2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

A Systems Software Architecture For Training Neural, Fuzzy Neural And Genetic Computational Intelligent Networks

Taner Arsan, Arif Selçuk Öğrenci, Tuncay Saydam
Kadir Has University
34230 Cibali-Istanbul, Turkey
{arsan, ogrenci, saydam}@khas.edu.tr

Abstract — **A systems software architecture for training distributed neural, fuzzy neural and genetic networks and their relevant information models have been developed. Principles of on-line architecture building, training, managing, and optimization guidelines are provided and extensively discussed. Qualitative comparisons of neural training strategies have been provided.**

I. INTRODUCTION

Neural networks (NN) have been extensively studied since they offer the techniques for solving the complex real-life optimization problems, such as, classification by learning from examples, without the need of a parametric model. Even though several architectures for network training have been developed so far, the optimal choice of a suitable network size and structural connections and efficient training of the network for a given application domain remain to be serious problems[13]. There have been efforts to construct "general purpose" simulation environments which would allow the users to define their NN model, choose the functionality of processing elements and carry out the training and simulation for specific problems [1]. Research has also been carried out for the simulation of NN in a distributed environment. A framework for concurrent simulation of NN has been proposed in [2], where a general purpose object-oriented concurrent programming language is used in conjunction with simple models of NN nodes. There exists also work about forming a library of software modules using object-oriented design and programming which would allow users to deploy prototype NN in order to solve specific tasks [3-7].

Generalized models for certain types of feed forward NN architectures and efficient training algorithms have been developed for solving classification and regression problems in [8], where specific types of nodes are used for building the NN. Even though, those work referenced above may address problems such as modular structure and training, the problem of finding the optimal software architecture remains unanswered. As also pointed out in [9] and [10], the evolution of the network to reach the optimal size is a very difficult problem. In a distributed environment, the problem becomes more critical, as each node operates autonomously so that a global minimization of the architecture is very hard. The lack of a general, open software architecture which not only incorporates the concurrent simulation but also integrates the various other strategies, such as fuzzy neural networks and genetic neural networks has attracted our interest. Our work aimed to construct an open software architecture for NN which exhibits the following attributes: A modular and flexible software architecture that will allow dynamic change of the new network topology and adjustment of training parameters during the training period ("online configurability"). Furthermore, our architecture aims to realize easy information retrieval by using a global control mechanism working in parallel with decentralized controllers on each node. This allows the system to be deployed on a distributed environment. This work shows clearly that the principle concepts and technology of software engineering can be successfully applied to neural networks to bring about distributed client-server oriented neural, fuzzy neural and genetic neural network architecture.

II. REVIEW OF NEURAL, FUZZY NEURAL AND GENETIC NEURAL NETWORK TRAINING

In our earlier work [13], we have used software architecture to describe the generic, distributed neural networks with its relevant information model. We are extending that model now by integrating other neural network technologies, such as fuzzy neural networks and genetic neural networks.
The concept of training, or learning from examples, is a very important issue. Neural networks are considered to operate in basically two modes: training and recall where training corresponds to the adaptation of the link weights when new patterns/vectors (training data) are applied to the neural network at the input layer. Depending on the presence of the desired responses for the applied inputs, the training is labeled as supervised or unsupervised. In any case, the neural network will change the values of its connection weights after the inputs are applied in such a manner that the outputs yield a satisfactory result. The degree to which the outputs are considered to be satisfactory also depends on the presence of desired given outputs. The process of recall, on the other hand, deals with applying new, unknown inputs at the input layer, and obtaining the neural network output. The success of the training is not only measured with the ability of the trained system to recall the original training data. The trained neural network also has to recall previously unknown patterns (input/output vectors) correctly which is the generalization property of the neural network.

