Journal of the IETE
Vol. 42, No. 4 & 5, July-October 1996, pp 195-204

Soft Computing: Goal, Tools and Feasibility

SANKAR K PAL, FIETE AND NIKHIL R PAL

Machine Intelligence Unit, Indian Statistical Institute, Calcutta 700 035, India.

This paper discusses what soft computing is, the need for soft computing and real world computing (RWC) systems, the essential ingredients necessary to realise this, *ie*, neural networks, fuzzy logic and probabilistic reasoning and their role in soft computing. The development of hybrid computational paradigms are also explored and they are projected as a frontier research area in the evolution of sixth generation computing systems.

Indexing terms: Neural networks, Fuzzy logic, Genetic alogorithms, Hybrid systems, Pattern recognition

WHAT IT IS ?

A computer can complete a job more efficiently than a human being when the job involves substantial amount of routine computation; like inversion of a matrix of large dimension. On the other hand, if the task requires perceptual power or cognitive capability of human beings, the Von-Neumann machine is far behind human beings. For example, human beings can recognize shapes of different sizes, orientations, even in an occluded environment much more efficiently than by a computer. A Von-Neumann machine is good for well structured problems. Typically, human brains are better for solving real world ill-defined, imprecisely formulated problems requiring huge computations. To overcome the limitations of traditional computing paradigm, scientists are searching for new computational approaches that can be used to model, at least partially, the functioning of brains and can be used to solve real world problems efficiently. As a result, in the recent past several novel modes of computation have emerged which are collectively known as soft computing.

The *raison d'ëtre* of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness and low cost [1]. Usually, it attempts to find an approximate solution to a precisely or imprecisely formulated problem. Neuro computing (NC) is one of the major components of soft computing. The other two important constituents are fuzzy logic (FL) and probabilistic reasoning (PR) with PR subsuming belief networks, genetic algorithms (GA) and chaotic systems. FL is mainly concerned with providing a machinery for dealing with imprecision and approximate reasoning. NC deals with learning and curve fitting. PR, on the other hand, deals with probabilistic uncertainty, propagation of belief, and searching and optimization.

Information processing in a biological system is a complex phenomenon, which enables us to survive by accomplishing tasks like recognition of surrounding, making prediction, planning, and acting accordingly. Human type information processing involves both logical information processing and intuitive information processing. Conventional computer systems are good for the former but their capability for the latter is far behind that of human beings. As a first step to accomplish human-like intuitive information processing, a computer system should be flexible enough to support the following three characteristics: openness, robustness an real-time processing. Openness of a system is its ability to adapt or extend itself on its own to cope with changes encountered in the real world. Robustness of a system means its stability and tolerability when confronted with distorted, incomplete or imprecise information. The real-time characteristic implies the ability of the system to react within a reasonable time in response to an event. Information processing systems with all these three characteristics are sometimes, known as Real World Computing (RWC) systems. A RWC system should, therefore, be capable of distributed representation of information, massively parallel processing, learning and self-organization in order to achieve flexibility in information processing. Thus, soft computing can be viewed as the key ingredient of RWC systems.

WHY IT IS NEEDED

Traditional hard computing paradigm is usually not suitable for many real life problems. Let us illustrate it with an example. Suppose X is driving a car and X watches a "Red light" (traffic signal). X has to stop. So X has to decide when to press the brake and how strongly. In a "precise framework" the steps followed by X may be to find the distance of the car from the "Light" and then depending on the current speed of the car press the brake. To realize this, the car should be provided with a laser-gun type arrangement so that the distance can be obtained. X also should know a set of rules of the form "If the car is at a distance of d ft and moving at a speed of s ft/sec then press the brake with p poundal for t seconds right now". This is a precise rule governed by the laws of Physics. Hence, if the brake is applied according to such rules, the car will stop where X wants it to. Theoretically, such concepts are fine, but practically they are useless because of the following reasons.

- Addition of a laser gun to a car increases its cost.
- Number of precise rules required will be too many to realize in a practical system.
- For the sake of arguments even if we assume that we know the rules to be followed, application of the brakes following the rule will be a very difficult task.

Paper No 48-A; Copyright ©1996 by the IETE.

This shows that precise models for solving such a simple real life problem under the conventional computing paradigm becomes not only very difficult but also expensive.

What do we learn from this example! Precise solutions are not always feasible. In fact, we do not need a precise solution to such a problem. The exact position where the car stops is not important, but it should stop before the "Red light" and should not hit any other car standing ahead of it. Hence, an approximate idea about the distance of the car and the speed of the car should be enough. Under this situation X can control the car using rules of the form "If the car is moving *very fast* and the "Red Light" is *close* then press the brake *pretty hard*".

Note that the rule has three vague clauses "very fast", "close" and "pretty hard". These make the rule an imprecise one and it will generate an approximate solution to the problem. The solution is less expensive and fast (real time) also. This is one facet of what soft computing paradigm for emulating the human like decision making (also, a real world computing system) attempts to achieve. Thus, to achieve higher machine IQ, the system should have the capability of modeling vagueness and making approximate decisions based on that. Fuzzy sets are good for handling this aspect of soft computing. In fact, this distinguished characteristic of fuzzy sets led to the emergence of soft computing.

Let us now make the driving problem a bit more complex. Suppose X is driving on a very crowded road and has to reach the destination D. From the present coordinate of X there are a couple of alternative paths to reach D. Depending on the traffic condition X should try to pick up an optimal path. Note that the traffic conditions (traffic flows in either direction, number of traffic signals that will appear on a path, raining or not *etc*) change with time, and hence what X thinks as the optimal path now may not remain optimal after some time. Consequently, X has to dynamically (adaptively) change his/her route. A human being makes approximate decisions for such problems based on his/her experience (learning from previous driving experiences). If we want an intelligent system to achieve this capability, it should have the ability to learn from experience and examples. Artificial neural networks (NN) are adaptive systems, and can deal with this aspect of the problem.

