

Introduction to Fuzzy Systems, Neural Networks, and Genetic Algorithms

Hideyuki TAKAGI *
Kyushu Institute of Design

1 INTRODUCTION

Soft Computing technologies are the main topics of this book. This chapter provides the basic knowledge of fuzzy systems (FSs), neural networks (NNs), and genetic algorithms (GAs). Readers who have already studied these technologies may skip the appropriate sections.

To understand the functions of FSs, NNs, and GAs, one needs to imagine a multi-dimensional input–output space or searching space. Figure 1 is an example of such a space.

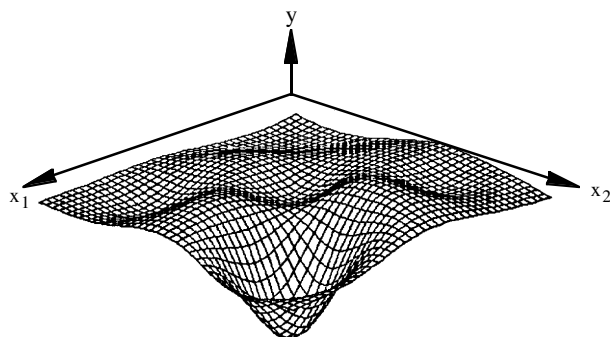


Figure 1: Example of a multi-dimensional space.

Suppose this space is a two-input and one-output space. FSs and NNs can form this nonlinear input–output relation. They realize the complex nonlinearity by combining multiple simple functions. The FS separates the space into several rule areas whose partial shapes are determined by membership functions and rule output. The NNs form the shape by combining sigmoidal, radial, or other simple functions that are enlarged, shrunked, upset, and/or shifted by synaptic weights. A simple example is shown in section 3.4.

4-9-1, Shiobaru, Minami-ku, Fukuoka, 815-8540 Japan,
TEL&FAX +81-92-553-4555, E-mail: takagi@kyushu-
id.ac.jp, URL: <http://www.kyushu-id.ac.jp/takagi>

⁰ This article appeared in *Intelligent Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms*, Ch.1, pp.1–33, edited by D. Ruan, Kluwer Academic Publishers (Norwell, Massachusetts, USA), (September, 1997).

Suppose, however, Figure 1 is a searching space. Then, the vertical axis shows evaluation values, such as error values for the NNs and fitness values for the GAs. The NNs and GAs determine the best evaluation position in the (x_1, x_2) searching space.

This chapter introduces the basic knowledge of these three technologies and provides an overview of how these technologies are cooperatively combined and have been applied in the real world.

2 WHAT ARE FUZZY SYSTEMS

2.1 Fuzzy theory and systems

Fuzzy sets are the basic concept supporting fuzzy theory. The main research fields in fuzzy theory are fuzzy sets, fuzzy logic, and fuzzy measure. Fuzzy reasoning or approximate reasoning is an application of fuzzy logic to knowledge processing. Fuzzy control is an application of fuzzy reasoning to control.

Although most applications of fuzzy theory have been biased toward engineering, these applications have recently reached other disciplines, such as medical diagnostics, psychology, education, economy, management, sociology, etc. The number of fuzzy applications in the field of *KANSEI* — a synthetic concept of emotion, impression, intuition, and other human subjective factors — has especially increased in Japanese fuzzy society.

It is not within the scope of this chapter to provide an overview for every aspect of fuzzy theory. We will focus on a fuzzy controller as an example of a simple FS to see how the output of the FS is calculated by fuzzy rules and reasoning.

2.2 Aspects of fuzzy systems

One feature of FSs is the ability to realize a complex nonlinear input–output relation as a synthesis of multiple simple input–output relations. This idea is similar to that of NNs (compare with Figure 11.) The simple input–output relation is described in each rule. The boundary of the rule areas is not sharp but ‘fuzzy.’ It is like an expanding sponge soaking up water. The system output from one rule area to the next rule area gradually changes. This is the essential idea of FSs and the origin of the term ‘fuzzy.’

Another feature of FSs is the ability to separate logic and fuzziness. Since conventional two-value logic-based systems cannot do this, their rules are modified when either logic or fuzziness should be changed. The FSs modify fuzzy rules when logic should be changed and modify membership functions which define fuzziness when fuzziness should be changed.

Suppose that the performance of an inverted pendulum controller is imperfect. Define a and \dot{a} as the angle between a pole in right side and vertical line and its angular velocity, respectively.

IF a is *positive big* and \dot{a} is *big*, THEN move a car to the right *quickly*. and

IF a is *negative small* and \dot{a} is *small*, THEN move a car to the left *slowly*.

are correct logic, and you need not change the fuzzy rules themselves. Only the definition of fuzziness must be modified in this case: *big*, *small*, *quickly*, *slowly*, and so on. On the other hand, two-value logic rules, such as

IF $40^\circ \leq a \leq 60^\circ$ and $50^\circ/\text{s} \leq \dot{a} \leq 80^\circ/\text{s}$, THEN move a car with 0.5 m/s. or

IF $-20^\circ \leq a \leq -10^\circ$ and $10^\circ/\text{s} \leq \dot{a} \leq 20^\circ/\text{s}$, THEN move a car with -0.1 m/s.

must be modified whenever the rules or the quantitative definitions of angle, angular velocity, and car speed are changed.

2.3 Mathematical model-based control and rule-based control

To understand FSs, fuzzy logic control will be used as a simple example of the FSs in this and the following sections. The task is to replace a skilled human operator in the Figure 2 with a controller.

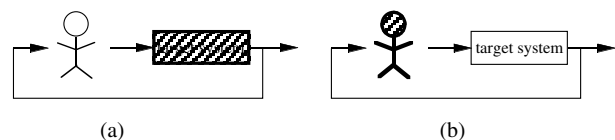


Figure 2: (a) Conventional control theory tries to make a mathematical model of a target system. (b) Rule-based control tries to make a model of a skilled human operator.

The mathematical model-based control approach is based on classic and modern control theory and is designed to observe the characteristics of the target system, construct its mathematical model, and build a controller based on the model. The importance is placed on the target system and the human operator is not a factor in the design (Figure 2(a).)

On the other hand, rule-based control does not utilize the target system in modeling but is based on the behavior of a skilled human operator (Figure 2(b).) Although most skilled operators do not know the mathematical behavior of their target systems, they can control their systems. For example, a skilled taxi driver probably does not know the mathematical equations of car behavior when his/her taxi turns to the right up an unpaved hill, but he/she can still handle the car safely. A fuzzy logic controller describes the control behavior of the skilled human operator using IF-THEN type of fuzzy rules.

2.4 Design of antecedent parts

Designing antecedent parts means deciding how to partition an input space. Most rule-based systems assume that all input variables are independent and partition the input space of each variable (see Figure 3.) This assumption makes it easy to not only partition the input space but also interpret partitioned areas into linguistic rules. For example, the rule of “IF temperature is A_1 and humidity is A_2 , THEN ...” is easy to understand, because the variables of temperature and humidity are separated. Some researchers propose multi-dimensional membership functions that aim for higher performance by avoiding the constraint of variable independence instead of linguistic transparency.

The difference between crisp and fuzzy rule-based systems is how the input space is partitioned (compare Figure 3(a) with (b).) The idea of FSs is based on the premise that in our real analog world, change is not catastrophic but gradual in nature. Fuzzy systems, then, allow overlapping rule areas to shift from one control rule to another. The degree of this overlapping is defined by membership functions. The *gradual* characteristics allow smooth fuzzy control.

