CompoNet: Programmatically Embedding Neural Networks into AI Applications as Software Components

Uzair Ahmad*, Uzma Nasir†, Sungyoung Lee*, Young-Koo Lee*

Abstract

The provision of embedding neural networks into software applications can enable variety of Artificial Intelligence systems for individual users as well as organizations. Previously, software implementation of neural networks remained limited to only simulations or application specific solutions. Tightly coupled solutions end up in monolithic systems and non reusable programming efforts. We adapt component based software engineering approach to effortlessly integrate neural network models into AI systems in an application independent way. As proof of concept, this paper presents componentization of three famous neural network models i) Multi Layer Perceptron ii) Learning Vector Quantization and iii) Adaptive Resonance Theory family of networks.

1. Introduction

Artificial Neural Networks are biologically inspired clusters of highly connected parallel processing nodes. Their remarkable ability to tolerate noise and adapt to unseen situations, wide scale application of neural networks has been carried out in many areas such as manufacturing [1], intelligent control systems [2], power engineering [3], pattern recognition [4] [5], speech recognition [6] and Ubiquitous Computing Systems [29] [30] to name a few. Mostly, the practical implementation considerations for neural network based AI systems are studied from hardware point of view [18]. However, several software applications demand incorporation of neural networks as well. This requirement holds for building hybrid intelligent systems based on combination of neural networks and knowledge-based system [9]. Moreover, AI researchers often need to verify and test their application of neural networks on novel problems in real life but software environment. Advanced software environments for neural network development and training are limited only to simulation of neural networks in a restricted simulation environment e.g. MATLAB Neural Network Toolbox [17]. Therefore, In order to train a model and verify the results, AI researchers have to bring training and test data from real life environment into a simulation environment. After the neural network model is optimized and trained, the integration of this model into real-life environment remains a secondary task and left to domain specific software programs which results in non-reusable effort.

Main premise of this paper states that the integration and deployment of neural networks into production environments in an application independent way can be greatly facilitated by employing component based software engineering approach. The CompoNet (Componentized-neural-Networks) facilitates development of neural network based software systems by designing and implementing neural networks as software components. These components can be reused effortlessly across different application and, thus, shipping of trained models from simulation to production environment is possible with minimal programming effort. Essentially CompoNet functions as binary unit of composition into larger systems. It can encapsulate arbitrary size and structure of underlying neural network models and exposes programmatic interfaces in order to i) embed a neural network into external software application and ii) persist neural network on permanent storage and restore it later. Section 3 explains the integration and deployment perspective of CompoNet into software systems, particularly an AI system middleware. For the sake of realization of the concept and in order to provide variety of choices to developer community, we componentized three famous neural network models.

i) Multi Layer Perceptron
ii) Learning Vector Quantization
iii) Adaptive Resonance Theory networks

Each of these models and its corresponding CompoNet implementation are described in section 4 by Unified Modeling Language (UML) and sample code snippets. Section 5 presents a feature wise comparison of

* Associated with UC Lab, Kyung Hee University, Korea; Primary Author email uzair@ieee.org
† Associated with Air University, Pakistan

CompoNet and similar approaches for object oriented modeling of neural networks.

2. Related Work

Several researchers have proposed object-oriented modeling of neural networks to achieve expressiveness, reusability and efficiency. Albuquerque and El-Emam have outlined fundamental abstractions and elaborated strong structural relationship between object oriented modeling and neural networks in [7] and [8]. Daikui developed high level procedural language, object oriented neural network language, in order to describe large scale neural networks simulation systems [9]. Leber presented design and implementation of an interactive object-oriented neural network simulator for recognition of acoustic signals [10]. Cedric *et al* presented an object oriented simulation kernel for feedforward neural networks [19].

The practical utility of previous work is limited to build neural network simulations using novel and useful programming constructs. Instead shipping the neural network model from simulation to production environment and integrating it neural network into larger AI software systems is equally important. Main contribution of this paper is to extend state of the art research on object-oriented modeling of neural networks, based on component technology, in such a way that neural networks could be composed into larger artificial intelligence systems as software components.

3. CompoNet Framework

Primary objective of the CompoNet framework is to ship neural network models from simulation environments to production environments by applying component-based approach on development of neural network based AI systems. The distinguishing features of CompoNet can be viewed as essential services offered by the componentization of neural network models to AI system developers.