Any artificial network that can be used for handling the problem just mentioned, must be fed with relevant information. In other words, the NN system has to be trained with adequate number of examples. The popular gradient decent (may be the backpropagation) type learning algorithms are usually very slow in learning and they may get stuck at some local minimum. Genetic algorithms, in such situations, may be very effectively used for learning. If the number of free parameters of the network is large, GAs may also become slow, but for GA-based learning the chance of getting stuck to a local minimum would be low. Consequently, we can expect a better generalization ability of the network.

The previous discussion suggests that the core and essential ingredients necessary to realize human-like computations are NN, FL, and PR. There are many such examples in the real world which support this. Next we discuss each of these three computational tools. Since the article is for a special issue on neural networks, we have devoted more on the NN component of soft computing.

NEURAL NETWORKS

In our body cells die all the time without affecting the performance of the system. This robustness possibly is achieved by the massive connectivity of the biological neural network. The death of a few cells or malfunctioning of a few of them are not expected to influence the systems behaviour. This suggests that computational neural nets may also be made to achieve this property with massive interconnection and parallel processing. Note that in a Von-Neumann kind machine error in one bit can affect severely the final outcome.

The concept of artificial neural networks has been inspired by biological neural networks, but the heart of this emerging technology is rooted in different disciplines. Biological neurons are believed to be the structural constituents of the brain and they are much slower than silicon logic gates. But inferencing in biological NN is faster than the fastest computer available today. Brain compensates for the relatively slower operation by a really large number of neurons with massive interconnections between them. Biological neural networks enjoy the following characteristics

- It is a nonlinear device highly parallel, robust and fault tolerant.
- It has a built-in capability to adapt its synaptic weights to changes in the surrounding environment.
- It can handle easily imprecise, fuzzy, noisy and probabilistic information.
- It can generalize from known tasks or examples to unknown ones.

Artificial NN (in short NN) is an attempt to mimic some or all of these characteristics [2-4]. This soft computational paradigm is different from programmed instruction sequence. Here information is stored in the synaptic connections. A neuron is an elementary processor with primitive types of operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. The computational neuron model proposed by Mc-Culloch-Pitts is a simple binary threshold unit as shown in Fig 1. The ith neuron computes the weighted sum of all its inputs from other units and outputs a binary value, zero or one, depending on whether this weighted sum is greater than equal or less than a threshold θ_i.

$$\text{Thus} \quad x_i(t+1) = f\left(\sum_i w_{ij} x_j(t) - \theta_i\right)$$

$$\text{where} \quad f(x) = 1 \text{ if } x \geq 0$$
$$= 0 \text{ otherwise.}$$

If the synaptic weight $w_{ij} > 0$, then it is called an excitatory connection; if $w_{ij} < 0$, it is viewed as an inhibitory connection.

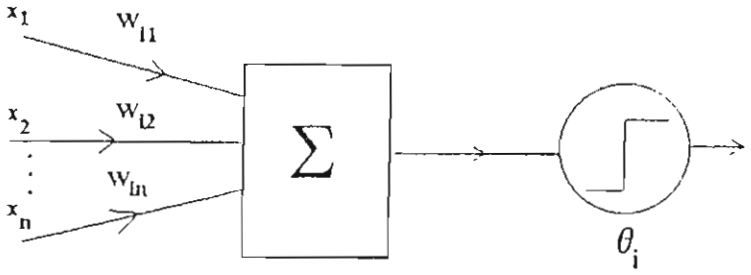

Fig 1 Mc-Culloch Pitts neuron

A simple generalization of Mc-Culloch-Pitts neuron by replacing the threshold function f with a more general non-linear function enhances the power of the networks built from such neurons. Even a synchronous assembly of Mc-Culloch-Pitts neurons is capable, in principle, of universal computation for suitably chosen weights [2]. Such an assembly can perform any computation that an ordinary digital computer can. Neural networks are naturally parallel computing devices. Although the development of neural networks is inspired by models of brains, the purpose is not just to mimic biological neurons, but to use principles from nervous systems to solve complex problem in an efficient manner.

A neural network is characterized by the network topology, connection strength between pairs of neurons (weights), node characteristics and the status updating rules. The updating or learning rules may be for weights and/or states of the processing elements (neurons). Normally an objective function is defined which represents the complete status of the network, and its set of minima corresponds to different stable states of the network.

The adaptability of a neural network comes from its capability of learning from "environments". Broadly, there are three paradigms of learning: Supervised, Unsupervised (or Self-Organized) and Reinforcement. Sometimes, reinforcement is viewed as a special case of supervised learning. Under each category there are many algorithms. In supervised learning (learning with a teacher) adaptation is done on the basis of direct comparison of the network output with known correct or desired answer. Unsupervised learning does not learn any specific input-output relation, rather the network is tuned to the statistical regularities of the input data to form categories by optimizing, with respect to the free parameters of the network, some task-independent measure of quality of representation of the categories by the net. The reinforcement learning, on the other hand, attempts to learn the input-output mapping through trial and error with a view to maximizing a performance index called reinforcement signal. Here the system only knows whether the output is correct or not, but not what the correct output is.

Some Popular Models

Artificial neural networks have become a technical folk legend of late. The market is flooded with new, more technical software (hardware) and products; many more are sure to come. Some of the popular networks are Hopfield Net (HN), Multilayer Perceptron (MLP), Self-Organizing Feature Map (SOFM), Learning Vector Quantization (LVQ), Cellular Neural Networks (CNN) and Adaptive Resonance Theory (ART) network.