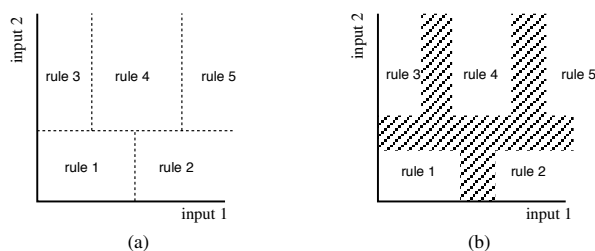


Figure 3: Rule partition of an input space: (a) partition for crisp rules and (b) partition for fuzzy rules.

2.5 Design of consequent parts

The next step following the partitioning of an input space is deciding the control value of each rule area.

Fuzzy models are categorized into three models according to the expressions of consequent parts:

- (1) Mamdani model: $y = A$
(A is a fuzzy number.)
- (2) TSK model: $y = a_0 + \sum a_i x_i$
(a_i is a constant, and x_i is an input variable.)
- (3) simplified fuzzy model $y = c$
(c is a constant.)

The Mamdani type of FS has a fuzzy variable defined by a membership function in their consequents, such as $y = \textit{big}$ or $y = \textit{negative small}$, which was used in the first historical fuzzy control [(Mamdani, 1974)]. Although it is more difficult to analyze this type of FS than a FS whose consequents are numerically defined, it is easier for this FS to describe qualitative knowledge in the consequent parts. The Mamdani type of FS seems to be suitable for knowledge processing expert systems rather than for control expert systems.

The consequents of the TSK (Takagi-Sugeno-Kang) models are expressed by the linear combination of weighted input variables [(Takagi&Sugeno, 1985)]. It is possible to expand the linear combination to nonlinear combination of input variables; for example, fuzzy rules which have NNs in their consequents [(Takagi&Hayashi, 1991)]. In this case, there is a tradeoff between system performance and the transparency of rules. The TSK models are frequently used in fuzzy control fields as well as the following simplified fuzzy models.

The simplified fuzzy model has fuzzy rules whose consequents are expressed by constant values. This model is the special case of both the Mamdani type of FSs and the TSK models. Even if each rule output is constant, the output of the whole FS is nonlinear, because the characteristics of membership functions are embedded into the system output. The biggest advantage of the simplified fuzzy models is that the models are easy to design. It is reported that this model is equivalent to Mamdani's model [(Mizumoto, 1996)].

2.6 Fuzzy reasoning and aggregation

Now that the IF and THEN parts have been designed, the next stage is to determine the final system output from the designed multiple fuzzy rules. There are two steps: (1) determination of rule strengths and (2) aggregations of each rule output.

The first step is to determine rule strengths, meaning how active or reliable each rule is. Antecedents include multiple input variables:

IF $x_1 \in \mu_1$ and $x_2 \in \mu_2$ and ... $x_k \in \mu_k$, THEN ...

In this case, one fuzzy rule has k membership values: $\mu_i(x_i)$ ($i = 1, \dots, k$). We need to determine how active a rule is, or its strength, from the k membership values. The class of the fuzzy operators used

for this purpose is called t -norm operator. There are many operators in t -norm category. One of the most frequently used t -norm operators is an algebraic product: rule strength $w_i = \prod_{j=1}^k \mu_j(x_j)$. The \textit{min} operator that Mamdani used in his first fuzzy control is also frequently introduced in fuzzy textbooks: rule strength $w_i = \textit{min}(\mu_j(x_j))$.

The final system output, y^* , is calculated by weighting each rule output with the obtained rule strength, w_i : $y^* = \sum w_i y_i / \sum w_i$. Mamdani type of fuzzy controllers defuzzify the aggregated system output and determine the final non-fuzzy control value.

Figure 4 is an example of a simple FS that has four fuzzy rules. The first rule is "IF x_1 is *small* and x_2 is *small*, THEN $y = 3x_1 + 2x_2 - 4$." Suppose there is an input vector: $(x_1, x_2) = (10., 0.5)$. Then, membership values are calculated. The first rule has membership values, 0.8 and 0.3, for the input values. The second one has 0.8 and 1.0. If the algebra product is used as a t -norm operator, then the rule strength of the first rule is $0.8 \times 0.3 = 0.24$. The rule strengths of the second, third, and fourth rules are 0.8, 1.0, and 0.3, respectively. If \textit{min} operator is used, the rule strength of the first rule is $\textit{min}(0.8, 0.3) = 0.3$. The output of each rule for the input vector, $(10., 0.5)$, is 27, 23.5, -9 , and -20.5 , respectively. Therefore, the final system output, y^* , is given as:

$$\begin{aligned}
 y^* &= \frac{\sum w_i y_i}{\sum w_i} \\
 &= \frac{0.24 \times 27 + 0.8 \times 23.5 + 1.0 \times (-9) + 0.3 \times (-20.5)}{0.24 + 0.8 + 1.0 + 0.3} \\
 &\cong 4.33
 \end{aligned}$$

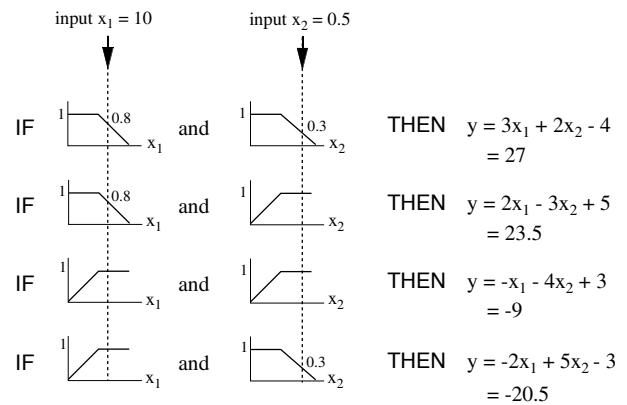


Figure 4: Example aggregation of TSK model.

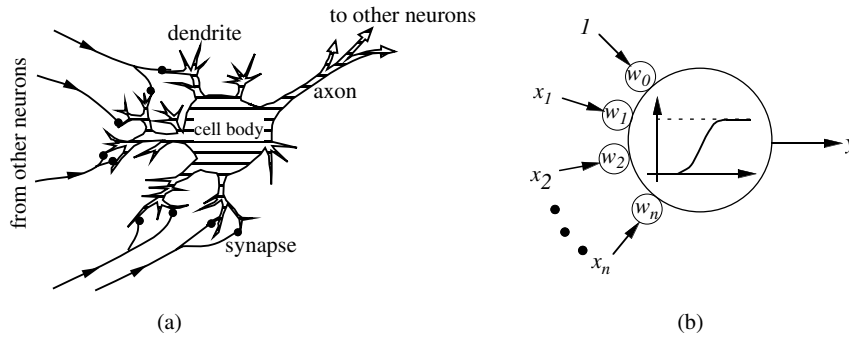


Figure 5: A biological neuron and an artificial neuron model.

3 WHAT ARE NEURAL NETWORKS

3.1 Analogy from biological neural networks

A biological neuron consists of dendrite, a cell body, and an axon (Figure 5(a)). The connections between the dendrite and the axons of other neurons are called synapses. Electric pulses coming from other neurons are translated into chemical information at each synapse. The cell body inputs these pieces of information and fires an electric pulse if the sum of the inputs exceeds a certain threshold. The network consisting of these neurons is a NN, the most essential part of our brain activity.

The main function of the biological neuron is to output pulses according to the sum of multiple signals from other neurons with the characteristics of a pseudo-step function. The second function of the neuron is to change the transmission rate at the synapses to optimize the whole network.