Parameterization: It refers to ability of software to change its internal structure according to certain parameters. In the context of neural networks parameterization enables a CompoNet to encapsulate arbitrary size and structure of underlying neural network models by means of topology parameters (e.g. size, structure and connectivity). CompoNet allows software developers to specify these *parameters* through programmatic interfaces.

Reusability: Black-box reusability refers to the ability of software component to provide cohesive, loosely coupled module that could be reused without accessing its internal implementation. This capability makes software components as binary units of independent production, acquisition, and deployment that interact to form a functioning system [13]. Modern software systems are in fact flexible compositions of “software components” that work together in a well defined component framework [11]. CompoNet achieves reusability by encapsulating the internal structure, functionality, configuration and execution of underlying neural network models into pluggable software modules.

Plug-ability: is related to the ease of replacement that is supported by the software modules. In real life, non-simulation applications; the dynamics of execution environment and usage context impose the need to replace certain modules with others. This replacement can be done either offline or at runtime during the execution. CompoNet framework allows both offline as well as runtime plug-ability of underlying neural network models.

Persistence: Refers to the ability of components to persist their state on permanent storage and reload it at later time when required. The state of a neural network is comprised by network topology, synaptic connections and their weights and preprocessing/post-processing parameters. CompoNet allows persistence functionality by means of input output interfaces.

3.1 Deployment View

Deployment view provides the architectural details regarding the placement of different entities in a neural network based intelligent system by using a layered model. Figure 1 shows different layers of an AI software system, deployment of CompoNet and interaction of different layers with each other.

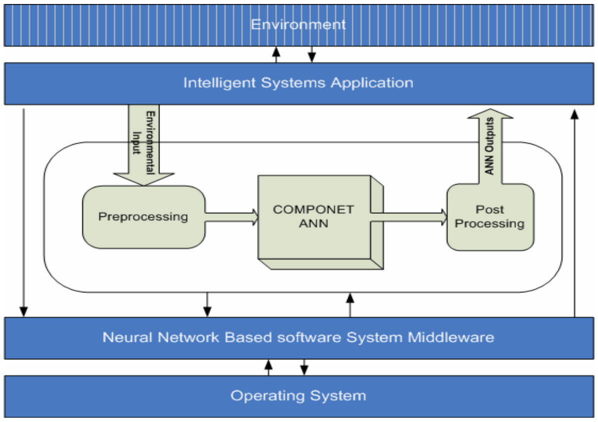

Figure 1: Deployment view of CompoNet in AI system

End user applications reside at the intelligent systems application layer and collect data from external environment and users. This data is passed down to the neural network based decision engine. Before inputs are fed into neural network an optional preprocessing step is carried out in order to normalize, filter and homogenize data for the network. The 'CompoNet ANN' depicts an embedded neural network in the form of a software component. This component encapsulates the execution of a whole neural network in an application independent way. Outputs of neural network model go through post-processing step for de-normalization and semantic mapping on system's knowledge base.

3.2 Execution View

Execution view explains operational aspects of CompoNet in the context of an AI software system. In order to model neural networks as software components, we identified a common set of higher level abstractions. These abstractions provide reusable structures for developing CompoNet across different neural network models. Two broad generic elements present in all neural models: static elements and dynamic elements. Static elements constitute topology and arrangement of synaptic connections and their respective weights. Static elements are responsible for representation and accessibility of neural network knowledge at execution time. Dynamic elements comprise mathematical formulation of how network learns the problem and maps the features to given classes or clusters corresponding features. Each of these elements has three modes of operation from execution perspective. All these elements and their respective functional descriptions are presented in Table 1.

Table 1. Modeling Neural Network Elements for CompoNet

	Operational Modes	Elements		
		Static	Dynamic	
CompoNet	Training	Topology parameters*	Synaptic Weights	Configuration
			Thresh hold values	
			Learning Functions	
			Input/Error Propagation Method	
	State Persistence	Neural Network Knowledge Representation Method	Load Configuration from Persistent storage	
			Save Configuration to persistent storage	
	Execution	Neural Network Configuration	Propagation Method	

Figure 2 graphically shows three operational modes of a neural network software component. Particularly it explains deployment of CompoNet in Middleware system that provides context-aware services to users through environment controller. System observes context parameters by means of environmental sensors. The sensor measurements contain discriminatory information about the surrounding context such as mobility and location of users. Once deployed, a component is available either for training, execution or persistence for later use. In simulation mode the CompoNet learns the patterns in training sensory measurements and how to map them onto specific outputs. In production mode, real time sensor inputs are provided to the CompoNet and based upon its outputs different context-aware services are realized using environment controllers and actuators. The dotted line shows the persistence capability of a neural network which allows storing both the static as well as dynamic information of neural network.