- *Hopfield Network (HN)*: The idea behind HN came from the analysis of some physical system whose spontaneous behavior can be used as a general form of content-addressable memory. Suppose the status of such a system can be described by a set of coordinates. Thus, a point in the state space represents the instantaneous status of the system. The time evolution of such a system is nothing but a flow in the state space. If the flow pattern of the system is such that it moves toward a stable point in state space from anywhere in the neighborhood of the corresponding stable point, and the system has a reasonable number of stable points, then the system can be used as a content-addressable memory. The statistical mechanics of spin-glasses provides us with such a physical system. The Hopfield network is derived from this. HN is a completely connected recurrent net; it acts as a nonlinear associative memory that can retrieve a pattern stored in it in response to the presentation of an imcomplete or noisy version of that pattern. It can also be used as a tool for optimization. The status of any neuron of the network can be updated at random times independent of others but in parallel. Since there are interactions between all neurons, the collective property inherently reduces the computational task. The Boltzmann machine is a generalization of HN whose operation is based on a concept of statistical thermodynamics known as simulated annealing [2, 3]. HN, Boltzmann machine and a derivative of Boltzmann machine, known as Mean-Field-Theorem machine have been used in many applications like image segmentation and restoration, combinatorial optimization (travelling salesman problem, graph partitioning *etc*) in addition to their use as content-addressable memory [6].
- *Multilayer Perceptron (MLP)*: The concept of perceptron was one of the most exciting developments during the early days of pattern recognition [2, 3]. This may be defined as a network of elementary processors arranged in a manner reminiscent of biological neural nets which will be able to learn how to recognize and classify patterns in an autonomous manner. In such a system the processors are

simple elements arranged in one layer. This classical (single-layer) perceptron, given two classes of patterns, attempts to find a linear decision boundary separating the two classes. If the two sets of patterns are linearly separable, the perceptron algorithm can find a separating hyperplane in a finite number of steps. However, if the pattern space is not linearly separable, the perceptron fails. A single-layer perceptron is inadequate for situations with multiple classes and nonlinear separating boundaries. Hence the invention of the multilayer perceptron network. In an MLP, there is complete connection between layers but within a layer there is no connection. MLP uses supervised learning, implemented in two phases. In the forward phase output of the network nodes are computed and in the backward phase weights are adjusted to minimize the error between the observed output and desired output. The weight updating can be done in several ways - the most celebrated one being the back-propagation algorithm [2, 3].

MLP is believed to have generalization capability, *ie*, the capability of producing correct or nearly correct output for input not used during training. The net can be viewed as a non-linear input-output mapping and the learning process as a fitting of a function to the given data set. MLP can do good interpolation, because with continuous activation functions, the output function is also continuous. But one should be cautious about its generalization also, because given a data set an MLP can pick up one of many possible generalizations corresponding to different minima, which may not be the desirable one. Since learning involves searching over a complex space, often it is time consuming. MLP has been applied in many applications starting from classifier design, function approximation, speech identification, scene analysis and pattern recognition.

The back-propagation algorithm is applicable even when each neuron in layer i receives signals directly from each input node and output signals of each neuron in layer j, $j < i$; *ie*, the output of the neurons in the proceeding layers. There are various other learning schemes, like quick-prop [7] for multilayerd feed forward networks.

Self-Organizing Feature Mapping (*SOFM*): The SOFM is an unsupervised learning network [8], which transform p-dimensional input patterns to a q-dimensional (usually $q = 1$ or 2) discrete map in a topologically ordered fashion. Input points which are close in p-dimension are also mapped closer on the q-dimensional lattice. Each lattice cell is represented by a neuron which has a p-dimensional adaptable weight vector associated with it. With every input the match with each weight vector is computed. Then the best matching weight vector and some of its topological neighbors are adjusted to match the input points a little better. Initially, the process starts with a large neighborhood; and with passage of time (with iteration) the neighborhood size is reduced gradually. At a given time instant, within the neighborhood, the weight vector associated with each neuron is not updated equally. The strength of interaction between the winner and a neighboring node is inversely related to the distance (on the lattice) between them. The feature map is obtained by lighting up only the nodes on the lattice which become winner for some data point. Although, this map preserves the topological order of the data to a reasonable extent, usually the display map for unlabeled data is not much informative. The reason is, if there are, say, two distinct clusters in the data in p-dimension, and each cluster has uniformly distributed points, then on the feature map also the winner nodes corresponding to each cluster will be closer but the distribution of the lighted nodes will be uniform over the lattice. The map is, thus, not coherent with the structure of the data. What we mean is this: two spheres in p-dimension do not produce two disks on the lattice. As a result, unless the feature map is labeled, which of course, cannot be done for unlabeled data, such maps are not useful for getting information about the structure of the data. But, when the data satisfy certain conditions [9] the structure of the display map becomes coherent with that of the data. For example, when the data set contains a few annular rings, the annular structure is also exhibited by the map. However, the information gathered in a SOFM can be used fruitfully too. SOFM has been used for generation of semantic maps, clustering, phonetic type writer, graph bipartitioning *etc*. There are some other variations of the basic SOFM. For example, in [10, 11] the neighborhood is defined using nodes from a minimum spanning tree of the prototypes associated with the lattice structure.

Learning Vector Quantization (*LVQ*): A special case of SOFM is known as LVQ network [8]. In LVQ, only the weight vector associated with the winner node is updated with every data point. Such a learning scheme where all nodes compete to become the winner is known as the competitive learning scheme. It is essentially a clustering network which does not care about preserving the topological order. Its main uses are for clustering and image data compression.

LVQ attempts to minimize an objective function that places all of its emphasis on the winning prototype for each data point. This is reasonable, but it ignores global information about the geometric structure of the data that is represented in the remaining losing prototypes. Thus, LVQ updating is a local strategy that ignores global relationship between the winner and the rest of the prototypes. For a better performance, the network should be allowed to influence the update of the winner, and perhaps be allowed to update some or all of the remaining prototypes also. This poses two questions: What would be the update neighborhood? What would be learning rate distribution over the neighbors? Note that, here we are referring to the metrical neighbors, not the topological neighbors. Keeping in view these prolems several new soft competition schemes, like Fuzzy LVQ (FLVQ), Generalized LVQ (GLVQ), Stochastic Relaxation Scheme (SRS) and Soft Competition Scheme (SCS) have been developed [12-14].

All these methods eliminate the need to define an update neighborhood by extending the update to all nodes and use learning rates that are functions of the distances of the prototypes to the data point.

In GLVQ, the LVQ objective function is modified so that the winner and non-winner get different importance, but all of them are updated [13]. FLVQ [12] is a hybridization of the fuzzy c-means (FCM) [15] clustering algorithm and the LVQ. Here the update neighborhood is implicitly defined through the use of FCM membership function in the update equations.