An artificial neuron model simulates multiple inputs and one output, the switching function of input-output relation, and the adaptive synaptic weights (Figure 5(b)). The first neuron model proposed in 1943 used a step function for the switching function [(McCulloch&Pitts, 1943)]. However, the perceptron [(Rosenblatt, 1958)] that is a NN consisting of this type of neuron has limited capability, because of the constraints of binary on/off signals. Today, several continuous functions, such as sigmoidal or radial functions, are used as a neuron characteristic functions, which results higher performance of NNs.

Several learning algorithms that change the synaptic weights have been proposed. The combination of the artificial NNs and the learning algorithms have been applied to several engineering purposes.

3.2 Several types of artificial neural networks

Many NN models and learning algorithms have been proposed. Typical network structures include feed-

back and feed-forward NNs. Learning algorithms are categorized into supervised learning and unsupervised learning. This section provides an overview of these models and algorithms.

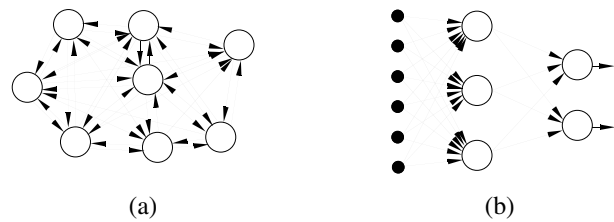


Figure 6: (a) a feed-back neural network, and (b) a feed-forward neural network.

The feed-back networks are NNs that have connections between network outputs and some or all other neuron units (see Figure 6(a).) Certain unit outputs in the figure are used as activated inputs to the network, and other unit outputs are used as network outputs.

Due to the feed-back, there is no guarantee that the networks become stable. Some networks converge to one stable point, other networks become limit-cycle, and others become chaotic or divergent. These characteristics are common to all non-linear systems which have feed-back.

To guarantee stability, constraints on synaptic weights are introduced so that the dynamics of the feed-back NN is expressed by the Lyapunov function. Concretely, a constraint of equivalent mutual connection weights of two units is implemented. The Hopfield network is one such NNs.

It is important to understand two aspects of the Hopfield network: (1) Synaptic weights are determined by analytically solving constraints not by per-

forming an iterative learning process. The weights are fixed during the Hopfield network runs. (2) Final network outputs are obtained by running feed-back networks for the solutions of an application task.

Another type of NN which is compared with the feed-back type is a feed-forward type. The feed-forward network is a filter which outputs the processed input signal. Several algorithms determine synaptic weights to make the outputs match the desired result.

Supervised learning algorithms adjust synaptic weights using input–output data to match the input–output characteristics of a network to desired characteristics. The most frequently used algorithm, the backpropagation algorithm, is explained in detail in the next section.

Unsupervised learning algorithms use the mechanism that changes synaptic weight values according to the input values to the network, unlike supervised learning which changes the weights according to supervised data for the output of the network. Since the output characteristics are determined by the NN itself, this mechanism is called *self-organization*. Hebbian learning and competitive learning are representative of unsupervised learning algorithms.

A Hebbian learning algorithm increases a weight, w_i , between a neuron and an input, x_i , if the neuron, y , fires.

$$\Delta w_i = ax_i,$$

where a is a learning rate. Any weights are strengthened if units connected with the weights are activated. Weights are normalized to prevent an infinite increase in weights.

Competitive learning algorithms modify weights to generate one unit with the greatest output. Some variations of the algorithm also modify other weights by lateral inhibition to suppress the outputs of other units whose outputs are not the greatest. Since only one unit becomes active as the winner of the competition, the unit or the network is called a *winner-take-all* unit or network. Kohonen’s self-organization feature map, one of the most well-known competitive NNs, modifies the weights connected to the winner-take-all unit as:

$$\Delta w_i = a(x_i - w_i),$$

where the sum of input vectors is supposed to be normalized as 1.

3.3 Feed-forward NN and the backpropagation learning algorithm

Signal flow of a feed-forward NN is unidirectional from input to output units. Figure 8 shows a numerical example of the data flow of a feed-forward NN.

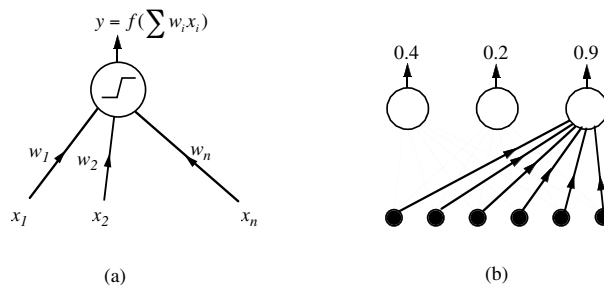


Figure 7: (a) Hebbian learning algorithms strength weight, w_i when input, x_i activates a neuron, y . (b) Competitive learning algorithms strength only weights connected to the unit whose output is the biggest.

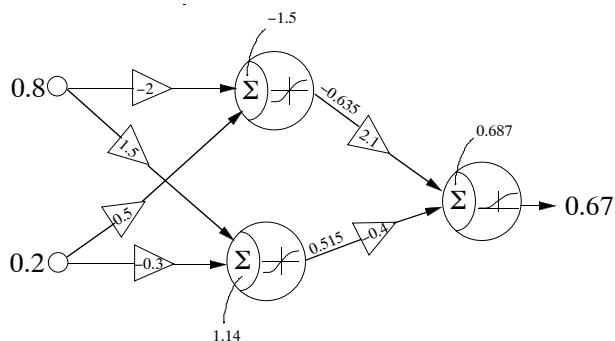


Figure 8: Example data flow in a simple feed-forward neural network.

One of the most popular learning algorithms which iteratively determines the weights of the feed-forward NNs is the backpropagation algorithm. A simple learning algorithm that modifies the weights between output and hidden layers is called a delta rule. The backpropagation algorithm is an extension of the delta rule that can train the weights, not only between output and hidden layers but also hidden and input layers. Historically, several researchers have proposed this idea independently: S. Amari in 1967, A. Bryson and Y-C. Ho in 1969, P. Werbos in 1974, D. Parker in 1984, etc. Eventually, Rumelhart, et al. and the PDP group developed practical techniques that gave us a powerful engineering tool [(Rumelhart, et al., 1986)].

Let E be an error between the NN outputs, \mathbf{v}^3 , and supervised data, \mathbf{y} . The number at superposition means the layer number. Since NN outputs are changed when synaptic weights are modified, the E

must be a function of the synaptic weights \mathbf{w} :

$$E(\mathbf{w}) = \frac{1}{2} \sum_{j=1}^{N_k} (v_j^3 - y_j)^2.$$

Supposed that, in Figure 1, the vertical axis is E and the $x_1 \dots x_n$ axes are the weights, $w_1 \dots w_n$. Then, NN learning is to find the global minimum coordinate in the surface of the figure.

Since E is a function of \mathbf{w} , the searching direction of the smaller error point is obtained by calculating a partial differential. This technique is called the gradient method, and the steepest decent method is the base of the backpropagation algorithm. The searching direction, $\mathbf{g} = -\partial E(\mathbf{w})/\partial \mathbf{w}$, and modification of weights is given as $\Delta \mathbf{w} = \epsilon \mathbf{g}$. From this equation, we finally obtain the following backpropagation algorithm.

$$\Delta w_{i,j}^{k-1,k} = -\epsilon d_j^k v_i^{k-1}$$

$$d_j^k = \begin{cases} (v_j^3 - y_j) \frac{\partial f(U_j^k)}{\partial U_j^k} & \text{for output layer} \\ \sum_{h=1}^{N_{k+1}} d_h^{k+1} w_{i,h}^{k,k+1} \frac{\partial f(U_j^k)}{\partial U_j^k} & \text{for hidden layer(s)} \end{cases}$$

where $w_{i,j}^{k-1,k}$ is the connection weight between the i -th unit in the $(k-1)$ -th layer and the j -th unit in the k -th layer, and U_j^k is the total amount of input to the j -th unit at k -th layer.

