Hybrid Intelligent Systems Design - A Review of a Decade of Research

Ajith Abraham & Baikunth Nath

School of Computing & Information Technology
Monash University (Gippsland Campus), Churchill 3842, Australia
Email: {Ajith.Abraham,Baikunth.Nath@infotech.monash.edu.au}

Abstract

The emerging need for Hybrid Intelligent Systems (HIS) is currently motivating important research and development work. The integration of different learning and adaptation techniques, to overcome individual limitations and achieve synergetic effects through hybridization or fusion of these techniques, has in recent years contributed to a large number of new intelligent system designs. Soft Computing (SC) introduced by Lotfi Zadeh [1] is an innovative approach to construct computationally intelligent hybrid systems consisting of Artificial Neural Network (ANN), Fuzzy Logic (FL), approximate reasoning and derivative free optimization methods such as Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS). Most of these approaches, however, follow an ad hoc design methodology, further justified by success in certain application domains. Due to the lack of a common framework it remains often difficult to compare the various hybrid systems conceptually and evaluate their performance comparatively. It has been over a decade since HIS were first applied to solve complicated problems. In this paper, we first aim at classifying state--of--the--art intelligent systems, which have evolved over the past decade in the HIS community. Some theoretical concepts of ANN, FL and Global Optimization Algorithms (GOA) namely GA, SA and TS are also presented. We further attempt to summarize the work that has been done and present the current standing of our vision on HIS and future research directions.

1. Introduction

Several adaptive hybrid intelligent systems have in recent years been developed for model expertise, decision support, image and video segmentation techniques, process control, mechatronics, robotics and complicated automation tasks etc. Many of these approaches use the combination of different knowledge representation schemes, decision making models and learning strategies to solve a computational task. This integration aims at overcoming limitations of individual techniques through hybridization or fusion of various techniques. These ideas have led to the emergence of several different kinds of intelligent system architectures [61][103][106].

It is well known that the intelligent systems, which can provide human like expertise such as domain knowledge, uncertain reasoning, and adaptation to a noisy and time varying environment, are important in tackling practical computing problems. In contrast with conventional Artificial Intelligence (AI) techniques which only deal with precision, certainty and rigor the guiding principle of hybrid systems is to exploit the tolerance for imprecision, uncertainty, low solution cost, robustness, partial truth to achieve tractability, and better rapport with reality [1]. In general HIS consists of 3 essential paradigms: Artificial neural networks, Fuzzy Logic and Global Optimization Algorithms (GOA). Nevertheless, HIS is an open instead of conservative concept. That is, it is evolving those relevant techniques together with the important advances in other new computing methods [2]. Table 1 lists the three principal ingredients together with their advantages.

Table 1. Hybrid intelligent system basic ingredients

Methodology	Advantage
Artificial Neural Networks	Adaptation, Learning and Approximation
Fuzzy Logic	Approximate Reasoning
Global Optimization Algorithms	Derivative Free Optimization

To realize a highly intelligent system, a synthesis of various techniques is required. Figure 1 shows the synthesis of ANN, FL and GOA and their mutual interaction leading to different architectures. Each technique plays a very important role in the development of HIS. Experience has shown that it is crucial for the design of HIS to primarily focus on the integration and interaction of different techniques rather than merge different methods to create ever-new techniques. Techniques already well understood, should be applied to solve specific domain problems within the system. Their weakness must be addressed by combining them with complementary methods.

ANNs offer a highly structured architecture with learning and generalization capabilities, which attempts to mimic the neurological mechanisms of the brain. A Neural Network stores knowledge in a distributive manner within its weights; which have been determined by learning with known samples. The generalization ability for new inputs is then based on the inherent algebraic structure of the ANN. However it is very hard to incorporate human a priori knowledge into an ANN. This is mainly due to the fact that the connectionist paradigm gains most of its strength from a distributed knowledge representation.

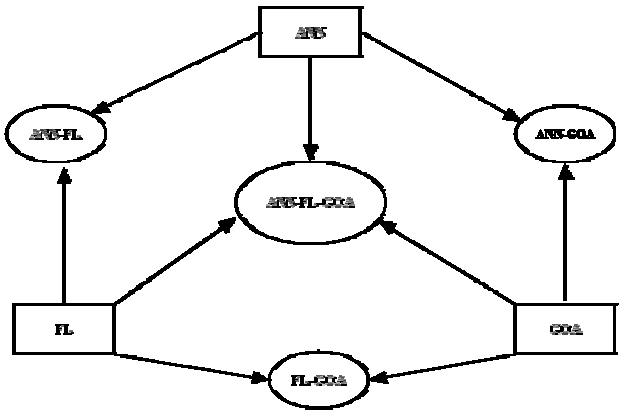


Figure 1. General framework for hybrid intelligent systems

In Contrast, FL exhibit complementary characteristics, offering a very powerful framework for approximate reasoning as it attempts to model the human reasoning process at a cognitive level. Fuzzy Systems (FS) acquires knowledge from domain experts and this is encoded within the algorithm in terms of the set of *if-then* rules. Fuzzy systems employ this rule based approach and interpolative reasoning to respond to new inputs. The incorporation and interpretation of knowledge is straight forward, whereas learning and adaptation constitute major problems.

Global optimization is the task of finding the absolutely best set of parameters to optimize an objective function. In general, it may be possible to have solutions that are locally optimal but not globally optimal. Consequently, global optimization problems are typically quite difficult to solve exactly; in the context of combinatorial problems, they are often NP-hard. A wide variety of methods have been proposed for solving these problems inexactly, including simulated annealing, genetic algorithms, and Tabu Search. Evolutionary Computing (EC) works by simulating evolution on a computer. From an historical point of view, the evolutionary optimization methods are divided into three main categories: Genetic Algorithms (GAs), Evolutionary Programming (EP) and Evolution Strategy (ES) [3]. These methods are fundamentally iterative generation and alteration processes operating on a set of candidate solutions that form a population. The entire population evolves towards better candidate solutions via the selection operation and genetic operators such as crossover and mutation. The selection operator decides which candidate solutions move on into the next generation, thus limits the search space.

Due to the complementarity of ANNs and FL, the trend in the design of hybrid system is to merge both of them into a more powerful integrated system, to overcome their individual weakness [60]. Global optimization algorithms could be useful to formulate an optimal combination of ANN and FL. A quick review of the literature shows that several Neuro-Fuzzy (NF) models have been developed [4-13][16][38-44][46-56][58][62-64][66-108]. A fuzzy neural network makes use of fuzzy rules to change the learning rate of the ANN or even by creating a network that works on fuzzy inputs [43] [44]. In a concurrent NF model both ANN and FL work independently without any interaction. Contrary, a cooperative NF mode makes use of ANN to determine the parameters of FL system [88][93]. Finally in the hybrid approach Fuzzy Inference System (FIS) is merged with an ANN. The topology of the hybrid NF model is very much similar to the feedforward ANN, i.e., nodes and layers. In a popular hybrid NF model the node functions are replaced by fuzzy rules and connection weights represents the fuzzy sets. Among the various NF models only the hybrid integrated NF model make use of the complementarity strength of ANN and FL. As we are the proponent of the integrated hybrid approach, in this review we will concentrate only on the various representations of integrated (fused) NF architectures.

2. Models Of Hybrid Systems

We broadly classify the various HIS architectures into 4 different categories based on the systems overall architecture: (1) Stand alone Architectures (2) Transformational Architectures (3) Hierarchical Hybrid Architectures and (4) Integrated Hybrid Architectures. The following sections discuss each of these strategies and expected uses of the model, and benefits and limitations of the approach.

2.1 Stand Alone Architecture

Stand-alone models of HIS applications consist of independent software components, which do not interact in anyway. Developing stand-alone systems can have several purposes. First they provide direct means of comparing the problem solving capabilities of different techniques with reference to a certain application [14-15]. Running different techniques in a parallel environment permits a loose approximation of integration. Stand-alone models are often used to develop a quick initial prototype, while a more time-consuming application is developed. Figure 2 displays a stand-alone system where neural network, fuzzy logic and expert system are being used separately.

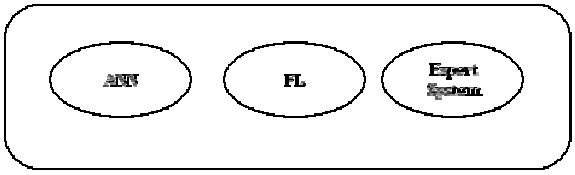


Figure 2. Stand –alone architecture

Some of the benefits are simplicity and ease of development using commercially available software packages. On the other hand, stand-alone techniques are not transferable; neither can support the weakness of the other technique.

2.2 Transformational Hybrid Architecture

In a transformational hybrid model the system begins as one type of system and end up as the other [57]. Determining which technique is used for development and which is used for delivery is based on the desirable features that the technique offers.

Expert Systems and ANNs have proven to be useful transformational models. Variously, either the expert system is incapable of adequately solving the problem, or the speed, adaptability, and robustness of neural network is required. Knowledge from the expert system is used to set the initial conditions and training set for ANN. Figure 3 shows the interaction between an ANN and expert system in a transformational hybrid model.

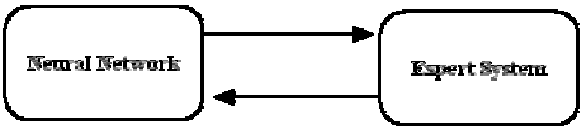


Figure 3. Transformational hybrid architecture

Transformational hybrid models are often quick to develop and ultimately require maintenance on only one system. Models can be developed suited to the environment and offer many operational benefits. Unfortunately, transformational models are significantly limited. Most of the developed models are just application oriented. For a different application, a totally new development effort might be required. A fully automated means of transforming an expert system to ANN and vice versa is required.

2.3 Hierarchical Hybrid Architectures

The architecture is built in a hierarchical fashion, associating a different functionality with each layer. The overall functioning of the model will depend on the correct functioning of all the layers [16]. Figure 4 demonstrates a hierarchical hybrid architecture involving an ANN, GA and Expert system. ANN uses a GA to optimize its topology and the output is fed directly to an expert system, which then creates the desired output. Possible error in one of the layers will directly affect the desired output.

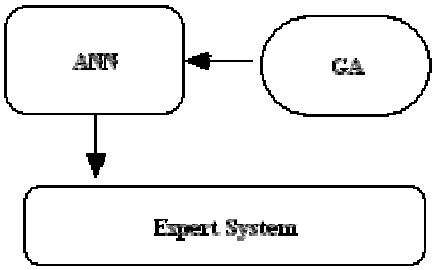


Figure 4. Hierarchical hybrid architectures

2.4 Integrated Hybrid Architectures

Fused architectures are the first true form of integrated intelligent systems. These models include systems, which combine different techniques into one single computational model. They share data structures and knowledge representations. Using ANN and FL, some of the major woks in this area are, Adaptive Network based Fuzzy Inference System (ANFIS) [4-5], NEFCON [6], NFCLASS [7], NEFPROX [42] FUN [8], SONFIN [9], FINEST [10], FuNN [51], EfuNN [49], dmFuNN [50], and many others [53-54] [58] [63-64]. Common to these approaches is their network-like architecture, which is often used, in one way or the other, on a multi-layered fuzzy rule base evaluation scheme (fuzzification, premise evaluation, truth value propagation, conclusion aggregation and defuzzification).

Another approach is to put the various techniques on a side-by-side basis and focus on their interaction in the problem-solving task. This method might allow integrating alternative techniques and exploiting their mutuality. Further more, the conceptual view of the agent allows one to abstract from the individual techniques and focus on the global system behavior, as well as study the individual contribution of each component.

Figure 5 shows a new framework based on the rational agent approach to characterize and analyze complex intelligent systems [11]. The architecture consists of four components and the environment (problem) on which it acts. The environment consists of the problem task and conceptually speaking it is the state vector perceived by the agent through the sensors and influenced by it through its effectors. The Performance Element (PE) is the actual controller mapping environment, states to actions. The Learning Element (LE) updates the knowledge represented in the PE in order to optimize the agent's performance with respect to an outside performance measure. It has access to the environment state, the agent's past actions, and an immediate reinforcement signal indicating the appropriateness of the action that last influenced the environment state. Given this information it updates the PE so that in future situations more pertinent actions are chosen over less pertinent ones. The critic faces the problem of transforming an external reinforcement signal into an internal one. The problem generator is to contribute to the exploration of the problem space in the most efficient way. This framework does not

require the use of a specific technique for realizing the individual components. These techniques may be chosen entirely according to their strength and according to the application. Table 2 summarizes some of the work done in the area with the components used [12-13].

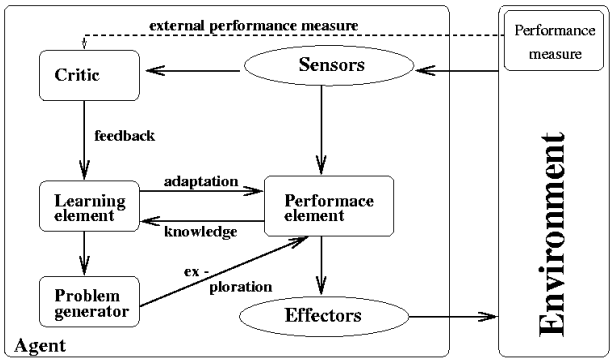


Figure 5. Conceptual learning agent architecture

Table 2. Actual components and techniques used in the conceptual learning agent architecture

Component	Technique Used
Performance Element	Fuzzy Rule Based System, Neuro Fuzzy System
Critic	Expert System, ANN trained with TD (λ) rule.
Learning element	Adaptive Reinforcement Algorithm, ANNs.

The benefits of fused architecture include robustness, improved performance and increased problem-solving capabilities. Finally, fully integrated models can provide a full range of capabilities such as adaptation, generalization, noise tolerance and justification. Fused systems have limitations caused by the increased complexity of the inter module interactions and specifying, designing, and building fully integrated models is complex. In this paper we further limit our review to fused architecture based systems.

3. Artificial Neural Networks

The study of Artificial Neural Networks (ANNs) originated in attempts to understand and construct mathematical models for neurobiology and cognitive psychology, and their current development continues to shed light in these areas. Although significant advance has been achieved in the area of conventional expert systems for mimicking human intelligence, there is still a long way to go for the current computational techniques before realizing the capability of carrying out certain man-dependent tasks.

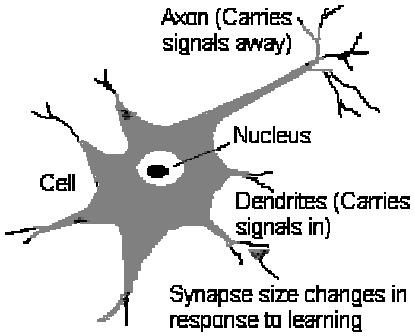


Figure 6. A mammalian nerve cell (neuron)

Human brains provide proof of the existence of ANNs that can succeed at those cognitive, perceptual and control tasks in which humans are successful. Rough arguments from neurobiology suggest that the cycle time of an individual human neuron is 10^{-3}seconds (1 millisecond) for a clock rate of less than 1 KHz. This compares with the current computers operating on a cycle time of 10^{-9}seconds for a clock rate of about 1 GHz, a factor more than a million (10^6). Nevertheless the brain is capable of computationally demanding perceptual acts (e.g., recognition of faces, speech) and control activities (e.g. body movements and body functions) that are now only on the horizon for computers. The advantage of the brain is its effective use of massive parallelism, the highly parallel computing structure, and the imprecise information processing capability. Figure 6 shows the mammalian nerve cell and associated components. The basic processing element in the nervous system is the neuron. Tree-like networks of nerve fiber called dendrites are connected to the cell body or soma, where the cell nucleus is located. Extending from the cell body is a single long fiber called the axon, which eventually branches into strands and sub strands, and are connected to other neurons through synaptic junctions, or synapses. The transmission of signals from one neuron to another at synapses is a complex chemical process in which specific transmitter substances are released from the sending end of the junction. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If the potential reaches a threshold, a pulse is sent down the axon - we then say the cell has "fired". In a simplified mathematical model of the neuron, the effects of the synapses are represented by *weights* that modulate the effect of the associated input signals, and the nonlinear characteristic

exhibited by neurons is represented by a transfer function, which is usually the sigmoid, gaussian, trigonometric functions etc. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm.

ANNs have been developed as generalizations of mathematical models of biological nervous systems. They have the advantageous capabilities of learning from training data, recalling memorized information, and generalizing to the unseen patterns. These capabilities do show great potential in such application areas as prediction [14], classification [15], control engineering [17], signal processing [18], and pattern recognition [19]. There are multitudes of different types of ANNs. Some of the more popular include the multilayer perceptron which is generally trained with the backpropagation of error algorithm, learning vector quantization, radial basis function, Hopfield, and Kohonen, to name a few. Some ANNs are classified as feedforward while others are recurrent (i.e., implement feedback) depending on how data is processed through the network. Another way of classifying ANN types is by their method of learning (or training), as some ANNs employ supervised training while others are referred to as unsupervised or self-organizing. Supervised training is analogous to a student guided by an instructor. Unsupervised algorithms essentially perform clustering of the data into similar groups based on the measured attributes or features serving as inputs to the algorithms.

ANNs are characterized by the network architecture, the connection strength between pairs of neurons (weights), node properties, and updating rules. The updating or learning rules control weights and/or states of the processing elements (neurons). Normally, an objective function is defined that represents the complete status of the network, and its set of minima corresponds to different stable states of the network. It can learn by adapting its weights to changes in the surrounding environment, can handle imprecise information, and generalise from known tasks to unknown ones. Each neuron is an elementary processor with primitive operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. Parallel implementations of ANNs offer an attractive way of speeding up both the learning and recall phases [20]. Parallel mapping of ANN models have been implemented on various hardware platforms and processor topologies for different ANN architectures.

There are all together more than one hundred ANN architectures and algorithms proposed by people from varying standpoints. However the most widely used ANNs are a few. We only present the Back Propagation (BP) Neural Network, which is widely used in HIS.

3.1 Back Propagation Neural Network

The simple perceptron is just able to handle linearly separable or linearly independent problems. For those non-linear problems (e.g. XOR), it is, however required that the network should have an appropriate intermediate representation of the input patterns by introducing nonlinear hidden layers. Back-Propagation Neural Network (BPNN) utilizes the delta rule for the learning process [21]. Backpropagation is an abbreviation for the backwards propagation of error. With the delta rule, as with other types of backpropagation, learning is a supervised process that occurs with each cycle or epoch (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random "guess" as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.

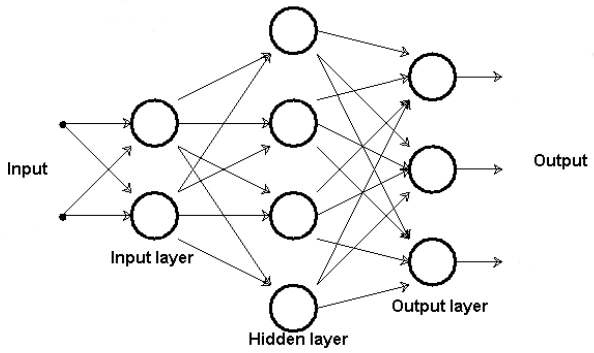


Figure 7. Typical three-layer feedforward network architecture

The network is initially randomized to avoid imposing any of our own prejudices about an application on the network. The training patterns can be thought of as a set of ordered pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_P, y_P)\}$ where x_i represents an input pattern and y_i represents the output pattern vector associated with the input vector x_i. The process of training the network then proceeds according to the following algorithm, which is derived as a natural result of finding the gradient of the error surface (in weight space) of the actual output produced by the network with respect to the desired result [22].