In SRS [14] the prototypes are updated probabilistically, where the probability of updating a prototype is a function of its distance from the training data point presented at that instant. On the other hand, in SCS [14] all prototypes are simultaneously updated by a scheme, which moves them towards the current training vector. The step size of each update is scaled by the probability of that prototype being the winner. The probability that the ith prototype be the winner is defined using an exponential function of the distance between the data point and the weight vectors. In SCS the learning rate is controlled in such a manner that, in the limit (as time goes to infinity) SCS behaves like the winner-take-all (LVQ) competition.

There exists another family of LVQs named as LVQ1 and LVQ2 [8] . These algorithms are conceptually different from the family of soft competition schemes just discussed. LVQ1 and LVQ2 are supervised learning schemes - essentially used as classifiers. The basic idea behind LVQ1 is this: if the winner prototype v_i has the same class label as that of the data point x, then bring v_i closer to x; otherwise, move the v_i away from x. Non-winner nodes are not updated. LVQ2, a modified form of LVQ1, is designed to make the learning scheme comply better with Bayes' decision making philosophy. This algorithm considers both the winner and the runner-up (second winner).

Adaptive Resonance Theory (*ART*): For a competitive learning scheme, there is no guarantee that the different clusters formed will be stable unless the learning rate gradually approaches zero with iteration. When the learning rate is reduced to zero with iteration the network loses its plasticity. Adaptive Resonance Theory net [16] overcomes the stability-plasticity dilemma. In ART, with the progress of learning, new steady states are formed as the system discovers and learns prototypes that represent invariants of the set of all experienced patterns. This is known as the plasticity property of the network. Formation of the steady states is controlled to avoid possible sources of system instability. The ART system is plastic enough to learn significant new events, yet stable to irrelevant events. In ART, a weight vector (prototype of a category) is adapted only when the input is sufficiently similar to the prototype; *ie*, when the input and a prototype resonate. When an input is not sufficiently similar to any prototype, a new category is formed using the input as the prototype. The condition "sufficient similar" is checked using a vigilance parameter. ART1 is designed for 0/1 input while ART2 is for continuous valued input.

Radial Basis Function (*RBF*) *Net*: The basic RBF network [2 ,3] has three layers: input layer, output layer and a hidden layer. In a RBF network, the output nodes make a linear combination of the basis functions computed by the hidden layer nodes. Such a network is also known as a localized receptive field network, because the hidden layer nodes produce localized responses to the input signals. The most commonly used basis function is the Gaussian kernel function. Like MLP, RBF networks can be used for both classifier design and function approximation, and it can make an arbitrary approximation to any continuous nonlinear mapping [3, 17]. The main difference between MLP and RBF nets lies in the basis functions used by the hidden layer nodes. RBF nets use Gaussian basis functions while an MLP uses sigmoidal basis functions. All computation nodes of an MLP share the same neural architecture, while in a RBF network the hidden layer neurons are structurally quite different from those at the output layer. In an MLP, each neuron computes the inner product of the input vector and the weight vector associated to it. On the other hand, in a RBF net each hidden node computes the Euclidean distance between the input vector and the center of the RBF associated to that node. For a RBF, the learning of the hidden layer parameters and the linear weights associated with the output nodes are usually treated differently. If we use fixed (non adaptive) RBFs to define the activation of the hidden nodes, then only the linear weights associated with the output nodes are to be learnt. This can be done using the pseudo-inverse method. Sometimes hybrid learning schemes can also be used to learn the free parameters of the net. The center of the RBFs can be adapted in a self-organized manner such that centers are placed in those areas of the input space which contain significant data; while the weights associated with the output nodes can be learnt using a supervised scheme. Of course, one may learn all the free parameters of the net using a supervised scheme with gradient descent. Note that, the choice between RBF and MLP depends on the problem at hand.

Cellular Neural Networks (*CNN*): A CNN [18] has an architecture similar to a cellular automata; *ie*; it is an assembly of neurons or cells where each node is connected only to its neighbor cells. A neighboring cell is not necessarily an adjacent cell, it depends on the definition of the neighborhood. The dynamics of a CNN has both output feedback and input control mechanisms. Asynchronous processing, continuous time dynamics and local interactions between cells are some key features of a CNN.

The network architectures discussed so far are, in a sense, general in nature as they can be used to solve many different problems. But there are other special type of application specific networks developed to cater some specific needs. For example, the time delay network is good for modeling systems

where the output has a finite temporal dependency on the input [19]. There are other families of networks for principal component analysis, mixed category perception, character recognition *etc* [20-22].

FUZZY SETS AND SYSTEMS

Fuzzy sets were introduced in 1965 by Zadeh [23] as a new way to represent vagueness in everyday life. They attempt to model human reasoning/thinking process. Fuzzy sets are generalization of crisp sets and have greater flexibility to capture faithfully the various aspects of incompleteness or imperfection in information [5, 24].

In ordinary set theory, an element either belongs to a set or not. For fuzzy sets, an element can partially belong to a set; for example, a set of TALL persons. Here there is no precise boundary to the set TALL, and hence we cannot really point out a collection of people labeled as tall. Fuzzy sets are conceptual sets, whose semantic is more important than its mathematical characterization. Mathematically, a fuzzy set is nothing but a mapping (known as membership function) from the universe of discourse to [0, 1]. The membership function has nothing to do with probability. Membership value of an object gives the extent to which a fuzzy property is possessed by the object. Another way to say this is to imagine that membership functions possess elasticity. Thus, higher the value of membership of an object to a class, the lesser the imprecisely defined concept of fuzzy set must be stretched to accommodate the object. Hard membership functions (characteristic function of a set), of course, are inelastic.

