Reconstructing and modelling gene regulatory network from experimental data is crucial in the understanding of fundamental cellular processes and disease mechanism. A GRN represents a set of regulatory interactions among genes in a cellular system. These interactions are involved directly or indirectly in controlling the production of gene products such as proteins and in mediating metabolic processes. Exploring GRNs can provide new ideas for treating complex diseases and breakthroughs for designing personalized medicine. In this chapter, four computational approaches applied to the problem of GRN reconstruction such as fuzzy logic approach, modified genetic algorithm, dynamic feed forward neural fuzzy network and TSK-type recurrent neural fuzzy network are presented.

6.1 Fuzzy Logic Approach

Fuzzy logic algorithm provides a systematic and unbiased way to transform precise numbers into qualitative descriptors in a process called fuzzification. When dealing with gene expression data the observed data is broken using fuzzy logic into discrete subsections which provides a
qualitative description of the data. Heuristic rules can be used to analyze this qualitative data, which in turn generate fuzzy solutions. Heuristic solutions can be transformed into a precise number using defuzzification operation.

There are three main advantages of applying fuzzy logic algorithm for the analysis of gene expression data. This includes

1. Fuzzy Logic inherently accounts for the noise in the data because it extracts trends, not precise values.

2. The algorithms in fuzzy logic make use of high level human like reasoning and are cast in the same language used in day-to-day conversation. As a result, predictions made by this algorithm can be easily interpretable.

3. Fuzzy logic approaches are computationally efficient and can be scaled to include an unlimited number of components. Thus a large number of biologically relevant patterns can be recognized using this approach.

Wolf and Wang [13] introduced fuzzy logic approach to search for the activator-repressor-target relationship of gene interaction from a normalized subset of Saccharomyces Cerevisiae data [103]. They applied fuzzy rules to every possible activator-repressor combination of genes and the output of the model was compared to the expression levels of the remaining genes. They concluded that those combinations of genes having low error are most likely to exhibit an activator/repressor relationship. The relations predicted by this approach agree well with the experimental results from the literature. But this algorithm suffered from high computational
time requiring 200 hours to analyze the relationship between 1898 genes [151]. In this work, fuzzy logic algorithm has been improved to infer causal relationship between genes by analyzing gene expression data. In the proposed model, clustering is used as the preprocessing step to remove the redundant computation performed by the model. Clustering will group the genes based on their similarity in the changes of expression over all samples available in the microarray data. Replacing the group with a reference expression profile, when executing fuzzy logic algorithm, will greatly reduce the computational cost.

6.1.1 The Fuzzy Logic Algorithm

The fuzzy logic algorithm [13] identifies the regulatory triplets consisting of activator, repressor and target genes by searching the microarray data set. The algorithm uses fuzzy logic to transform the gene expression levels into qualitative descriptors such as HIGH, MEDIUM and LOW states. The Zadeh – Mamdani’s fuzzy model with two inputs and one output has been adopted to realize fuzzy inference and fuzzy IF-THEN rules are used to perform a map from the input space to the output space. The inputs are the normalized expression value of genes (A and B) paired into activator and repressor. The first step is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership function of the triangular form shown in figure 6.1. Next, these fuzzified expression values of A and B are evaluated against a set of heuristic rules summarized in the decision matrix shown in Table 6.1. The output of the fuzzy system is the fuzzified value of the predicted target C, which encompasses a range of output values, and so it must be defuzzified in order to resolve into a single output value. The predicted expression
values of C for each sample were calculated in the similar manner. The mean square error (MSE) between the entire series of predicted values and target gene’s actual expression values were calculated. Those triplets that have low MSE value may exhibit the activator-repressor-target relationship.

![Figure 6.1 Membership Function](image)

**Figure 6.1 Membership Function**

<table>
<thead>
<tr>
<th></th>
<th>Table 6.1 Decision Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>if repressor is</strong></td>
</tr>
<tr>
<td></td>
<td>HIGH</td>
</tr>
<tr>
<td><strong>if activator is</strong></td>
<td></td>
</tr>
<tr>
<td>LOW</td>
<td>Target is LOW</td>
</tr>
<tr>
<td>MED</td>
<td>Target is LOW</td>
</tr>
<tr>
<td>HIGH</td>
<td>Target is MED</td>
</tr>
</tbody>
</table>

The second score is calculated based on the exploration of decision matrix. If the expression value of gene A is always high and that of gene B is always low, the dataset cannot properly explore the fuzzy rules. A hit ratio is defined as the ratio of the number of rules used to the total number of rules. A high hit ratio value implies that all of the rules are used equally in generating the output and the resulting predictions are more likely to be credible. The low value implies that all the rules have not been tested and the predictions may or may not be true. In this study only those gene pairs
Computational Methods

with hit ratio greater than 0.8 were considered for further analysis. The algorithm implemented in Matlab requires 232secs seconds to analyze the relationship between 100 genes from Plasma RNA dataset of CRC patients.