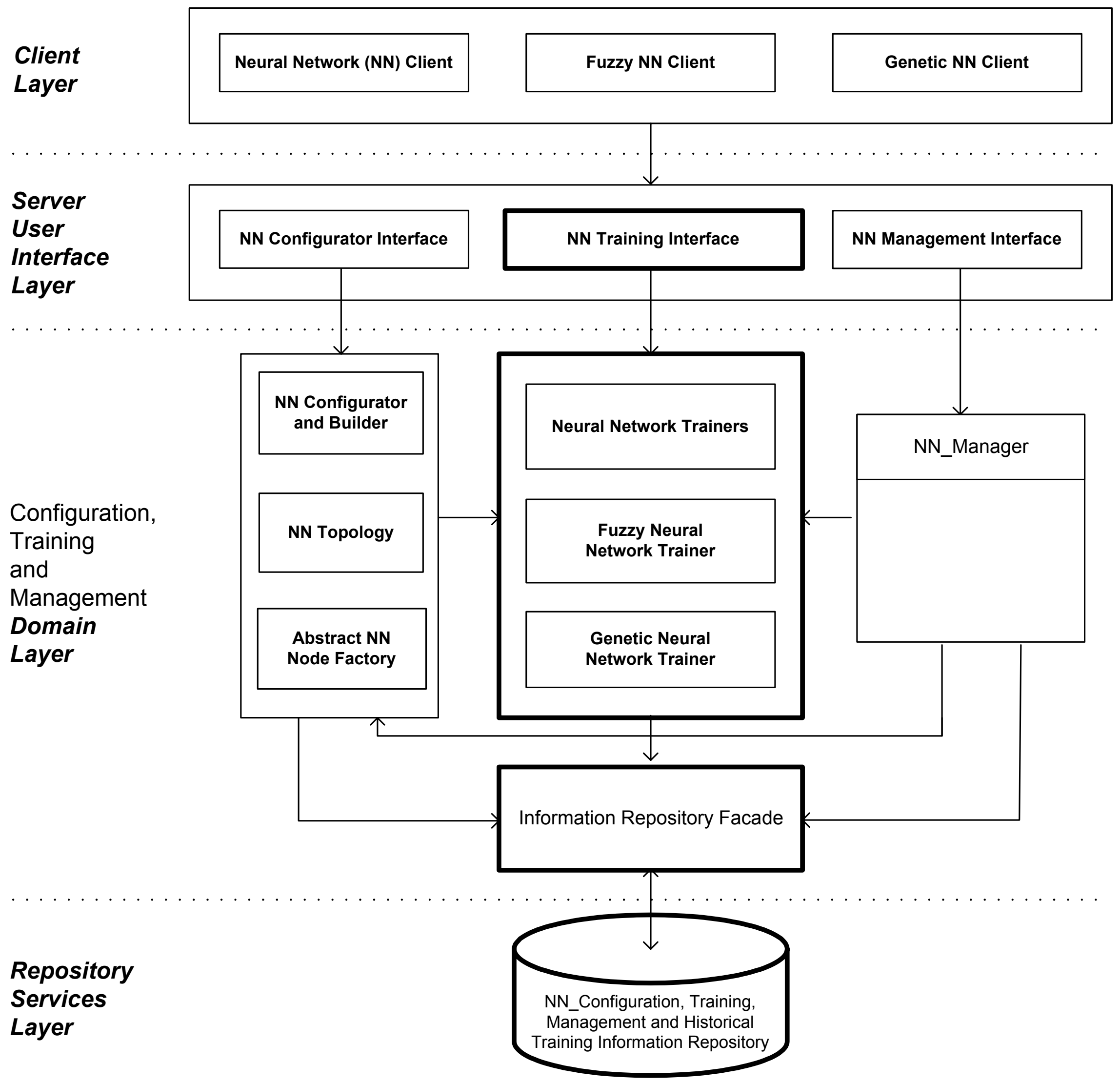
Client Layer
Neural Network (NN) Client
Fuzzy NN Client
Genetic NN Client
Server User Interface Layer
NN Configurator Interface
NN Training Interface
NN Management Interface
Configuration, Training and Management Domain Layer
NN Configurator and Builder
NN Topology
Abstract NN Node Factory
Neural Network Trainers
Fuzzy Neural Network Trainer
Genetic Neural Network Trainer
NN_Manager
Information Repository Facade
Repository Services Layer
NN_Configuration, Training, Management and Historical Training Information Repository

Fig. 1. A Software Systems Architecture for Neural Network Training Architecture.