The set TALL can be modeled by a function shown in Fig 2*a*. Note that this is not the only function that can be used to characterize the fuzzy set TALL. There may be many other functions, but all of them should have the general non-decreasing characteristic to keep the membership function consistent with the semantic of the fuzzy set. Such functions are known as S-type membership functions. Similarly, the fuzzy set, "CLOSE TO 7" can be modeled by the membership function shown in Fig 2*b*. Again, there could be many other choices for this membership function. But they should have a membership value of 1 at 7 or over a neighborhood of 7 and the membership values should decrease as we move away from 7. Such functions are called Π-type membership functions. Note that the flexibility of a fuzzy system is associated with the concept of membership function. The non-uniqueness of membership function raises a natural question-where do we get it from? Well, one can get the membership function from an expert (subjective computation) or may compute it based on data and a model (objective computation).

Since fuzzy sets characterize imprecise properties, they can be effectively used to model vagueness associated with real-life systems. Fuzzy logic is based on the theory of fuzzy sets and approximate reasoning. It is much closer in spirit to human reasoning and natural language than the traditional logical system. Thus, fuzzy logic provides an effective means to capture faithfully the approximate and inexact nature of the real world.

A fuzzy system usually has three components: fuzzification, processing and defuzzification. The fuzzification step provides a fuzzy set representation to a crisp input or makes a fuzzy representation of the input signals obtained otherwise. Triangular (more general Π-type) membership functions are used quite often to define fuzzy inputs and outputs required to characterize the fuzzy model of a system.

The processing component essentially transforms the fuzzy or crisp information to come out with a fuzzy conclusion (output). The type of processing depends on the fuzzy model being used. If a fuzzy clustering model is used, then the processing may be an iterative optimization of an objective function [15]; while for a fuzzy controller the processing means the use of some reasoning scheme to compute a fuzzy consequent [25].

For any real system the final action has to be crisp. The defuzzification phase produces crisp action from a fuzzy conclusion. Depending on the type of fuzzy system, the defuzzification scheme may be different. For example, in the case of a fuzzy classifier, the defuzzification scheme may be to find the class with maximum membership value. On the other hand, for a fuzzy control system, the defuzzification is obtained by aggregating several output fuzzy sets using one of several schemes, like center of mass, height method [25].

GENETIC ALGORITHMS (GAs)

Genetic algorithms [26-28], another biologically inspired evolutionary method, provide an adaptive, robust, parallel and randomized searching technique where a population of solutions evolves over a sequence of generations to a globally optimal solution. Based on a fitness function, good solutions are selected for reproduction using two genetic recombination op-

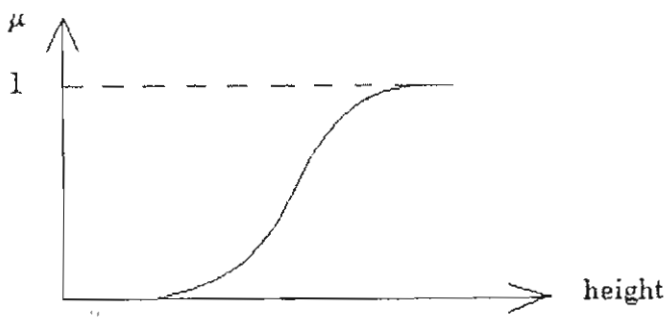

Fig 2*a* Fuzzy set "TALL"

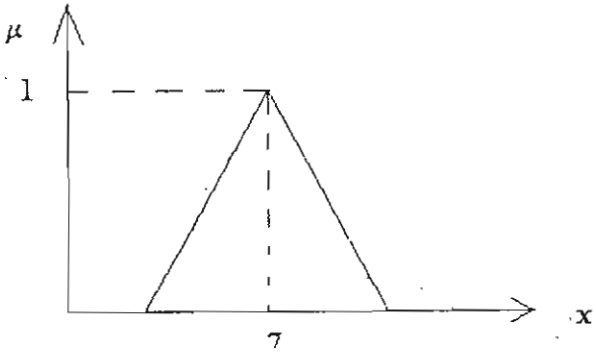

Fig 2*b* Fuzzy set "CLOSE TO 7"

erators, crossover and mutation.

GAs [26, 27] are optimization algorithms which can solve problems resistant to other known optimization methods. They do not require differentiability or continuity of the fitness function. Even if the fitness function is not mathematically well structured, they can be used to find an optimal solution. GAs work simultaneously on multiple points in the search space, not on a single point, unlike conventional search techniques. Due to the stochastic characteristic, they have a very low chance of getting stuck to a local minimum. The criterion of "survival of the fittest" provides evolutionary pressure for populations to grow with increasingly fit individuals. Although there are many variants, the basic mechanism of GAs (conventional GAs) consists of the following steps

(*i*) Start with an initial population (a set of strings/chromosomes).

(*ii*) Evaluation of fitness of every string and selection of appropriate candidate strings to form the mating pool.

(*iii*) Crossover and mutation.

(*iv*) Repetition of steps (*ii*) and (*iii*) until the system ceases to improve, or some stopping criterion is reached.

Each member (which corresponds to a solution to the problem) of the population is represented by a fixed length coded string. Different coding schemes are possible, but binary coding is the most common. Selection or reproduction creates the population for the next generation using a probabilistic selection process which gives a string with higher fitness a greater chance of selection. Mutation corresponds to random flipping of one or more bits of an individual string. Mutation increases the diversity in the population and ensures that the probability of attaining any point in the search space is greater than zero. The simplest implementation of crossover selects two parents (randomly) from the mating pool and then after choosing a random position each parent string exchanges its tail at that position. The resulting offsprings form the subsequent population for the next generation.

SOFT COMPUTING: HOW IT IS ACHIEVED

As mentioned before, soft computing is an essential ingredient of RWCS. The role of different soft computational tools and of their hybridization in the development of RWCS is obvious. So far we have discussed various tools for soft computing, their objectives, and how they can be used, but not mentioned explicitly how each tool contributes towards achieving the goals of soft computing.