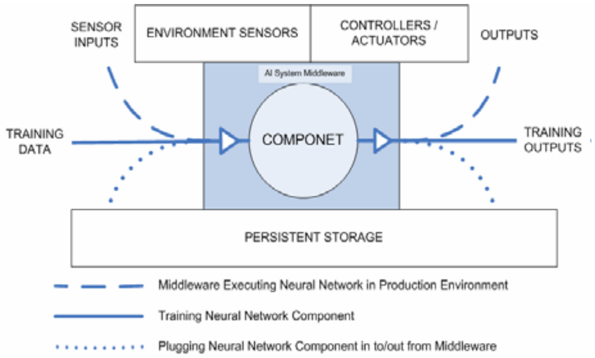
SENSOR INPUTS
ENVIRONMENT SENSORS
CONTROLLERS / ACTUATORS
OUTPUTS
AI System Middleware
TRAINING DATA
COMPONET
TRAINING OUTPUTS
PERSISTENT STORAGE
Middleware Executing Neural Network in Production Environment
Training Neural Network Component
Plugging Neural Network Component in to/out from Middleware

Figure 2: Operation modes of a CompoNet

Our current implementations of three neural networks use specific descriptive formats, so called *netconfig* file. It is a simple comma separated file which provides a standard mechanism to persist, parse and load/reload neural networks to and from stable storage at runtime. Moreover, this provision allows downloading and invoking a neural network on a network using regular data networks. Table 2 presents semantic structure of an example *netconfig* file.

Table 2. Netconfig File for CompoNet

Element	Description	Example
Structure	Topological information of network	2,0,4,2,2,2,1,2;
Synaptic Weights	Weights of every connection starting from 1st Neuron of 1st layer onwards	1,3.5,-3.9,-0.4,3.4,0.05,1.4,-0.9,3.1,-0.8,2.6,-2.9,-5.5,2.8,-2.1,0.1,7.3,7.9;
Biases	Bias values of each neuron	-1.3,2,0.1,-4.6,0.6,2.5,-7.2;
Preprocessing	Normalization values	9,11,9,11;
Post Processing	De-normalization values	0,1;

Example column of Table 2 presents actual values of corresponding elements of a multi layer perception neural network trained for solving XOR problem. As specified in the example column of structure row the topology of this network comprises 2 hidden layers with 4 neurons at the first and 2 neurons at the second hidden layer. The input and output layers contain 2 and 1 neuron respectively. The activation functions at the hidden neurons are specified as tan-sigmoid, which is represented as 2, and output neuron is log-sigmoid, represented as 1.

4. CompoNet Implementation

Apart from the high level common abstractions detailed in previous section, CompoNet framework offers object oriented modeling and componentized implementation of three neural network models. These components can be utilized using standard application programming interfaces (API). Here we present the OO design of CompoNet and example code to illustrate the programmatic usage of API. Current implementation of CompoNet is targeted for Microsoft .Net framework using C# language. We share our source code, binaries and demo applications as open source project [12]. CompoNet modeling and implementation of all three operational modes of underlying neural network is presented in following sections.

4.1 Multilayer Perceptron

Multilayer Perceptron (MLP) network is one of most frequently used neural network models. It uses back propagation training algorithm to learn arbitrarily nonlinear class boundaries in a feature space. Typically, these networks are employed for supervised learning of pattern recognition problems. Figure 3 shows topological structure of Multi Layer Perceptron (MLP) network. MLP networks are composed have three layers i) One input layer ii) Arbitrary number of hidden layers iii) One out put Layer.