Fuzzy neural network is functionally equivalent to a fuzzy inference model. It is basicly a five layer network, an input layer which reads in crisp inputs, an output layer which produces a crisp output and three hidden layers that represent fuzzy membership functions and fuzzy rules. Membership function nodes are mapped to fuzzy rule nodes. An output membership neuron will receive inputs from the corresponding fuzzy rule neurons combine them through the union operator into a crisp output. Training input/output data is presented to the system, the computed output will be compared to a given one, and as with the regular neural network, the system will learn through a back propagation algorithm.

Problem domain in genetic networks is represented by a chromosome and chromosome population [14,15,16]. Chromosome's performance is evaluated through a chromosome fitness function. The sum of squared errors is calculated on the training set of examples. The smaller the sum, the fitter the chromosome. Crossover operator takes two parent chromosomes and creates a single child with genetic properties taken from these parents. Mutation operator randomly selects a gene in a chromosome, while adding a small randomness to each weight in this gene. Training will until a previously specified number of generations (stopping criteria) have been built and considered.

We consider the proposed system to form a bridge between various neural networks architectures and contemporary software architecture methodology. In the following sections, we first give a Neural, Fuzzy Neural and Genetic Neural Network Software Architecture. This will be followed by a detailed information model. Then, the essential problems of training and optimization are addressed, where the NN Management functions are heavily utilized. After considering the relative properties of the three training strategies, the paper will conclude by analyzing the principal research contributions.

III. TRAINING ARCHITECTURE IN NEURAL, FUZZY NEURAL AND GENETIC NEURAL NETWORKS

We have developed a software architecture to enable clients to configure, train, and manage their neural networks. As shown in **Figure 1,** there four horizontal layers:

1 Client Layer, that normally run the client machines, **2 User Interface Layer,** that acts as client interface to training domain functionalities, **3 Configuration, Training and Management Domain Layer,** which include the key server domain functionalities and **4 Repository Services Layer,** which provides database services. *Information repository Façade* provides database access services to domain layer objects. As the most important part of this architecture, domain layer has the following objects: *Neural Network (NN) Configurator and Builder*. Through the use of a management interface, clients use this object and *NN-Manager* object to help build the neural nodes, nodal attributes, links and link attributes. The functional details of training and management objects are elaborated in Figure 2. ***Horizontal layering and vertical structuring*** inherent in this architecture help us achieve the following important software engineering advantages:

- Each neural network client is able to configure its own virtual neural network, run and train it ***independent*** of other users,
- The architecture integrates regular neural, fuzzy neural and genetic neural components into a single, object-oriented software. This allows ***distribution*** (implementations may run on several networked machines), ***transparency*** (both logical/functional, location and language transparency),
- It fosters an architecture that is ***open, modular*** and ***manageable***, and allows us to ***control design complexity*** through ***layering***,
- Network configurations and training results may be deposited at a ***common repository*** for future reference and information sharing,
- Enables the reuse of ***higher level architectural and design abstractions***, such as

Patterns,

- In general, neural networks are trained by the following principles:
  1 Compute alone, learn together,
  2 Compute forward, learn backward.

IV. INFORMATION MODEL OF THE TRAINING ARCHITECTURE

The central column of **Figure 1**, the training architectural objects, have been developed into an information model by specifying their detailed functionalities. The methods relevant to each object are indicated. Neural network training interface acts as training domain interface to calling clients. Likewise, information repository interface will act as database access interface to domain objects. Network management interface object, along with a manager object, provide functionalities so that clients can generate network nodes, set/get network configuration and set/get the needed training data for each training session. Manager communicates with the repository to set and get such information. Separation of actual training objects from their management, is an important distinction that provides an encapsulation and hiding of management functions. All interface objects are considered as *abstract objects* having virtual method signatures to access the domain objects where real objects and methods are implemented.