1. Select the first training vector pair from the training pair vectors. Call this the vector pair (x,y).
2. Use the input vector x, as the out put from the input layer of processing elements.
3. Compute the activation to each unit on the subsequent layer.
4. Apply the appropriate activation function, which we denote as $f(net^h)$ for the hidden layer and as $f(net^0)$ for the output layer, to each unit on the subsequent layer.
5. Repeat steps 3 and 4 for each layer in the network.
6. Compute the error, δ^o_{pk}, for this pattern p across all K output layer units by using the formula: $\delta^o_{pk} = (y_k - o_k) f'(net^o_k)$

7. Compute the error δ_{pj}^{h}, for all J hidden layer units by using the recursive formula. $\delta_{pj}^{h} = f'(net_j^h) \sum_{k=1}^{K} \delta_{pk}^{o} w_{kj}$

8. Update the connection weight values to the hidden layer by using the equation: $w_{ji}(t+1) = w_{ji}(t) + \eta \delta_{pj}^{h} x_i$. Where η is a small value used to limit the amount of change allowed to any connection during a single pattern training cycle.

9. Update the connection weight values to the output layer by using the equation: $w_{kj}(t+1) = w_{kj}(t) + \eta \delta_{pk}^{o} f(net_j^h)$.

10. Repeat steps 2 to 9 for all vector pairs in the training set. Call this one training epoch.

Steps 1 to 10 are to be repeated for as many epochs as it takes to reduce the sum of squared error to a minimal value according to the formula $E = \sum_{p=1}^{P} \sum_{k=1}^{K} (\delta_{pk}^{o})^2$ (1)

Figure 7 shows a typical *L*-layer feed-forward network (the input nodes are not counted as a layer) consists of an input layer, (*L*-1) hidden layers, and an output layer of nodes successively connected in a feed-forward fashion with no connections between neurons in the same layer. It has been proved that BPNN with sufficient hidden layers can approximate any nonlinear function to arbitrary accuracy. This makes BPNN a good candidate for signal prediction, forecasting and system modeling systems. In the batched mode variant the descent is based on the gradient ∇E for the total training set [23].

$$\Delta w_{ij}(n) = -\varepsilon * \frac{\delta E}{\delta w_{ij}} + \alpha * \Delta w_{ij}(n-1) \quad (2)$$

ε and α are the learning rate and momentum respectively. A good choice of both the parameters are required for training success and speed of the ANN. BP often gets stuck in a local minimum mainly because of the random initialization of weights. For some initial weight settings, BP may not be able to reach global minimum of weight space, while for other initializations the same network is able to reach a optimal minimum. BP usually generalizes quite well, but sometimes it does not. Generalization can be compared to interpolation in mathematics and when an ANN fails to learn it is often referred as "over training". This usually happens when the ANN is trained for a very long time for a particular task. The ANN initially learns to detect the global features of the input and as a consequence generalizes very well. But after prolonged training the network will start to recognize individual input/output pair rather than settling for weights that generally describe the mapping for the whole training set. Conventional ANNs are not able to learn many problems at one time. The different problems seem to be in each other's way: if one of the problems is represented in the weights of the network the other problem is forgotten and vice versa.

An optimal design of an ANN can only be achieved by the adaptive evolution of connection weights, architecture and learning rules that progress on different time scales [24] [25]. Figure 8 illustrates the general interaction mechanism with the architecture of the ANN evolving at the highest level on the slowest time scale.

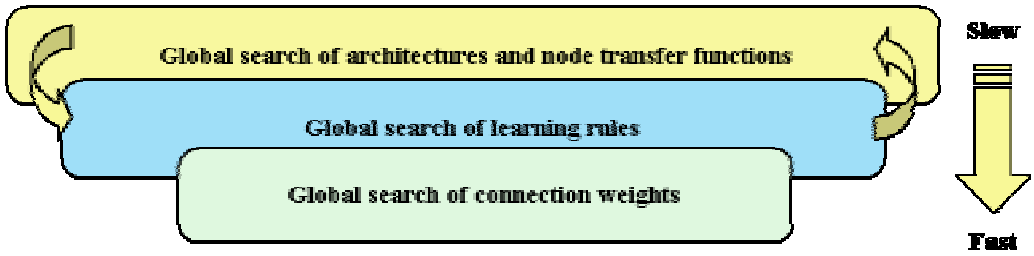


Figure 8 Interaction of evolutionary search mechanisms for the formation of a optimal neural network

For every architecture, there is the evolution of learning rules, which proceeds on a faster time scale in an environment decided by the architecture. For each learning rule, evolution of connection weights proceeds at a faster time scale in an environment decided by the problem, the learning rule and the architecture. Hierarchy of the architecture and learning rules rely on the prior knowledge. If there is more prior knowledge about the learning rules than the architecture then it is better to implement the learning rule at a higher level.

4. Fuzzy Logic

Zadeh [1] introduced the concept of Fuzzy Logic (FL) to present vagueness in linguistics, and further implement and express human knowledge and inference capability in a natural way [26]. Figure 9 shows a basic configuration of a fuzzy logic system. They are a fuzzification interface, a fuzzy rule base (knowledge base), an inference engine (decision-making logic), and a defuzzification interface. The following section summarizes basic concepts and notations of fuzzy set theory and fuzzy logic, which will be required to understand the complete literature.

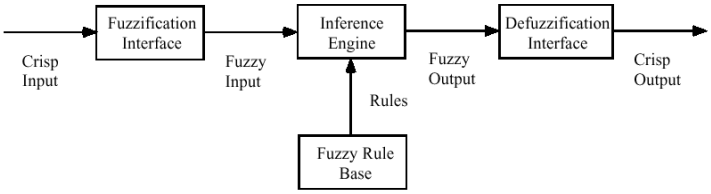


Figure 9. Basic architecture of a fuzzy logic system

Let X be a space of objects and x be a generic element of X. A classical set A is defined as a collection of elements or objects x ϵ X, such that each x can either belong to or not belong to the set A, A ⊑ X. By defining a characteristic function (or membership function) on each element x in X, a classical set A can be represented by a set of ordered pairs (x,0) or (x,1), where 1 indicates membership and 0 non-membership. Unlike Conventional set mentioned above fuzzy set expresses the degree to which an element belong to a set. Hence the characteristic function of a fuzzy set is allowed to have value between 0 and 1, denoting the degree of membership of an element in a given set. If X is a collection of objects denoted generically by x, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A= \{(x, \mu_A (x)) \mid x \epsilon X\} \quad (3)$$

$\mu_A (x)$ is called the Membership Function (MF) of linguistic variable x in A, which maps X to the membership space M, M= [0,1], where M contains only two points 0 and 1, A is non-fuzzy (crisp) and μ_A is identical to the characteristic function of a crisp set.

Fuzzification is necessary in the fuzzy logic scheme, because the observed input data from the existing sensors is always crisp and numerical, while the inference engine can only deal with linguistic values. Generally there are 3 ways for the fuzzification operation. The first approach is to convert a crisp value into a fuzzy singleton within the specified universe. In fact, a fuzzy singleton is a precise value, and therefore no fuzziness is introduced at this stage. The second method copes with the case of random noise disturbed input data. The fuzzification operator will convert the probabilistic data into fuzzy numbers. Thus the computational efficiency of fuzzy reasoning is improved, since fuzzy numbers are easy to manipulate rather than random variables. The third approach is a hybrid scheme incorporating both certainty (fuzzy numbers) and randomness (random numbers) combining the advantages of both the above schemes.

The fuzzy rule base is characterized in the form of i*f-then* rules in which preconditions and consequents involve linguistic variables. The collection of these fuzzy rules forms the rule base for the fuzzy logic system. Due to their concise form, fuzzy if-then rules are often employed to capture the imprecise modes of reasoning that play an essential role in the human ability to make decisions in an environment of uncertainty and imprecision. Most fuzzy systems employ the inference method proposed by Mamdani [110] in which the rule consequence is defined by fuzzy sets and has the following structure:

$$if \;\; x_1 \; is \; A_1^k \; and \; ...x_{N_x} \; is \; A_{N_x}^k \; then \; y_1 \; is \; B_1^k, ..., \; y_{N_Y} \; is \; B_{N_Y}^k \quad (4)$$

Takagi, Sugeno and Kang (TSK) [109] proposed an inference scheme in which the conclusion of a fuzzy rule is constituted by a weighted linear combination of the crisp inputs rather than a fuzzy set.

$$if \;\; x_1 \; is \; A_{1,k} \; and \; ...and.....x_{N_X} \; is \; A_{N_X,k} \; then \; y_1 = f_{1,k}(x_1,...., x_{N_X}),...., y_{N_Y} = f_{N_Y,k}(x_1,...., x_{N_X}) \quad (5)$$

TSK fuzzy controller usually needs a smaller number of rules, because their output is already a linear function of the inputs rather than a constant fuzzy set. The fuzzy rule base is usually constructed manually or by automatic adaptation by some learning techniques as used in neuro-fuzzy systems. There exist four modes of obtaining fuzzy rules for the knowledge base.

1. From expert experience.
2. Based on a skilled operator's action on a real time task.
3. Fuzzy model of the process to be monitored.
4. Self-organizing fuzzy controller.

It is typically advantageous if the fuzzy rule base is adaptive to a certain application. The self-organizing fuzzy controller [27] is a good example to demonstrate the characteristics of this approach. Adapting fuzzy rule base using neural network learning mechanism is another technique [28].

4.1 Fuzzy inference methods

Compositional operation consists of two phases: a combination and a projection phase. Aggregation is the combining of fuzzy relations, representing fuzzy rules, into a single fuzzy relation. There are 4 different methods for the fuzzy rules inference. They are *Max-min* operation [26], *Max-product* inference [29], *Sum-product* or *sum-dot* method [30] and *Max drastic product* operation [30]. In the following section we will discuss the first method.

Max-min operation is based on choosing a *min* operator for the conjunction in the premise of the rule as well as for the implication function and the *max* operator for the aggregation. The application of the compositional rule of inference results in:

$$\mu_{B^\circ}(y) = \overbrace{\max_{k}}^{aggregation} \underbrace{\min}_{implication} (\beta_k \mu_{B_k}(y)) \quad (6)$$

Max product inference [29] method is another commonly applied inference method in fuzzy control. The max-product is also known as max-dot method. This inference method is characterized by scaling (product) the consequent B_k of a fuzzy rule r_k with the degree of fulfillment β_k of that rule and aggregating these results B_k° to obtain the fuzzy controller output by means of a max operator:

$$\mu_{B^\circ}(y) = \overbrace{\max_{k}}^{aggregation} \beta_k \underset{implication}{*} \mu_{B_k}(y)) \text{; where" * " represents multiplication.} \quad (7)$$

The defuzzification interface is a mapping from a space of fuzzy actions defined over an output universe of disclosure into a space of non-fuzzy actions, because the output from the inference engine is usually a fuzzy set while for most practical applications crisp values are often required. The three commonly applied defuzzification techniques are, *max-criterion*, *center-of-gravity* and the *mean- of-maxima*. The *max-criterion* is the simplest of these three to implement. It produces the point at which the possibility distribution of the action reaches a maximum value.

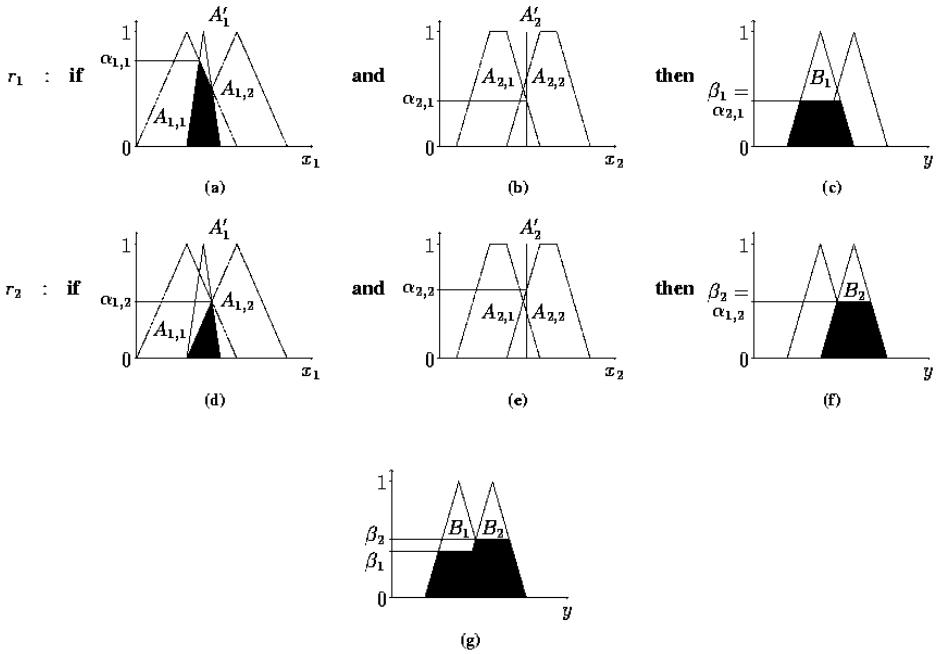


Figure 10. Fuzzy inferencing system using the *max-min* method.(a) , (b) and (c) illustrates a fuzzy *if-then* rule (r_1) and (d), (e) and (f) illustrates a fuzzy *if-then* rule (r_2). The rules r_1 and r_2 are then aggregated using the max-min method to obtain the final output (g).

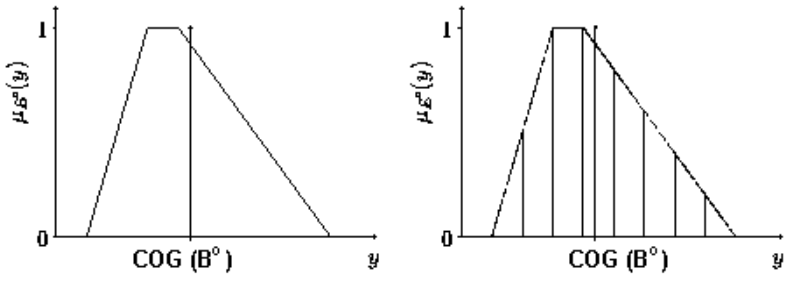


Figure 11. Center-of-gravity defuzzification method (a) continuous and (b) discrete

The Center-Of-Gravity (COG) is an averaging technique. The difference is that the (point) masses are replaced by the membership values. This method is often called the center of area defuzzification method in the case of 1-dimensional fuzzy sets. In a continuous form the COG is defined by:

$$COG(B^{\circ}) = \frac{\int_{Y} \mu_{B^{\circ}}(y) y dy}{\int_{Y} \mu_{B^{\circ}}(y) dy} \tag{8}$$

and the discrete form is defined by:

$$COG(B^{\circ}) = \frac{\sum_{q=1}^{N_q} \mu_{B^{\circ}}(y_q) y_q}{\sum_{q=1}^{N_q} \mu_{B^{\circ}}(y_q)} \tag{9}$$

Where N_q is the number of quantization used to discretize membership function $\mu_{B^{\circ}}(y)$ of fuzzy output B°. Figure 11 illustrates the COG defuzzification method for a continuous and discrete defuzzification process.

Mean-Of-Maxima (MOM) strategy generates the crisp action that represents the mean value of all local actions, whose membership functions reach the maximum. It can be expressed as follows:

$$MOM(B^{\circ}) = \sum_{j=1}^{m} \frac{z_j^s}{m^s} \tag{10}$$

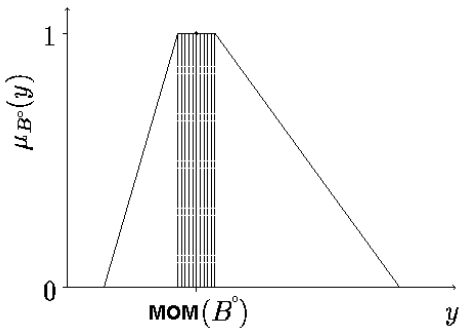


Figure 12. Mean-of–maxima defuzzification method

Where z_j^s is the support value at which the membership function reaches the maximum value of $\mu_B(z_j)$ and m^s is the number of such support values. Figure 12 illustrates the mean-of–maxima defuzzification method. Mean-of-maxima method can achieve a better transient performance while the COG leads to a better steady state performance.

4.2 Evolutionary Fuzzy Control Systems

Several research works are going on exploring the integration of evolutionary algorithms with fuzzy logic [113-118]. Majority of the work are concerned with the automatic design or optimization of fuzzy logic controllers either by adapting the fuzzy membership functions [118] or by learning the fuzzy *if-then* rules [113-114][116-117][121]. The first method results in a self-tuning controller in which a GA adapts the fuzzy membership functions. The genome encodes parameters such as center and width of trapezoidal, triangle or Gaussian membership functions. This approach requires a previously defined rule base and is primarily useful in order to optimize the performance of an already existing controller.

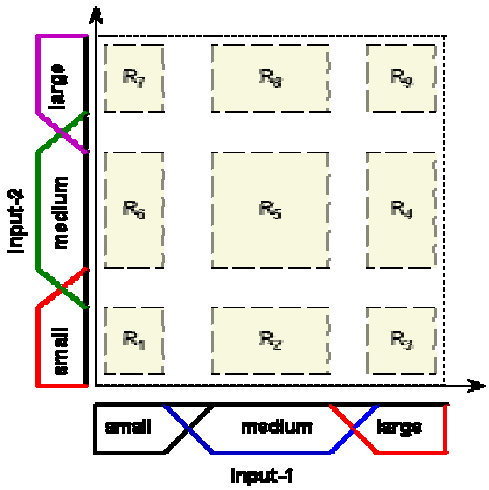


Figure 13. Example showing how the 2 dimensional space is partitioned using 3 trapezoidal membership functions per input dimension

The second approach is based on a self-organizing process that learns the appropriate relationship between control input and output starting without any previous knowledge about the rules. The size of the genotype depends on the number of input variables and fuzzy sets. Referring to Figure 13, a Mamdani or TSK rule may be formed as:

If input-1 is *medium* and input 2 is *large* then rule R_8 is fired.