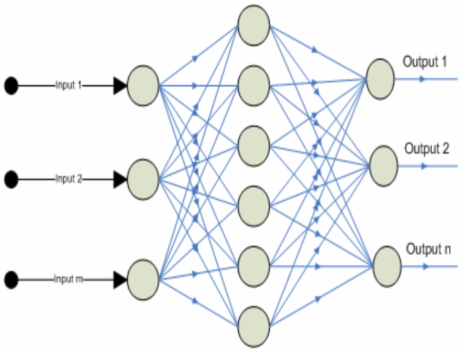

Figure 3: Topological structure of Multi Layer Perceptron

In order to represent arbitrary size and structure of MLP network in a software component, UML class structure is shown in Figure 4. Each Layer of an MLP is modeled as Layer class which can contain an arbitrary number of Neurons. Neuron class provides high level abstraction which specifies the operations that a neuron should perform. The InputNeuron and OutputNeuron are specialized neurons which provide extra functionality other than the basic neuron provides. Input layer may contain m neurons depending on dimensions of feature space. All hidden layers can have arbitrary number of neurons in each layer. Interconnectivity of different layers and their respective neurons is modeled by Synapse class. All input neurons are connected to every neuron of first hidden layer. Similarly, every hidden layer neurons have synaptic connections to the next layer neurons. The NeuralReader class provides the loading and persistence operations to the programmers. It implements parsing logic of a *netConfig* file which allows a loading a neural network from local storage or even from network. The Feed Forward execution mechanism is implemented in NeuralNetExe class which receives the flow of inputs and executes the underlying neural network. Final output an MLP is provided to the system by means of API.

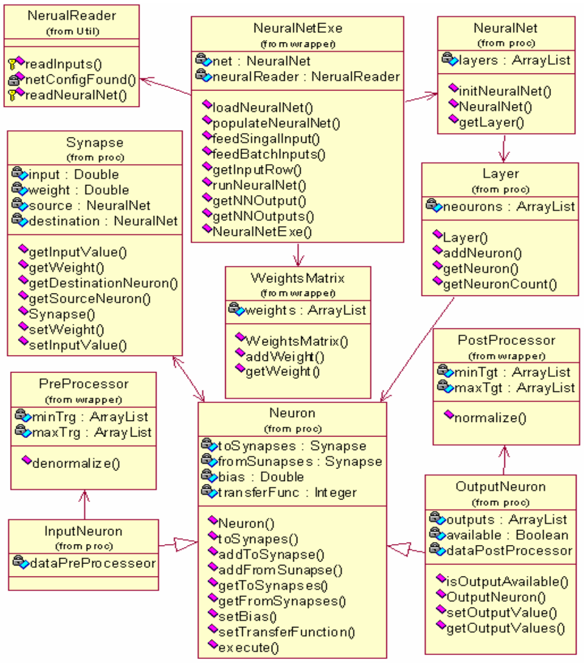

Figure 4: Class structure of MLP.NET CompoNet

Listing 1 presents the example code to create, load and execute a multilayer perceptron neural network programmatically. Please notice that only C# language code snippets are presented in this paper. Nevertheless the application developers can choose their own implementation language within .Net framework. These components can easily be implemented in other platforms using Java by virtue of object oriented models presented here. Detailed object interaction models are not presented here because of limited space, however interested readers can freely access them as open source [12].

Listing 1: Example code for embedding MLP into System

```
// Creating an MLP Neural Network model
NeuralNetExe exe= new
NeuralNetExe();
// Loading a Neural Network model from
Persisten Storage
try
{
    exe.loadNeuralNet(netconfigFilePa
th);
}
catch (Exception er)
{
    MessageBox.Show("This is not a
valid .net file)"
}
// Feeding Input Vector to Neural Network
exe.feedInput(input);
// Executing Neural Network to classify
the input
exe.runNeuralNet();
// Getting Neural Network classification
decision
exe.getNNOutput();
```

The composing system feeds input vectors using feedInput(inputVector) function and receives the output of MLP using getOutput() function of the NeuralNetExe class. The internal operations involve preprocessing the input vector by means of normalize() operation and then feeding the individual components of input vector to the appropriate neurons layer by layer. MLP.NET encapsulates all of these complexities and provides neural network outputs to the external system.

4.2 Learning Vector Quantization

Learning Vector Quantization (LVQ) networks are widely employed for supervised classification tasks. Fundamental work on LVQ is formulated and described by Kohonen [14]. Figure 6 shows general topological structure of Learning Vector Quantization networks. Design of LVQ networks specifies three layers of neurons. Input layer contains as many neurons as components of input vector.