To calculate d_j^k , d_j^{k+1} must be previously calculated. Since the calculation must be conducted in the order of the direction from the output layer to input layer, this algorithm is named the *backpropagation* algorithm.

When a sigmoidal function is used for the characteristic function, $f()$, of neuron units, the calculation of the algorithm becomes simple.

$$f(x) = \frac{1}{1 + \exp^{-x+T}}$$

$$\frac{\partial f(x)}{\partial x} = (1 - f(x))f(x)$$

Figure 9 illustrates the backpropagation algorithm.

3.4 Function approximation

The following analysis of a simple NN that has one input, four hidden nodes, and one output will demonstrate how NNs approximate the nonlinear relationship between input and outputs (Figure 10). The '1's in the figure are offset terms.

Figure 11(a1) – (a5) shows the input–output characteristics of a simple NN during a training epoch, where triangular points are trained data, and horizontal and vertical axes are input and output ones. After 480 iterations of training, the NN has learned the nonlinear function that passes through all training data points.

As a model of the inside of the NN, Figure 11(b1) – (b4) shows the output of four units in the hidden

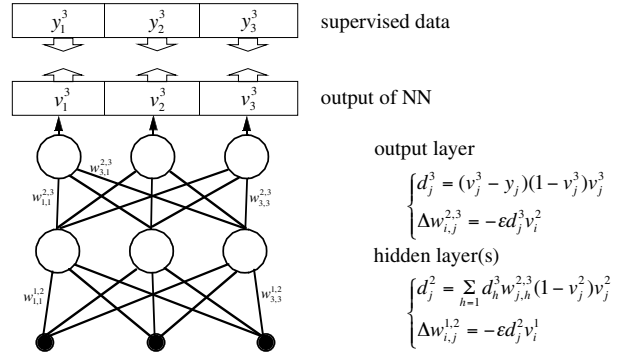


Figure 9: Visual-aid of understanding the programming backpropagation algorithm.

layer. For example, the unit (b1) has the synaptic weight of -22.3 and -16.1 between the unit and the input layer and outputs $f()$ whose input is $-22.3x - 16.1$.

One can understand how the NN forms the final output characteristics visually when the four outputs of the hidden layer units with the final output characteristics are displayed (see Figure 11(c1) and (c2).) The output characteristics of the NN consist of four sigmoidal functions whose amplitudes and center positions are changed by synaptic weights. Thus, NNs can form any nonlinear function with any precision by theoretically increasing the number of hidden units. It is important to note that a learning algorithm cannot always determine the best combination of weights.

4 WHAT ARE GENETIC ALGORITHMS

4.1 Evolutionary computation

Searching or optimizing algorithms inspired by biological evolution are called *evolutionary computa-*

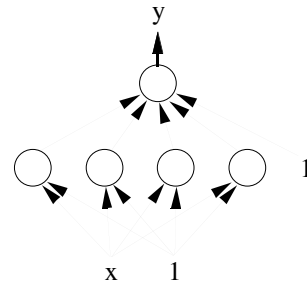


Figure 10: Simple neural network to analyze nonlinear function approximation.

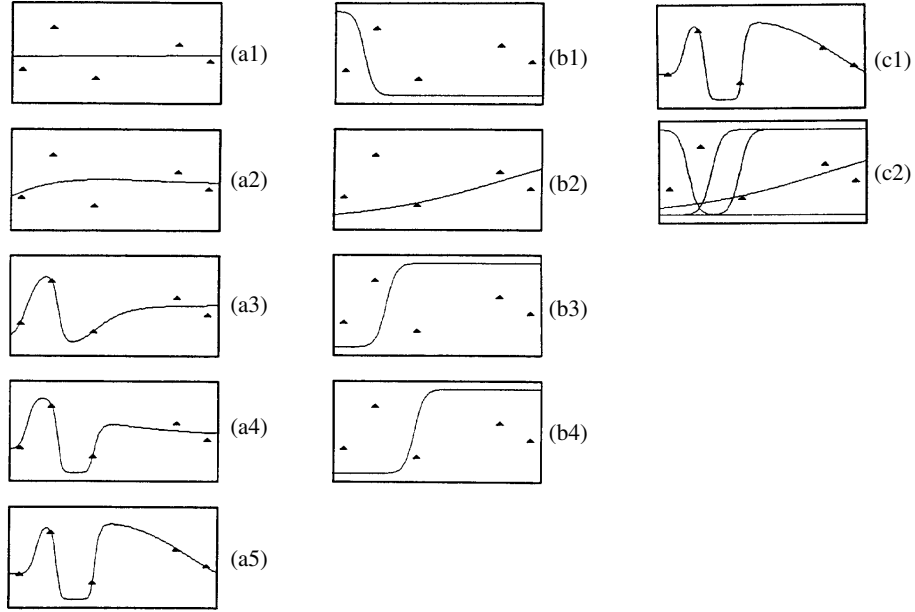


Figure 11: Analysis of NN inside. Triangular points are training data, and horizontal and vertical axes are input and output axes. (a1) – (a5) are the input–output characteristics of a NN with the training of 10, 100, 200, 400, and 500 iterations, respectively. (b1) – (b4), the characteristics of four trained sigmoidal functions in a hidden layer, are $f(-22.3x - 16.1)$, $f(-1.49x - 0.9)$, $f(-20.7x + 10.3)$, and $f(-21.5x + 4.9)$, respectively; where $w_1x + w_0$ is trained weights, w_i , and input variable, x . (c1) is the same as (a5): the input–output characteristics of the trained NN. (c2) is the overlapped display of (b1) – (b4). Comparison of (c1) and (c2) shows the final NN input–output characteristics are formed by combining sigmoidal functions inside with weighting the function.

tions. The features of the evolutionary computation are that its search or optimization is conducted (1) based on multiple searching points or solution candidates (population based search), (2) using operations inspired by biological evolution, such as crossover and mutation, (3) based on probabilistic search and probabilistic operations, and (4) using little information of searching space, such as differential information mentioned in section 3.3. Typical paradigms which consist of the evolutionary computation include GA (genetic algorithm), ES (evolution strategies), EP (evolutionary programming), and GP (genetic programming).

GAs usually represent solutions for chromosomes with bit coding (genotype) and searches for the better solution candidates in the genotype space using GA operations of selection, crossover, and mutation. The crossover operation is the dominant operator.

ESs represent solutions as expressed by the chromosomes with real number coding (phenotype) and searches for the better solution in the phenotype space using the ES operations of crossover and mutation. The mutation of a real number is realized

by adding Gaussian noise, and ES controls the parameters of a Gaussian distribution allowing it to converge to a global optimum.

EPs are similar to GAs. The primary difference is that mutation is the only EP operator. EPs use real number coding, and the mutation sometimes changes the structure (length) of EP code. It is said that the similarities and differences come from their background; GAs started from the simulation of genetic evolution, while EPs started from that of environmental evolution.

GPs use tree structure coding to represent a computer program or create new structures of tasks. The crossover operation is not for a numerical value but for a branch of the tree structure. Consider the application’s relationship with NNs. GAs and ESs determine the best synaptic weights, which is NN learning. GP, however, determines the best NN structure, which is a NN configuration.