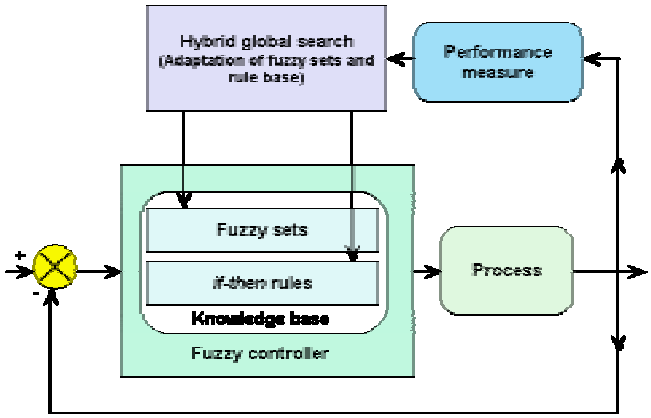


Figure 14. Adaptive fuzzy control system architecture

A conventional fuzzy controller makes use of a model of the expert who is in a position to specify the most important properties of the process [119-120]. Figure 14 shows the architecture of the adaptive fuzzy control system architecture wherein the fuzzy membership functions and the rule bases are optimized using a global search procedure. An optimal design of an adaptive fuzzy control system can only be achieved by the adaptive evolution of membership functions and learning rules which progress on different time scales. Figure 15 illustrates the general interaction mechanism with the global search of fuzzy rules evolving at the highest level on the slowest time scale. For each fuzzy rule base, global search of membership functions proceeds at a faster time scale in an environment decided by the problem [122].

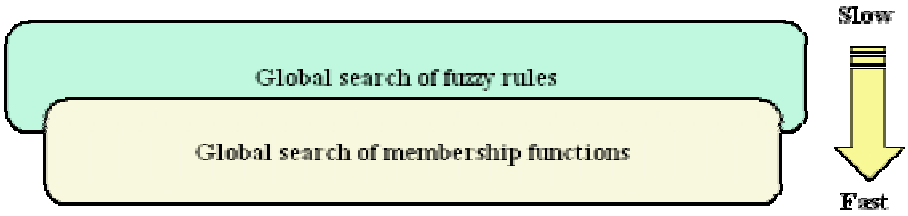


Figure 15. Interaction of evolutionary search mechanisms in the design of optimal adaptive fuzzy control system.

Automatic adaptation of membership functions is popularly known as self tuning and the genome encodes parameters of trapezoidal, triangle, logistic, Laplace, hyperbolic-tangent, Gaussian membership functions etc.

Global search of fuzzy rules can be carried out using two approaches. In the first approach the fuzzy knowledge base is adapted as a result of antagonistic roles of competition and cooperation of fuzzy rules. Each genotype represents a single fuzzy rule and the entire population represents a solution. A classifier rule triggers whenever its condition part matches the current input, in which case the proposed action is send to the process to be controlled. The global search algorithm will generate new classifier rules based on the rule strengths acquired during the entire process. The fuzzy behavior is created by an activation sequence of mutually collaborating fuzzy rules. The entire knowledge base is build up by a cooperation of competing multiple fuzzy rules. The second approach evolves a population of knowledge bases rather than individual fuzzy rules. The disadvantage is the increased complexity of search space and additional computational burden especially for online learning.

5. Global Optimization Techniques

Global optimization problems fall within the broader class of nonlinear programming (NLP). In the following sections we discuss three popular nature's heuristic methods namely genetic algorithms, simulated annealing and tabu search, which have been successfully applied to hybrid systems to find approximate solutions [111].

5.1 Genetic Algorithms

Genetic algorithms are adaptive methods, which may be used to solve search and optimization problems, based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "Survival of the Fittest", first clearly stated by Charles Darwin in "The Origin of Species". By mimicking this process, genetic algorithms are able to "evolve" solutions to real world problems, if they have been suitably encoded [31].

GAs deal with parameters of finite length, which are coded using a finite alphabet, rather than directly manipulating the parameters themselves. This means that the search is unconstrained neither by the continuity of the function under investigation, nor the existence of

a derivative function. Figure 16 depicts the functional block diagram of a GA and the various aspects are discussed below. It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as genes) are joined together to form a string of values (known as a chromosome). A gene (also referred to a feature, character or detector) refers to a specific attribute that is encoded in the chromosome. The particular values the genes can take are called its alleles. The position of the gene in the chromosome is its locus.

Encoding issues deal with representing a solution in a chromosome and unfortunately, no one technique works best for all problems. A fitness function must be devised for each problem to be solved. Given a particular chromosome, the fitness function returns a single numerical fitness or figure of merit, which will determine the ability of the individual, which that chromosome represents. Reproduction is the second critical attribute of GAs where two individuals selected from the population are allowed to mate to produce offspring, which will comprise the next generation. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation.

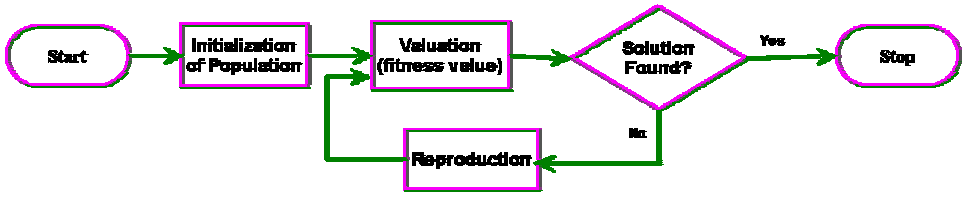


Figure 16. Flow chart of genetic algorithm iteration

There are many ways in which crossover can be implemented. In a single point crossover (Figure 17), two chromosome strings are cut at some randomly chosen position, to produce two "head" segments, and two "tail" segments. The tail segments are then swapped over to produce two new full-length chromosomes. Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0.

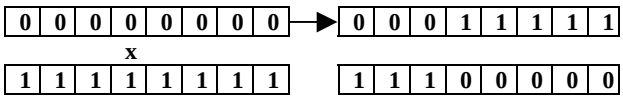


Figure 17. A single point crossover operation

Another genetic operation is mutation, which is an asexual operation that only operates on one individual. It randomly alters each gene with a small probability. Traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and helps ensure that no point in the search space has a zero probability of being examined. If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum.

Selection is the survival of the fittest within GAs. It determines which individuals are to survive to the next generation. The selection phase consists of three parts. The first part involves determination of the individual's fitness by the fitness function. A fitness function must be devised for each problem; given a particular chromosome, the fitness function returns a single numerical fitness value, which is proportional to the ability, or utility, of the individual represented by that chromosome. For many problems, deciding upon the fitness function is very straightforward, for example, for a function optimization search; the fitness is simply the value of the function. However, there are cases where there may be many performance measures to optimize; in a bridge design task, we may need to optimize a combination of strength/weight ratio, span, width, maximum load, cost, and construction time. Ideally, the fitness function should be smooth and regular so that chromosomes with reasonable fitness are close in the search space, to chromosomes with slightly better fitness. However, it is not always possible to construct such ideal fitness functions. For any type of search to be successful the fitness function must not have too many local maxima nor a very isolated global maximum. The second part involves converting the fitness function into an expected value followed by the last part where the expected value is then converted to a discrete number of offspring. Some of the commonly used selection techniques are roulette wheel, genetic annealing and stochastic universal sampling.

Even though GAs have found themselves applicable in solving many real world problems, sometimes, the amount of time spend to achieve a successful application is too long. In extreme cases, GAs may perform poorly when hampered by interference from mutation and genetic drift. Some of the techniques to overcome such difficulties are discussed below:

1. Sharing allows the simultaneous exploration of many local maximas, rather than a single peak. This algorithm does not emphasize having copies of one superior individual. It maintains copies of many varying relatively superior schemata. The number of these copied schemata depends upon their relative values.

2. Crowding slows the convergence rate of genetic algorithms. Each (produced) child replaces one individual in the population. The probability of replacement increases when the individual contains similar alleles. The most effective technique combines a small generation gap (indicating the fraction of the population that has been changed with each generation) with a small amount of crowding which decreases duplicate genetic material (leading to a diverse population).

3. Elitism is used to keep the best individual identified thus far, maintaining the most superior individuals in the population. The most superior schemata are maintained since the maximal element replaces the minimal element during each generation. This method increases the rate of convergence of the genetic algorithm.

4. Parallel genetic algorithms perform standard genetic algorithm selection and mating on various sub-populations to achieve global optimization in the entire population. These sub-populations are concurrently evaluated, as separate genetic algorithms, and propagated through generations. Individuals have a small probability of migration between sub-populations. Superior schemata will dominate all (or many) of the sub-populations if they are superior to schema in other sub-populations.

Sharing and crowding may be used to decrease the amount of duplicate schemata in the population, thus preventing premature convergence. Elitism may be incorporated to keep the most superior individuals (and superior schemata) within the population. Parallel genetic algorithms [32] use the convergence of its sub-populations to superior schemata(s) of low order and, then, propagation of individuals between sub-populations to achieve global optimization.

5.2 Simulated Annealing

As its name implies, the Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. The algorithm [33] was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. SA's major advantage over other methods is its ability to avoid becoming trapped at local minima. Figure 18 shows a Flowchart of a SA iteration.

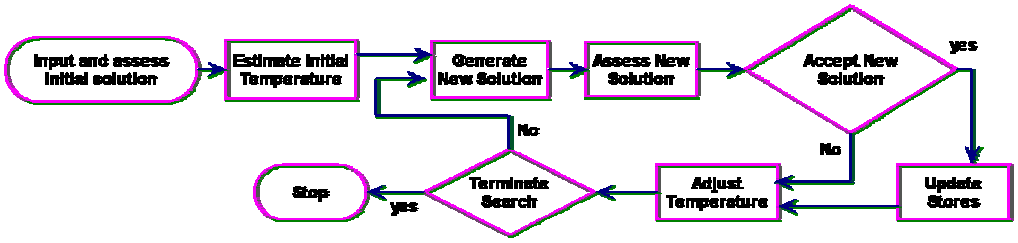


Figure 18. Flow chart of simulated annealing iteration

The algorithm employs a random search, which not only accepts changes that decrease objective function *f*, but also some changes that increase it. The latter are accepted with a probability $p = \exp\left(-\frac{\delta f}{T}\right)$, where δf is the increase in objective function, and *f* and *T* are control parameters. Several SAs have been developed with annealing schedule inversely linear in time (Fast SA), exponential function of time (Very Fast SA), etc. We explain an SA algorithm, which is exponentially faster than the Very Fast SA [34] whose annealing schedule is given by $T(k) = \frac{T_0}{\exp(e^k)}$, where T_0 is the initial temperature, $T(k)$ is the temperature we wish to approach to zero for *k=1,2,.....*

Representing the generation function of the simulated annealing algorithm as:

$$g_k(Z) = \prod_{i=1}^{D} g_k(z_i) = \prod_{i=1}^{D} \frac{1}{2(|z_i| + \frac{1}{\ln(1 / T_i(k))}) \ln(1 + \ln(1 / T_i(k)))} \tag{11}$$

where $T_i(k)$ is the temperature in dimension *i* at time *k* and D is the dimension of the state space.

The generation probability will be given by

$$G_k(Z) = \int_{-1}^{z_1} \int_{-1}^{z_2} .....\int_{-1}^{z_D} g_k(Z) dz_1 dz_2 .... dz_D = \prod_{i=1}^{D} G_{ki}(z_i) \tag{12}$$

$$\text{where } G_{ki}(z_i) = \frac{1}{2} + \frac{\operatorname{sgn}(z_i) \ln(1 + |z_i| \ln(1 / T_i(k)))}{2 \ln(1 + \ln(1 / T_i(k)))} \tag{13}$$

It is straightforward to prove that for an annealing schedule

$$T_i(k) = T_{0i} \exp(-\exp(b_i k^{1/D})) \tag{14}$$

a global minimum (statistically) can be obtained. That is,

$$\sum_{k=k_o}^{\infty} g_k = \infty \tag{15}$$

where b_i>0 is a constant parameter and k_0 is a sufficiently large constant to satisfy (15).

5.3 Tabu Search

Tabu Search (TS) is a meta-strategy for guiding known heuristics to overcome local optimality and has now become an established optimization approach that is rapidly spreading to many new fields [35]. The method can be viewed as an iterative technique which explores a set of problem solutions, denoted by *X*, by repeatedly making moves from one solution *s* to another solution *s′* located in the neighborhood *N(s)* of *s*. These moves are performed with the aim of efficiently reaching optimal solution by the evaluation of some objective function f(s) to be minimized. In the sequel we sketch the basic ingredients of TS [112].

Initialization
Generate a random solution s in X;
iteration counter (iteration)= 0;
bestiteration =0 [for tracking the best iteration];
Set tabu list (T=0);
Initialize the aspiration function A(f);
Begin loop
while *(iteration-bestiteration < nmax) do**;***
iteration= iteration+1;
generate a set of feasible V solutions* $s_i = s \oplus m_i$ *,such that* $m_i \notin T$
or $f(s_i) < A(f(s))$*;*
choose the best solution s′ in V;*
update tabu list T and A(f);
If *(f(s′) < f(s*))* ***then***
s=s′*
bestiteration=iteration;
s=s′
end loop *when number of iterations has reached nmax.*

Figure 19. Tabu search algorithm

Let us define the notion of neighborhood N(s) for each solution *s* in X. By definition N(s) is a set of solutions in X reachable from *s* via a slight modification *m*.

$$N(s) = \{s' \in X \mid s' = s \oplus m, m \in M\} \quad (1.16)$$

Where M contains all possible modifications and $s' = s \oplus m$ is obtained by applying modification *m* to *s*. TS starts from an initial solution randomly generated in X and moves repeatedly from a solution to a neighbor. At each step of the procedure, a subset *V** of the neighborhood of the current solution *s* is generated and the local optimization problem $min\{f(x) \mid x \in V^* \subseteq N(s)\}$ is solved. In order to escape from local minima, the idea is to move to the best neighbor *s′* in *V** even if *f(s′) > f(s)*. Following a steepest descent / mildest ascent approach, a move may result in a best possible improvement (or a least possible deterioration) of the objective function value. Without additional control, however, such a process can cause a locally optimal solution to be re-visited immediately after moving to a neighbor, or in a future stage of the search process. To prevent the search from endlessly cycling between the same solutions, a tabu list T is introduced. This list keeps track of the reverses of the last $|T|$ modifications that have been done enacted during the search process. A move from *s* to *s′* will be considered tabu if it is performed via a modification contained in T. However the concept of tabu list sometimes appears to be restrictive. Since only parts of the neighborhood are explored, it might be worth returning after a while to a solution visited previously to search in another direction. An aspiration function *A* deals precisely with the rigidity of the tabu list. It permits the tabu status of a move to be dropped under certain favorable circumstances.

We define an aspiration level *A(z)* for each value *z* of the objective function. Then a tabu move from *s* to *s′* is permitted if *f(s')<A(f(s))*. Initially *A(z)=z* , for all possible values of *z=f(s)*. This aspiration function is updated as follows whenever we move from *s* to *s′*.

A(f(s))=min(A(f(s)),f(s')) and *A(f(s'))=min(A(f(s')),f(s))* (17)

The above shows that the reverse move from *s′* to *s* is considered in the updating of *A* even though it was not done explicitly. Generally the process is stopped as soon as a given number of iterations have been performed without improving the best solution obtained.

6. Neuro-Fuzzy Systems

A Fuzzy Inference System (FIS) [59] can utilize human expertise by storing its essential components in rule base and database, and perform fuzzy reasoning to infer the overall output value. The derivation of *if-then* rules and corresponding membership functions depends heavily on the *a priori* knowledge about the system under consideration. However there is no systematic way to transform experiences of knowledge of human experts to the knowledge base of a FIS. There is also a need for adaptability or some learning algorithms to produce outputs within the required error rate. On the other hand, ANN learning mechanism does not rely on human expertise. Due to the homogenous structure of ANN, it is hard to extract structured knowledge from either the weights or the configuration of the ANN. The weights of the ANN represent the coefficients of the hyper-plane that partition the input space into two regions with different output values. If we can visualize this hyper-plane structure from the training data then the subsequent learning procedures in an ANN can be reduced. However, in reality, the a priori knowledge is usually obtained from human experts and it is most appropriate to express the knowledge as a set of fuzzy if-then rules and it is not possible to encode into an ANN. Table 3 summarizes the comparison of FIS and ANN.

Table 3. Complementary features of neural network and fuzzy inference system

Artificial Neural Network	Fuzzy Inference System
Black box	Interpretable
Learning from scratch	Making use of linguistic knowledge

To a large extent, the drawbacks pertaining to these two approaches seem complementary. Therefore it seems natural to consider building an integrated system combining the concepts of FIS and ANN modeling. A common way to apply a learning algorithm to a fuzzy system is to represent it in a special ANN like architecture. However the conventional ANN learning algorithms (gradient descent) cannot be applied directly to such a system as the functions used in the inference process are usually non differentiable. This problem can be tackled by using differentiable functions in the inference system or by not using the standard neural learning algorithm.

In the following sections we briefly discuss the different integrated neuro-fuzzy models that make use of the complementarities of ANNs and FIS to form a better system.

6.1 Adaptive Network Based Fuzzy Inference System

Adaptive Network Based Fuzzy Inference System (ANFIS) [4-5] [52] [55] perhaps the first integrated hybrid neuro-fuzzy model. ANFIS structure is capable of implementing the following fuzzy rules:

Type 1: The overall output is the weighted average of each rule's crisp output induced by the rule's firing strength (the product or minimum of the degrees of match with the premise part) and output membership functions (Figure 20). The output membership functions used in this scheme must be monotonically non-decreasing [36].

Type 3: According to Takagi and Sugeno's fuzzy if-then rules, the output of each rule is a linear combination of input variables plus a constant term, and the final output is the weighted average of each rules output (Figure 21) [37].

Figures 22 and 23 illustrate the layered architectures of ANFIS-type 3 and type 1 implementation respectively. The architecture of the ANFIS - type 3 is composed of 5 layers and the functionality of each layer is as follows:

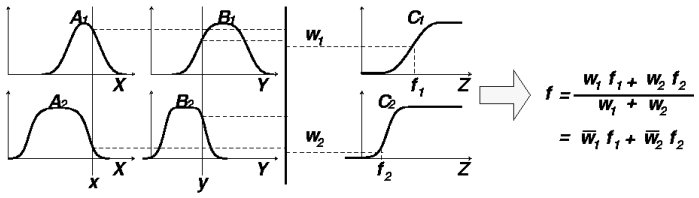


Figure 20. Type 1 fuzzy reasoning

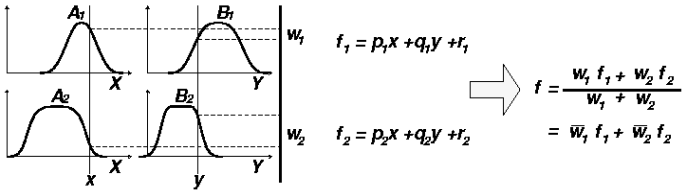


Figure 21. Type 3 fuzzy reasoning

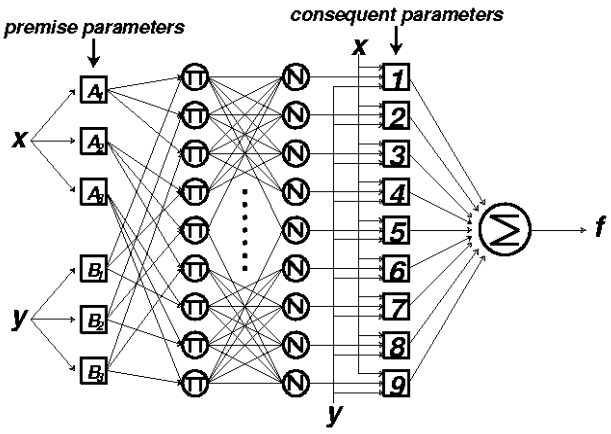


Figure.22. Architecture of the ANFIS - type 3 fuzzy rule implementation

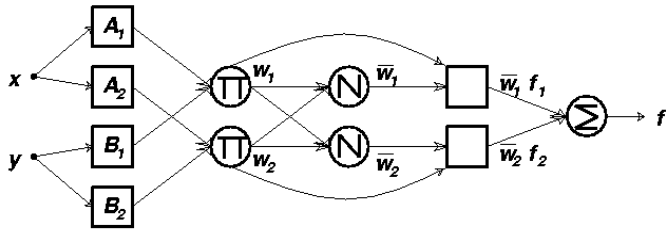


Figure 23. Architecture of ANFIS- type 1 implementation

Layer-1 Every node in this layer has a node function $O_i^1 = \mu_{A_i}(x)$. Usually the node function is the bell shaped or the Gaussian with maximum equal to 1 and minimum equal to 0. Parameters in this layer are referred to premise parameters.