The key aspect of this model is naturally the three training objects as well as the manager object, as shown in the middle part of **Figure 2.** We distinguish three networks and training strategies: Artificial neural network literature is quite rich in variations of multilayer neural network learning algorithms, such as unsupervised (self-organizing maps, competitive,..),supervised (bidirectional associative memory, Hopfield,..) learning. Rather than considering each of these as learning objects in column 1, we have given the information models of only the two. *The first column*, standard neural network trainer has three objects, ***preprocessor, competitive trainer*** **and** ***adaptive multilayer trainer***. In competitive learning, neurons are expected to compete among themselves, and only a single output neuron is activated at any time. The winning neuron is the one that best matches the input vector, based on the minimum distance Euclidean criterion. Adaptive multilayer NN trainer does everything a regular NN trainer performs, such as weight initialization, output function activation, error computation and distribution through back propagation, but with one important addition: in order to improve convergence of back propagation learning, learning rate parameter is continuously adjusted during training. *The second column* has a ***fuzzy neural network trainer***. This is a rather complex object, which, during design, may be divided functionally into, *fuzzifier/defuzzifier, fuzzy function builder* and *fuzzy rule engine and trainer* objects. This system is a neural network that is functionally equivalent to a fuzzy inference model. In addition to input and output layers that read in crisp inputs and produces a crisp output, a neuro-fuzzy system has three hidden layers that represent membership functions and fuzzy rules. Membership function nodes are mapped to fuzzy rule nodes. An output membership neuron receives inputs from the corresponding fuzzy rule neurons and combines them through the union operator. Training input/output data is presented to the system, the computed output will be compared to a given one, and the system will learn through a back propagation algorithm. The neuron activation functions are modified as the output error is propagated backward. *The third column* is a ***genetic neural network trainer*** object. This also is a complex object that, during design, may be decomposed into following functionally coherent objects, such as : *chromosome population builder, fitness function evaluator, mutation and crossover, population trainer*. Problem domain is represented here as a chromosome and chromosome