It is beyond the scope of this chapter to describe these paradigms. We will focus only on GAs in the following sections and see how the GA searches for solutions.

Table 1: Technical terms used in GA literatures

chromosome	vector which represents solutions of application task
gene	each solution which consists of a chromosome
selection	choosing parents' or offsprings' chromosomes for the next generation
individual	each solution vector which is each chromosome
population	total individuals
population size	the number of chromosome
fitness function	a function which evaluates how each solution suitable to the given task
phenotype	expression type of solution values in task world, for example, 'red,' "13 cm", "45.2 kg"
genotype	bit expression type of solution values used in GA search space, for example, "011," "01101."

4.2 GA as a searching method

It is important to be acquainted with the technical terms of GAs. Table 1 lists some of the terms frequently used.

There are advantages and one distinct disadvantage to using GAs as a search method.

The advantages are: (1) fast convergence to near global optimum, (2) superior global searching capability in the space which has complex searching surface, and (3) applicability to the searching space where we cannot use gradient information of the space.

The first and second advantages originate in population-based searching. Figure 12 shows this situation. The gradient method determines the next searching point using the gradient information at the current searching point. On the other hand, the GA determines the multiple next searching points using the evaluation values of multiple current searching points. When only the gradient information is used, the next searching point is strongly influenced by the local geometric information of the current searching points. Sometimes it results in the searching being trapped at a local minima (see Figure 12.) On the contrary, the GA determines the next searching points using the fitness values of the current searching points which are widely spread throughout the searching space, and it has the mutation operator to escape from a local minima. This is why these advantages are realized.

The key disadvantage of the GAs is that its convergence speed near the global optimum becomes slow. The GA search is not based on gradient information but GA operations. There are several proposals to combine the two searching methods.

4.3 GA operations

Figs. 13 and 14 show the flows of GA process and data, respectively. Six possible solutions are expressed in bit code in Figure 14. This is the genotype

expression. The solution expression of the bit code is decoded to values used in an application task. This is phenotype expression. The multiple possible solutions are applied to the application task and evaluated by each. These evaluation values are fitness values. GA feed-backs the fitness values and selects current possible solutions according to their fitness values. They are parent solutions that determine the next searching points. This idea is based on the expectation that better parents can probabilistically generate better offspring. The offspring in the next generation are generated by applying the GA operations, crossover and mutation, to the selected parent solution. This process is iterated until the GA search converges to the required searching level.

The GA operations are explained in the following sections.

4.4 GA operation: selection

Selection is an operation to choose parent solutions. New solution vectors in the next generation are calculated from them.

Since it is expected that better parents generate better offspring, parent solution vectors which have higher fitness values have a higher probability to be selected.

There are several selection methods. Roulette wheel selection is a typical selection method. The probability to be a winner is proportional to the area rate of a chosen number on a roulette wheel. From this analogy, the roulette wheel selection gives the selection probability to individuals in proportion to their fitness values.

The scale of fitness values is not always suitable for the scale of selection probability. Suppose there are a few individuals whose fitness values are very high, and others whose are very low. Then, the few parents are almost always selected and the variety of their offspring becomes small, which results in convergence to a local minimum. Rank-based selection

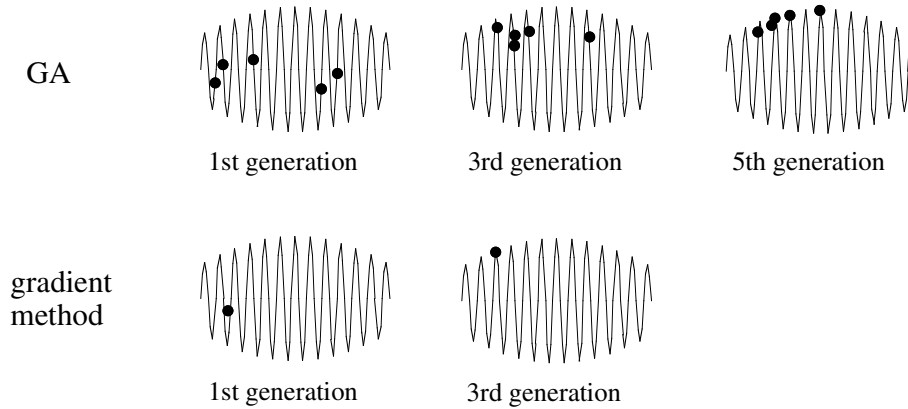


Figure 12: GA search and gradient-based search.

uses an order scale of fitness values instead of a distance scale. For example, it gives the selection probability of (100, 95, 90, 85, 80, ...) for the fitness values of (98, 84, 83, 41, 36, ...).

Since the scales of fitness and selection are different, scaling is needed to calculate selection probabilities from fitness values. The rank-based selection can be called linear scaling. There are several scaling methods, and the scaling influences GA conversion.

Elitist strategy is an approach that copies the best n parents into the next generation as they are. The fitness value of offspring does not always become better than those of its parents. The elitist strategy prevents the best fitness value in the offspring generation from becoming worse than that in the previous generation by copying the best parent(s) to the offspring.

4.5 GA operation: crossover

Crossover is an operation to combine multiple parents and make offspring. The crossover is the most essential operation in GA. There are several ways to combine parent chromosomes. The simplest crossover is called *one-point crossover*. The parent chromosomes are cut at one point, and the cut parts are exchanged. Crossover that uses two cut points is called *two-point crossover*. Their natural expansion is called *multi-point crossover* or *uniform crossover*. Figure 15 shows these standard types of crossover.

There are several variations of crossover. One unique crossover is called the simplex crossover [(Bersini&Scront, 1992)]. The simplex crossover uses two better parents and one poor parent and makes one offspring (the bottom of Figure 15.) When both better parents have the same ‘0’ or ‘1’ at a certain bit position, the offspring copies the bit into the same bit position. When better parents

have different bit at a certain bit position, then a complement bit of the poor parent is copied to the offspring. This is analogous to learning something from bad behavior. The left end bit of the example in Figure 15 is the former case, and the second left bit is the latter case.

4.6 GA operation: mutation

When parent chromosomes have similar bit patterns, the distance between the parents and offspring created by crossover is close in a genotype space. This means that the crossover cannot escape from the local minimum if individuals are concentrated near the local minimum. If the parents in Figure 16 are only the black and white circles, offspring obtained by combining bit strings of any of these parents will be located nearby.

Mutation is an operation to avoid this trapping at a local area by exchanging bits of chromosome. Suppose the white individual jumps to the gray point in the figure. Then, the searching area of GA spreads widely. The mutated point has the second highest fitness value in the figure. If the point that has the highest fitness value and the mutated gray point are selected and mate, then the offspring can be expected to be close to the global optimum.

If the mutation rate is too high, the GA searching becomes a random search, and it becomes difficult to quickly converge to the global optimum.

4.7 Example

The knapsack problem provides a concrete image of GA applications and demonstrates how to use GAs. Figure 17 illustrates the knapsack problem and its GA coding. The knapsack problem is a task to find the optimum combination of goods whose total amount is the closest to the amount of the knapsack,

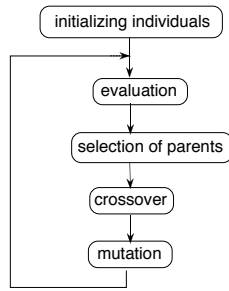


Figure 13: The flow of GA process.