UML class diagram of LVQ.NET component is shown in Figure 7. InputPattern class represents this layer. Neurons of hidden layer, also called competitive layer, determine distance of input vector from codebook vectors based on any of several nearest neighbor rules. HiddenLayer class performs this function.

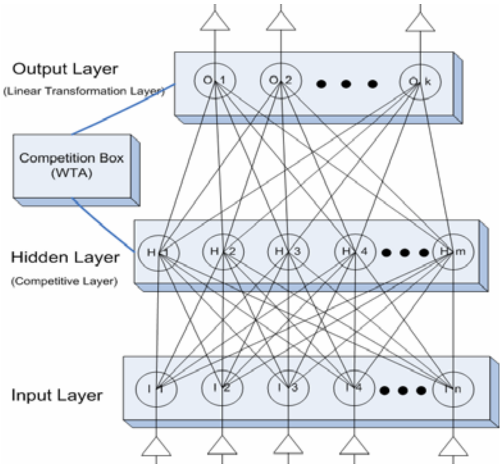

Figure 6: Topological structure of LVQ Network

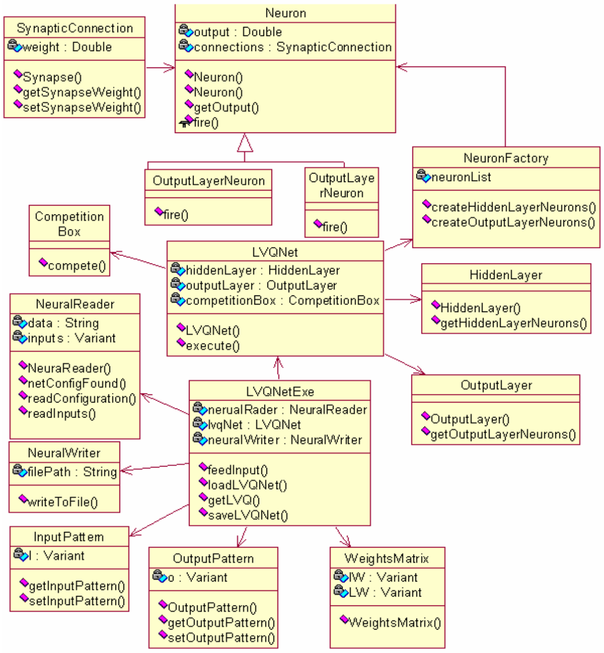

Figure 7: Class structure of LVQ.NET CompoNet

Once the input vector is classified at competitive layer, the second layer, also called linear layer, transforms the output of competitive layer into target classification vectors defined by developers. This transformation involves competition among competitive layer neurons in order to select the 'winner' neuron; closest matching neuron which best represents the class of current input.

Listing 2: Example code for embedding LVQ into System

```
// Creating an LVQ Neural Network
model
LVQ.NET.LVQExe exe = new LVQExe();
// Loading a LVQ model from
Persistent Storage
exe.loadLVQNet(netconfigFilePath);
// Creating an Input Vector
InputPattern i = new
InputPattern(input);
// Executing the LVQ model
OutputPattern o = exe.feedInput(i);
```

Figure 8 diagram shows sequence of interaction LVQ.NET objects in order to classify input vector given by composing systems.

4.3 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) based neural networks were developed by Grossberg and Carpenter [15], [16]. These networks are widely used for supervised learning, pattern classification, and unsupervised learning, clustering, problems especially where online and incremental learning is required. Several variants of both types of ART neural networks exist but here we present componentization of only the basic ones.

4.3.1 ART-1 and FuzzyART

ART-1 network is widely used for unsupervised clustering of distinct classes in a feature space. Both ART-1 and FuzzyART have same topological structure, shown in Figure 9, and learning mechanism. The only difference is that FuzzyART network can handle analog input vectors but ART-1 can accept only binary features. In order to be able to recognize analog patterns, the FuzzyART employs fuzzy logic.