Layer-2 Every node in this layer multiplies the incoming signals and sends the product out. Each node output represents the firing strength of a rule. $w_i = \mu_{A_i}(x)\ x\ \mu_{B_i}(y), i = 1,2.....$.

Layer-3 Every i-th node in this layer calculates the ratio of the i-th rule's firing strength to the sum of all rules firing strength. $\overline{w_i} = \frac{w_i}{w_1 + w_2}, i = 1,2....$.

Layer-4 Every node i in this layer is with a node function $O_1^4 = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i)$, where $\overline{w_i}$ is the output of layer3, and $\{p_i, q_i, r_i\}$ is the parameter set. Parameters in this layer will be referred to as consequent parameters.

Layer-5 The single node in this layer labeled Σ computes the overall output as the summation of all incoming signals: $O_1^5 = Overall\ output = \sum_i \overline{w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$.

ANFIS makes use of a mixture of back propagation to learn the membership functions and least mean square estimation to determine the coefficients of the linear combinations in the rule's conclusions. A step in the learning procedure has two parts: In the first part the input patterns are propagated, and the optimal conclusion parameters are estimated by an iterative least mean square procedure, while the antecedent parameters (membership functions) are assumed to be fixed for the current cycle through the training set. In the second part the patterns are propagated again, and in this epoch, back propagation is used to modify the antecedent parameters, while the conclusion parameters remain fixed. This procedure is then iterated.

6.1.1 ANFIS hybrid learning rule

ANFIS uses a hybrid learning rule with a combination of gradient descent and least squares estimate [5]. Assuming a single output ANFIS represented by $output = F(\vec{I}, S)$, where I is the set of input variables and S is the set of parameters, if there exist a function H such that the composite function H ○ F is linear in some of the elements of S, then these elements can be identified by the least squares method. More formally the parameter set S can be decomposed into two sets:

$S = S_1 \oplus S_2$ (where $\oplus$ represents direct sum), (18)

such that H ○ F is linear in the elements of S_2 . Then upon applying H to equation (18), we have:

$H(output) = H \circ F(\vec{I}, S)$ (19)

which is linear in the elements of S_2 . Now the given values of elements of S_1 , we can plug *P* training data into equation (19), and obtain a matrix equation:

AX=B (X = unknown vector whose elements are parameters in S_2) (20)

If $|S_2|$ =M, (M= number of linear parameters) then the dimensions of *A, X* and *B* are $P \times M$, $M \times 1$ and $P \times 1$ respectively. Since P is always greater than M, there is no exact solution to equation (20). Instead a Least Square Estimate (LSE) of *X,* X^*, is sought to minimize the squared error $\|AX - B\|^2$. X^* is computed using the pseudo-inverse of X:

$X^* = (A^T A)^{-1} A^T B$ (21)

where A^T is the transpose of A and $(A^T A)^{-1} A^T$ is the pseudo-inverse of A where $A^T A$ is non-singular. Due to computational complexity, in ANFIS a sequential method is deployed as follows:

Let the *i-th* row vector of matrix A defined in equation (20) be a_i^T and *i-th* element of matrix B defined be b_i^T , then X can be calculated iteratively using the following sequential formulae:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0,1,....., P-1 \end{aligned} \tag{22}$$

Where S_i is often called the covariance matrix and the least squares estimate X^* is equal to X_P. The initial condition to bootstrap equation (22) are $X_O=0$ and $S_O=\gamma I$, where γ is a positive large number and I is the identity matrix of dimension M x M. For a multi output ANFIS equation (1.22) is still applicable except the $output = F(\vec{I}, S)$ will become a column vector. Each epoch of this hybrid learning procedure is composed of a forward pass and a backward pass. In the forward pass, we supply input data and functional signals go forward to calculate each node output until the matrices A and B in equation (20) are obtained, and the parameters in S_2 are identified by the sequential least squares formulae given in (22). After identifying parameters in S_2, the functional signals keep going forward till the error measure is calculated. In the backward pass, the error rates propagate from the output layer to the input layers, and the parameters in S_1 are updated by the gradient method given in (23). The gradient

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{23}$$

where α is the generic parameter, η is a learning rate and E the error measure. For given fixed values of parameters in S_1, the parameters in S_2 thus found are guaranteed to be the global optimum point in the S_2 parameter space due to the choice of the squared error measure.

The procedure mentioned above is mainly for offline learning version. However the procedure can be modified for an online version by formulating the squared error measure as a weighted version that gives higher weighting factors to more recent data pairs. This amounts to the addition of a forgetting factor λ to the original sequential formulae (22).

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= \frac{1}{\lambda}\left[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}} \right] \quad i = 0,1,....., P-1 \end{aligned} \tag{24}$$

The value of λ is between 0 and 1. The smaller the λ is, faster the effects of old data decay. But a smaller λ sometimes causes numerical instability and should be avoided.

6.2 NEFCON (Neuro-Fuzzy Controller)

The learning algorithm defined for NEFCON [6] [40] is able to learn fuzzy sets as well as fuzzy rules. This method can be considered as an extension to GARIC (Generalized Approximate Reasoning based Intelligent Control) [38] [69-71] that also use reinforcement learning but need a previously defined rule base. Figure 24 illustrates the basic NEFCON architecture with 2 inputs and five fuzzy rules. The NEFCON-Model is based on a generic fuzzy perceptron [39]. The inner nodes R_1, ..., R_5 represent the rules, the nodes ξ_1, ξ_2, and η the input and output values, and μ_r, V_r the fuzzy sets describing the antecedents and consequents . In contrast to ANNs the connections in NEFCON are weighted with fuzzy sets instead of real numbers. Rules with the same antecedent use so-called shared weights, which are represented by ellipses drawn around the connections as shown in the figure. They ensure the integrity of the rule base. The node R_1 for example represents the rule:

R_1: If ξ_1 is $A_1^{(1)}$ and ξ_2 is $A_1^{(2)}$ then η is B_1. (25)

The optimal state of a system (plant) can be described by a vector of state variable values. This means that the plant has reached the desired state if all of its state variables have reached their value by this vector. If we use membership functions to describe how good the current value is compared with the desired value, it is possible to derive fuzzy error that characterizes the performance of our natural fuzzy controller.

The knowledge base of the fuzzy system is implicitly given by the network structure. The input units assume the task of fuzzification interface, the inference logic is represented by the propagation functions, and the output unit is the defuzzification interface. The learning process of the NEFCON model can be divided into two main phases. The first phase is designed to learn an initial rule base, if no prior knowledge about the system is available. Furthermore it can be used to complete a manually defined rule base. The second phase

optimizes the rules by shifting or modifying the fuzzy sets of the rules. Both phases use a fuzzy error E, which describes the quality of the current system state, to learn or to optimize the rule base. In the third stage the fuzzy sets are modified according to one of the fuzzy error backpropagation algorithms as detailed below.

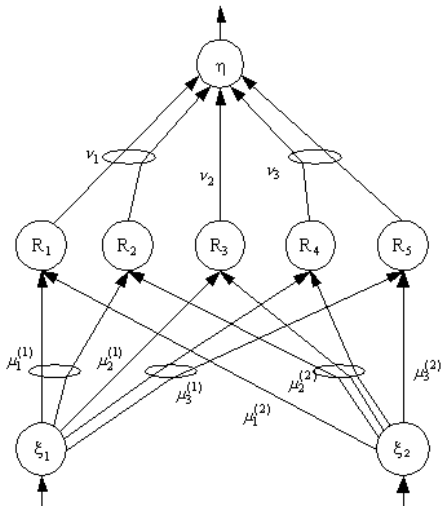


Figure 24. A NEFCON system with 2 input variables and five rules.

The extended fuzzy error E of a NEFCON system is given by:

$$E(x_1, ..., x_n) = \mathrm{sgn}(\eta_{opt})\, e(x_1, ..., x_n) \tag{26}$$

where $(x_1, ..., x_n)$ is the current input , e is the fuzzy error, and $\mathrm{sgn}(\eta_{opt})$ is the sign of the (otherwise unknown) optimal control output.

6.2.1 Fuzzy Error Back Propagation Algorithm (FEBP)

Let S be a process with *n* state variables $\xi_i \in X_i\ (i \in \{1,....n\})$ and one control variable $\eta \in Y$, with *k* rule units $R_1,\ldots R_k$. The algorithm for adapting the membership functions is defined by the following steps that have to be repeated until a certain criterion is met.

1. Compute the output O_η using the current state, apply O_η to S, and determine its new state.
2. Determine the extended fuzzy error E from the new state of S.
3. Determine for each rule unit R_r its contribution t_r to the output value O_η, and calculate the fuzzy rule error E_{R_r} for each rule unit R_r ($r \in \{1,\ldots,k\}$): $E_{R_r} = O_{R_r} \cdot E \cdot \mathrm{sgn}(t_r)$
4. Determine the changes in the parameters of the membership functions: $V_{j_r} = (j_r \in \{1,..., q\}, r \in \{1,...., k\}) O_{R_r} \cdot E \cdot \mathrm{sgn}(t_r)$: $\Delta d_{j_r} = \sigma \cdot E_{R_r} \cdot (e_{j_r} - d_{j_r})$, with a learning rate $\sigma > 0$. Apply these changes to the fuzzy sets V_{j_r}, such that the constraints $\Psi(V_{j_r})$ are met.
5. Determine the changes in the parameters of the membership functions $\mu_{j_r}^{(i)} (i \in \{1,..., n\}, j_r \in \{1,...., p_i\}, r \in \{1,...., k\})$:

$$\Delta a_{j_r}^{(i)} = -\sigma \cdot E_{R_r} \cdot (b_{j_r}^{(i)} - a_{j_r}^{(i)}), \tag{27}$$

$$\Delta c_{j_r}^{(i)} = \sigma \cdot E_{R_r} \cdot (c_{j_r}^{(i)} - b_{j_r}^{(i)}). \tag{28}$$

Apply these changes to the fuzzy sets $\mu_{j_r}^{(i)}$ such that the constraints $\psi(\mu_{j_r}^{(i)})$ are met. The learning process can be stopped when the fuzzy error E is smaller than a certain value for a certain number of cycles. However the above algorithm relies on monotonic membership functions in the rules consequents. For common triangular membership functions (or other forms) to be included in the consequents the algorithm is modified as follows.

Let S be a process with *n* state variables $\xi_i \in X_i\ (i \in \{1,....n\})$ and one control variable $\eta \in Y$, with *k* rule units $R_1,\ldots R_k$. For each input variable ξ_i there are p_i triangular fuzzy sets $\mu_j^{(i)} (j \in \{1,...., p_i\})$ and for the output variable η there are q triangular fuzzy sets $v_j\ (j \in \{1,...., q\})$. For the fuzzy sets there are constraints Ψ that determine whether a modification can be carried out. The FEBP for adapting the membership functions for k rule units is defined by the following steps that have to be repeated until a certain criterion is met.

1. Compute the output O_η using the current state, apply O_η to S, and determine its new state.
2. Determine the extended fuzzy error E from the new state of S.

3. Determine for each rule unit R_r its contribution t_r to the output value O_η, and calculate the fuzzy rule error E_{R_r} for each rule unit R_r ($r \in \{1,\ldots,k\}$): $E_{R_r} = O_{R_r} \cdot E \cdot \mathrm{sgn}(t_r)$

4. Determine the modifications for the consequent fuzzy sets V_{j_r}:

$$\begin{aligned} \Delta b_{j_r} &= \sigma \cdot or \cdot E, \\ \Delta a_{j_r} &= \Delta b_{j_r}, \\ \Delta c_{j_r} &= \Delta b_{j_r}. \end{aligned} \tag{29}$$

where learning rate $\sigma > 0$. Apply these changes to the fuzzy sets V_{j_r}, such that the constraints $\Psi(V_{j_r})$ are met.

5. Determine the modifications for the antecedent fuzzy sets $\mu_{j_r}^{(i)}(i \in \{1,\ldots,n\}, j_r \in \{1,\ldots,p_i\}, r \in \{1,\ldots,k\})$:

$$\Delta b_{j_r}^{(i)} = \sigma \cdot E_{R_r} (\xi_i - b_{j_r}^{(i)}) \cdot (c_{j_r}^{(i)} - a_{j_r}^{(i)}), \tag{30}$$

$$\Delta a_{j_r}^{(i)} = -\sigma \cdot E_{R_r} (b_{j_r}^{(i)} - a_{j_r}^{(i)}) + \Delta b_{j_r}^{(i)}, \tag{31}$$

$$\Delta c_{j_r}^{(i)} = \sigma \cdot E_{R_r} \cdot (c_{j_r}^{(i)} - b_{j_r}^{(i)}) + \Delta b_{j_r}^{(i)}. \tag{32}$$

Apply these changes to the fuzzy sets $\mu_{j_r}^{(i)}$ such that the constraints $\psi(\mu_{j_r}^{(i)})$ are met. Both the learning algorithms presented above are to adapt the membership functions, and can be applied only if there is already a rule base of fuzzy rules. The idea of the learning algorithm is identical: increase the influence of a rule if its action goes in the right direction (rewarding), and decrease its influence if a rule behaves counter productively (punishing). If there is no absolutely no knowledge about initial membership function, a uniform fuzzy partition of the variables should be used.

6.2.2 NEFCON – Algorithm for learning rule base

Two methods are available for learning the rule base. I*ncremental rule learning* is used when the correct out put is not known and rules are created based on estimated output values. As the learning progresses more rules are added according to the requirement. For *decremental rule learning,* initially rules are created due to fuzzy partitions of process variables and unnecessary rules are eliminated in the course of learning. Decremental rule learning is less efficient compared to incremental approach. However it can be applied to unknown processes without difficulty, and there is no need to know or to guess an optimal output value. For a process with n state variables which are partitioned by p_i fuzzy sets each and a control variable with q linguistic terms, we can create at most N linguistic rules, where $N = q\prod_{i=1}^{n} p_i$. However if we create N rule units, then the rule base is of course inconsistent. After the learning process the rules must be consistent and limited to k rules, and $k \le \frac{N}{q}$ must hold.

The learning algorithm is divided into three stages. During the first stage all rule units are deleted which provide an output value with a sign different from the optimal output value. In the second stage subsets of rules with identical antecedents are considered. From all rule subsets whose antecedents match the current state (i.e. have a degree of fulfillment greater than zero), one rule is selected at random to contribute to the NEFCON output. The resulting error is ascribed only to the rules selected. At the end of this stage, from each rule subset only the rule with the smallest error is kept, and all other rules are deleted from the NEFCON system.

6.2.2.a Decremental rule learning

Let R denote the set of all rule units and Ant (R_r) denote the antecedent and Con (R_r) the consequent of a fuzzy rule corresponding to the rule unit R_r Let S be a process with *n* state variables $\xi_i \in X_i$ ($i \in \{1,\ldots n\}$) which are partitioned by p_i fuzzy sets each and one control variable $\eta \in Y$, partitioned by q fuzzy sets. Let there also be $N = q\prod_{i=1}^{n} p_i$ initial rules with

$$(\forall R, R^\circ \in R)((Ant(R) = Ant(R^\circ) \wedge Con(R) = Con(R^\circ)) \Rightarrow R = R^\circ). \tag{33}$$

1) For each rule unit R_r a counter C_r (initialized to 0) is defined $r \in \{1,\ldots,N\}$. For a fixed number m_1 of iterations the following steps are carried out.

- Determine the current NEFCON output O_η using the current state of S.
- For each rule R_r determine its contribution t_r to the overall output O_η, $r \in \{1,\ldots,N\}$.
- Determine $\mathrm{sgn}(\eta_{opt})$ for the current input values.

- Delete each rule unit R_r with $sgn(t_r) \neq sgn(\eta_{opt})$ and update the value of N.
- Increment the counters C_r for all R_r with $o_{R_r} > 0$.
- Apply O_η to S and determine the new input values.

2) For each R_r a counter Z_r (initialized to 0) is defined. For a fixed number m_2 of iterations the following steps are carried out.

- From all subsets, $R_j = \{R_r \mid Ant(R_r) = Ant(R_s)(r, s \in \{1,...., N\})\} \subseteq R$, one rule unit R_{r_j} is selected arbitrarily.
- Determine the current NEFCON output O_η using only the rule units selected and the current state of S.
- Apply O_η to S and determine the new input values.
- For each rule R_{r_j} determine its contribution t_{r_j} to the overall output O_η, $r_j \in \{1,...., N\})$.
- Determine sgn (η_{opt}) for the current input values.
- Add $\left| O_{R_{r_j}} \cdot E \right|$ to the counter Z_r of each selected rule unit R_{r_j}.
- For all selected rule units R_{r_j} with $O_{R_{r_j}} > 0$, C_{r_j} is incremented.

3) Delete all rule units R_{s_j} for all subsets R_j from the network for which there is a rule unit $R_{s_j} \in R_j$ with $Z_{r_j} < Z_{s_j}$, and delete all rule units R_r with $C_r \leq \beta.(m_1 + m_2)$, $0 \leq \beta < 1$, from the network, and update the value of N.

4) Apply appropriate FEBP algorithm to the NEFCON system with k=N remaining rule units.

6.2.2.b Incremental rule learning

The incremental learning rule creates a rule base from scratch by adding rule after rule. It does this by first classifying an input vector, i.e. finding that membership function for each variable that yields the highest membership value for the respective input value. A rule antecedent is thus formed. Then the algorithm tries to guess the output value by deriving it from the current fuzzy error. In the second phase the rule base is optimized by changing the consequent to an adjacent membership function if necessary. To start the algorithm, an initial fuzzy partition must be assigned to each variable. Let S be a process with *n* state variables $\xi_i \in X_i$ ($i \in \{1,\ldots n\}$) which are partitioned by p_i fuzzy sets each and one control variable $\eta \in [y_{min}, y_{max}]$, partitioned by q fuzzy sets. Let there also be initial k' predefined rule units, where k' may be zero. The following steps give the incremental rule-learning algorithm.