6.1.2 Clustering to Improve Run time

Clustering is a data mining process used to reveal natural structures and identify interesting patterns in the gene expression data. Cluster analysis partitions a given set of genes in the dataset into groups so that genes having similar expression profiles form a group. In this work, hybrid clustering algorithm described in chapter 5 has been applied to cluster genes that behave similarly. The figure 6.2 illustrates the use of clustering in modelling the gene regulatory network. For example, top figure shows that an increasing activator and decreasing repressor would cause the target gene to increase quickly. These triplets make sense intuitively and should be included in the analysis. Whereas, in the bottom figure, an increasing activator and decreasing repressor would cause the target gene to decrease. These cluster triplets do not make sense intuitively and should not be included in the analysis. Using the fuzzy logic algorithm, one can easily determine whether or not large group of genes, represented by these cluster centers, are likely to fit the model. The fuzzy logic algorithm is executed using centroid of each cluster as input and the cluster triplets ranked according to how well they fit the model. Based on the knowledge of how cluster centers fit to the model, the total number of gene combination analyzed could be reduced. This will enhance the performance of the above algorithm by reducing the computation time with minimal effect on results.
6.2 Modified Genetic Algorithm

Regulatory relationships between genes can be represented as linear coefficients of weights, with the “net” regulation influence on a gene’s expression being the mathematical summation of the independent regulatory inputs [113]. This representation is practical for analysis of microarray experiment, which provides snapshots of concentration at various samples. The regulatory networks generated from microarray data with this approach, display stable gene expression levels, consistent with known biological system [113]. In this approach GRN is represented by a weighted graph $G = (V, E, W)$, where $V$ is the set of nodes (genes), $E$ is the set of edges (regulatory relationships) and $W$ is the weight matrix. Figure 6.3 shows an example of a gene network and the corresponding weight matrix. The value of $w_{ij}$ is limited to a range between -1 and 1. The positive $w_{ij}$ means gene $i$ activating gene $j$, as opposed to a negative value representing an inhibition.
Zero indicates no influence. The expression level of a transcriptional regulatory network containing N genes is represented by a vector $\mathbf{x}$. Each column of weight matrix $W$ represents all regulatory inputs to a gene.

$$\begin{array}{ccc}
G1 & G2 & G3 \\
G1 & 0 & 0.4 & 0 \\
G2 & 0 & 0.8 & -0.9 \\
G3 & 0.7 & 0 & 0 \\
\end{array}$$

**Figure 6.3** A Sample Gene Network and the corresponding Weight Matrix.

The net regulatory input to a gene $i$, $S_i$ is determined by taking the weighted sum of the expression level of other genes.

$$S_i = \sum_{j=0}^{n} w_{ij}x_j$$

The response of gene $i$ to the regulatory input is defined using the sigmoid function as

$$x_i = \frac{m_i}{1 + e^{-s_i}}$$

where $m_i$ is the maximal expression level.
6.2.1 Genetic Algorithm Implementation

Genetic algorithms are search procedures, based on the mechanics of natural genetics, able to provide robust search in complex problem spaces. Genetic algorithm was applied for optimizing the weight matrix for gene regulatory network. For the GA implementations, gene networks have to be coded into chromosomes. Each row of the weight matrix is aligned in an array to be the one dimensional real number array chromosome. The fitness function of the GA is defined by the Euclidean error $\delta$ between the observed expression pattern and the actual expression pattern of the target gene.

$$\delta = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Since GA is a probabilistic search, several generations and greater computation power were required to model smaller network with good sensitivity and precision. Simple regulatory networks with few numbers of genes can be inferred effectively using GA [14, 152]. However, when applied to large microarray dataset this approach is slow and computationally complex. So, in order to reduce the number of generations, some preprocessing steps were done to initialize the genotypes.