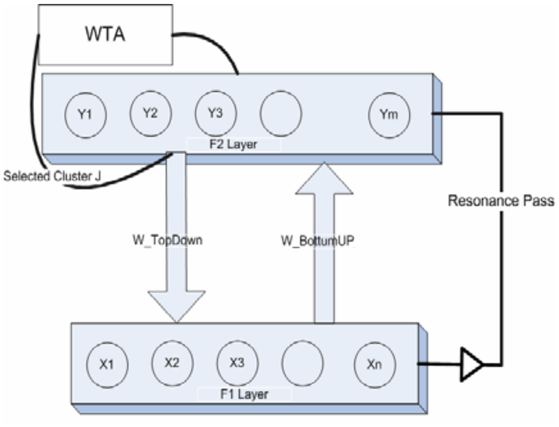

Figure 9: Topological structure of ART and FuzzyART

ART systems can utilize an optional preprocessing step called complement coding. It prevents category proliferation problem in FuzzyART networks as described in [16]. This preprocessing technique is internalized by CompoNet and can be invoked by using appropriate parameters. F1 Layer contains *n* input neurons each of which are connected to *m* category neurons in F2 Layer. Bottom up synaptic connections contain analog valued weights are used to perform distance measurement among different categories stored in F2 Layer. Each of F2 Layer neurons is connected to input neurons in F1 Layer. Top down synaptic perform selection of candidate category that current input is assigned to. After a category neuron is selected, vigilance mechanism checks if current input passes resonance threshold or not. If resonance pass is cleared, weights of both Bottom UP and Top Down connection are updated. Otherwise a new category node is created in order to represent current input. The UML class diagram of ART-1 and FuzzyART networks is presented in Figure 10.

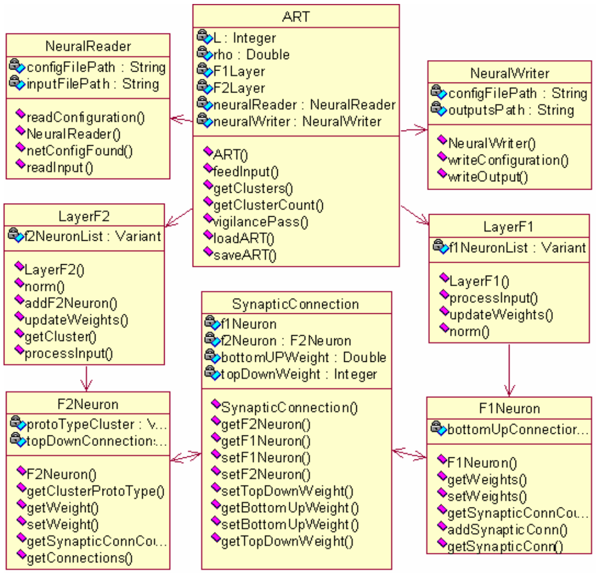

Figure 10: Class structure of ART.NET CompoNet

Building blocks of ART.NET CompoNet are modeled as LayerF2 and LayerF1 classes. Each of these classes can have associations with multiple F1Neuron and F2neuron, forming a 1 to many relationship. Dynamic elements of ART networks are encapsulated in ART class. Example code for embedding a FuzzyART neural network in a software application is shown in Listing 3.

Listing 3: Example code for embedding FuzzyART

```
// Creating a FuzzyART Neural Network
model
FuzzyART.NET.FuzzyART  artexe =
new
FuzzyART.NET.FuzzyART(3,.8700001,1,.5,tru
e);

// Preprocessing Input vector
trg = normalize(trg);

// Feeding input vector to FuzzyArt
object[,] o = artexe.feedInput(trg);
```

Figure 11 shows a sequence of operations that takes place inside ART.NET CompoNet in order to assign a given input vector to most suitable existing clusters.

4.3.2 ARTMAP and Fuzzy ARTMAP

ARTMAP or predictive ART [25] extends basic ART model in order to serve the purpose of pattern classification of binary patterns. Similar to ART, ARTMAP has its counterpart Fuzzy ARTMAP [26] model for analog valued inputs and Fuzzy ARTMAP has same complement coding option as FuzzyART in order to control category proliferation. Simple ARTMAP introduces a category layer on top of basic ART model. F2 Layer neurons are connected to each category neuron in category layer as shown in Figure 12.