1) For a fixed number m_1 of iterations the following steps are carried out.

- For the current input vector $(x_1,\ldots,x_n)$ find those fuzzy sets $\hat{\mu}^{(1)}, \ldots \hat{\mu}^{(n)}$ for which

$$(\forall_i, j)(\hat{\mu}^{(1)}(x_i) \geq \mu_j^{(i)}(x_i)) \quad \text{holds.} \tag{34}$$

- If there is no rule with the antecedent

If ξ_i is $\hat{\mu}^{(1)}$ and ....and ξ_n is $\hat{\mu}^{(n)}$, then the fuzzy set $\hat{v}$ such that $(\forall j)(\hat{v}(o) \geq v_j(o))$ holds, where the heuristic output value o is determined by

$$o = \begin{cases} m + |E| \cdot (y_{max} - m) \ \ if \ \ E \geq 0 \\ m - |E| \cdot (m - y_{min}) \ \ if \ \ E < 0 \end{cases} \qquad m = \frac{y_{max} + y_{min}}{2} \tag{35}$$

- Enter the rule

If ξ_i is $\hat{\mu}^{(1)}$ and ξ_n is $\hat{\mu}^{(n)}$, then η is $\hat{v}$; into the NEFCON system.

2) For a fixed number m_2 of iterations the following steps are carried out.

- Propagate the current input vector through the NEFCON system and estimate for each rule unit R_r determine its contribution t_r to the overall output O_η. Compute the desired contribution to the system output value by

$$t_r^* = t_r + \sigma \cdot o_r \cdot E, \tag{36}$$

where E is the extended fuzzy error, and $\sigma > 0$ is a learning rate.

- For each rule unit each rule unit R_r, determine the new output membership function $\hat{v}_r$ such that

$$(\forall_i)(\hat{v}_r \ (t_r^*) \geq v_j(t_r^*)) \tag{37}$$

and change the consequent of the rule unit R_r accordingly.

- Delete all rules that have not been used in more than in p% of all propagations.
- Apply appropriate FEBP algorithm to the NEFCON system.

The idea of the rule-learning algorithm is to try out the existing rules and to assess them. Rule units that do not pass this test are eliminated from the network. In the second phase the algorithm has to choose the rule with the lowest error value from each subset that consists of rules with identical antecedents, and delete all other rules of each sub set. To obtain a good rule base it must be ensured that the state space of the process is sufficiently covered during the learning process, i.e. m_1 *and* m_2 must be large enough. Due to the complexity of the calculations required, the decremental learning rule can only be used, if there are only a few input variables with not too many fuzzy sets. For larger systems the incremental learning rule will be optimal. Two measures can make use of partial knowledge for the learning process and reduce the cost of the learning procedure, especially for decremental rule learning:

- If for a given antecedent a consequent is known, then an equivalent rule unit is put into the NEFCON system, and the learning algorithm is not allowed to remove it. Other rules with this antecedent are not created.
- If for some states only a subset of all possible rules need be considered, then only these rules are put into the network.

A similar situation is applicable to Incremental learning rule, where by if prior knowledge is available, then rule learning does not need to start from scratch.

6.3 NEFCLASS (Neuro-Fuzzy Classification)

NEFCLASS [7] [40-41] is used to derive fuzzy rules from a set of data that can be separated in different crisp classes. The rule base of a NEFCLASS system approximates an (unknown) function φ that represents the classification problem and maps an input pattern x to its class C_i :

$$\varphi : R^n \rightarrow \{0,1\}^m, \varphi(x) = (c_1,...., c_{m)}, \text{with } c_i = \begin{cases} 1 \ \text{if} \ x \in C_i \\ 0 \ \text{otherwise.} \end{cases} \tag{38}$$

Because of the propagation procedures used in NEFCLASS the rule base actually does not approximate φ but a function $\varphi^\circ : R^n \rightarrow \{0,1\}^m$. We obtain φ (x) from the equality φ (x) =φ (φ (x)), where φ reflects the interpretation of the classification result obtained from a NEFCLASS system. Figure 25 illustrates the NEFCLASS system that maps patterns with two features into two distinct classes by using five linguistic rules. The NEFCLASS very much resemble the NEFCON system except the slight variation in the learning algorithm and the interpretation of the rules.

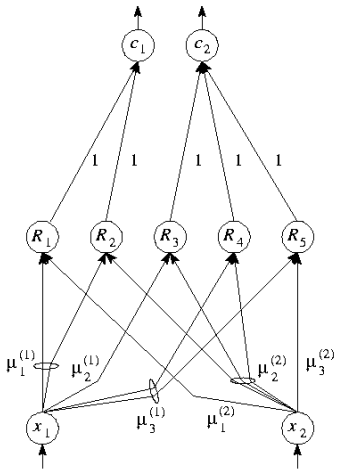


Figure 25. A NEFCLASS system with 2 input variables, 5 rules and 2 output classes

As in NEFCON system in NEFCLASS identical linguistic values of an input variable are represented by the same fuzzy set. As classification is the primary task of NEFCLASS, there should be two rules with identical antecedents and each rule unit must be connected to only one output unit. The weights between rule layer and the output layer only connect the units. A NEFCLASS system can be built from partial knowledge about the patterns, and can then be refined by learning, or it can be created from scratch by learning. A

user must define a number of initial fuzzy sets that partition the domains of the input features, and specify a value for k, i.e. the maximum number of rule nodes that may be created in the hidden layer.

NEFCLASS makes use of triangular membership functions and the learning algorithm of the membership functions uses an error measure that tells whether the degree of fulfillment of a rule has to be higher or lower. This information is used to change the input fuzzy sets. Being a classification system, we are not much interested in the exact output values. In addition we take a winner-takes-all interpretation for the output, and we are mainly interested in the correct classification result.

The incremental rule learning in NEFCLASS is much less expensive than decremental rule learning in NEFCON. It is possible to build up a rule base in a single sweep through the training set. Even for higher dimensional problems, the rule base is completed after at most three cycles. Compared to ANNs, NEFCLASS uses a much simpler learning strategy. There is no vector quantization involved in finding the rules (clusters, and there is no gradient information needed to train the membership functions. Some other advantages are interpretability, possibility of initialization (incorporating prior knowledge) and its simplicity.

6.4 NEFPROX (Neuro-Fuzzy Function Approximation)

NEFPROX system is based on plain supervised learning (fixed learning problem) and it is used for function approximation. It is a modified version of the NEFCON model without the reinforcement learning. The advantage of Neuro-Fuzzy models is that we can incorporate prior knowledge; where as conventional ANNs have to learn from scratch.

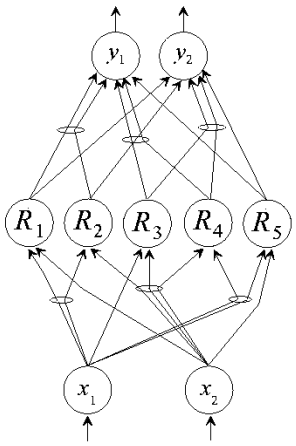


Figure 26. Architecture of the NEFPROX system

NEFPROX is very much similar to NEFCON and NEFCLASS except the fact that NEFCON have only a single output node, and NEFCLASS systems do not use membership functions on the conclusion side. We can initialize the NEFPROX system if we already know suitable rules or else the system is capable to incrementally learn all rules. NEFPROX architecture is as shown in Figure 26.

While ANFIS is capable to implement only Sugeno models with differentiable functions, NEFPROX can learn common Mamdani type of fuzzy system from data. Further NEFPROX is much faster compared to ANFIS to yield results. However ANFIS yields better approximation results.

6.5 Fuzzy Inference Environment Software with Tuning (FINEST)

FINEST [10] [56] is designed to tune the fuzzy inference itself. FINEST is capable of two kinds of tuning process, the tuning of fuzzy predicates, combination functions and the tuning of an implication function. The three important features of the system are:

- The generalized modus ponens is improved in the following four ways (1) Aggregation operators that have synergy and cancellation nature (2) A parameterized implication function (3) A combination function which can reduce fuzziness (4) Backward chaining based on generalized modus ponens.

- Aggregation operators with synergy and cancellation nature are defined using some parameters, indicating the strength of the synergic affect, the area influenced by the effect, etc., and the tuning mechanism is designed to tune also these parameters. In the same way the tuning mechanism can also tune the implication function and combination function.

- The software environment and the algorithms are designed for carrying out forward and backward chaining based on the improved generalized modus ponens and for tuning various parameters of a system.

6.5.1 FINEST Tuning Algorithm

FINEST make use of a back propagation algorithm for the fine-tuning of the parameters. Figure 27 shows the layered architecture of FINEST and the calculation process of the fuzzy inference. The input values (x_i) are the facts and the output value (y) is the conclusion of the fuzzy inference.

Consider we have the following fuzzy rules and facts:

$$\textit{Rule } i: \textit{If } x_1 \textit{ is } A_{i1} \textit{ and } \ldots\ldots \textit{and } x_n \textit{ is } A_{in} \textit{ then } y \textit{ is } B_i.$$

$$\textit{Fact } j: x_j \textit{ is } A_j^{\circ} \ (j = 1,\ldots,n, i = 1,\ldots,m).$$

The calculation procedure in each layer is as follows:

Layer-1: Converse truth value qualification for condition *j* of rule *i,*

$$\tau_{Aij}(a_{ij}) = \sup_{\mu_{Aij}(x_j) = a_{ij}} \mu_{A_j^{\circ}}(x_j). \tag{39}$$

Layer-2: Aggregation of the truth values of the conditions of Rule *i*,

$$\tau_{Ai}(a_i) = \sup_{and_i(a_{i1},\ldots,a_{in}) = a_i} \{\tau_{Ai}(a_{i1}) \wedge \cdots \wedge \tau_{Ain}(a_{in})\}. \tag{40}$$

Layer-3: Deduction of the conclusion from Rule *i*,

$$\mu_{B_I^{\circ}}(y) = \sup_{a_i} \{\tau_{Ai}(a_i) \wedge I_i(a_i, \mu_{B_i}(y))\} \tag{41}$$

Layer-4: Combination of the conclusions derived from all the rules,

$$\mu_{B_I^{\circ}}(y) = comb(\mu_{B_1^{\circ}}(y) \wedge \cdots \wedge \mu_{B_m^{\circ}}(y)). \tag{42}$$

In the above equations τ_{Aij}, τ_{Ai}, B_i° respectively represent the truth value of the condition " x_i *is* A_{ij} "of rule *i*, the truth value of the condition part of rule *i*, and the conclusion derived from rule *i.* Besides, the function and_i, I_i and *comb* respectively represent the function characterizing the aggregation operator of rule *i*, the implication function of rule *i*, and the global combination function. The functions and_i, I_i, *comb* and membership functions of each fuzzy predicate are defined with some parameters.

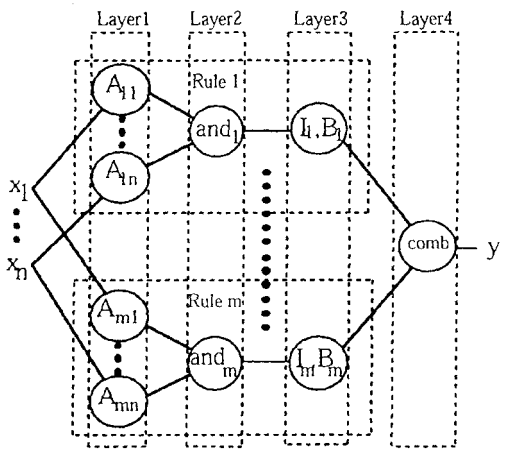


Figure 27. FINEST architecture.

Back-propagation method is used to tune the network parameters. All data inside the network are treated as fuzzy sets. The actual output of the network B° is given by,

$$B^{\circ} = b^{\circ(1)} / y^{(1)} + \cdots + b^{\circ(p)} / y^{(p)} \tag{43}$$

The teaching signal *T* is given by,

$$T = t^{(1)} / y^{(1)} + \cdots + t^{(p)} / y^{(p)} \tag{44}$$

Error function E is expressed by,

$$E = \frac{1}{2} \sum_{h=1}^{p} (t^{(h)} - b'^{(h)})^2. \quad (45)$$

It is possible to tune any parameter, which appears in the nodes of the network representing the calculation process of the fuzzy data if the derivative function with respect to the parameters is given. So it is very much important to parameterize the inference procedure. Some details on parameterized aggregation functions, parameterized implication functions and parameterized combination functions are given in [10].

6.6 Self Constructing Neural Fuzzy Inference Network (SONFIN)

SONFIN [9] is inherently a modified Takagi-Sugeno-Kang (TSK) type fuzzy rule based model possessing ANN learning ability. Fuzzy rules are created and adapted as online learning proceeds via a simultaneous structure and parameter identification. In the structure identification of the precondition part, the input space is partitioned in a flexible way according to an aligned clustering based algorithm. As to the structure identification of the consequent part, only a singleton value selected by a clustering method is assigned to each rule initially. Afterwards, some additional significant terms (input variables) selected via a projection-based correlation measure for each rule will be added to the consequent part (forming a linear equation of input variables) incrementally as learning proceeds. For parameter identification, the consequent parameters are tuned optimally by either Least Mean Squares [LMS] or Recursive Least Squares [RLS] algorithms and the precondition parameters are tuned by back propagation algorithm. To enhance knowledge representation ability of SONFIN, a linear transformation for each input variable can be incorporated into the network so that much fewer rules are needed or higher accuracy can be achieved. Proper linear transformations are also learned dynamically in the parameter identification phase of SONFIN. Figure 28 illustrates the 6-layer structure of SONFIN.

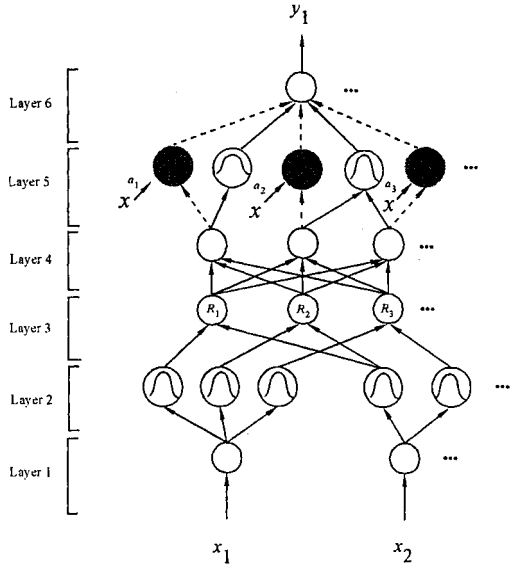


Figure 28. Six layered architecture of SONFIN

The six-layered SONFIN structure realizes a fuzzy model of the following form:

$$Rule\ i:\ If\ x_1\ is\ A_{i1}\ and\ .....and\ x_n\ is\ A_{in}\ then\ y\ is\ m_{0i} + a_{ji}x_j + ...$$

where A_{ij} is a fuzzy set, m_{0i} is the center of a symmetric membership function on y, and a_{ji} is a consequent parameter. At any node, if the net input is defined as:

$$net - input = f[u_1^{(k)}, u_2^{(k)}, ..., u_p^{(k)};\quad w_1^{(k)}, w_2^{(k)}, ..., w_p^{(k)}],$$

where $u_1^{(k)}, u_2^{(k)}, ..., u_p^{(k)}$ are inputs to the node and $w_1^{(k)}, w_2^{(k)}, ..., w_p^{(k)}$ are the associated link weights. The superscript in the above equation relates to the layer number. A second action of each node is to output an activation value of its net-input,

$$output = o_i^k = a(net - input) = a(f)$$

where a(.) denotes the activation function. Detailed functioning of the each layers is as follows:

Layer 1: Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly.

$$f = u_i^1 \qquad and \qquad a^{(1)} = f. \quad (46)$$

Layer-2: Each node in this layer corresponds to one linguistic label (small, big, etc.) of one of the input variables in Layer 1. Membership functions shall be Gaussian, triangular, trapezoidal etc. With a Gaussian membership function, the operation performed in this layer is given by,

$$f[u_{ij}^{(2)}] = -\frac{[u_i^{(2)} - m_{ij}]^2}{\sigma_{ij}^2} \tag{47}$$

and $a^{(2)}(f) = e^f$

where m_{ij} and σ_{ij}^2 are respectively, the center (or mean) and the width (or variance) of the Gaussian membership function of the *jth* term of the ith input variable x_i. The link weight of this layer can be interpreted as m_{ij} .

Layer-3: A node in this layer represents one fuzzy logic rule and performs precondition matching of a rule. Here we use the following AND operation for each layer 3 node

$$\begin{aligned} f[u_i^{(3)}] &= \prod_i u_i^{(3)} \\ &= e^{-[D_i(x-m_i)]^T[D_i(x-m_i)]} \end{aligned} \tag{48}$$

and $a^{(3)}(f) = f$

where

$D_i = diag(1/\sigma_{i1}, 1/\sigma_{i2}, ..., 1/\sigma_{in})$ *and*

$m_i = (m_{i1}, m_{i2}, ..., m_{in})^T$.

The link weight in layer-3 $w_1^{(3)}$ is unity. The output of a layer-3 node represents the firing strength of the corresponding fuzzy rule.

Layer-4: The number of nodes in this layer is equal to that in layer 3 and the firing strength calculated in layer-3 is normalized in this layer by

$$f[u_i^{(4)}] = \sum_i u_i^{(4)} \tag{49}$$

and $a^{(4)}(f) = \frac{u_i^{(4)}}{f}$.

Like layer-3, the link weight in this layer $w_1^{(4)}$ is unity too.

Layer-5: This layer is called the consequent layer. Two different types are used in this layer as denoted by shaded and blank circles in figure 20. The node denoted by a blank circle is the essential node representing a fuzzy set of the output variable. Only the center of each Gaussian membership function is delivered to the next layer for the local mean of maximum (LMOM) defuzzification operation and the width is used for output clustering only. Different nodes in layer-4 may be connected to a same blank node in layer-5, meaning that the same fuzzy set is specified for different rules. The function of the blank node is given by,

$$f = \sum_i u_i^{(5)} \tag{50}$$

and $a^{(5)}(f) = f.a_{0i}$.

where $a_{0i} = m_{0i}$, the center of the Gaussian membership function. As to the shade mode, it is generated only when necessary. Each node in layer-4 has its own corresponding shaded node in layer-5. One of the inputs to a shaded node is the output delivered from layer-4 and the other possible inputs (terms) are the input variables from layer-1. The function of the shaded node is given by,

$$f = \sum_j a_{ji} x_j \tag{51}$$

and $a^{(5)}(f) = f \cdot u_i^{(5)}$.

Combining functions of the shaded and blank nodes, the obtain the function of the whole layer as,

$$a^{(5)}(f) = \left(\sum_j a_{ji} x_j + a_{0i} \right) u_i^{(5)}. \quad (52)$$

Layer-6: Each node in this layer corresponds to one output variable. The node integrates all the actions recommended by layer-5 and acts as a defuzzifier with

$$fu_i^{(6)} = \sum_i u_i^{(6)} \quad (53)$$

and $a^{(6)}(f) = f$.