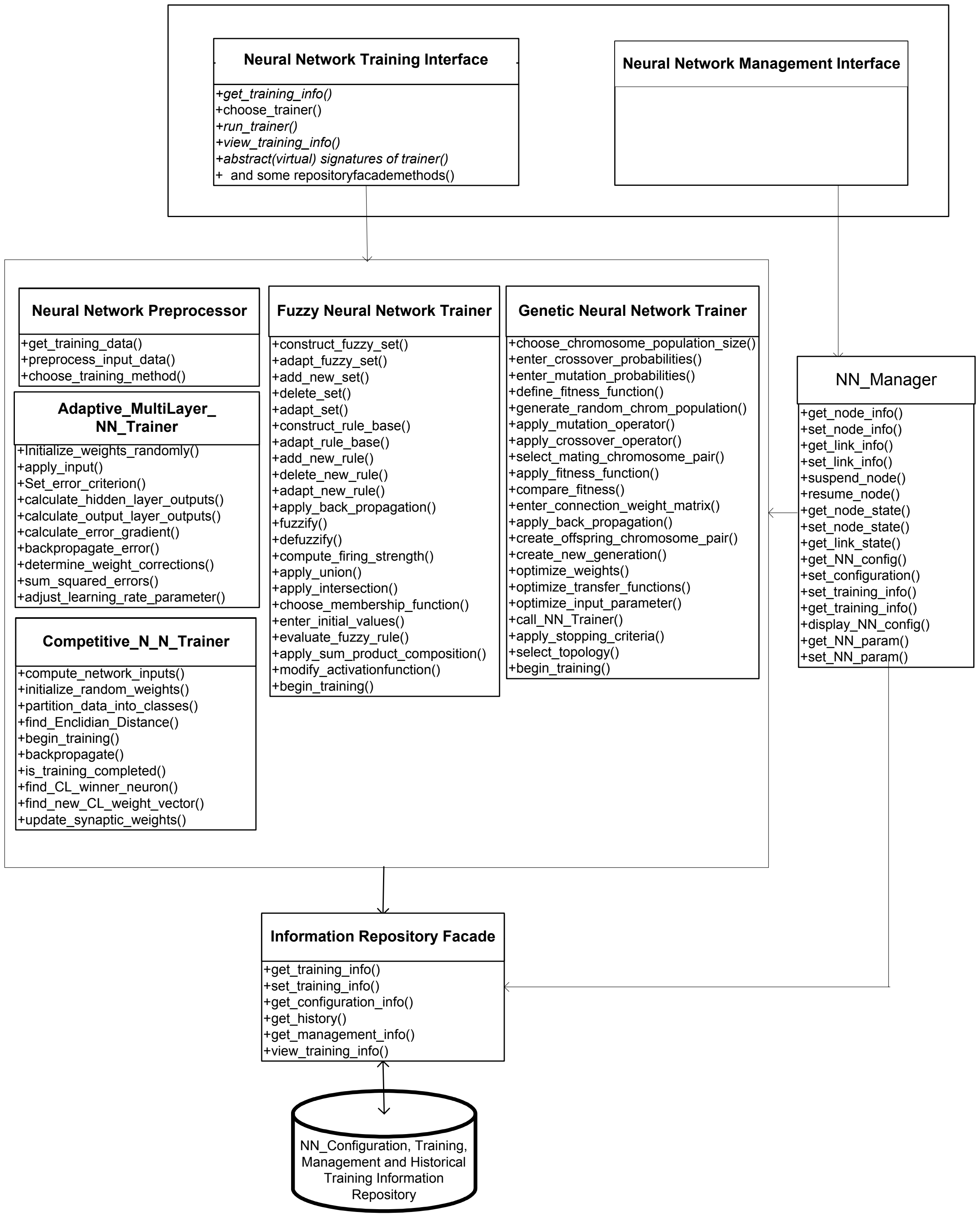
Neural Network Training Interface
+get_training_info()
+choose_trainer()
+run_trainer()
+view_training_info()
+abstract(virtual) signatures of trainer()
+ and some repositoryfacademethods()
Neural Network Management Interface
Neural Network Preprocessor
+get_training_data()
+preprocess_input_data()
+choose_training_method()
Adaptive_MultiLayer_ NN_Trainer
+Initialize_weights_randomly()
+apply_input()
+Set_error_criterion()
+calculate_hidden_layer_outputs()
+calculate_output_layer_outputs()
+calculate_error_gradient()
+backpropagate_error()
+determine_weight_corrections()
+sum_squared_errors()
+adjust_learning_rate_parameter()
Competitive_N_N_Trainer
+compute_network_inputs()
+initialize_random_weights()
+partition_data_into_classes()
+find_Enclidian_Distance()
+begin_training()
+backpropagate()
+is_training_completed()
+find_CL_winner_neuron()
+find_new_CL_weight_vector()
+update_synaptic_weights()
Fuzzy Neural Network Trainer
+construct_fuzzy_set()
+adapt_fuzzy_set()
+add_new_set()
+delete_set()
+adapt_set()
+construct_rule_base()
+adapt_rule_base()
+add_new_rule()
+delete_new_rule()
+adapt_new_rule()
+apply_back_propagation()
+fuzzify()
+defuzzify()
+compute_firing_strength()
+apply_union()
+apply_intersection()
+choose_membership_function()
+enter_initial_values()
+evaluate_fuzzy_rule()
+apply_sum_product_composition()
+modify_activationfunction()
+begin_training()
Genetic Neural Network Trainer
+choose_chromosome_population_size()
+enter_crossover_probabilities()
+enter_mutation_probabilities()
+define_fitness_function()
+generate_random_chrom_population()
+apply_mutation_operator()
+apply_crossover_operator()
+select_mating_chromosome_pair()
+apply_fitness_function()
+compare_fitness()
+enter_connection_weight_matrix()
+apply_back_propagation()
+create_offspring_chromosome_pair()
+create_new_generation()
+optimize_weights()
+optimize_transfer_functions()
+optimize_input_parameter()
+call_NN_Trainer()
+apply_stopping_criteria()
+select_topology()
+begin_training()
NN_Manager
+get_node_info()
+set_node_info()
+get_link_info()
+set_link_info()
+suspend_node()
+resume_node()
+get_node_state()
+set_node_state()
+get_link_state()
+get_NN_config()
+set_configuration()
+set_training_info()
+get_training_info()
+display_NN_config()
+get_NN_param()
+set_NN_param()
Information Repository Facade
+get_training_info()
+set_training_info()
+get_configuration_info()
+get_history()
+get_management_info()
+view_training_info()
NN_Configuration, Training, Management and Historical Training Information Repository