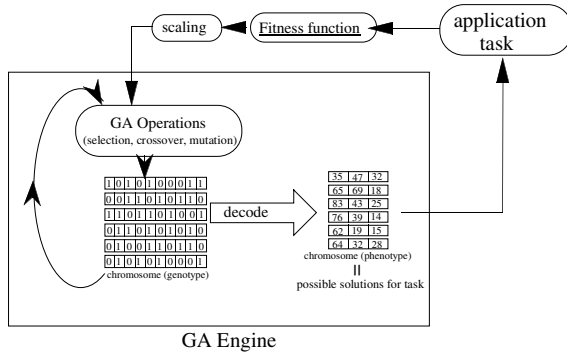


Figure 14: The flow of GA data.

or to find the optimum combination of goods whose total value is the highest under the condition that the total amount of goods is less than that of the knapsack. Although the knapsack problem itself is a toy task, there are several similar practical problems such as the optimization of CPU load under a multi-processing OS.

Since there are six goods in Figure 17 and the task is to decide which goods are input into the knapsack, the chromosome consists of six genes with the length of 1 bit. For example, the first chromosome in the figure means to put A, D, and F into the knapsack. Then, the fitness values of all chromosome are calculated. The total volume of A, D, and F is 60, that of B, E, and F is 53, that of A, D, E, and F is 68, and so on. The fitness values do not need to be the total volume itself, but the fitness function should be a function of the total volume of the input goods. Then, GA operations are applied to the chromosome to make the next generation. When the best solution exceeds the required minimum condition, the searching iteration is ended.

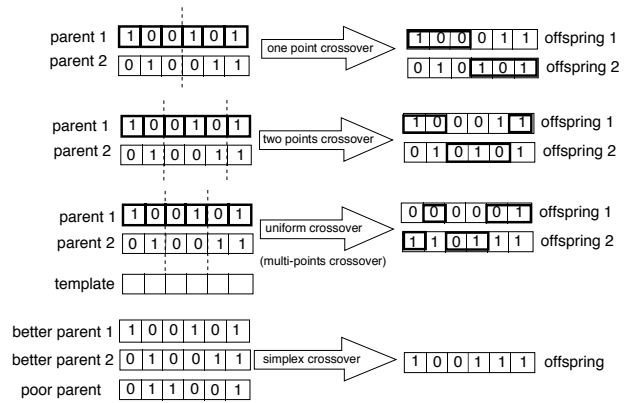


Figure 15: Several variations of crossover.

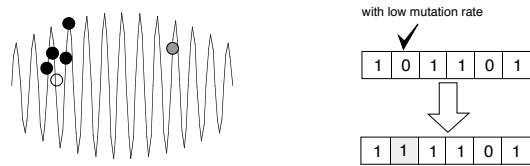


Figure 16: Effect and an example of mutation.

5 MODELS AND APPLICATIONS OF COOPERATIVE SYSTEMS

There are many types of cooperative models of FSs, NNs, and GAs. This section categorizes these types of models and introduces some of their industrial applications. Readers interested in further study of practical applications should refer to detailed survey articles in such references as [(Takagi, 1996), (Yen et al. eds., 1995)].

5.1 Designing FSs using NN or GA

NN-driven fuzzy reasoning is the first model that applies an NN to design the membership functions of an FS explicitly [(Hayashi&Takagi,1988),

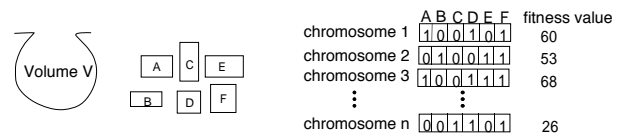


Figure 17: Knapsack problem and example of GA coding.

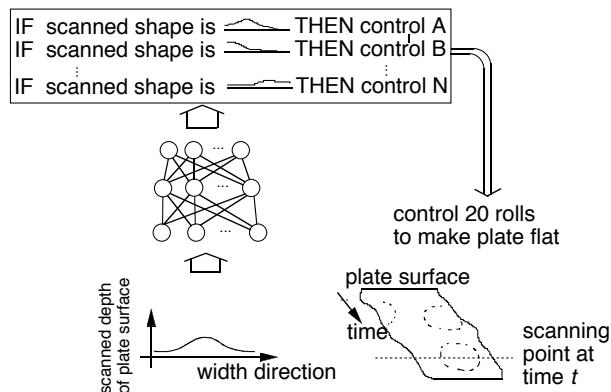


Figure 18: Rolling mill control by fuzzy and neural systems. Scanned pattern of plate surface is recognized by an NN. The output value of the each output unit of the NN is used as a rule strength for the corresponding fuzzy rule.

(Takagi&Hayashi, 1991)]. The purpose of this model is to design the entire shape of non-linear multi-dimensional membership functions with an NN. The outputs of the NN are the rule strengths of each rule which are a combination of membership values in antecedents.

The Hitachi rolling mill uses the model to control 20 rolls that flatten iron, stainless steel, and aluminum plates to a uniform thickness [(Nakajima et al., 1993)]. An NN inputs the scanned surface shape of plate reel and outputs the similarity between the input shape pattern and standard template patterns (see Figure 18). Since there is a fuzzy control rule for each standard surface pattern, the outputs of the NN indicate how each control rule is activated. Dealing with the NN outputs as rule strengths of all fuzzy rules, each control value is weighted, and the final control values of the 20 rolls are obtained to make plate flat at the scanned line.

Another approach parameterizes an FS and tunes the parameters to optimize the FS using an NN [(Ichihashi&Watanabe, 1990)] or using a GA [(Karr et al., 1989)]. For example, the shape of a triangular membership function is defined by three parameters: left, center, and right. The consequent part is also parameterized.

The approach using an NN was applied to develop several commercial products. The first neuro-fuzzy consumer products were introduced to the Japanese market in 1991. This is the type of Figure 20, and some of the applications are listed in section 5.3.

The approach using a GA has been applied to

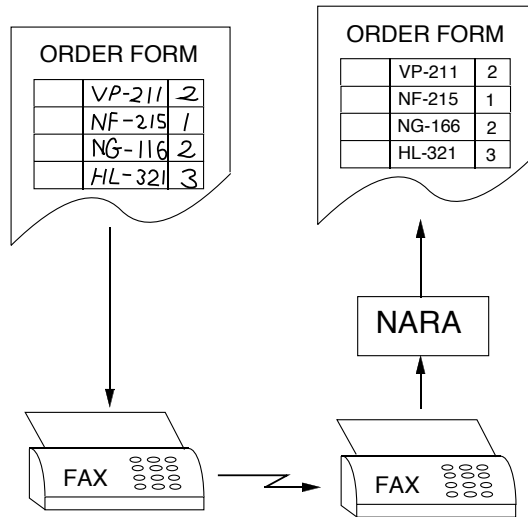


Figure 19: FAX OCR: hand-written character recognition system for a FAX ordering system.

some Korean consumer products since 1994. Some of them are: cool air flow control of refrigerators; slow motor control of washing machines to wash wool and/or lingerie [(Kim et al., 1995)]; neuro-fuzzy models for dish washers, rice cookers, and microwave ovens to estimate the number of dishes, to estimate the amount of rice, and to increase control performance; and FSs for refrigerators, washing machines, and vacuum cleaners [(Shin et al., 1995)]. These neuro-fuzzy systems and FSs are tuned by GAs.

5.2 NN configuration based on fuzzy rule base

NARA is a structured NN construct based on the IF-THEN fuzzy rule structure [(Takagi et al., 1992)]. An a priori knowledge of tasks is described by fuzzy rules, and small sub-NNs are combined according to the rule structure. Since a priori knowledge of the task is embedded into the NARA, the complexity of the task can be reduced.