6.6.1 Learning Algorithm for SONFIN

Learning progresses concurrently in two stages for the construction of SONFIN. The *structure* learning includes both the precondition and consequent structure identification of a fuzzy *if-then* rule. The parameter learning is based on supervised learning algorithms, the parameters of the linear equations in the consequent parts are adjusted by either LMS or RLS algorithms and the parameters in the precondition part are adjusted by the backpropagation algorithm. SONFIN can be used for normal operation at anytime during the learning process without repeated training on the input-output pattern when online operation is required. In SONFIN rule base is dynamically created as the learning progresses by performing the following learning processes:

- **Input-output space partitioning.**

The way the input space is partitioned determines the number of rules extracted from the training data as well as the number of fuzzy sets on the universal of discourse of each input variable. For each incoming pattern x the strength a rule is fired can be interpreted as the degree the incoming pattern belongs to the corresponding cluster. The firing strength derived from layer-3 nodes can be directly used as degree measure.

$$F^i(x) = e^{-[D_i(x-m_i)]^T [D_i(x-m_i)]} \quad (54)$$

where $[D_i(x - m_i)]^T [D_i(x - m_i)]$ is, in fact, the distance between x and the center of cluster *i*. For generation of a new fuzzy rule, if x(t) is the incoming pattern find

$$J = \arg \max_{1 \le j \le c(t)} F^j(x) \quad (55)$$

where c(t) is the number of rules existing at time t. If $F^J \le \overline{F}(t)$, then a new rule is generated where $\overline{F}(t) \in (0,1)$ is a pre-specified threshold that decays during the learning process. The center and width of the corresponding membership functions (of the newly formed fuzzy rules) are assigned according to the first-neighbor heuristic. For each rule generated, the next step is to decompose the multidimensional membership function to corresponding 1-D membership function for each input variable. For the output space partitioning ($\overline{F}_{out}$), almost a similar measure is adopted. Performance of SONFIN can be enhanced by incorporating a transformation matrix R into the structure, which accommodates all the *a priori* knowledge of the data set.

- **Construction of fuzzy rule base.**

Generation of new input cluster corresponds to the generation of a new fuzzy rule, with its precondition part constructed by the learning algorithm in process. At the same time we have to decide the consequent part of the generated rule. This is done using a algorithm based on the fact that different preconditions of rules may be mapped to the same consequent fuzzy set. Since only the center of each output membership function is used for defuzzification, the consequent part of each rule may simply be regarded as a singleton. Compared to the general fuzzy based rule models with singleton output where each rule has its own singleton value, fewer parameters are needed in the consequent part of the SONFIN, especially for complicated systems with a large number of rules.

- **Optimal consequent structure identification**

TSK model can model a sophisticated system with a few rules [45]. In SONFIN, instead of using the linear combination of all input variables as the consequent part, only the most significant input variables are used as the consequent terms of the SONFIN. The significant terms will be chosen and added to the network incrementally any time when the parameter learning cannot improve the network output accuracy anymore during the online learning process. The consequent structure identification scheme in SONFIN is a kind of node growing method in ANNs [9]. When the effect of the parameter learning diminished (output error is not decreasing), additional terms are added to the consequent part.

- **Parameter identification.**

After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Parameter learning is performed on the whole network after structure learning, no matter whether the nodes (links) are newly added or are existent originally. Backpropagation algorithm is used for this supervised learning. For each training data set, starting at the input nodes, a forward pass is used to compute

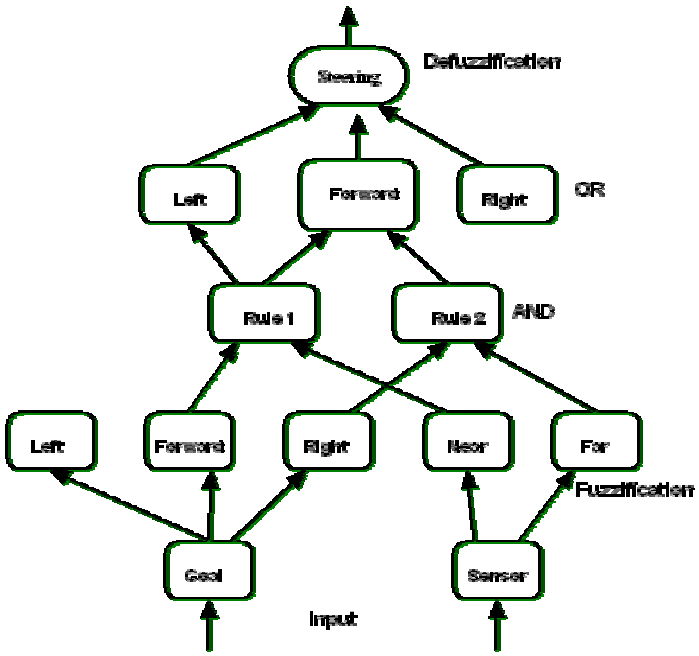
the activity levels of all the nodes in the network to obtain the current output. Then starting at the output nodes, a backward pass is used to compute $\frac{\partial E}{\partial w}$ for all the hidden nodes of all the layers. If w is the adjustable parameter in a node, the general rule used is:

$$W(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right), \tag{56}$$

where η is the learning rate.

6.7 FUN [FUzzy Net]

In FUN [8] inorder to enable an unequivocal translation of fuzzy rules and membership functions into the network, special neurons have been defined, which, through their activation functions, can evaluate logic expressions. The network consists of an input, an output and three hidden layers. The neurons of each layer have different activation functions representing the different stages in the calculation of fuzzy inference. The activation function can be individually chosen for problems. The network is initialized with a fuzzy rule base and the corresponding membership functions. Figure 29 illustrates the FUN network. The input variables are stored in the input neurons. The neurons in the first hidden layer contain the membership functions and this performs a fuzzification of the input values. In the second hidden layer, the conjunctions (fuzzy-AND) are calculated. Membership functions of the output variables are stored in the third hidden layer. Their activation function is a fuzzy-OR. Finally the output neurons contain the output variables and have a defuzzification activation function.



Rule: *IF (Goal IS forward AND Sensor IS near) OR (goal IS right AND sensor IS far) THEN steering: = forward*

Figure 29. Architecture of the FUN showing the implementation of a sample rule

6.7.1 FUN: Learning strategies

The rules and the membership functions are used to construct an initial FUN network. The rule base can then be optimized by changing the structure of the net or the data in the neurons. To learn the rules, the connections between the rules and the fuzzy values are changed. To learn the membership functions, the data of the nodes in the first and three hidden layers are changed. FUN can be trained with the standard neural network training strategies such as reinforcement or supervised learning.

- **Learning of the rules**

The rules are represented in the net through the connections between the layers. The learning of the rules is implemented as a stochastic search in the rule space: a randomly chosen connection is changed and the new network performance is verified with a cost function. If the performance is worse, the change is undone, otherwise it is kept and some other changes are tested, until the desired output is achieved. As the learning algorithm should preserve the semantic of the rules, it has to be controlled in such a way that no two values of the same variable appear in the same rule. This is achieved by swapping connections between the values of the same variable.

- **Learning of membership functions**

FUN makes use of triangular membership functions with three membership function descriptors (MFDs). It is presumed that the membership functions are normalised, i.e. they always have a constant height of 1. The learning algorithm is a combination of gradient

descent and a stochastic search. A maximum change in a random direction is initially assigned to all MFDs. In a random fashion one MFD of one linguistic variable is selected, and the network performance is tested with this MFD altered according to the allowable change for this MFD. If the network performs better according to the given cost function, the new value is accepted and next time another change is tried in the same direction. Contrary if the network performs worse, the change is reversed. To guarantee convergence, the changes are reduced after each training step and shrink asymptotically towards zero according to the learning rate.

6.8 Evolving Fuzzy Neural Networks (EfuNNs and mEfuNNs)

EfuNNs [49] and dmEfuNNs [50] are based on the ECOS (Evolving COnnectionist Systems) [48] framework for adaptive intelligent systems formed as a result of evolution and incremental, hybrid (supervised/unsupervised), online learning [46-47]. They can accommodate new input data, including new features, new classes, and etc. through local element tuning.

EFuNNs are FuNN (Fuzzy Neural Networks) [51] structures that evolve according to the ECOS principles. FuNN is connectionist feedforward architecture with five layers of neurons and four layers of connections (Figure 30). The first layer (input layer) passes the data to the second layer, which calculates the fuzzy membership degrees to which the input values belong to predefined fuzzy membership functions. The third layer of neurons represents associations between the input and the output variables, fuzzy rules. The fourth layer calculates the degrees to which output membership functions are matched by the input data, and the fifth layer does defuzzification and calculates exact values for the output variables. The number of neurons in each of the layers can potentially change during operation through growing or shrinking. The number of connections is also modifiable through learning with forgetting, zeroing, pruning and other operations. FuNN uses triangular membership functions and can be modified (width and center) through learning. Several training algorithms (independent and combinations) including modified BP algorithm and GAs were tested successfully in FuNN.

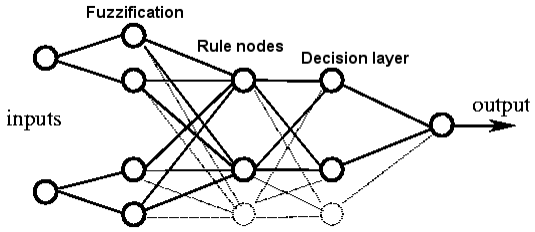


Figure 30. Five layered architecture of FuNN

Genetic Algorithms were also used to evolve an optimal topology for a FuNN [46]. Each chromosome was encoded with a sequence of values, one for each input, which represents the number of membership functions to be attached for each input, number of membership functions attached to the outputs, number of rule nodes in the structure, sequence of values whether or not a particular input feature is to be used by the network. To evaluate each individual, the chromosome is decoded into a FuNN structure and is trained for a specific number of epochs with the training data set. When training is completed, based on the error fitness value is calculated. Initial population was generated; with mutation and one point crossover, and tournament selection was used to pick the best-fit individuals for the next generation. After the set number of generations, the most-fit individual of the final generation was decoded into a FuNN and subsequently trained using the bootstrapped training algorithm across the entire data set. Genetically evolved FuNN performed better than the manually designed FuNN.

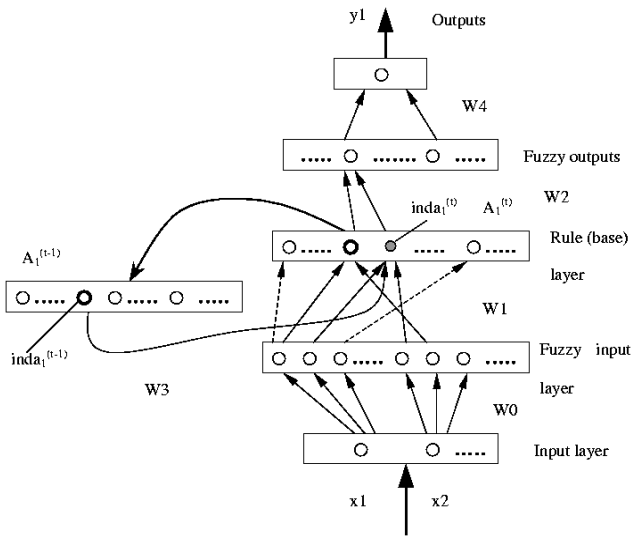


Figure 31. Architecture of EFuNN

In EfuNNs (Figure 31) all nodes are created during learning. The nodes representing membership functions (MF) can be modified during learning. As in FuNN, each input variable is represented here by a group of spatially arranged neurons to represent a fuzzy quantization

of this variable. For example, three neurons can be used to represent "small", "medium" and "large" fuzzy values of the variable. Different membership functions can be attached to these neurons (triangular, Gaussian, etc.). New neurons can evolve in this layer if, for a given input vector, the corresponding variable value does not belong to any of the existing MF to a degree greater than a membership threshold. A new fuzzy input neuron, or an input neuron, can be created during the adaptation phase of an EFuNN. A new Rule Node *rn* is created and its input and output connection weights are set as follows:

W1 (*rn*)=EX; W2 (*rn*) = TE, (57)

where TE is the fuzzy output vector for the current fuzzy input vector EX. In case of "one-of-n" EFuNNs, the maximum activation of a rule node is propagated to the next level. Saturated linear functions are used as activation functions of the fuzzy output neurons. In case of "many-of-n" mode, all the activation values of rule nodes that are above an activation threshold of Ahtr are propagated further in the connectionist structure.

6.8.1 EFuNN learning algorithm

EFuNN evolving algorithm is given as a procedure of consecutive steps:

1) Initialize an EFuNN structure with a maximum number of neurons and no (or zero value) connections. Initial connections may be set through inserting fuzzy rules in a FuNN structure. FuNNs allow for insertion of fuzzy rules as an initialization procedure thus allowing for prior information to be used prior to the evolving process. If initially there are no rule nodes connected to the fuzzy input and fuzzy output neurons, then create the first node *rn*=1 to represent the first example EX=x_1 and set its input W1 (*rn*) and output W2 (*rn*) connection weights as follows:

   Create a new rule node *rn* to represent an example EX: W1(*rn*)=EX; W2(*rn*) = TE, where TE is the fuzzy output vector for the (fuzzy) example EX.

2) WHILE <there are examples> DO
   Enter the current, example xi, EX being the fuzzy input vector (the vector of the degrees to which the input values belong to the input membership functions). If there are new variables that appear in this example and have not been used in previous examples, create new input and/or output nodes with their corresponding membership functions.

3) Find the normalized fuzzy similarity between the new example EX (fuzzy input vector) and the already stored patterns in the case nodes j=1,2,...,*rn*:

   Dj= sum (abs (EX - W1(j))/ 2) / sum (W1(j))

4) Find the activation of the rule (case) nodes j, j=1:rn Here radial basis activation function, or a saturated linear one, can be used on the Dj input values i.e. A1 (j) = *radbas* (Dj), or A1 (j) = *satlin* (1 - Dj).

5) Update the local parameters defined for the rule nodes, e.g. age, average activation as pre-defined.

6) Find all case nodes j with an activation value A1(j) above a sensitivity threshold S*thr*.

   If there is no such case node, then <Create a new rule node> using the procedure from step 1.
   ELSE

7) Find the rule node *inda1* that has the maximum activation value *(maxa1).*

8) *(a)* In case of "one-of-n" EFuNNs, propagate the activation *maxa1* of the rule node *inda1* to the fuzzy output neurons. Saturated linear functions are used as activation functions of the fuzzy output neurons:

   A2 = *satlin* (A1(*inda1*) * W2)

   (b) *in case of "many-of-n" mode, only the activation values of case nodes that are above an activation threshold of A*thr *are propagated to the next neuronal layer.*

9) Find the winning fuzzy output neuron *inda2* and its activation *maxa2*.

10) Find the desired winning fuzzy output neuron *indt2* and its value *maxt2*.

11) Calculate the fuzzy output error vector: Err=A2 - TE.

12) IF (*inda2* is different from *indt2*) or (abs (Err (*inda2*)) > Err*thr*) <Create a new rule node>
    ELSE

13) Update: (a) the input, and (b) the output connections of rule node k=*inda1* as follows:
    (a) Dist=EX-W1(k); W1(k)=W1(k) + lr1. Dist, where lr1 is the learning rate for the first layer;
    (b) W2(k) = W2 (k) + lr2. Err. *maxa1*, where lr2 is the learning rate for the second layer.

14) Prune rule nodes j and their connections that satisfy the following fuzzy pruning rule to a pre-defined level representing the current need of pruning:

IF *(node (j) is OLD) and (average activation A1av(j) is LOW) and (the density of the neighboring area of neurons is HIGH or MODERATE) and (the sum of the incoming or outgoing connection weights is LOW) and (the neuron is NOT associated with the corresponding "yes" class output nodes (for classification tasks only))* THEN *the probability of pruning node (j) is HIGH.*

The above pruning rule is fuzzy and it requires that the fuzzy concepts as OLD, HIGH, etc. are defined in advance.

16) END of the while loop and the algorithm

17) Repeat steps 2-16 for a second presentation of the same input data.

6.8.2 Dynamic Evolving Fuzzy Neural Networks (dmEfuNNs)

Dynamic Evolving Fuzzy Neural Networks (dmEfuNN) model is developed with the idea that not just the winning rule node's activation is propagated but a group of rule nodes is dynamically selected for every new input vector and their activation values are used to calculate the dynamical parameters of the output function. While EfuNN and FuNN make use of the weighted fuzzy rules of Zadeh-Mamdani type, dmEfuNN uses the Takagi-Sugeno fuzzy rules.

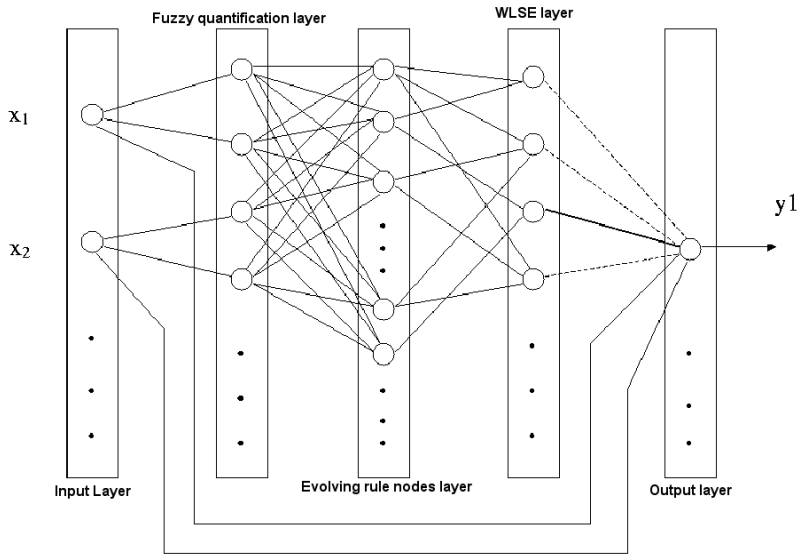


Figure 32. The structure of dmEFuNN

Takagi-Sugeno fuzzy rules are implemented in a dmEFuNN recall (before its adaptation). The method is explained here for n inputs and one output system:

1) For each input vector X_{di} , find m rule nodes r_{i1} , r_{i2} , ..., r_{im} with the closest fuzzy normalised local distance Ds to the fuzzy input vector X_{dif} , calculated as:

$$Ds(X_{dif} , r_{ij})= sum (abs(X_{dif} -- W1(r_{ij})) / sum (X_{dif} + W1(r_{ij}))$$

where $W1(r_{ij})$ is the weight matrix of the connections from the fuzzy quantification layer to the rule nodes layer.

2) Use the formula, $A = [a_0 a_1 a_2 ... a_n]^T = (X^T Z X)^{-1} X^T ZY$, to calculate the weighted least- square estimator (WLSE), where Z is weighting matrix for placing heavier emphasis on more important data. In this case, Z is a diagonal matrix and $z_{jj} = 1 - Ds (X_{dif} , r_{ij})$.