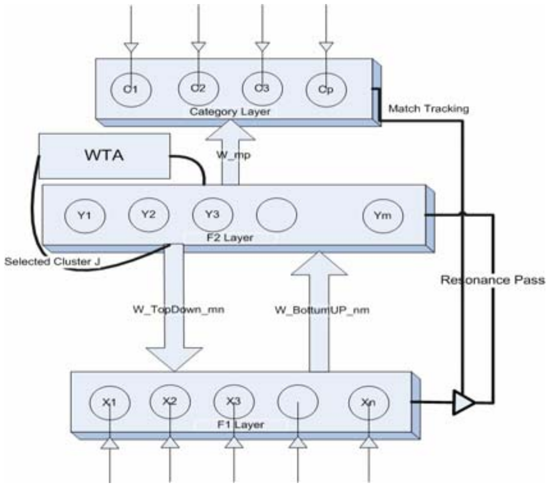

Figure 12: Topological structure of ARTMAP and FuzzyARTMAP

Like other pattern classification methods, ARTMAP has two modes of operation

i) Learning new pattern-class pairs

ii) Recalling best matching pattern stored in ARTMAP for given input pattern.

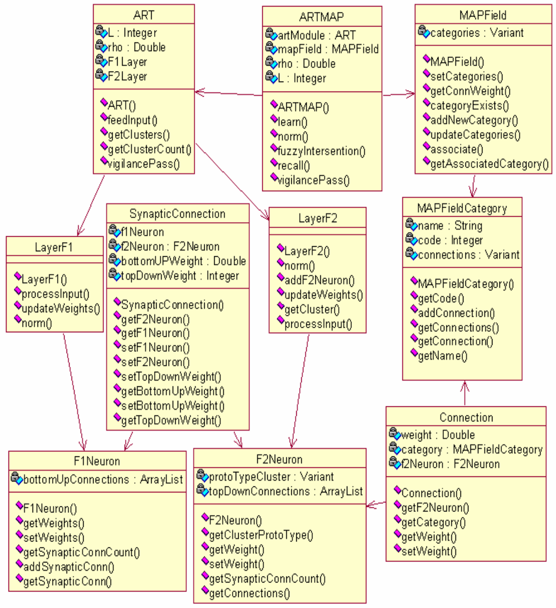

Figure 13: Class structure of ARTMAP.NET CompoNet

Figure 13 shows CompoNet class diagram model of ARTMAP networks. Since ArtMAP network can be built on ART-1, as can be seen from Figure 11, CompoNet reuses ART.NET internally to implement ARTMAP networks. Figure 12 class diagram has three extra classes other than those already present in ART.NET, Rest of the elements are being reused inside CompoNet. Although FuzzyARTMAP neural network is not presented here due to space limitation, it has similar structure but different dynamics for learning analog pattern-class pairs. Listing 4 shows example program snippet to embed a FuzzyARTMAP neural network into an online learning application.

Listing 4: Example code for embedding FuzzyART Map

```
FuzzyARTMAP.NET.FuzzyARTMAP artMAPExe;
artMAPExe                =              new
FuzzyARTMAP.NET.FuzzyARTMAP(
/* inputsize */ 4,
/* rho */ .8,
/* rhoInc */ .8,
/* aplha */ .1,
/* beta */ .5,
/* Complement Coding*/ true,
/* Dalay Update*/ false);

trg = normalize(trg);
int output = artMAPExe.learn(trg, tgt);
```

5. Comparative View

The object oriented modeling and programming of neural networks is not a new idea itself but the scope of previous approaches is limited to only finding the useful abstractions to facilitate the object-oriented artificial neural network programming or building object oriented ANN simulation engines. CompoNet framework leverages the strengths of component technology and enables neural network models to be trained in simulation environment as well as to be embedded into real life production environments as software components. Table 3 summarizes a comparative perspective of CompoNet and related works with respect to the features that each approach offers to the developers. MATLAB provides similar features to CompoNet but it has one major disadvantages. The target machine should have MATLAB installation or MATLAB component runtime environment in order to achieve plug-ability in production environments. On the other hand CompoNet approach is lightweight enough to be used even on resource constrained hand held devices such a PDA.

