecting adequate computing modules, using optimum options and parameter settings.

3.2 Data flow

In RICBIS Java environment, data are first collected either from local file systems or from a URL link on the Web.

A *DataFiler* is used to generate a DataSet object from the data resource. For data manipulation such as normalisation or dimension reduction by PCA for example, the DataSet object can be directly used. If it needs to go through a unsupervised learning procedure, data entries are extracted as signals. Otherwise for the case of supervised learning, they are extracted as DataInstance objects.

4 Usage examples

Here we give two usage examples on well-known benchmark data sets to illustrate how the environment can be used to solve different problems.

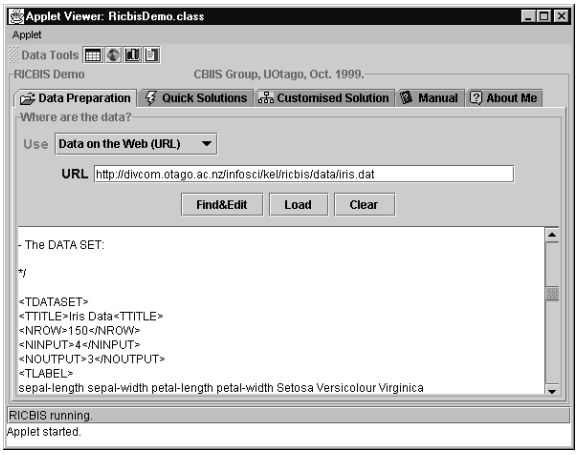

Fig. 1: Data preparation interface.

4.1 Example 1: Clustering of Iris data using RICBIS/SOM module

The procedure can be summarised as follows:

1. Click on the *Data Preparation* tab (see Fig.1). Select *Use data on the Web (URL)*. Type in the URL for the data file, click on *Find&Edit*. The data file should be loaded into the text area. If the data file is in wrong format, click into it and modify the text. Afterwards click on the *Load* button to load the data set into memory.

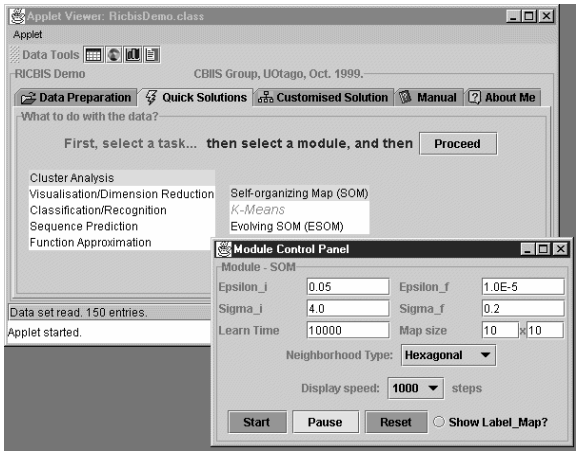

Fig. 2: The task/module panel and a control panel for SOM.

2. Click on the *Quick Solutions* tab. Select *Clustering* from the task list, and then after the module list is generated, select *SOM* from it. A control panel for the SOM will pop out (see Fig.2).

3. There are default settings available on the control panel. Modify the settings if needed. Click on the check box of *Show Label_Map?*. Then click on the *Start* button. Now the SOM module is started and a progress bar will be displayed.

4. Once the learning process is finished, the progress bar disappears and a beep sound can be heard. Meanwhile a labelled mapping of the data set is displayed on screen with quantisation error message.

4.2 Example 2: Applying EFuNN for gas-furnace dynamic time series prediction

1. Choose *Load Training File* menu item from the Train menu (see Fig.3). Enter the URL of the training file in the text field of the dialog window. If the format of the data file is not correct, an error message will appear in the status bar. Otherwise, the data parameters (inputs, outputs and entries) will be shown in the respective fields of the EFuNN's control panel. The file can be visualized by selecting the menu item *View Training File* from the menu *Train*.

2. Set the parameters of the EFuNN algorithm

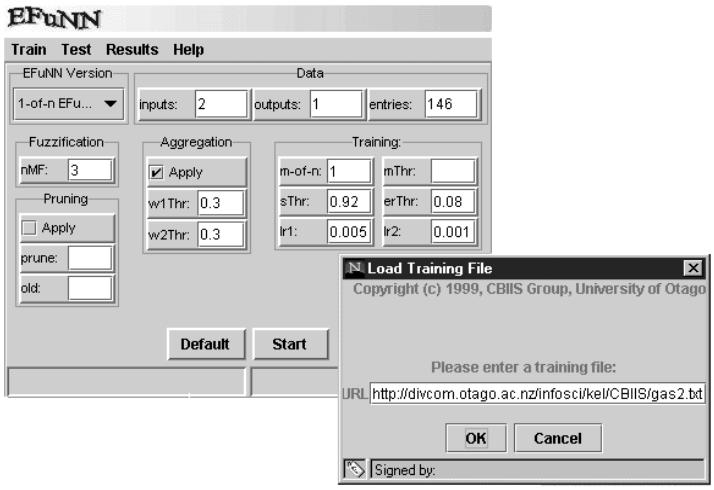

Fig. 3: The EFuNN Applet: main control panel and *Load Training File* dialog

or click the *Default* button to use their default values.

3. If you wish to test the EFuNN network off-line, load a training file using the first item of the menu *Test*. Again, the testing file can be viewed by choosing the second menu item.