3) The output of dmEFuNN is calculated as:

$$y = X_{dif} A$$

The first, second and third layers of dmEFuNN have exactly the same structures and functions as the EFuNN. The fourth layer, the fuzzy inference layer, selects m rule nodes from the third layer which have the closest fuzzy normalised local distance to the fuzzy input vector, and then, a Takagi-Sugeno fuzzy rule will be formed using the weighted least square estimator. The last layer calculates the output of dmEFuNN. Please refer to Figure 32 for details about the dmEFuNN architecture.

The number m of activated nodes used to calculate the output values for a dmEFuNN is not less than the number of the input nodes plus one. The dmEFuNNs calculate the output using a Takagi-Sugeno fuzzy rule, which is formed by weighted least square estimator. A first-order Takagi-Sugeno fuzzy rule has a consequent part that is a linear function:

$$y = a_0 + a_1 x_1 + a_2 x_2 + ... + a_n x_n.$$

where, $x_1, x_2, \ldots, x_n$ are n elements of the input vector.

On the basis of the theory of Least Square Estimator, the experiments have to be performed to obtain a training data set composed of m data pairs in order to get the n + 1 coefficients $a = [a_0, a_1, a_2, \ldots, a_n]$. It is necessary that $m \geq n+1$ to identify uniquely the unknown vector *a*. If m = n +1 and the matrices composed of m data pairs is a nonsingular, then a can be obtained by solving the linear equations. In the case of dmEFuNNs, these m data pairs come from the m activated nodes that means the number m is not less than the number of input elements n + 1.

Like the EFuNNs, the dmEFuNNs can be used for both off-line learning and online learning thus optimising global generalization error, or a local generalization error. In dmEFuNNs, for a new input vector (for which the output vector is not known), a subspace consisted of m rule nodes are found and a first order Takagi-Sugeno fuzzy rule is formed using the least square estimator method. This rule is used to calculate the dmEFuNN output value. In this way a dmEFuNN acts as a universal function approximator using *m* linear functions in a small *m*-dimensional node subspace. The accuracy of approximation depends on the size of the node subspaces, the smaller the subspace is, the higher the accuracy. It means that if there are sufficient training data vectors and sufficient rule nodes are created, a satisfying accuracy can be obtained.

7 Conclusions

In this paper we have attempted to classify the state-of-the art hybrid intelligent systems and summarized the current standing of the research work done particularly in the field of fused neuro-fuzzy systems. Neuro-fuzzy systems could be seen as heuristics to determine parameters of a fuzzy inference system by processing training data with a learning algorithm [66].

For evaluating the different NF models we used the chaotic time series given by Mackey-Glass differential equation [65]:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1\,x\,(t). \qquad (58)$$

We used the value x(t-18), x(t-12), x(t-6), x(t) to predict x(t+6). Fourth order Runge-Kutta method was used to generate 1000 data series. The time step used in the method is 0.1 and initial condition were x (0)=1.2, τ=17, x (t)=0 for t<0 and x(t) is thus derived for $0 \leq t \leq 2000$. We used a Pentium II, 450 MHZ platform for training the NF models. Offline training was done using 500 data sets (t= 118 to 617), by giving 4 inputs x (t-18), x (t-12), x (t-6), x (t) and we attempted to predict the output for x (t+6). NF models were tested with another 500 data sets (t=618 to 1117) and the results are plotted in Figure 33 and 34. We also trained an ANN using backpropagation algorithm to compare the performance with NF systems. Figure 35 shows the results of the predicted values against the desired values for an ANN. Table 4 gives a comparative performance of NF systems and ANN. Figure 36 illustrates Root Mean Square Error (RMSE) of NF systems and ANN for time series prediction problem[79].

Table 4. Comparative performance of NF systems and ANN.

System	Epochs	CPU Time(s)	RMSE (test)
ANFIS	75	165	0.0017
NEFPROX	216	*75	0.0332
EfuNN	1	30	0.014
dmEFuNN	1	†52	0.0042
SONFIN	1	-	0.018
ANN	1580	165	0.0047

(*Sun Ultra platform † Platform unknown)

Among NF models ANFIS has the lowest RMSE error and NEPROX the highest. This is probably due to Takagi-Sugeno rules implementation in ANFIS compared to the Mamdani-type fuzzy system in NEFPROX. However NEFPROX outperformed ANFIS in terms of computational time.

In ANFIS the adaptation (learning) process is only concerned with parameter level adaptation within fixed structures. For large-scale problems, it will be too much complicated to determine the optimal premise-consequent structures, rule numbers etc. The structure of ANFIS ensures that each linguistic term is represented by only one fuzzy set. However the learning procedure of ANFIS does not provide the means to apply constraints that restrict the kind of modifications applied to the membership functions. When using Gaussian membership functions, operationally ANFIS can be compared with a radial basis function network.

ANFIS [5] implements a Takagi-Sugeno fuzzy system and applies a mixture of back propagation and least mean squares procedure to train the system. However the adaptation (learning) process is only concerned with parameter level adaptation within fixed structures. For large-scale problems, it will be too much complicated to determine the optimal premise-consequent structures, rule numbers etc. The structure of ANFIS ensures that each linguistic term is represented by only one fuzzy set. However the learning procedure of ANFIS does not provide the means to apply constraints that restrict the kind of modifications applied to the membership functions. When using Gaussian membership functions, operationally ANFIS can be compared with a radial basis function network.

FUN system [8] is initialized by specifying a fixed number of rules and a fixed number of initial fuzzy sets for each variable and there after uses a stochastic procedure (learning technique) that randomly changes parameters of membership functions and connections within the network structure. The learning process is driven by a cost function, which is evaluated after the random modification. If the

modification resulted in an improved performance the modification is kept, otherwise it is undone. Since no formal neural network learning technique is used it is questionable to call FUN a neuro-fuzzy system.

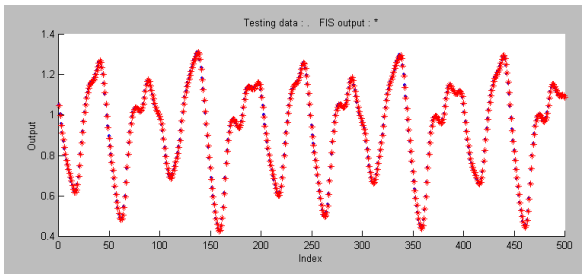


Figure 33. Mackey-Glass time series prediction using ANFIS

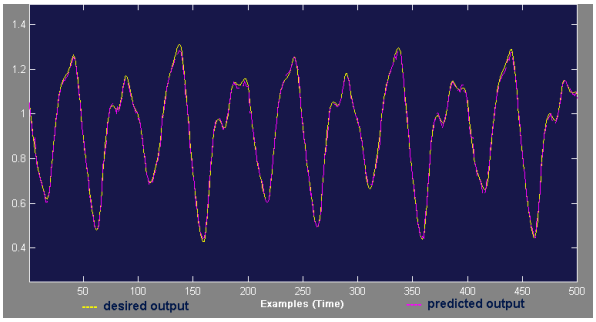


Figure 34. Mackey-Glass time series prediction using EfuNN.

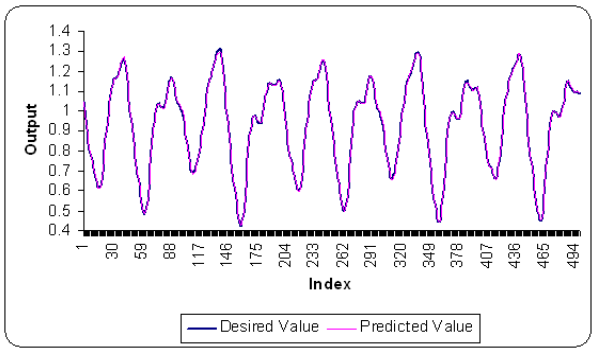


Figure 35. Mackey-Glass time series prediction using ANN

NEFCON [6] make use of an incremental or decremental rule learning algorithm for learning the rule base (structure learning) and a fuzzy backpropagation algorithm for learning the fuzzy sets (parameter learning). NEFCON system is capable of incorporating prior knowledge as well as learning from scratch. However the performance of the system will very much depend on heuristic factors like learning rate, error measure etc. NEFCLASS [7] is mainly useful for classification assignments and it uses a supervised algorithm. NEFCLASS does not use membership functions in the rules consequents. Compared to ANNs the advantages of NEFCLASS are interpretability, ability to incorporate prior knowledge and simplicity. NEFPROX [42] is a modified version of NEFCON with more than 1 output and based on supervised learning algorithm and basically used as a function approximator. Since NEFPROX uses a neural network learning technique too much learning could lead to poor generalization.

FINEST [10] provides a mechanism based on the improved generalized modus ponens for fine tuning of fuzzy predicates & combination functions and tuning of an implication function. FINEST uses a gradient descent technique to tune the various parameters. Parameterization of the inference procedure is very much essential for proper application of the tuning algorithm. In many cases the parameterized function is designed only to cover the mathematical part leaving the semantics part ignored very often. FINEST presents a novel technique for parameterizing aggregation function, implication function and combination functions.

SONFIN [9] learns from scratch and the rules are created and adapted as online learning proceeds via simultaneous structure and parameter identification. For the structure identification of the precondition part, the input space is partitioned based on a clustering algorithm and for the structure identification of the consequent part only a singleton value selected by the clustering method is assigned to each rule initially. Further, as the learning proceeds, rules will get modified incrementally. Precondition parameters are tuned by back propagation algorithm and consequent parameters by least mean squares or recursive least squares algorithms.

FuNN [51] uses a fixed structure and based on a multiple iteration learning procedure based on gradient descent, backpropagation algorithm. EfuNN [49] implements a Mamdani type of fuzzy rule base and is based on a dynamic structure (creating and deleting

strategy), single rule inference, established on the winner-takes all rule for the rule node activation, with a one-pass training, instance based learning and reasoning. dmEFuNN [50] is an improved version of the EfuNN using several (m) of the highest activated rule nodes instead of one and implements a Takagi-Sugeno fuzzy system. The rule node aggregation is achieved by a C-means clustering algorithm.

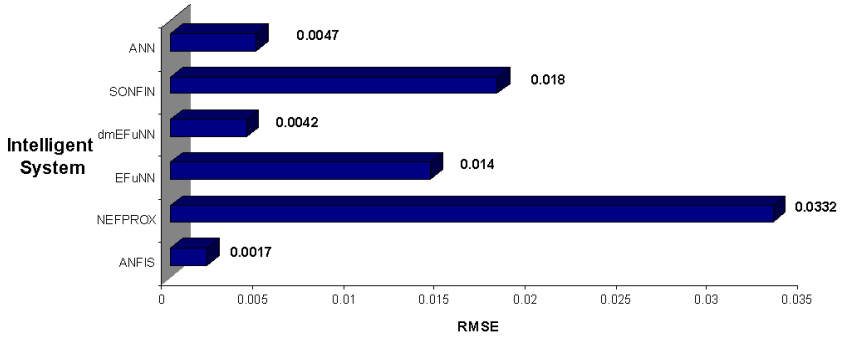


Figure 36. Graph showing RMSE error of NF systems and ANN for time series prediction problem.

8 Further Research Directions

From the success in integrating neural network and fuzzy logic and knowing their strengths and weaknesses, we can construct better neuro-fuzzy systems to mitigate the limitations and take advantage of the opportunities to produce more powerful hybrids than those that could be built with stand alone systems. As a guideline, for neuro-fuzzy systems to be in the top of the ladder, some of the major requirements are fast learning (memory based - efficient storage and retrieval capacities), on-line adaptability (accommodating new features like inputs, outputs, nodes, connections etc), achieve a global error rate and computational inexpensive.

As in many of the neuro-fuzzy models, the user should always supervise the learning process and try to interpret the results. User has to specify the type of membership functions, clustering methods, learning parameters and many others depending on the model used. The data acquisition and preprocessing training data is also quite important for the success of neuro-fuzzy systems. Like in ANNs the success of the learning process is not guaranteed, as the designed network might not be optimal. Global optimization procedures like GAs or a combination of hybrid global search techniques (SA, TS etc.) might be useful for adaptive evolution of network architecture, learning algorithms, node functions etc., to prevent the network parameters being trapped in local optima due to gradient search or back propagation algorithms. For online learning, global optimization procedures might sound computational expensive. Fortunately GAs work with a population of independent solutions, which makes it easy to distribute the computational load among several processors. The design of parallel GA's involves choices such as using one population or multiple populations. In both cases, the size of population or populations must be determined carefully, and when multiple populations are used, one must decide how many to use [32]. In addition, the populations may remain isolated or they may communicate by exchanging individuals. Communication involves extra costs and additional decision on topologies, on how many individuals are exchanged, and on the frequency of communications

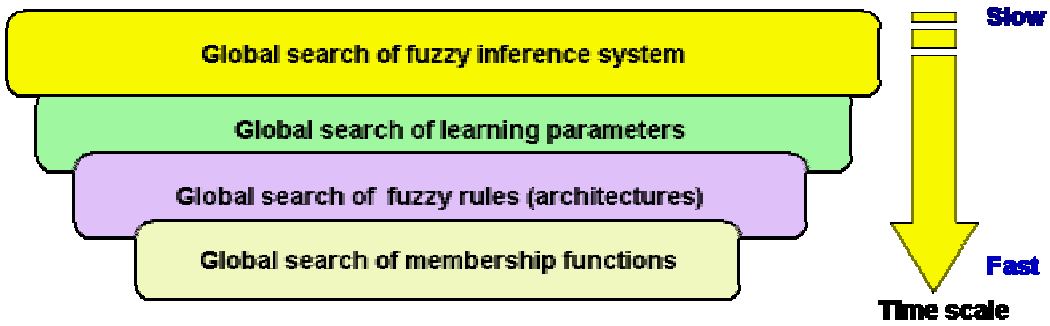


Figure 37. General framework for global search of optimal neuro-fuzzy system

Sugeno-type fuzzy systems are high performers (less RMSE) but often requires complicated learning procedures and computational expensive. However, Mamdani-type fuzzy systems can be modeled using faster heuristics but with a compromise on the performance (high RMSE). Hence there is always a compromise between performance and computational time. An optimal design of a neuro-fuzzy system can only be achieved by the adaptive evolution of membership functions, rule base (architecture) and learning rules, which progress on different time scales as illustrated in Figure 37. Most of the neuro-fuzzy systems are either based on Mamdani-type or Takagi-Sugeno-type. As shown in Figure 37, the fuzzy inference procedure (Mamdani FIS, Takagi Sugeno FIS, etc.) will evolve at the highest level on the slowest time scale to adapt the inference system according to the problem.

For every (rule base) architecture, there is the evolution of learning rules that proceeds on a slower time scale in an environment decided by the architecture. For each architecture, evolution of membership functions proceeds at a faster time scale in an environment decided by the problem, the architecture, learning rule and the inference system. Hierarchy of the different adaptation procedures will rely on the prior knowledge. For example, if there is more prior knowledge about the architecture than the learning rules then it is better to implement the learning rule at a higher level.

9 References

[1] Zadeh LA, *Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent Systems*, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, O Kaynak, LA Zadeh, B Turksen, IJ Rudas (Eds.), pp1-9, 1998.

[2] Dasgupta D (Ed.), *Artificial Immune Systems and Their Applications*, Publisher: Springer-Verlag, Berlin, January 1999.

[3] Thomas B, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*; Oxford University Press, 1996.

[4] Jang J S R, ANFIS: *Adaptive Network based Fuzzy Inference Systems*, IEEE Transactions, Man & Cybernetics, 1991.

[5] Jang J S R, *Neuro-Fuzzy Modeling: Architectures, Analyses and Applications*, PhD Thesis, University of California, Berkeley, July 1992.

[6] Nauk D and Kruse R, *A Neuro Fuzzy Controller Learning by Fuzzy Error Propagation*, In Proceedings of Conference of the North American Fuzzy Information Processing Society NAFIPS '92, Mexico, pp. 388-397, 1992.

[7] Nauk D, Nauck U and Kruse R, *Generating Classification Rules with Neuro-Fuzzy System NEFCLASS*, In proceedings of the Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS '96, Berkeley, 1996.

[8] Sulzberger S M, Tschicholg-Gurman N N and Vestli S J, *FUN: Optimization of Fuzzy Rule Based Systems Using Neural Networks*, In Proceedings of IEEE Conference on Neural Networks, San Francisco, pp 312-316, March 1993.

[9] Juang Chia Feng and Lin Chin Teng, *An Online Self Constructing Neural Fuzzy Inference Network and its Applications*, IEEE Transactions on Fuzzy Systems, Vol 6, No.1, pp. 12-32, 1998.

[10] Tano S, Oyama T and Arnould T, *Deep combination of Fuzzy Inference and Neural Network in Fuzzy Inference*, Fuzzy Sets and Systems, 82(2) pp. 151-160, 1996.

[11] Russel S and Norwig P, *A modern Approach to Artificial Intelligence*, Prentice Hall, Englewood Cliffs*, NJ, 1995.*

[12] Jacobsen H A and Iordanova I*, Approach to Extraction of Fuzzy Production Rules from a Connectionist Component of a Hybrid Expert System.* Proceedings of the 6th International Conference on AI: Methodology, Systems and Applications, Bulgaria, September 1994.

[13] Jacobsen H A, *A Generic Architecture for Hybrid Intelligent Systems,* In Proceedings of The IEEE Fuzzy Systems, Alaska, 1998.

[14] Abraham A and Nath B, *Failure Prediction Of Critical Electronic Systems in Power Plants Using Artificial Neural Networks*, First International Power & Energy Conference, INT-PEC'99, November 1999.

[15] Abraham A and Nath B, *Artificial Neural Networks for Intelligent Real Time Power Quality Monitoring Systems*, First International Power & Energy Conference, INT-PEC'99, November 1999.

[16] Arao M, Tsutsumi, Fukuda T and Shimokima K*, Flexible Intelligent System based on Fuzzy Neural Networks and Reinforcement Learning*, In proceedings of IEEE International Conference on Fuzzy Systems, Vol. 5(1), pp 69-70, March 1995.

[17] Hunt K J, Sbarbaro D, Zbikowski R and Gawthorp P J, *Neural Networks for Control Systems – A survey*, Automatica, Vol. 28, pp. pp. 1083-1112, June 1992.

[18] Widrow B and Winter R, *Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition*, Computer, Vol.21, no.3, pp. 25-39, 1988.

[19] Bishop CM, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1995.

[20] Sundararajan N and Saratchandran P, *Parallel Architectures for Artificial Neural Networks: Paradigms and Implementations,* IEEE Computer Society Press, 1998.

[21] Rumelhart D E, Hinton G E and Williams R J, *Learning Internal Representations by Error Propagation*. In D. E. Rumelhart and J. L. McClelland, (Eds.), Parallel Distributed Processing: Explorations in the microstructure of cognition; Vol. 1: Foundations*,* Cambridge, MIT Press, 1986.

[22] Skapura D M, *Building Neural Networks,* Addison-Wesley Publishing Company, 1996.

[23] Schiffmann W, Joost M and Werner R, *Comparison of Optimized Backpropagation Algorithms,* Proceedings. Of the European Symposium on Artificial Neural Networks, Brussels, M. Verleysen (Ed.), de Facto Press, pp. 97-104, 1993.

