FuzzyCOPE: A Software Environment for Building Intelligent Systems - The Past, The Present and the Future

Michael Watts, Brendon J. Woodford and Nik Kasabov
Department of Information Science
University of Otago
PO Box 56
Dunedin
New Zealand
Email: mike@kel.otago.ac.nz

Abstract

The paper discusses the motivation for, heritage, architecture and future development plans of the FuzzyCOPE software environment. FuzzyCOPE is a free software environment for teaching, research and intelligent system development.

1.Introduction

Artificial neural network (ANN) methods have long been known to offer powerful means of solving a wide variety of problems. The teaching of these methods to undergraduate students requires a low cost software toolbox that is comprehensive in terms of the number of models available, while still being easy to use. Teaching at the postgraduate level also requires a comprehensive system that is capable of being utilised for research projects, while rapid changes in the state of the art of ANN systems require a system that is easily expandable. Here, an attempt at fulfilling these requirements, the FuzzyCOPE family of software tools, is described. The capabilities of the software are described, as well as the architecture of the system and the reasoning behind the design decisions made. The paper is structured as follows: section 2 describes the development of the FuzzyCOPE family of software. Section 3 describes the architecture of the software, section 4 describes the user interface design, and section 5 discusses plans for future development. Conclusions are given in section 6.

2.The FuzzyCOPE Family of Software

Since the paradigm of hybrid connectionist rule based systems was established [4] there are now several software environments that implement this paradigm. The first generation of such environments (see for example COPE [5]) implemented in a logical way different types of ANN (such as multi-layer perceptrons, Kohonen self-organising maps [3], adaptive-resonance theory ANN [2]), to be combined with the CLIPS-based production systems. Here, an ANN could be called for training, or for recall of the action part of the production rules [4] [5]. The second generation of such environments included fuzzy rules and fuzzy neural networks. Such an environment was FuzzyCOPE [6]. This environment further developed the main principles of COPE [5] through a combination of Fuzzy-CLIPS (an extension of CLIPS) developed by the NRC in Canada in 1994, `http://ai.iit.nrc.ca/fuzzy/fuzzy.html` and fuzzy inference and fuzzy-neural network modules. FuzzyCOPE 1 was released in 1995, and contained some basic data manipulation tools, a MLP simulator, and a Kohonen SOM simulator. FuzzyCOPE 2 was released in 1996, and added a fuzzy neural network (FuNN) simulator [6, 8]. After the successful construction of a large C++ ANN code base in 1997 as part of the CBIS project [9], it was decided to re-implement FuzzyCOPE using this code. The results of this effort were released in 1998.

The current release version of FuzzyCOPE is version three. The modules in version three include:

- Multilayer Perceptrons
- Kohonen Self Organising Maps
- Fuzzy Neural Networks
- data manipulation and transformation functions (data shuffling, normalisation, denormalisation)

FuzzyCOPE 3 is currently being used in the teaching of three courses at the University of Otago, at second, third and fourth year levels. In these courses it is being used both as a teaching environment, for introducing students to the concepts and principles of intelligent systems, and as a basis for student projects.

The successor to version three is currently in development. Unlike the radical change from version two to version three, version four is intended as an evolutionary development of it's predescessor. New modules added to FuzzyCOPE 4 include:

- Elman Simple Recurrent Networks
- Four layer Fuzzy Neural Networks

- Evolving Fuzzy Neural Networks (EFuNNs) [7]
- additional data manipulation and transformation tools

FuzzyCOPE 3 may be obtained from
`http://kel.otago.ac.nz/software/FuzzyCOPE3/`

The FuzzyCOPE 4 website is at
`http://kel.otago.ac.nz/software/FuzzyCOPE4/`

3.Architecture of FuzzyCOPE

3.1.Requirements

The architecture of any piece of software is determined by the requirements of the system. In the case of FuzzyCOPE 3 and 4, the requirements are as follows:

- The computational engine must be functionally separate from the Graphical User Interface (GUI), to allow it to be used for other applications.
- The Application Programming Interface (API) to the engine must be available for the development of intelligent applications.
- The engine must support the existence of an arbitrary number of objects, of whatever type, in memory at any one time.
- The engine must maintain all objects in memory, with secondary storage access occurring only when loading or saving objects from or to file.
- The system should be as open as possible to future expansion and enhancement.
- For reasons of speed and portability, the computational engine should be written in ANSI compliant C++.