Correlation analysis can be used to capture the regulation and co-regulation among the genes and have been proven useful for identifying biologically relevant groups of genes and samples [15]. High correlation between gene A and B can be caused by (i) A regulates B or vice versa (ii) A and B are co-regulated by other genes (iii) there is no causal relationship just coincidence. Here regulations may be indirect, i.e. interactions through immediates. Pearson’s correlation coefficient is widely used to measure the
expression pattern similarity between two genes. Pearson’s correlation coefficient considers each gene as a random variable and calculates the similarity of expression patterns by computing the linear relationship between gene expressions. However, experimental study has shown that it is sensitive to outliers and sometimes yields a high similarity score to a pair of dissimilar patterns. If two patterns have a common peak or valley at a single feature, the correlation will be dominated by this feature, although the patterns at the remaining features may be completely dissimilar. This observation evoked an improved measure called Jackknife correlation. Given two data objects \( O_i, O_j \), Jackknife correlation coefficient is defined as \( \text{Jackknife}(O_i, O_j) = \min\{\rho_{ij}(1), \rho_{ij}(2), \ldots, \rho_{ij}(p)\} \) where \( \rho_{ij}(k) \) is the Pearson’s correlation coefficient of data objects \( O_i \) and \( O_j \) with the \( k^{th} \) feature deleted. Use of the Jackknife correlation avoids the “dominance effect” caused by single outliers.

A Genetic Algorithm operates through a simple cycle of stages as shown in figure 6.4. Each cycle in Genetic Algorithm produces a new generation of possible solutions for a given problem. In the first phase, an initial population, set of individuals each representing a possible solution to a given problem is created to initiate the search process. For that the correlation coefficient of the target gene with every other gene is calculated. If the correlation is significant, a random number is generated to initialize the corresponding data point of each of the genotypes in the population. Based on the value of the fitness function, chromosomes are selected for a subsequent genetic manipulation process. In this work, Roulette Wheel selection algorithm was used to breed a new generation. In Roulette Wheel selection, the fitness level is used to associate a probability of selection with each individual chromosome. In order to enhance the adaptability of the GA
as well as the reverse engineering method, the genetic manipulation process consisting of two steps is carried out. In the first step, the crossover operation that interchanges the bits (genes) of two parent chromosomes at the crossover point is executed. The second step is termed mutation, where the bits at one or more randomly selected positions of the chromosomes are modified to form a new population. The mutation process avoids the algorithm to get trapped at local maxima. The offsprings produced by the genetic manipulation process form the next population to be evaluated. The process is repeated until the specified numbers of generations are completed. Finally, the chromosome with the best fitness score has the weights of the regulators of the target gene. The algorithm is repeated for each of the target gene.
By incorporating statistical techniques viz. correlation analysis, while generating initial population, search space can be reduced significantly. This will improve the computational time of modified GA when compared to that of basic GA.

6.3 Dynamic Feed Forward Neural Fuzzy Network

A Neural Fuzzy Network is a learning system that finds the parameters of a fuzzy system (i.e., fuzzy sets, fuzzy rules) by exploiting approximation techniques from neural networks [153]. Both neural networks and fuzzy systems can be used for solving a problem if there does not exist any mathematical model of the given problem. They solely do have certain disadvantages which almost completely disappear by combining both concepts. For example, neural networks can only come into play if the problem is expressed by a sufficient amount of observed examples. These observations are used to train the network. On the contrary, a fuzzy system demands linguistic rules instead of learning examples as prior knowledge. If the knowledge is incomplete or wrong, then the fuzzy system must be tuned in a heuristic way. This is usually very time consuming and error-prone.

In this work, a novel dynamic neural fuzzy hybrid system has been proposed to extract information from microarray data in the form of fuzzy rules. This method combines both the advantages of neural network and fuzzy system; it brings the low-level learning and computational power of neural networks into fuzzy systems and provides the high–level human like reasoning of fuzzy system into neural network. Unlike other neural fuzzy architectures [146], where the network structure is fixed and rules should be
assigned in advance, there is no predetermination of rules; all of them are constructed during online learning.