[24] X Yao, *Evolving Artificial Neural Networks*, Proceedings of the IEEE, 87(9): pp. 1423-1447, September 1999.

[25] Abraham A and Nath B, *Hybrid Heuristics for Optimal Design Of Neural Networks*, International Conference on Recent Advances in Soft Computing, RASC2000, June 2000.

[26] Zadeh L A, *Outline of a New Approach to the Analysis of Complex Systems and Decision Process*, IEEE Transactions, System, Man, and Cybernetics, Vol.3, no.1, pp. 28-44, 1973.

[27] Procyk T J and Mamdani EH, *A Linguistic Self Organising Process Controller*, Automatica, vol.15, no.1, pp. 15-30, 1979.

[28] Kosko B, *Fuzzy Engineering*, Upper Saddle River, NJ: Prentice Hall, 1997.

[29] Kaufmann A, *Introduction to the Theory of Fuzzy Subsets*, New York, Academic Press, 1975.

[30] Mizumoto M, *Fuzzy sets and their operations*, Information and Control, vol. 48, pp. 30-48, 1981.

[31] Goldberg, DE, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[32] Cantu-Paz Erick, *Designing Efficient and Accurate Parallel Genetic Algorithms*, PhD thesis, Graduate College of the University of Illinois at Urbana Champaign, 1999.

[33] Metropolis N, Rosenbluth A W, Rosenbluth MN, Teller AH and Teller E, *Equations of State Calculations by Fast Computing Machines*, J. Chem. Phys. 21, pp. 1087- 1092, 1958.

[34] X Yao, *A New Simulated Annealing Algorithm*, International Journal of Computer Mathematics, 56: pp.161-168, 1995.

[35] Glover F, Taillard E and De Werra D, *A User's Guide to Tabu Search*, In: Hammer PL (Ed.) Annals Of Operations Research, Volume 41, 3-28, pp 3-27, 1993.

[36] Tsukamoto Y, *An Approach to Fuzzy Reasoning Method*, Gupta MM et al (Eds.), Advances in Fuzzy Set Theory and Applications, pp. 137-149, 1979.

[37] Takagi T and Sugeno M, *Derivation of Fuzzy Control Rules from Human Operators Control Actions*, Proceedings of the IAFC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, pp 55-60, July 1983.

[38] Berenji H R, Lea R N, Jani Y, Khedkar P, Malkani A and Hoblit J, *Space Shuttle Attitude Control by Reinforcement Learning with Fuzzy Logic*, In Proceedings of IEEE International Conference on Neural Networks, San Francisco, pp. 1396-1401.1993.

[39] Nauck D and Kruse R, *Designing Neuro-Fuzzy Systems Through Backpropagation*, Fuzzy Modeling: Paradigms and Practice, Witold Pedryz, (Ed), Kluwer Academic Publishers, pp. 203-228, 1996.

[40] Nauck D, Klawonn F and Kruse R, *Foundations of Neuro-Fuzzy Systems*, Wiley, 1997.

[41] Nauck D and Kruse R, *New Learning Strategies for NEFCLASS*, In Proceedings of Seventh International Fuzzy Systems Association World Congress, IFSA'97, Vol. IV, pp. 50-55, 1997.

[42] Nauck D and Kruse R, *Neuro-Fuzzy Systems for Function Approximation,* 4th International Workshop Fuzzy-Neuro Systems, 1997.

[43] Halgamuge SK, Mari A and Glesner M, *Fast Perceptron Learning by Fuzzy Controlled Dynamic Adaption of Network Parameters*, Kruse. R et al (Eds.) Fuzzy Systems in Computer Science, pp 129-139, 1994.

[44] Narazaki H and Ralescu, *A Synthesis Method for Multi-Layered Neural Network Using Fuzzy Sets*, Proceedings of the International Workshop on Fuzzy Logic in Artificial Intelligence, pp 54-66, Sydney.

[45] Sugeno M and Tanaka K, *Successive Identification of a Fuzzy Model and its Applications to Prediction of a Complex System,* Fuzzy Sets Systems, Vol.42, no.3, pp. 315-334, 1991.

[46] Kasabov N, *Evolving Connectionist and Fuzzy Connectionist Systems –Theory and Applications for Adaptive On-line Intelligent Systems*, In: Neuro-Fuzzy Techniques for Intelligent Information Processing, Kasabov N and Kozma R, (Eds.), Physica Verlag, 1999.

[47] Kasabov N, *Evolving Connectionist Systems for Fast Identification, Classification and Decision Making*, Australian Journal of Intelligent Information Processing Systems, 1999.

[48] Kasabov N, *The ECOS Framework and the 'ECO' Training Method for Evolving Connectionist Systems*, Journal of Advanced Computational Intelligence, Vol.2, No.6, pp.1-8, 1998.

[49] Kasabov N, *Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation*, in Yamakawa T and Matsumoto G (Eds), Methodologies for the Conception, Design and Application of Soft Computing, World Scientific, pp. 271-274, 1998.

[50] Kasabov N and Qun Song, *Dynamic Evolving Fuzzy Neural Networks with 'm-out-of-n' Activation Nodes for On-line Adaptive Systems*, Technical Report TR99/04, Department of information science, University of Otago, 1999.

[51] Kasabov N, *Adaptable Connectionist Production Systems*, Neurocomputing,13(2-4), pp.95-117, 1996.

[52] Jang J S R, *Input Selection for ANFIS Learning*, In Proceedings of the IEEE International Conference on Fuzzy Systems, New Orleans, 1996.

[53] Zhang Q Y and Kandel A, *Compensatory Neurofuzzy Systems with Fast Learning Algorithms*, IEEE Transactions on Neural Networks, Volume 9, No. 1, pp.83-105, 1998.

[54] Kandel A, Zhang QY and Bunke H, *A genetic Fuzzy Neural Network for Pattern Recognition,* In IEEE Transactions on Fuzzy Systems, pp. 75-78, 1997.

[55] Jang JSR and Sun Chuen-Tsai, *Neuro-Fuzzy Modeling and Control*, Proceedings of the IEEE, Vol. 83, pp. 378-406, 1995.

[56] Oyama T, Tano S, Miyoshi T, Kato Y, Arnould T and Bastian A, *FINEST: Fuzzy Inference Environment Software with Tuning*, In Proceedings of IEEE International Conference on Fuzzy Systems, pp 3-4, 1995.

[57] Medsker LR, *Hybrid Intelligent Systems*, Kluwer Academic Publishers, 1995.

[58] Mizumoto M and Shi Yan, *A New Approach of Neurofuzzy Learning Algorithm*, Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms, Ruan D (Ed.), Kluwer Academic Publishers, pp. 109-129, 1997.

[59] Cherkassky V, *Fuzzy Inference Systems: A Critical Review*, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Kayak O, Zadeh LA et al (Eds.), Springer, pp.177-197, 1998.

[60] Kruse R and Nauck D, *A Neuro-Fuzzy Systems*, *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, Kayak O, Zadeh LA et al (Eds.), Springer, pp.230-259, 1998.

[61] Kandel A and Langholz G, (Eds), *Hybrid Architectures for Intelligent Systems*, CRC Press, 1992.

[62] Takagi H, Hayashi I, *NN-Driven Fuzzy Reasoning*, International Journal of Approximate Reasoning, Vol.5, pp. 191-212, 1991.

[63] Halgamuge S K and Glesner M, *The Fuzzy Neural Approach for Pattern Classification with the Generation of Rules Based on Supervised Learning*, In Proceedings of Neuro-Nimes 92, pp. 167-173, 1992.

[64] Zimmermann H G, Neuneier R, Dichtl H and Siekmann S, *Modeling the German Stock Index DAX with Neuro-Fuzzy*, In Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing, 1996.

[65] Mackey MC, Glass L, *Oscillation and Chaos in Physiological Control Systems*, Science Vol 197, pp.287-289, 1977.

[66] Nauck D, *Neuro-Fuzzy Systems: Review and Prospects*, In Proceedings of Fifth European Congress on Intelligent Techniques and Soft Computing (EUFIT'97), pp. 1044-1053, 1997.

[67] Abe S and Lan M S, *Fuzzy Rule Extraction Directly from Numerical Data for Function Approximation*, IEEE Trans. Systems, Man & Cybernetics, 25: pp. 119-129, 1995.

[68] Abe A and Lan M S, *A Method for Fuzzy Rules Extraction Directly from Numerical Data and its Application to Pattern Classification*, IEEE Transactions on Fuzzy Systems, 3(1): pp. 18-28, 1995.

[69] Berenji H R, *A Reinforcement Learning-Based Architecture for Fuzzy Logic Control*. International. Journal of Approximate Reasoning, 6: pp. 267 –292, 1992.

[70] Berenji H R and Khedkar P, *Fuzzy Rules for Guiding Reinforcement Learning*, In International. Conference. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'92), pp. 511-514, 1992.

[71] Berenji H R and Khedkar P, *Learning and Tuning Fuzzy Logic Controllers Through Reinforcements*, IEEE Transactions. Neural Networks, 3: pp.724-740, 1992.

[72] Esogbue A O, *A Fuzzy Adaptive Controller Using Reinforcement Learning Neural Networks*. In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 178-183, 1993.

[73] Brown M, Bossley K and Mills D, *High Dimensional Neurofuzzy Systems: Overcoming the Course of Dimensionality*, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 2139-2146, 1995.

[74] Andlinger P and Reichl E R, Fuzzy-Neunet: *A Non Standard Neural Network*. In Prieto et al., 1991], pp. 173-180, 1991.

[75] Bersini H, Nordvik J P and Bonarini A, *A Simple Direct Adaptive Fuzzy Controller Derived from its Neural Equivalent*, In Proceedings IEEE International. Conference on Fuzzy Systems, pp. 345-350, 1993.

[76] Buckley J J and Hayashi Y, *Hybrid neural nets can be fuzzy controllers and fuzzy expert systems*, Fuzzy Sets and Systems, 60: pp. 135-142, 1993.

[77] Buckley J J and Hayashi Y, *Neural Networks for Fuzzy Systems*, Fuzzy Sets and Systems, 71: pp. 265-276, 1995.

[78] Carpenter G A, Grossberg S, Markuzon N, Reynolds J H, and D. B. Rosen, Fuzzy ARTMAP: *A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps*. IEEE Transactions Neural Networks, 3(5), pp 698-712, 1992.

[79] Abraham A and Nath B, "*Designing Optimal Neuro-Fuzzy Systems for Intelligent Control* " (Invited Paper), The Sixth International Conference on Control, Automation, Robotics and Vision, (ICARCV 2000), December 2000.

[80] Eppler W, *Pre-structuring of neural networks with fuzzy rules*, In Neuro-Nimes'90. 3rd International Workshop on Neural Networks and their Applications, pp 227-241, 1990.

[81] Freisleben B and Kunkelmann T, *Combining Fuzzy Logic and Neural Networks to Control an Autonomous Vehicle*, In Proceedings IEEE International. Conference on Fuzzy Systems, pp 321-326, 1993.

[82] Furuya T, Kokubu A and Sakamoto T, FS: *Neuro Fuzzy Inference System*, In International Workshop on Fuzzy System Applications, pp. 219-230, 1988.

[83] Glorennec P Y, *Associating a Neural Network and Fuzzy Rules for Dynamic Process Control*, 3rd Int. Workshop on Neural Networks and Their Applications Neuro-Nimes'90, pp. 211-225, 1990.

[84] Goode P and Chow M Y, *A Hybrid Fuzzy/Neural System Used to Extract Heuristic Knowledge from a Fault Detection Problem*, In Proceedings. IEEE International. Conference. on Fuzzy Systems, pp. 1731-1736, 1994.

[85] Gupta M M and Rao D H, *On the Principles of Fuzzy Neural Networks*, Fuzzy Sets and Systems, 61: pp.1-18, 1994.

[86] Halgamuge S K, Glesner M, *Neural Networks in Designing Fuzzy Systems for Real World Applications*, Fuzzy Sets and Systems, 65: pp. 1-12, 1994.

[87] Hayashi I, Nomura H, Yamasaki H and Wakami N, *Construction of Fuzzy Inference Rules by Neural Network Driven Fuzzy Reasoning and Neural Network Driven Fuzzy Reasoning With Learning Functions*, International. Journal of Approximate Reasoning, 6: pp. 241-266, 1992.

[88] Hayashi Y and Imura A, *Fuzzy Neural Expert System with Automated Extraction of Fuzzy If-Then Rules from a Trained Neural Network*, In First International. Symposium on Uncertainty Modeling and Analysis, pp 489-494, 1990.

[89] Hollatz J, *Neuro-Fuzzy in Legal Reasoning*, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 655-662, 1995.

[90] Ichihashi H and Turksen I, *Neuro-Fuzzy Data Analysis and Its Future Directions*, In Proceedings. IEEE International Conference on Fuzzy Systems, pp 1919-1925, 1995.

[91] Ishibuchi H, Okada H and Tanaka H, *Interpolation of fuzzy of fuzzy if-then rules by neural networks*. In Proceedings, 2nd Int. Conf. on Fuzzy Logic and Neural Networks, pp. 337-340, 1992.

[92] Kasabov N K, *Hybrid Fuzzy Connectionist Rule-based Systems and the Role of Fuzzy Rules Extraction*, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp 49-56, 1995.

[93] Keller M, Yager R R and Tahani H, *Neural Network Implementation of Fuzzy Logic. Fuzzy Sets and Systems*, 45: pp.1-12, 1992.

[94] Khan E and Venkatapuram P, Neufuz: *Neural Network Based Fuzzy Logic Design Algorithms*, In Proceedings IEEE International Conference on Fuzzy Systems, pp. 647-654, 1993.

[95] Li W, *Optimization of a fuzzy controller using neural network*, In Proceedings IEEE International. Conference on Fuzzy Systems, pp 223-227, 1994.

[96] Lin C T and Lee C S G, *Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems*. In Proceedings IEEE International. Conference. on Fuzzy Systems, pp. 88-93, 1993.

[97] Mitra S and Pal S, *Fuzzy Multi-Layer Perceptron: Inferencing and Rule Generation*, IEEE Transactions Neural Networks, 6: pp. 51-63, 1995.

[98] Miyoshi T, Tano S, Kato Y and Arnould T, *Operator Tuning in Fuzzy Production Rules using Neural Networks*, In Proceedings. IEEE Int. Conf. on Fuzzy Systems, pp. 641-646, 1993.

[99] Narazaki H and Ralescu A L, *A Synthesis Method for Multi-Layered Neural Network using Fuzzy Sets*, International Workshop on Fuzzy Logic in Artificial Intelligence, pp. 54-66, 1991.

[100] Rueda A and Pedrycz W, *A Hierarchical Fuzzy-Neural-PD Controller for Robot Manipulators,* In Proceedings. IEEE International Conference on Fuzzy Systems, pp. 672-677, 1994.

[101] Simpson P, *Fuzzy Min-Max Neural Networks - Part 1: Classification*, IEEE Transactions Neural Networks, 3: pp.776-786, 1992.

[102] Simpson P, *Fuzzy Min-Max Neural Networks - Part 2: Clustering,* IEEE Transactions Fuzzy Systems, 1: pp. 32-45, 1992.

[103] Takagi H, *Fusion Technology of Fuzzy Theory and Neural Networks - Survey and Future Directions*. In Proceedings 1st International Conference on Fuzzy Logic & Neural Networks, pp. 13-26, 1990.

[104] Tschichold-Gurman N, Ghazvini M and Diez D, *M-RCE: A Self-Configuring ANN with Rule-Extraction Capabilities*. In Proceedings. of International Conference on Artificial Neural Networks, 1992.

[105] Vasilakos A V and Zikidis K C, *A Novel Neuro-Fuzzy Architecture for Fuzzy Computing Based on Functional Reasoning*, In Proceedings IEEE International Conference on Fuzzy Systems, pp. 671-678, 1995.

[106] Yager R, *On the Interface of Fuzzy Sets and Neural Networks*, In International Workshop on Fuzzy System Applications, pp. 215-216,1988.

[107] Yao C C and Kuo Y H, *A Fuzzy Neural Network Model with Three-layered Structure*, In Proceedings. IEEE International. Conference on Fuzzy Systems, pp. 1503-1510, 1995.

[108] Yen J, Wang H and Daugherty W C, *Design Issues of a Reinforcement-based Self-Learning Fuzzy Controller for Petrochemical Process Control*, In Proceedings Workshop of North American Fuzzy Information Processing Society, pp. 135-142, 1992.

[109] Sugeno M, *Industrial Applications of Fuzzy Control*, Elsevier Science Pub Co., 1985.

[110] Mamdani EH and Assilian S, *An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller*, International Journal of Man-Machine Studies, Vol. 7, No.1, pp. 1-13, 1975.

[111] Colorni A., M.Dorigo, F.Maffioli, V. Maniezzo, G. Righini and M. Trubian, *Heuristics from Nature for Hard Combinatorial Problems,* International Transactions in Operational Research, Vol (3) 1, pp 1-21, 1996.

[112] Costa D., *An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem*, INFOR Vol. 33, No.3, pp.161-178, 1995.

[113] Mohammadian M and Stonier R J, *Generating Fuzzy Rules by Genetic Algorithms,* Proceedings of 3rd IEEE International Workshop on Robot and Human Communication, Nagoya, pp.362-367. 1994.

[114] Frank Hoffmann, *The Role of Fuzzy Logic in Evolutionary Robotics*, Fuzzy Logic Techniques for Autonomous Vehicle Navigation, A. Saffiotti and D. Driankov (Ed.), Springer-Verlag, 1999.

[115] Lee M.A. and Esbensen H., *Fuzzy / Multiobjective Genetic Systems for Intelligent Systems Design Tools and Components*, Fuzzy Evolutionary Computation, Pedrycz W (Ed.), Kluwer Academic Publishers, pp.107-125, 1997.

[116] Hoffmann F and Pfister G, *Automatic Design of Hierarchical Fuzzy Controllers Using Genetic Algorithms*, 2nd European Congress on Intelligent Techniques and Soft Computing (EUFIT '94), 1994

[117] Furuhashi T, *Development of If-Then Rules with the Use of DNA Coding,* Fuzzy Evolutionary Computation, Pedrycz W (Ed.), Kluwer Academic Publishers, pp.107-125, 1997.

[118] Herrera F., Lozano M and Veregay L., *Tuning Fuzzy Logic Control by Genetic Algorithms*, International Journal of Approximate Reasoning, 12(3/4), pp. 299-315, 1995.

[119] Mendel J.M and Wang L.X., *Generating Fuzzy Rules by Learning from Examples,* IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 6, pp.1414-1427, 1992.

[120] Jager R, *Fuzzy Logic in Control*, PhD Thesis, Technische Universiteit Delft, Netherlands, 1995.

[121] Hoffmann F, Soft computing techniques for the design of mobile robot behaviors, Journal of Information Sciences, 122 (2-4) pp. 241-258, 2000.

[122] Abraham A and Nath B, Evolutionary Design of Fuzzy Control Systems - An Hybrid Approach, The Sixth International Conference on Control, Automation, Robotics and Vision, (ICARCV 2000), December 2000.