Figure 2. A Systems Information Model of Neural, Fuzzy Neural and Genetic Neural Networks Training Objects.

population. Chromosome's performance is evaluated through a fitness function. Each weight in the chromosome is assigned to a respective link in the network. The sum of squared errors is calculated on the training set of examples. The smaller the sum, the fitter the chromosome. Crossover operator takes two parent chromosomes and creates a single child with genetic properties taken from these parents. Mutation operator randomly selects a gene in a chromosome, while adding a small randomness to each weight in this gene. Training continues until a specified number of generations have been built and considered. Back propagation algorithm with certain variations are used in all three different training networks. In genetic networks, back propagation is normally used after each certain number of generations, e.g.10, have been generated. Recurrent network methods needed for the above cases have not been considered in this information model. In all the cases, the training data is set through *the network manager object*. More detailed comparisons of these three training objects are given in **Table 1.**

TABLE 1. COMPARISON OF NEURAL TRAINING AND LEARNING STRATEGIES.

Comparison Attribute	**Neural Networks (NN)**	**Fuzzy Neural Networks**	**Genetic Neural Networks**
Components	Artificial Neurons (AF)	AF + Fuzzy Rule Nodes	Strings representing chromosomes + AF
Topology Structure	Fixed, Networked Neurons	Fixed, Multilayer Networked Neurons + Rule Nodes	Dynamic, Networked Neurons
Soft Intelligence	Yes	Yes	Yes
Learning Through	Link weight adjustment	Fuzzy inference and link weight adjustment	Link weight optimization and selection combined
Optimization Solution	Back propagation	Back propagation	Apply genetic operators over generations plus back propagation
Optimization Nature	Suboptimal (local)	Suboptimal (local)	Near Optimal (hopefully local and Global)
Topology Selection	Difficult to select	Difficult to select	Topology concept is based on neurons and chromosome lengths
Layered Architecture	Yes, low degree	Yes, high degree	Yes, low degree
Stopping criteria	Error limit or number of training epochs	Fuzzy inference / error limit, number of training epochs	Error limit and/or number of generations created
Computational Overhead *(application dependent)*	Medium ?	Higher ?	High / Medium ?
Robustness to noise / Can deal with incomplete data	Yes / No	Yes / Yes	Yes / No ?!
Probabilistic Attributes	No	Yes – Fuzzy	Crossover mutation probabilities, random initial population generation
Activation Function	Continuous / Discrete Crisp Activation Function	Fuzzy Membership Activation Function	Selection based on crisp activation function
Treshold	Influences Neural Activation	Influences Neural Activation	Influences genetic variation mechanism and neural activation function

Neural Network Management

A distributed software architecture that provides transparency and asynchronous virtual network machine to its clients requires online software management. NN-Manager, a neural network management object: The main functionality of this object is to monitor neural network environment, collect network state information, configure network by setting each network node's input and output data, port connections and keep up-to-date network information repository. Initially, all domain objects work with this management objects in order to build network configuration (set/change/get node and topology information) as well as set/change/collect training data.

V. QUALITATIVE COMPARISONS OF NEURAL TRAINING STRATEGIES

In order to help the clients using different approaches to neural network training software, we have developed **Table 1**. Based on our experience with neural networks and a recent literature survey, we have used fourteen important attributes to compare the relative properties, similarities, differences and performance advantages of the three training strategies namely, regular neural, fuzzy neural and genetic neural networks. This table is designed to guide the users choose their neural learning strategy based on their problem domain, input data, computational and performance requirements.

VI. RESEARCH CONTRIBUTIONS AND CONCLUSIONS

From the foregoing software architecture work, we would like to derive the following conclusions:

- Each neural network client may configure its own virtual neural network server, runs and trains it ***independently,***

- The architecture that integrates regular neural, fuzzy neural and genetic neural components into a single, object-oriented software that is ***distributed*** (may run on several networked machines), ***transparent*** (both logical and location transparency, allowing for components working independently), ***asynchronous*** and ***autonomous,***

- The architecture is ***layered, open,*** and ***manageable*** by a management component. The architecture is ***modular***. It can be expanded by adding more neural network clients or servers, or, by adding objects with new and enhanced functionalities.