NARA has been used for a FAX ordering system. When electric shop retailers order products from a Matsushita Electric dealer, they write an order form by hand and send it by FAX. The FAX machine at the dealer site passes the FAX image to the NARA. The NARA recognizes the hand-written characters and sends character codes to the dealer's delivery center (Figure 19).

5.3 Combination of NN and FS

There are many consumer products which use both NN and FS in a combination of ways. Although there are many possible combinations of the two sys-

(MATSUSHITA ELECTRIC)

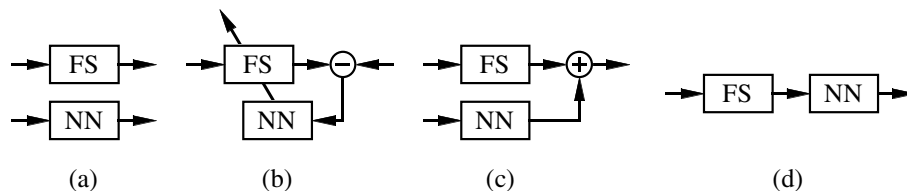


Figure 20: Combination types of an FS and an NN: (a) independent use, (b) developing tool type, (c) correcting output mechanism, and (d) cascade combination.

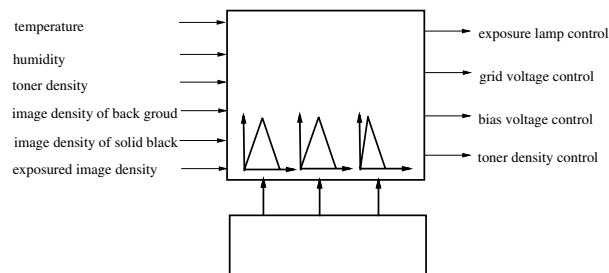


Figure 21: Photo copier whose fuzzy system is tuned by a neural network.

tems, the four combinations shown in Figure 20 have been applied to actual products. See the reference of [(Takagi, 1995)] for details on these consumer applications.

Figure 20(a) shows the case where one piece of equipment uses the two systems for different purposes without mutual cooperation. For example, some Japanese air conditioners use an FS to prevent a compressor from freezing in winter and use an NN to estimate the index of comfort, the PMV (Predictive Mean Vote) value [(Fanger,1970)] in ISO-7730, from six sensor outputs.

The model in Figure 20(b) uses the NN to optimize the parameters of the FS by minimizing the error between the output of the FS and the given specification. Figure 21 shows an example of actual applications of this model. This model has been used to develop washing machines, vacuum cleaners, rice cookers, clothes driers, dish washers, electric thermo-flask, inductive heating cookers, oven toasters, kerosene fan heaters, refrigerators, electric fans, range-hoods, and photo copiers since 1991.

Figure 20(c) shows a model where the output of an FS is corrected by the output of an NN to increase the precision of the final system output. This model is implemented in washing machines manufactured by Hitachi, Sanyo, and Toshiba, and oven ranges manufactured by Sanyo.

Figure 20(d) shows a cascade combination of an FS and an NN where the output of the FS or NN becomes the input of another NN or FS. An electric fan developed by Sanyo detects the location of its remote controller with three infrared sensors. The outputs from these sensors change the fan's direction according to the user's location. First, a FS estimates the distance between the electric fan and the remote controller. Then, a NN estimates the angle from the estimated distance and the sensor outputs. This two-stage estimation was adopted because it was found that the outputs of three sensors change according to the distance to the remote controller even if the angle is same.

Oven ranges manufactured by Toshiba use the same combination. An NN first estimates the initial temperature and the number of pieces of bread from sensor information. Then an FS determines the optimum cooking time and power by inputting the outputs of the NN and other sensor information.

Figure 22 shows an example of this cascade model applied to the chemical industry. Toshiba applied the model to recover expensive chemicals that are used to make paper from chips at a pulp factory [(Ozaki, 1994)]. The task is to control the temperature of the liquid waste and air (or oxygen) sent to a recovering boiler, deoxidize liquid waste, and recover sulfureted sodium effectively.

The shape of the pile in the recovering boiler influences the efficiency of the deoxidization, which, in turn, influences the performance of recovering the sulfureted sodium. An NN is used to recognized the shape of the pile from the edge image detected from the CCD image and image processing. An FS determines the control parameters for PID control by using sensor data from the recovering boiler and the recognized shape pattern of the pile.

5.4 NN learning and configuration based on GA

One important trend in consumer electronics is the feature that adapts to the user environment or preference and customizes mass produced equipment at the user end. An NN learning function is the leading technology for this purpose.

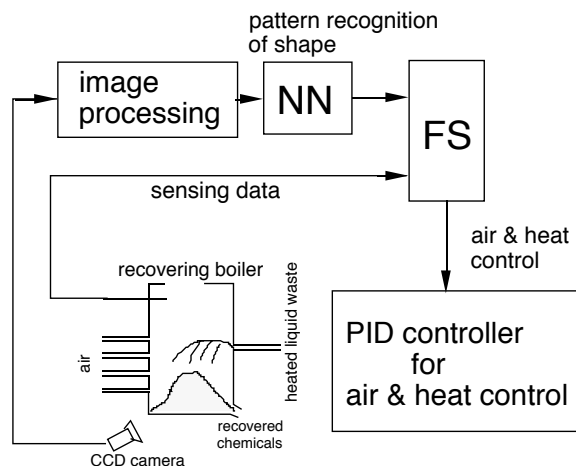


Figure 22: Chemicals recycling system at a pulp factory. An NN identifies the shape of the chemical pile from the edge image, and a fuzzy system determines the control values for air and heat control to recover chemicals optimally.

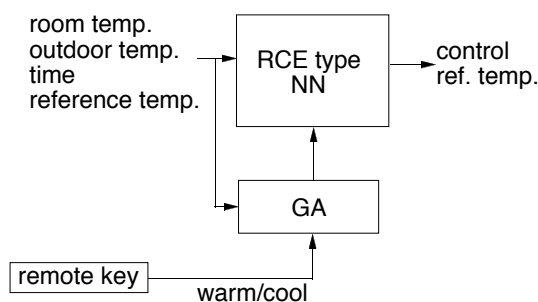


Figure 23: Temperature control by a RCE neural network controller designed by GA at the user end.

Japanese electric companies applied a single user-trainable NN to (1) kerosene fan heaters that learn and estimate when their owners use the heater [(Morito et al., 1991)], and (2) refrigerators that learn when their owners open the doors to pre-cool frozen food [(Ohtsuka et al., 1992)].

LG Electric developed an air conditioner that implemented a user-trainable NN trained by a GA [(Shin et al., 1995)]. The NNs in RCE's air conditioners inputs room temperature, outdoor temperature, time, and user-set temperature, and outputs control values to maintain the user-set temperature [(Reilly et al., 1982)]. Suppose a user wishes to change the control to low to adapt to his/her preference. Then, a GA changes the characteristics of the NN by changing the number of neurons and weights (see Figure 23).

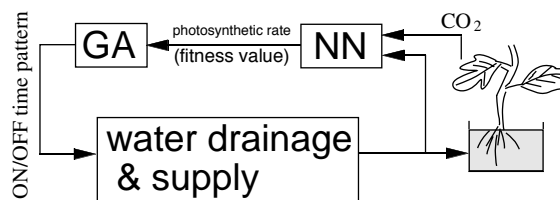


Figure 24: Water control for a hydroponics system.

5.5 NN-based fitness function for GA

A GA is a searching method that multiple individuals apply to a task and evaluate for the subsequent search. If the multiple individuals are applied to an on-line process in addition to the usual GA applications, the process situation changes before the best GA individual is determined.