Role of NN

We talked about several networks, but all of them do not have the same soft computational features to the same extent. For example, in a Hopfield network, the processing elements can work in parallel and can solve a precisely formulated optimization problem like the travelling salesman problem. Besides parallelism, the network is robust because of its massive connectivity. On the other hand, all neurons in an MLP cannot work in parallel. Nodes in a layer can work in parallel, but between layers a precedence must be maintained - layer i can work only after layer $(i-1)$ has finished computation. An MLP can solve approximately several complex problems. Most system identification problems can be reduced to finding the relationship (at least approximately) between a set of input-output data. An MLP can do it very effectively. Other conventional tools like regression (linear/nonlinear) can also provide an approximate solution but they are not viewed as soft computation tools because usually they are not adaptive, and the level of approximation that can be achieved is strongly dependent on the functional form that one assumes. For an MLP, we do not have to care about any explicit functional form. Just by choosing a large network, followed by a network pruning one can easily find a good solution. On the other hand, for an ART net, the most important characteristics, which are related to soft computation, are the stability and plasticity properties of the structures (clusters) that the network finds. The plasticity property enables the system to learn new events, thereby making it more flexible. The stability to irrelevant events makes the system robust. Conventional clustering methods, even the LVQ network, do not have these characteristics.

The massive connectivity of NN also provides a kind of tolerance to uncertainty arising from system malfunctioning. For example, in a Hopfield network or in a multilayer network, like the one used by Ghosh *et al* [29]; the malfunctioning of some neurons or destruction of some of the links gracefully alters the collective behaviour of the system [30]. This type of ruggedness is not expected from LVQ and ART as they do not have massive connectivity in their achitectures.

Role of FL

Let us now explain the role of fuzzy sets in soft computing paradigm. As mentioned before, the distinguishing feature of fuzzy logic is that it gives the emergence of the basic concept of soft computing as a key ingredient to realize high machine IQ (MIQ). Let us consider the problem of target identification from remotely sensed images, where the impreciseness in a target arises from both grayness ambiguity and spatial ambiguity. Remotely sensed images are normally poorly illuminated, dependent on environmental conditions and have low spatial resolution. Often a scene contains many objects which are ill-defined because of grayness and spatial ambiguity. Moreover, the gray value assigned to a pixel is the average reflectance of different types of ground covers present in the corresponding area. Consequently assigning a crisp label to every pixel is not only difficult, it may be conceptually incorrect also.

In conventional decision making systems, after segmentation of the image each pixel is assigned a crisp label representing a particular type of object. Such a process may assign incorrect labels to pixels that correspond to areas with different types of ground covers (*ie*, border areas of different regions). Consequently, the final decision from the system may be erroneous. Fuzzy sets can be effectively used to model different

types of objects or ground covers and a fuzzy segmentation algorithm can be used to partition the image. Under this framework, each pixel may partially belong to more than one class. These fuzzy membership values may then be integrated at higher levels with other evidences to result in better decisions.

To further illustrate the relevance of FL in soft computing, let us consider an example from the AI domain. A rule based expert system is driven by the rules (knowledge) extracted from experts. Expert's rules are of the form "If the temperature of the patient is *High* and the blood pressure is *High* then the person is suffering form diabetes. Fuzzy set theoretic tools can be used to model and reason for such systems. Π-type triangular membership functions, as discussed earlier (Fig 2*b*) can be used to model *High* and a suitable fuzzy implication operator can then be used compute the truth value of such a statement. Thus we see, FL enables us to achieve tolerance to imprecision. It helps us to obtain an approximate solution to a problem. For example, suppose we have a rule "if x is A then y is B". Now if we find that x is A' where A' is nearly equal to A, then the approximate solution B' can be found though fuzzy reasoning.

Role of GAs

Genetic algorithms are also considered to be an important ingredient of soft computing. It is an evolutionary process, in which better and better solutions are generated based on the principle of "survival of the fittest". It is an adaptive process, where the solutions are not seriously affected by initialization and thus it provides robustness to the system. Another important characteristic of GAs that is relevant to soft computing is their parallelism. GAs work on several solutions simultaneously (in parallel). The method is also universal in nature in the sense that it can be applied to any problem as long as a fitness function can be defined for it. The fitness function may not even be differentiable or continuous.

HOW IT CAN BE IMPROVED THROUGH HYBRIDIZATION

We have discussed, so far, the role of individual tool towards soft computing. While the development of this individual tool was in progress, a group of researchers felt the need of integrating them in order to enjoy the merits of different computational techniques into one system. The result is the development of several hybrid computational paradigms, like, neuro-fuzzy, neuro-genetic, fuzzy-genetic and neuro-fuzzy-genetic. These hybrid paradigms are suitable for solving complex real world problems for which only one tool may not be adequate. Let us explain the hybridization between the three major tools of soft computing. A feed-forward neural network can recognize patterns, but the way of making decision is not well explained in the literature. As mentioned before, NNs attempt to model the way brain computes/works; they have the generic advantages of parallelism, fault tolerance and robustness. Fuzzy logic, on the other hand, can model, to a reasonable extent, vagueness present in a system and reason or explain happenings. Thus, a judicious integration of the two approaches may lead to some application specific advantages in addition to their generic merits. GAs are effective parallel stochastic searching techniques and many fuzzy systems or neural systems require finding a set of optimal parameters for their proper functioning. Therefore, the integration of any two or all the three promises to provide more (artificial) intelligent systems having all the generic advantages of each tool, and in some cases system specific advantages. Before we talk about different hybrid systems, let us draw an analogy with hybridization that is involved in biological information processing.

Human beings collect different types of information through different sensory organs and process them to make decisions in everyday life. Each sensory organ is efficient in collecting and processing information in its respective domain. Finally, integration of all these information leads to optimal decisions. This is true for any real life complex multi-facet decision making problem also.

Neuro-Fuzzy Systems

Neuro-fuzzy hybridization is done broadly in two ways: a neural network equipped with the capability of handling fuzzy information, and a fuzzy system augmented by neural networks to enhance the flexibility, speed and adaptability of the fuzzy system. The former may be called fuzzy-neural-networks (FNN); whereas the latter is named neural-fuzzy-system (NFS). Both types of integrations are grouped, in the literature, under the popular name neuro-fuzzy computing.

A neural network may be called fuzzy, *ie*, a FNN when either the input signals and/or connection weights and/or the outputs are fuzzy subsets or membership values to some fuzzy sets [31-33]. Usually fuzzy numbers (represented by triangular functions for ease of computations) are used to model fuzzy signals.