6.3.1 Dynamic Neural Fuzzy Network Architecture

The dynamic neural fuzzy network (DNFN) adapts the Zadeh-Mamdani’s fuzzy model [154] to realize the fuzzy inference and uses fuzzy if-then rules to perform the map from input space to output space. In contrast to other neural fuzzy model, there are no fuzzy rules initially in DNFN. They are generated on-line via learning. There are two learning phase - structural and parameter learning. The structural learning phase is responsible for the construction of dynamic fuzzy if-then rules. The parameter learning phase is for tuning the free parameters such as weights of the rules, which is accomplished through the repeated training of input output patterns. Parameter learning process is done concurrently with the structural learning phase. Owing to its simplicity back propagation algorithm is adopted to adjust the weights of DNFN.

Architecture of DNFN is illustrated in figure 6.5. The network consists of 5 layers.

The detailed functions of each layer are as follows.

Layer 1 (Input Layer): Dimension of this layer matches the number of input variables. No computation is of data is done in this layer. Each node transmits input values to the next layer direct.

Layer 2 (Fuzzification layer): Each node in the layer corresponds to one linguistic label- low expressed, medium expressed, highly expressed (LOW, MEDIUM, HIGH) of an input variable. Each node in this layer
computational methods calculates the membership value specifying the degree to which an input value belongs to a fuzzy set. The triangular membership function shown in figure 6.1 is used in this layer.

Layer 3 (Rule Antecedent Layer): Nodes in this layer are called rule nodes. The number of nodes in this layer will be incremented each time when a new rule is generated during the structural learning phase. Each rule node represents a fuzzy logic rule and performs precondition matching of the rule. Fuzzy AND operator max is used to evaluate the antecedent part of each rule. The output of a rule node represents the firing strength of the corresponding rule.

Layer 4 (Rule Consequent Layer): The number of nodes in this layer will be equal to the number of rule nodes. Every rule has a weight assigned to it. The weighted sum of the firing strength of each rule is calculated and it
is used to reshape the fuzzy set associated with the consequent part. The output of this layer is the reshaped fuzzy set.

Layer5 (Aggregation and Defuzzification Layer): This layer is the output layer. This node combines the fuzzy sets from the previous layer to a single fuzzy set. The fuzzy set thus obtained is then defuzzified. The centroidal defuzzification technique is used in this layer.

6.3.2 Construction of Fuzzy Rules

The way the input space is partitioned determines the number of rules. Each rule generated has the following form

\[ R : \text{if } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } A_2 \text{ and } \ldots \text{ and } x_n \text{ is } A_n \text{ then } y \text{ is } B \]

where \( A \) and \( B \) are the fuzzy sets used to perform a map from the input space vector to the output space. The algorithm for the construction of fuzzy rules is as follows.

Suppose no rule exist initially, let \((x, y)\) be the vector representing input and output pattern respectively.

If \( x \) is the first incoming pattern

**Step 1:** Generate a new rule with \( A_i \) as the fuzzy set having maximum membership value for \( x_i \) and the output fuzzy set \( B_i \) is the fuzzy set having maximum membership value for \( y \).

Else for each newly incoming pattern \((x_k, y_d)\) do the following steps.

**Step 2:** Find \( y_k \), the actual output with the existing rules.

**Step 3:** Compute the overall error using the equation
$E_k = \text{abs}(y_d - y_k)$, where $y_d$ is the desired output.

**Step 4:** If $E_k > \lambda_{\text{error}}$ Create new rule, do step 1.

else

**Step 5:** Adjust the parameters of the network using the same input pattern.

### 6.3.3 Deletion of Redundant Rules

In order to reduce the complexity of the model two criteria for deleting redundant rules were proposed.

1. If a fuzzy set is near zero over its own universe of discourse for a certain number of time steps then the rule with specific fuzzy set as precondition should be removed as this fact indicates that the output of the rule is also near zero.

2. When the two fuzzy rules have same consequent and similar antecedent part, we have a case of selecting one among the rules. The solution is to select the rule whose firing strength is high for more number of samples.

The main reason for the presence of redundant rule is due to the inherent noise in the microarray input data.

The multilayered dynamic neural fuzzy network was modeled to extract regulatory relationship among genes and reconstruct gene regulatory network from microarray data. This method combines the merits of connectionist and fuzzy approaches. It encodes the knowledge learned in the form of fuzzy rules and processes data following fuzzy reasoning principles. While the dynamic aspect of gene regulation was taken into account through the on-line learning of fuzzy rules, the structural learning together with the
parameter learning form a fast learning algorithm for building a small, yet powerful, dynamic neural fuzzy network.