- The architecture has a well-developed **distributed software management hierarchy**. While assigning local management functions at each neural network node, the NN server management as shown in **Figure 2**. is placed and separated at a higher hierarchical level. An integrated neural network operation is accomplished by the interaction of these management functions.

- Three different types of clients can have their own ***special user interfaces*** to access the server's configuration, training and management functionalities. It is implied that a given user can have multiple copies as "simultaneous" windows to work on various aspects of his solution,

- Each neural network training result may be deposited at a ***common repository*** for future reference and information sharing,

- ***Detailed information model*** of the three different neural network training schemes, including the regular, fuzzy and genetic trainers, is an important part of this paper. Each object in this model has methods to show its highly detailed working functionalities in a clear manner. This model will prove to be very helpful during the detailed software design process.

- ***Functional separation, encapsulation*** and ***information hiding*** principles have been carefully incorporated into the architecture and its information model.

- **Table 1,** which compares the basic attributes of the three training techniques, might well be the most ***comprehensive*** of such ***comparisons*** in the literature. Based on several key attributes it tries to identify the advantages of each technique.

- In this paper, we believe to have successfully applied the current object-oriented software engineering principles, techniques and methodology to derive an ***integrated neural network training architecture***.

References

[1] E. Mesrobian & J. Skrzypek, A Software Environment For Studying Computational Neural Systems, *IEEE Transactions on Software Engineering, 18*(7), 1992, 575-589.

[2] G.L.Heileman et al., A General Framework for Concurrent Simulation of Neural Network Models, *IEEE Transactions on Software Engineering*, *18*(7), 1992, 551-562.

[3] G. Valentini & F. Masulli, NEURObjects: An object-oriented library for neural network development, *Neurocomputing*, *48*(1-4), 2002, 623-646.

[4] A. Weitzenfeld et al., A Neural Schema Architecture for Autonomous Robots, *Proc. of 1998 International Symposium on Robotics and Automation*, Coahuila, Mexico, 1998.

[5] G. E. Mobus & P. S. Fisher, Mavric's Brain, IEA/AIE '94: Proceedings of the Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Austin, Texas. 1994, 315-320.

[6] W. Pree et al., OO Design & Implementation of a Flexible Software Architecture for Decision Support System, *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE'97)* Madrid, Spain, 1997, 382-388.

[7] T. P. Caudell et al, eLoom and Flatland: specification, simulation and visualization engines for the study of arbitrary hierarchical neural architectures, *Neural Networks*, *16*, 2003, 617-624.

[8] G. Arulampalam & A. Bouzerdom, A generalized feedforward neural network architecture for classification and regression, *Neural Networks*, *16*, 2003, 561-568.

[9] L. A. Coward, A Functional Architecture Approach to Neural Systems, *International Journal of Systems Research and Information Systems*, *9*(2-4), 2000, 69-120.

[10] N. Garcia-Pedrajas et al., Multi-objective cooperative coevolution of artificial neural networks, *Neural Networks*, *15*, 2002, 1259-1278.

[11] E. D. Karnin, A Simple Procedure for Pruning Back-Propagation Trained Neural Networks, *IEEE Transactions on Neural Networks*, *1*(2), 1990, 239-242.

[12] T.F. Rathbun et al., MLP Iterative Construction Algorithm, *Neurocomputing*, *17*, 1997, 195-216.

[13] A.S.Ogrenci, T. Arsan and T. Saydam An Open Software Architecture of Neural Networks: Neurosoft, *Proceedings of SEA2004, Boston,MA*. November 2004.

[14] Nikola Kasabov, Evolving Connectionist Systems, *Springer Verlag*, 2003.

[15] Michael Negnevitsky, Artificial Intelligence, *Addison Wesley*, 2002.

[16] L.N.Castro and J. Timmis, Artificial Immune Systems, *Springer Verlag*, 2002.