Lee and Lee [32] are probably the first to unite the concepts of fuzzy sets and neural networks. The fuzzy neuron of Lee and Lee allows the excitatory and inhibitory inputs, and the outputs to be fuzzy. In other words, it entertains graded input and output. Following this, several models of fuzzy neurons have been proposed [34] and developed [35]. Neural networks with fuzzy neurons fall in the category of FNN as they are capable of processing fuzzy information.

FNN and NFS both usually exhibit some universal advantages that are applicable to all hybrid systems, and some application specific advantages as well. For example, a FNN generally enjoys the generic advantages of NNs and is capable of handling imprecise information. Besides, they may have some additional capabilities which are not available with a pure fuzzy system and/or with a pure neural system. As an illustration, in Ghosh *et al* [29], the multilayer perceptron is converted to an unsupervised network by the introduction of concepts from fuzzy theory. This FNN can extract objects in a noisy environment in a completely unsupervised manner by minimizing a measure of fuzziness computed on the output of the network.

Let us consider another kind of FNN which does not enhance or change the capability of the NN but makes its implementation efficient. It is well known that back-propagation learning is very slow and the choice of the learning parameters (like learning coefficient and momentum factor) is an important factor. If the learning rate is high, the network may oscillate; if it is low, then the convergence may be slow. Neuro-fuzzy hybridization can be done to accelerate the training by an adaptive choice of the learning rate using the fuzzy control paradigm. Let Δe be the change in error and $\Delta \dot{e}$ be change in Δe. We can use fuzzy rule like the following to adapt the learning rate and the momentum factor [36]:

"If Δe is SMALL and $\Delta \dot{e}$ is SMALL and no sign change of error over SEVERAL consecutive iterations, then increase the learning rate and momentum factor a LITTLE."

In a neural-fuzzy system (NFS) designed to realize the process of fuzzy reasoning the connection weights of the network correspond to the parameters of fuzzy reasoning [37, 38]. Using the backpropagation type learning algorithms, an NFS can identify fuzzy rules and learn membership functions of the fuzzy reasoning. Usually for a NFS, it is easy to establish a one-to-one correspondence between the network and the fuzzy system. In other words, the NFS architecture has distinct nodes for antecedent clauses, conjunction operators and consequent clauses. There can be, of course, another black-box type NFS where a multilayer network is used to learn the input-output relation represented by a fuzzy system. For such a system the network structure has no such relation to the architecture of the fuzzy reasoning system.

Genetic-Fuzzy Systems

Apart from the usual merits of parallelism, robustness, GAs are sometimes essential to support another soft computational tool, fuzzy logic. Fusion of fuzzy systems and genetic algorithms can be done for tuning of the former. Consider a fuzzy system (a controller, as an example) with two antecedent variables (x, y) and one consequent variable (u). The fuzzy system consists of rules of the form:

If x is X and y is Y then u is U.

Here X, Y and U are fuzzy sets defined on the linguistic variables x, y, and u respectively. Suppose there are seven fuzzy sets defined on each of the three linguistic variables. So there can be 49×7 possible rules for this system, of which there could be several subsets with at most 49 rules that are consistent in a given implementation. In fact, for most applications, we will not require all the 49 rules. The use of more rules is not only time expensive, but it can also confuse the system and degrade its performance. Thus the selection of a right and adequate subset of rules is an important problem for designing an efficient fuzzy system. The rule selection should be such that it maximizes the performance of the system. The calculus based techniques are difficult (the difficulty being associated with the analytical representation of the objective function) to use for this type of optimization problems. Genetic algorithms have been successfully applied for such problems [39, 40]. Such a fusion of the two technologies, again does not enhance the power of the fuzzy system, but makes it easy to get an optimal design of the same. In this context we mention that the merits of integrating FL with GAs to improve the capability of the latter has not yet been established.

Genetic-Neural Systems

While integrating neural networks and genetic algorithms, the two components may interact in many ways [41, 42]. For example, GA can be used to select the cloning templates of a CNN, again it can help to avoid the tedious back-propagation algorithm for an MLP, thereby overcoming some limitations of neural networks. Such an integrated system may be termed as genetic-neural system. As it stands, the use of NN for improving the performance of GAs is not yet explored.

Neuro-Fuzzy-Genetic Systems

Finally, we can have systems that exploit benefits of all the three soft-computation tools. Such systems may be called neuro-fuzzy-genetic (NFG) systems. For example, a fuzzy reasoning system may be implemented using a multilayer network, where the free parameters of the system may be learnt using GAs. Similarly, the parameters of a FNN may be also learnt using GAs. The benefits of these three soft computational tools on a single system need to be investigated [43].

WHERE IT IS AIMING AT

We have discussed so far what is soft computing, its primary tools, their role and relation with RWC. Over the last few decades, information technology has passed through a revolution. The growth of information technology in terms of computing power ranges from conventional computing (whose kernel is data processing) to RWCS (whose kernel is flexible information processing) via fifth generation computing (whose kernel is knowledge information processing). Thus, as it stands these hybrid paradigms will not only continue to remain in the forefront research area for the coming decades, but also will play a key role in the development of future technology including sixth generation computing systems.

REFERENCES

1. L A Zadeh, Fuzzy logic and soft computing: Issues, contention and perspection, *Proc 3rd Int Conf on Fuzzy Logic, Neural nets and Soft Computing*, Iizuka, Japan, 1994.
2. J Hertz, A Krogh & R G Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City CA, 1991.
3. S Haykin, Neural networks - a comprehensive foundation, *Macmillan College, Proc Con Inc*, NY, 1994.
4. R Rosenfeld & J Anderson, (Ed), *Neuro Computing*, MIT Press, 1988.
5. G J Klir & B Yuan, *Fuzzy Sets and Fuzzy Logic - Theory and applications*, Prentice Hall PTR, NJ, 1995.
6. J J Hopfield & D W Tank, Neural computation of decisions in optimization problems, *Bio Cybern*, vol 52, pp 141-152, 1985.

7. S Fahlman, Fast-learning variations on back-popagation: An emperical study, *Proc Connectionist Models Summer School*, 1988.