6.4 TSK-type Recurrent Neural Fuzzy Network

Recurrent neural networks (RNNs), in general, can deal with temporal and spatial/temporal problems by adapting feedback connections to their topologies which are used to memorize past information. Although RNNs are generally efficient for temporal sequence production problems, it has been proved that recurrent neural fuzzy networks are superior to recurrent neural networks in dealing with the problems like GRN reconstruction which involves concurrent spatial and temporal mapping [146,147]. Additionally, due to their high-level, humanlike reasoning, fuzzy-based approaches are better candidates in dealing with the uncertainties of modelling noisy data [148, 149].

In this work, a TSK type recurrent neural fuzzy network (TRNFN) has been applied to extract gene regulatory relations from microarray data. Although recurrent neural fuzzy based algorithms have been used for inferring GRN by some investigators [147], the use of TSK-type fuzzy model for the inference of gene regulatory network has not been successfully explored. The TRNFN extends DNFN by the inclusion of memory elements in the form of feedback connections. Unlike DNFN, whose output is a function of its current inputs only, TRNFN can perform dynamic mapping using their ability to store prior system states. Moreover, the rules in DNFN are of ordinary Mamdani-type fuzzy rule, whereas TRNFN uses TSK-type fuzzy rule. In [155], where TSK-type recurrent fuzzy network TRNFN is used for dynamic system control has shown that the performance and learning accuracy is superior to those of Mamdani type
fuzzy network. In TRNFN a global feedback structure is adopted where the output of all rule nodes are fed back and summed, so each rule’s firing strength depends not only on its previous value but also on others. With the global feedback structure, TRNFN achieve better performance than local feedback structure where, the rule’s firing strength is influenced only by its previous value. Furthermore, the inclusion of TSK-type consequence can significantly reduce the rule number [155]. TRNFN is characterized by small network size and fast learning speed.

6.4.1 Architecture

TSK-type Recurrent Neural Fuzzy Network (TRNFN) was proposed by Chia-Feng Juang in 2002 [155]. TRNFN is constructed from a series of fuzzy if-then rules, with the consequence of each rule being of TSK-type fuzzy reasoning. The network precondition part includes external variables and internal variables derived from fuzzy firing strengths, and the consequence is a linear combination of them plus a constant term. Each rule i has the following form:

\[
R(i) = IF x_1(t) \text{is } A_{i1} \text{ and } x_2(t) \text{is } A_{i2} \text{ and } \ldots x_n(t) \text{is } A_{in} \text{ and } h_i(t) \text{ is } G
\]

\[
THEN y(t + 1) = a_{i0} + \sum_j a_{ij}x_j(t) + a_{in+1} h_i(t)
\]

where A and G are fuzzy sets, h is the internal variable and a is the parameter for the inference output y.

Two learning phases, structural and parameter learning are used concurrently for constructing TRNFN. The structural learning phase includes input-output space partitioning, construction of fuzzy if-then rules and the feedback structure identification. The parameter learning phase is responsible for tuning of free parameters of the network structure. TRNFN
works for supervised learning environment with available gradient information. Since the gradient information is available, the neural network learning approach is used for learning the network parameters.

Architecture of TRNFN with two external inputs and one single output is illustrated in Figure 6.6. TRNFN is a six layered network, including a feedback layer that brings the temporal processing ability into a feed forward neural fuzzy network. The detailed function of each layer is described below. In the following descriptions the symbol $u_i^{(k)}$ denotes the input of the $i^{th}$ node in the $k^{th}$ layer; correspondingly, the symbol $Y_i^{(k)}$ denotes the output of the $i^{th}$ node of layer $k$.