One solution is to design a simulator of the task process and embed the simulator into a fitness function. An NN can then be used as a process simulator, trained with the input-output data of the given process.

A GA whose fitness function uses an NN as a simulator of plant growth was used in a hydroponics system [(Morimoto et al., 1993)]. The hydroponics system controls the timing of water drainage and supply to the target plant to maximize its photosynthetic rate. The simulation NN is trained using the timing pattern as input data and the amount of CO_2 as output data. The amount of CO_2 is used as an alternative to the photosynthetic rate of the plant. Timing patterns of water drainage and supply generated by the GA are applied to the pseudo-plant, the trained NN, and evaluated according to how much CO_2 they create (Figure 24). The best timing pattern is selected through the simulation and applied to the actual plant.

6 SUMMARY

This chapter introduced the basic concepts and concrete methodologies of fuzzy systems, neural networks, and genetic algorithms to prepare the readers for the following chapters. Focus was placed on: (1) the similarities between the three technologies through the common keyword of *nonlinear relationship in a multi-dimensional space* visualized in Figure 1 and (2) how to use these technologies at a practical or programming level.

Readers who are interested in studying these applications further should refer to the related tutorial papers.

REFERENCES

- [(Bersini&Scront, 1992)] Bersini, H. and Scront, G., "In search of a good evolution-optimization crossover," in *Parallel Problem Solving from Nature*, R. Manner and B. Mandrick (eds.), pp.479–488, Elsevier Science Publishers, (1992).
- [(Fanger,1970)] Fanger, P. O., "Thermal Comfort — Analysis and Application in Environmental Engineering," McGraw-Hill (1970)
- [(Hayashi&Takagi,1988)] Hayashi,I. and Takagi, H., "Formulation of Fuzzy Reasoning by Neural Network," 4th Fuzzy System Symposium of IFSA Japan, pp.55–60 (May, 1988) (*in Japanese*)
- [(Ichihashi&Watanabe, 1990)] Ichihashi, H. and Watanabe, T., "Learning control by fuzzy models using a simplified fuzzy reasoning," *J. of Japan Society for Fuzzy Theory and Systems*, Vol.2, No.3, pp.429–437 (1990) (*in Japanese*)
- [(Karr et al., 1989)] Karr, C., Freeman, L., Meredith, D., "Improved Fuzzy Process Control of Spacecraft Autonomous Rendezvous Using a Genetic Algorithm," *SPIE Conf. on Intelligent Control and Adaptive Systems*, pp.274–283 (Nov., 1989)
- [(Kim et al., 1995)] Kim, J.W., Ahn, L.S., and Yi, Y.S., "Industrial applications of intelligent control at Samsung Electric Co., – in the Home Appliance Division –,” in *Fuzzy Logic for the Applications to Complex Systems*, ed. by W. Chiang and J. Lee, World Scientific Publishing, pp.478–482 (1995)
- [(McCulloch&Pitts, 1943)] McCulloch, W. S. and Pitts, W. H., "A logical calculus of the ideas immanent in nervous activity," *Bullet. Math. Biophysics*, Vol.5, pp.115–133 (1943)
- [(Mamdani, 1974)] Mamdani, E. H., "Applications of fuzzy algorithms for control of simple dynamic plant," *Proc. of IEEE*, Vol. 121, No. 12, pp.1585–1588 (1974)
- [(Mizumoto, 1996)] Mizumoto, M., "Product-sum-gravity method = fuzzy singleton-type reasoning method = simplified fuzzy reasoning method," *Fifth IEEE Int'l Conf. on Fuzzy Systems*, New Orleans, USA, pp.2098–2102, (Sept., 1996)
- [(Morimoto et al., 1993)] Morimoto, T., Takeuchi, T., and Hashimoto, Y., "Growth optimization of plant by means of the hybrid system of genetic algorithm and neural network," *Int'l Joint Conf. on Neural Networks (IJCNN-93-Nagoya)*, Nagoya, Japan, Vol.3, pp.2979–2982 (Oct. 1993).
- [(Morito et al., 1991)] Morito, K., Sugimoto, M., Araki, T., Osawa, T., and Tajima, Y., "Kerosene fan heater using fuzzy control and neural networks (CFH-A12JD)", *Sanyo Technical Review*, Vol. 23, No. 3, pp.93–100 (1991) (*in Japanese*)
- [(Nakajima et al., 1993)] Nakajima, M., Okada, T., Hattori, S., and Morooka, Y., "Application of pattern recognition and control technique to shape control of the rolling mill," *Hitachi Review*, Vol.75, No.2, pp.9–12 (1993) (*in Japanese*)
- [(Ohtsuka et al., 1992)] Ohtsuka, H., Suzuki, N., Minagawa, R., and Sugimoto, Y. "Applications of Intelligent Control Technology to Home Appliances," *Mitsubishi Electric Technical Report*, Vol.66, No.7, pp.728–731 (1992) (*in Japanese*)
- [(Ozaki, 1994)] Ozaki, T., "Recovering boiler control at a pulp factory," *Neuro-Fuzzy-AI Handbook*, Ohm Publisher, pp.1209–1217 (1994) (*in Japanese*)
- [(Reilly et al., 1982)] Reilly, D.L., Cooper, L.N., and Elbaum, C., "A neural model for category learning," *Biological Cybernetics*, Vol.45, No.1, pp.35–41 (1982)
- [(Rosenblatt, 1958)] Rosenblatt, F., "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, Vol.65, pp.386–408 (1958)
- [(Rumelhart, et al., 1986)] Rumelhart, D.E., McClelland, J.L., and the PDP Research Group, *Parallel distributed processing: explorations in the microstructure of cognition*, Cambridge, MA: MIT Press (1986)
- [(Shin et al., 1995)] Shin, M.S., Lee, K.L., Lim, T.L., and Wang, B.H., "Applications of evolutionary computations at LG Electronics," in *Fuzzy Logic for the Applications to Complex Systems*, ed. by W. Chiang and J. Lee, World Scientific Publishing, pp.483–488 (1995)
- [(Takagi&Sugeno, 1985)] Takagi, T. and Sugeno, M., "Fuzzy Identification of Systems and Its Applications to Modeling and Control," *IEEE Trans. SMC*-15-1, pp.116–132 (1985)
- [(Takagi&Hayashi, 1991)] Takagi, H. and Hayashi, I., "NN-driven Fuzzy Reasoning," *Int'l J. of approximate Reasoning (Special Issue of IIZUKA '88)*, Vol. 5, No.3, pp.191–212 (1991)
- [(Takagi et al., 1992)] Takagi, H., Suzuki, N., Kouda, T., Kojima, Y., "Neural networks designed on Approximate Reasoning Architecture," *IEEE Trans. on Neural Networks*, Vol. 3, No. 5, pp.752–760 (1992)
- [(Takagi, 1995)] Takagi, H., "Applications of Neural Networks and Fuzzy Logic to Consumer Products," ed. by J. Yen, R. Langari, and L. Zadeh, in *Industrial Applications of Fuzzy Control and Intelligent Systems*, Ch.5, pp.93–106, IEEE Press, Piscataway, NJ, USA (1995)
- [(Takagi, 1996)] Takagi, H., "Industrial and Commercial Applications of NN/FS/GA/Chaos in 1990s," *Int'l Workshop on Soft Computing in Industry (IWSCI'96)*, Muroran, Hokkaido, Japan, pp.160–165 (April, 1996)
- [(Yen et al. eds., 1995)] Yen, J., Langari, R., and Zadeh, L., (edit.) *Industrial Applications of Fuzzy Control and Intelligent Systems*, IEEE Press, Piscataway, NJ, USA (1995)