8. T Kohonen, *Self-Organization and Associative Memory*, 3rd Edition, Springer Verlag, Berlin, 1989.

9. H Ritter & K Schulten, On the stationary states of Kohonen's self-organizing sensory mapping, *Biological Cybern*, vol 54, pp 99-106, 1986.

10. T Kohonen, The self-organized map, *Proc IEEE*, vol 78, no 9, pp 1464-1480, 1990.

11. J A Kangas, T Kohonen & J T Laaksonen, Variants of self-organizing maps, *IEEE Trans Neural Networks*, vol 1, no 1, pp 93-99, 1990.

12. E C K Tsao, J C Bezdek & N R Pal, Fuzzy Kohonen Clustering Networks, *Pattern Recognition*, vol 27, no 5, pp 757-764, 1994.

13. N R Pal, J C Bezdek & E Tsao, Generalized Clustering Networks and Kohonen's Self-Organizing Scheme, *IEEE Trans Neural Networks*, vol 4, no 4, pp 549-558, 1993.

14. E Yair, K Zeger & A Gersho, Competitive Learning and Soft Competition for Vector Quantizer design, *IEEE Trans Signal Processing*, vol 4, no 2, pp 294-309, 1992.

15. J C Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum, New York, 1981.

16. G A Carpenter & S Grossberg, The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network, *Computer*, pp 77-88, March 1988.

17. D R Hush & B G Horne, Progress in Supervised Neural Networks, *IEEE Signal Processing Magazine*, pp 8-38, 1993.

18. L O Chua & L Yang, Cellular Neural Networks: Theory, *IEEE Trans Circuits and Systems*, vol 35, pp 1257-1272, 1988.

19. K J Lang, A H Waibel & G E Hinton, A Time-delay Neural Network Architecture for Isolated Word Recognition, *Neural Networks*, vol 3, no 1, pp 23-44, 1990.

20. E Oja, Principal Components, Minor Components and Neural Networks, *Neural Networks*, vol 5, pp 927-936, 1992.

21. J Basak & S K Pal, X-tron: An Incremental connectionist Model for Category Perception, *IEEE Trans Neural Networks*, vol 6, no 5, pp 1091-1108, 1995.

22. K Fukushima, Neocognitron: A Self-Organizing Multilayer Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, *Biological Cybernetics*, vol 36, pp 193-202, 1980.

23. L A Zadeh, Fuzzy Sets, *Information and Control*, vol 8, pp 338-353, 1965.

24. J C Bezdek & S K Pal, *Fuzzy Models for Pattern Recognition*, IEEE Press, NJ, 1992.

25. D Driankov, H Hellendoorn & M Reinfrank, *An Introduction to Fuzzy Control*, Springer-Verlag, New York, 1993.

26. L Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

27. David E Goldberg, *Genetic Algorithms: Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989.

28. S K Pal & P P Wang (Eds), *Genetic Algorithms for Pattern Recognition*, CRC Press, Boca Raton, 1996.

29. A Ghosh, N R Pal & S K Pal, Self-organization for Object Extraction using Multilayer Neural Network and Fuzziness Measures, *IEEE Trans Fuzzy Systems*, vol 1, pp 54-68, 1993.

30. A Ghosh, N R Pal & S K Pal, Modeling of component failure in neural networks for robustness evaluation: An application to object extraction, *IEEE Trans Neural Nets*, vol 6, pp 648-656, 1995.

31. J J Buckley & Y Hayashi, in *Fuzzy Neural Networks, Fuzzy Sets, Neural Networks, and Soft Computing*, R R Yager & L A Zadeh (Eds), Van Nostrand Reinhold, NY, pp 233-249, 1994.

32. S C Lee & E T Lee, Fuzzy Neural Networks, *Math Bio Sc*, vol 23, pp 151-177, 1975.

33. S K Pal & S Mitra, Multilayer Perceptron, Fuzzy Sets and Classification, *IEEE Trans Neural Networks*, vol 3, no 5, pp 683-697, 1992.

34. M M Gupta & J Qi, On Fuzzy Neuron Models, *Proc Int Joint Conf on Neural Networks (IJCNN)*, Seattle, II, pp 431-436, 1991.

35. T Yamakawa, Pattern Recognition Hardware System Employing a Fuzzy Neuron, *Proc Int Conf on Fuzzy Logic*, Iizuka, pp 943-948, 1990.

36. J J Choi, R J Arabshahi, R J Marks II & T P Caudell, Fuzzy Parameter Adaptation in Neural Systems, *Int Conf on Neural Networks*, vol 1, pp 232-238, MD, 1992.

37. J Keller & H Tahani, Implementation of Conjunctive and Disjunctive Fuzzy Logic Rules in Neural Networks, *Int Jour Approx Reasoning*, vol 6, pp 221-240, 1992.

38. J Keller, R Yager & H Tahani, Neural Network Implementation of Fuzzy Logic, *Fuzzy Sets and Systems*, vol 45, pp 1-12, 1992.

39. C L Karr & E J Gentry, Fuzzy Control of pH using Genetic Algorithms, *IEEE Trans, Fuzzy Systems*, vol 1, no 1, pp 46-53, 1993.

40. A Homaifar & E McCormick, Simultaneous Design of Membership functions and Rule Sets for Fuzzy Controllers using Genetic Algorithms, *IEEE Trans, Fuzzy Systems*, vol 3, no 2, pp 129-139, 1995.

41. H Muhlenbein, Limitations of Multi-layer Perceptrons Networks-Steps Towards Genetic Neural Networks, *Parallel Computing*, vol 14, pp 249-260, 1990.

42. S K Pal, D Bhandari, P Harish & M K Kundu, Cellular neural networks, genetic algorithms and Object Extractrion, *Far East Jour of Mathematical Sciences*, vol 1, no 2, pp 139-155, 1993.

43. S K Pal & D Bhandari, Genetic Algorithms with Fuzzy Fitness Function for Object Extraction using Cellular Neural Network, *Fuzzy Sets and Systems*, vol 65, pp 129-139, 1994.