Table 3. Feature comparison of different approaches

	Param*	Sim Env**	Prod Env+	Plug ability	Persist ability	Code Reuse
CompoNet	√	√	√	√	√	√
MATLAB[17]	√	√	√	√	√	√
OOP [7]	√	√	x	x	x	√
NSK [19]	√	√	x	x	√	√
OONL [9]	√	√	x	x	x	x

* Parameterization ** Simulation Environment + Production Environment

6. Conclusions and Future Work

In this paper we presented novel application of component technology on development of neural networks based AI systems. Until now software implementations of neural models are restricted to simulation environments or weaved into target application logic. Unlike other neural network training and development methods, CompoNet neural network models are designed as software components. This enables AI system developers to embed neural networks in target software platform as plug in. This approach provides: i) flexible development of neural networks based or hybrid intelligent systems ii) Easy learning and relearning of neural network during exploitation of system iii) Flexibility in replacement of neural network by other one during exploitation of system iv) Relative independence between implementation languages of neural network and software connecting with it. Our current implementation is targeted for Microsoft's .Net platform. In future, we plan to provide CompoNet solutions for J2EE component framework. Currently our neural network components store their state in flat files. We also plan to incorporate structural strengths and rich express-ability of XML into CompoNet persistence mechanisms.

Acknowledgements

This work is financially supported by the Ministry of Education and Human Resources Development (MOE), the Ministry of Commerce, Industry and Energy (MOCIE) and the Ministry of Labor (MOLAB) through the fostering project of the Lab of Excellency.

7. References

[1] Huang, S.H. Hong-Chao Zhang, "Artificial neural networks in manufacturing: concepts, applications, and perspectives" IEEE Transactions on Components, Packaging, and Manufacturing Technology

[2] Vitthal, R.; Durgaprasada Rao, C.; "Process control via artificial neural networks and learning automata" Industrial Automation and Control, 1995 (I A & C'95), IEEE/IAS International Conference on (Cat. No.95TH8005) 5-7 Jan. 1995 Page(s):329 – 334

[3] Bretas, A.S.; Phadke, A.G.;, **"**Artificial neural networks in power system restoration"**,** Power Delivery, IEEE Transactions on Volume 18, Issue 4, Oct. 2003 Page(s):1181 – 1186

[4] Javad Haddadnia, KarimFaez, Majid Ahmadi. N-Feature Neural Network Human Face Recognition.

[6] Polur, P.D.;*et al* "Isolated speech recognition using artificial neural networks", Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE, Volume 2, pp.1731 - 1734 vol.2

[7]Albuquerque et al, "The adherence of the object oriented programming paradigm on the simulation of artificial neural networks" IEEE World Congress on Computational Intelligence., 1994 Volume 6, Issue , 27 Jun- 2 Jul 1994 Page(s):3900 - 3904 vol.6

[8] El Emam, K. et al,"Object oriented neural networks", International Conference on Control 1991. Control, 25-28 Mar 1991 pp: 1007-1010

[9] Daikui Shouren Hu, "An object-oriented neural network language" IEEE International Joint Conference on Neural Networks, 18-21 Nov 1991, pp: 1606-1611 vol.2

[10] Leber, J.F. Moschytz, G.S. "An interactive object-oriented neural network simulator applied to the recognition of acoustical signals", IEEE International Symposium on Circuits and Systems, 10-13 May 1992, pp: 2937-2940 vol.6

[11] Jean-Guy Schneider and Jun Han, "Components: Past, Present and Future", WCOP 2004

[12] CompoNet Open Source project: http://componet.sourceforge.net/

[13] Clemens Szyperski, Dominik Gruntz and Stephan Murer, "*Component Software – Beyond Object-Oriented Programming* Second Edition", Addison-Wesley / ACM Press, 2002 ISBN 0-201-74572-0

[14] Kohonen, T.; The Self-Organizing Map, Procs. IEEE, 78, 1464 ss, 1990.

[15] Carpenter, G.A., Grossberg, S., & Reynolds, J.H. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network, Neural Networks (Publication), 4, 565-588

[16] Carpenter *et al*, Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Transactions on Neural Networks, 3, 698-713

[17] MATLAB Neural Networks ToolBox: http://www.mathworks.com

[18] Teresa Serrano *et al* "Adaptive resonance theory microchips". Boston : Kluwer Academic Publishers, c1998.

[19] Cédric Gégout, Bernard Girau, Fabrice Rossi: "NSK, an Object-Oriented Simulator Kernel for Arbitrary Feedforward Neural Networks." pp: 95-104 ICTAI 1994: New Orleans, Louisiana, USA