3.2.Architectural Philosophy

Given the system requirements above, the paradigm chosen for the architecture of FuzzyCOPE is that of a Client-Server system. Commands and data are passed between the client (for example, the user interface) and the server (the module that contains the computational processes) via formatted strings. Command and result strings are assembled and parsed by Application Programming Interface (API) libraries written in a variety of programming languages. The server itself maintains a registry of objects that are currently available in memory, where each object is identified by a unique string identifier (or alias). This paradigm was selected for the following reasons:

- Eliminates problems with C++ pointers. Manipulating pointers is one of the most error-prone features of the languages that support them, and pose many representation problems in those languages that do not. By assigning each object a unique alias, pointers need not be considered at the user level.
- Avoids problems with passing data in proprietary formats. Many programming products, especially on the Windows platform, use proprietary formats for representing data types such as floating point numbers. By passing all data and commands as strings, such proprietary impediments are obviated.
- API library flexibility. Any programming language that supports strings can be used to create an API library. Languages currently being used to develop API libraries include C++, Borland Delphi and MicroSoft Visual Basic.
- Simplifies use of the system. Only the API functions need be considered at the application level. The complexities of the command strings and the computational engine can be ignored.
- Readily lends itself to future expansion. The computational engine can be expanded into a network enabled server without fundamental changes to the code, while the command language can be further expanded into a specialised scripting language.

3.3.The Frigate Command Language

As mentioned above, communication between the client application and the computational engine is via formatted strings. The format of these strings and the commands they carry are referred to as the Frigate command language.

3.3.1.Philosophy of Frigate

The Frigate command language is designed around the following principles:

- Extensibility. The language must be easily extensible, to incorporate future functionality.
- Consistency. Commands that do similar tasks must have a similar structure.
- Comprehensible. Commands and responses must be unambiguous and easy to understand.
- Forgiving. The syntax must not be rigid, to ease interaction with the engine and development of API libraries.

3.3.2.Structure of Frigate

A Frigate command string consists of a single command, usually followed by one or more tag-argument pairs. A tag is a string meaningful to the interpreter, while the argument is the value associated with that tag. Arguments may be strings, quoted (multi-part) strings, numerics, booleans or arrays. Presented below are some examples of Frigate commands.

```
create type MLP alias IrisMLP inputs 4
hidden {10,15} outputs 3 bias true
```

In this example, the command is `create`. The type of object to create (the argument to the `type` tag) is a Multi-Layer Perceptron (MLP). The `alias` tag specifies the name to assign the MLP when it has been created. The architecture of the MLP is specified by the `inputs`, `outputs`, `hidden` and `bias` tags. Here the arguments to the `inputs` and `outputs` tags are 4 and 3, respectively. The argument to the `hidden` tag is an array, and specifies that the MLP is to have two hidden layers, one of 10 nodes and the other of 15 nodes. The final tag, `bias`, is a boolean specifying the presence or absence of a bias layer.

Note that the order in which the tag-argument pairs are presented is irrelevant. The only restrictions are that the command keyword must be the first item of the command string, and that the argument values must follow the correct tags.

To load an MLP from disk, the command is:

```
load type MLP alias IrisMLP filename
''iris.mlp''
```

Here the command `load` will load a MLP type object from the file `iris.mlp` and assign it to the alias `IrisMLP`

4.The User Interface of FuzzyCOPE 4

4.1.Requirements

It is desirable for the Graphical User Interface (GUI) of FuzzyCOPE 4 to satisfy these criteria:

- Be developed using a platform independent GUI development environment. This has the advantage of maximising the number of platforms and operating systems that can use FuzzyCOPE 4 whilst minimising the development time of the GUI.
- Is able to be extended to meet the future development of the FuzzyCOPE 4 engine.
- Uses a object orientated paradigm for the development of the GUI which mirrors that of the FuzzyCOPE 4 engine and API.
- Has comprehensive visualisation capabilities for both data and ANN

Figure 1 is a screenshot of the intended look of the FuzzyCOPE 4 GUI. The user interface paradigm is that of a workbench where the various objects that are stored in memory are represented graphically using different icons. Connections between icons reflect the associations between the data objects and the processing elements of the FuzzyCOPE 4 engine. By right-clicking on any of the icons, a set of functions that can be applied to an object can be selected.

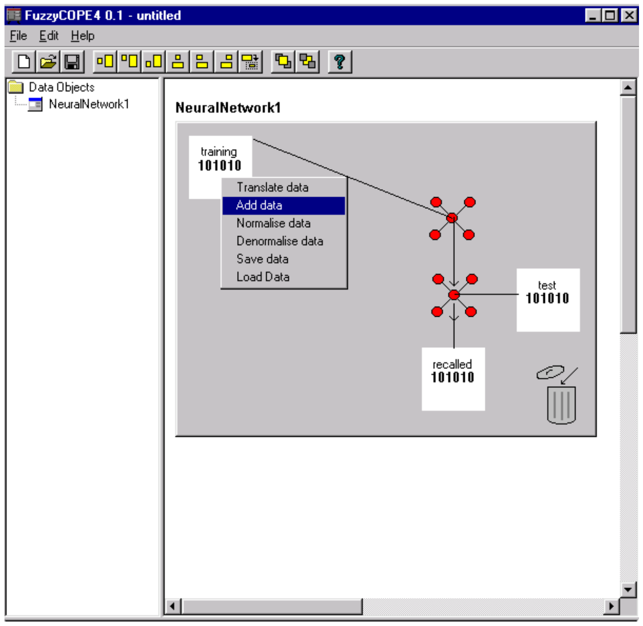

Figure 1: The intended look of the FuzzyCOPE 4 GUI

By using this method of visual development of the user's neural network design, they are abstracted away from the processing of the FuzzyCOPE 4 API and engine. We believe this to be a desirable feature of the FuzzyCOPE paradigm in terms of its use as a teaching tool to explain the concepts of neural network design and implementation.