4. To start training (or training and testing) EFuNN, click the *Start* button. If there is no testing file loaded, only training with one step ahead prediction will be performed, otherwise also off-line (batch) testing on the testing set will be done. When the training (or training and testing) is (are) finished, a message in the status bar will appear. Results can be viewed by selecting *View Results* item from the *Results* menu. In addition to the one step ahead prediction error for each of the training examples, the following errors for the whole training set will be presented: sum square error, root mean square error, mean error and standard deviation. If off-line testing was performed, similar errors for the testing set examples will be given as well.

5. Note that error messages will be generated in the following cases: 1) no correspondence between the training and testing data parameters; 2) mismatch between the parameters specified in the control panel and those read from the files; 3) missing or invalid parameter values. On-line help is available from the menu *Help*.

5 RICBIS2: The Future Direction

Integrating a number of intelligent information processing modules via a platform independent Java interface, the first release of RICBIS shows satisfactory performance in solving real world problems. In the future new methods and systems will be included, and RICBIS will migrate to an agent-based architecture which allows for intelligent information processing in a distributive environment. Meanwhile we will improve the RICBIS interface by using XML in storage and exchange of data and network information, and by introducing more intelligent features to assist users in the process of problem solving.

Further details about RICBIS can be found on the CBIIS site at URL
http://divcom.otago.ac.nz/infosci/kel/CBIIS/CBIIS.html.

Acknowledgements

RICBIS was developed within the framework of the Connectionist Based Intelligent Information Systems (CBIIS) research programme supported by the New Zealand Foundation for Research, Science and Technology, Contract UOO-808.

References

[1] D. Deng, N. Kasabov, Evolving self-organizing map and its application in gen-

eration of a world macroeconomic map, in the same volume.

[2] Elman J.L. (1990). Finding structure in time, Cognitive Science, 14, 179-211.

[3] C.F. Goldfarb, P. Prescod, *The XML Handbook*, Prentice Hall PTR, 1998.

[4] Iliev G., Kasabov N. (1999). Adaptive blind noise suppression in some speech processing applications, Proc. Intern. Conference on Neural Information Processing (ICONIP'99), Perth, Australia, in press.

[5] Kasabov, N. (1996) Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, MIT Press.

[6] Kasabov, N. (1997) Fuzzy rule extraction, reasoning and rule adaptation in fuzzy neural networks, Proceedings of the Intern. Conference on Neural Networks, ICNN'97. Houston, May 1997, IEEE Press, 102-107.

[7] Kasabov, N. (1998). The ECOS framework and the 'eco' training method for evolving connectionist systems, Journal of Advanced Computational Intelligence, vol.2, No.6, 1-8.

[8] Kasabov, N. (1999). Evolving connectionist and fuzzy connectionist systems: theory and applications for adaptive, on-line intelligent systems, Neuro-Fuzzy Techniques for Intelligent Information Processing, N. Kasabov and R. Kozma (eds.), Heidelberg Physica Verlag.

[9] Kasabov N., Erzegoveri, L. et. al. (2000). Hybrid Intelligent Decision Support Systems and Applications for Risk Analysis and Prediction of Evolving Economic Clusters in Europe, in N. Kasabov (ed), Future Directions for Intelligent System and Information Sciences, Physcia Verlag, Springer Verlag, in press.

[10] Kasabov, N., Kim J., Kozma, R. (1999). Knowledge-based neural networks: Rule extraction from fuzzy neural networks through structural learning with forgetting, Journal of Advanced Computational Intelligence, in press.

[11] Kasabov, N., Kozma, R., et.al. (1999). Hybrid connectionist-based methods and systems for speech data analysis and phoneme-based speech recognition. In: Neuro-Fuzzy Techniques for Intelligent Information Processing, N. Kasabov and R. Kozma, (eds.), Heidelberg Physica Verlag.

[12] Kasabov, N.K., Israel, S.A., Woodford, B.J. (1999). Hybrid Evolving Connectionist Systems for Image Classification, Journal of Advanced Computational Intelligence (in press).

[13] Kasabov, N. and Woodford B. (1999). Rule insertion and rule extraction from evolving fuzzy neural networks: algorithms and applications for building adaptive, intelligent expert systems, Proc. of the FUZZ-IEEE'99 International Conference on Fuzzy Systems, August, Seoul, Korea.

[14] Kim, J., Kasabov, N. et.al. (1998). Connectionist methods for classification of fruit populations based on visible-near infrared spectrophotometry data, Technical Report, Univ. of Otago.

[15] Kohonen T. (1982). Self-organizing formation of topologically correct feature maps, Biological Cybernetics, v. 43, 59-69.

[16] Rumelhart D.E., Hinton G.E., Williams R.J. (1986). Learning internal representations by error propagation, in Parallel Distributed Processing: Explorations in the Microstructure of Cognition, v.1., chapter 8, MIT Press.

[17] Watts, W., Woodford, B., Kasabov, N. (1999) FuzzyCOPE: A comprehensive environment for building intelligent systems - the past, the present and the future, in the same volume.

[18] Wolf, A., Swift, J.B. et.al. (1985) Determining Lyapunov exponents from a time series, *Physica*, 16D, 285-317.

[19] Woodford B.J. (1999). Analysing images of pest damage to apples using wavelets, Proceedings of The Third New Zealand Computer Science Research Students Conference. Yeates, S. (ed), 101-108.

[20] URL http://kel.otago.ac.nz/translator/.