5.Future Development of FuzzyCOPE: FuzzyCOPE 5

Further developments of the FuzzyCOPE family are also planned. Version five is intended to be a network based server, with the capability of both communicating with clients and reading and writing data across the WWW. Conceptual designs so far have focussed on a low level sockets based interface for the server. Since commands and results are passed between the client and server as strings, the relatively primitive interface sockets provide will not be a disadvantage. Basing the interface on sockets will also allow API libraries to be written in a variety of popular languages, including Java, Perl, PHP, Python and TCL. Implementation of a Java API library would allow the creation of platform independent applications that offload the heavy computational load to a fast server. Implementation of API libararies for such popular Common Gateway Interface (CGI) languages as Perl and PHP would allow the creation of intelligent websites that process data taken from web pages with connectionist and other intelligent models.

Other advantages of a sockets based network server include a possible use in distributed genetic algorithms: GA's that seek to optimise the structure, weights or training parameters of ANNs classically require large amounts of processing power. Previous methods of dealing with this problem have involved specially written servers [1]: the existence of a general purpose server and API libraries would assist in solving these problems.

The capability of communicating with search engines is also desirable, as this would allow the user to search the WWW for existing data sets and ANN solutions.

The user interface for version five in envisaged to be either a Java application, or a Java applet.

As more Java implementations of the various modules become available, a Java implementation of the server also becomes possible. This would allow for platform independence for the server, and opens the possibility of adapting the server into a mobile agent.

6.Conclusion

The paper has introduced the software environment FuzzyCOPE. It has described the motivation and history of the system, as well as the architecture of the current versions and the reasons behind the design decisions made. Plans for future development have also been discussed. The FuzzyCOPE environment is part of the New Zealand Repository for Intelligent Connectionist-Based Systems (RICBIS) that contains about 100 methods, modules, tools and systems readily available and easily accessible for both teaching and intelligent system development. The RICBIS website is at:
`http://divcom.otago.ac.nz/infosci/kel/-CBIIS.html`.

7.Acknowledgements

This research is partially supported by the Foundation for Research, Science and Technology, New Zealand through a grant UOO808 to the University of Otago. We would also like to acknowledge the work of Roy Ward on the CBIS code base upon which the FuzzyCOPE 3 and 4 engine was built, and of Sue Pearson in doing the initial interface work for FuzzyCOPE 3.

References

[1] Richard K. Belew, John McInerney, and Nicol N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *Artificial Life II, SFI Studies in the Sciences of Complexity, vol X*, pages 511–547. Addison-Wesley, 1991.

[2] G. Carpenter, S. Grossberg, J.H. Markuzon, and D.B. Rosen. FuzzyARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional amps. *IEEE Transactions of Neural Networks*, 3(5):698–713, 1991.

[3] T.M. Heskes and B. Kappen. On-line learning processes in artificial neural networks. In *Math. foundation of neural networks*. Elsevier, Amsterdam, 1993.

[4] N. Kasabov. Hybrid connectionist rule based systems. In *Artificial Intelligence IV Methodology, Systems, Applications*, pages 227–235. North-Holland, Amsterdam, 1990.

[5] N. Kasabov. COPE-a hybrid connectionist production system environment. In *Proceedings of the Third Australian Conference on Neural Networks (ACNN'92)*, pages 135 – 138. Sydney University Electrical Engineering, 1992.

[6] N. Kasabov. *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*. MIT Press, 1996.

[7] Kasabov N. Evolving fuzzy neural networks - algorithms, applications and biological motivation. In Takeshi Yamakawa and Gen Matsumoto, editors, *Methodologies for the Conception, Design and Application of Soft Computing*, pages 271–274. World Scientific Publishing Co, 1998.

[8] Kasabov N., Kim J., Watts M., and Gray A. FuNN/2 - a fuzzy neural network architecture for adaptive learning and knowledge acquisition in multi-modular distributed environments. *Information Sciences - Applications*, 1997.

[9] R. Ward, M. Purvis, R. Raykov, F. Zhang, and M. Watts. An architecture for distributed connectionist computation. In *Progress in Connectionist-Based Information Systems, Proceedings of the ICONIP / ANZIIS / ANNES '97, Dunedin, 24-28 November 1997*, 1997.