![Figure 6.6 Architecture of TRNFN](image)
Layer 1: Dimension of this layer matches the number of input variable. No computation is done in this layer. The node only transmits input values to the next layer directly. That is

\[ Y_i^{(1)} = u_i^{(1)} = x_i \]

Layer 2: Node in the layer corresponds to one linguistic label - low expressed, medium expressed, highly expressed etc. of an input variable. Each node in this layer calculates the membership value specifying the degree to which an input value belongs to a fuzzy set. Gaussian membership function is employed in this layer since it can be easily decomposed in to the product of one-dimensional membership functions. With this choice, operation performed in this layer is

\[ Y_i^{(2)} = \exp \left( -\frac{(u_j^{(2)} - m_{ij})^2}{\sigma_{ij}^2} \right) \text{ and } u_j^{(2)} = Y_j^{(1)} \]

where \( m_{ij} \) and \( \sigma_{ij} \) represents the center and width of the Gaussian membership function of the \( i^{th} \) term of the \( j^{th} \) input variable \( x_j \). For the internal variable \( h_i \) the following sigmoid member function is used:

\[ Y_i^{(2)} = \frac{1}{1 + \exp(-u_i^{(2)})} \text{ and } u_i^{(2)} = Y_i^{(5)} \]

The link weights are all set to unity.

Layer 3: Nodes in this layer are called rule nodes. The number of nodes in this layer will be incremented each time when a new rule is generated during the structural learning phase. Each rule node represents a fuzzy logic rule and performs precondition matching of the rule. The input to each node
comes from two sources: one from layer 2 and the other from the feedback layer. Former, the output of the rule node represents the spatial firing strength of its corresponding rule and the latter, the output of the feedback term node, represents the rule’s temporal firing degree. The output of each node in this layer is determined by fuzzy AND operation. Here, the product operation is utilized to determine the firing strength of each rule. The function of each rule is

\[ Y^{(3)}_i = \prod_{j=1}^{n+1} Y^{(2)}_j = \frac{1}{1 + \exp\left(-Y^{(5)}_i\right)} \cdot \exp\left\{-\left(\sum_{j=1}^{n} \frac{Y^{(1)}_j - m_{ij}}{\sigma_{ij}^2}\right)^2\right\} \]

where \( n \) is the number of external inputs. The link weights in this layer are set to unity.

**Layer 4:** Nodes in this layer perform a linear summation. The mathematical function for each node \( n \) in this layer is

\[ Y^{(4)}_i = \sum_{j=0}^{n+1} a_{ij} u^{(4)}_j = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j + a_{in+1} h_i \]

where \( n \) is the number of external inputs, and \( a_{ij}, j = 0, ..., n + 1 \) are the parameters to be tuned in the structural learning phase. All link weights from this layer to layer 6 are set to unity.

**Layer 5:** This layer is also called feedback layer, which calculates the value of internal variable \( h \). Each rule node has a corresponding internal variable \( h \) which is used to decide the influence of the firing history to the current rule. The link weights are assigned random values initially and are updated
during the parameter learning phase. The weighted sum of outputs from rule nodes is calculated in each node.

\[ h_i = Y_i^{(5)} = \sum_{j=1}^{r} Y_j^{(3)} \cdot w_{ij} \]

where \( r \) is the number of rules generated so far. The delayed value of \( h_i \) is connected to layer 2 as shown in the figure 6.6.

**Layer 6:** Each node in this layer is called an output linguistic node. This node performs the defuzzification operation and computes the output signal \( Y \). The node in this layer together with the links attached to it performs this task. The mathematical function is

\[ Y = Y^{(6)} = \frac{\sum_{j=1}^{r} Y_j^{(3)} \cdot Y_j^{(4)}}{\sum_{j=1}^{r} Y_j^{(3)}} \]

**Learning Process**

Upon receiving the training data TRNFN performs structural as well as parameter learning process simultaneously. In the structural learning phase TRNFN decides the number of fuzzy rules and initializes the network parameters. In the parameter learning phase it optimally adjusts the free parameters of the network structure. The way the input space is partitioned determines the number of rule. The creation of new rule corresponds to the creation of new cluster in the input space. The spatial firing strength can be regarded as the degree to which an input pattern belongs to the corresponding cluster. Based on this concept, the spatial firing strength
\[ F^i(x) = \prod_{k=1}^{n} Y_k^{(2)} \]
\[ = \exp \left\{ -\sum_{j=1}^{n} \frac{(x_j - m_{ij})^2}{\sigma_{ij}^2} \right\} \in [0,1] \]

is used as a criterion to decide whether a fuzzy rule should be generated or not. As there was no rules initially, for the first incoming pattern \(x(0)\), a new fuzzy rule is generated with center and width of gauss membership function is assigned as

\[ m_{1i} = x_i(0) \text{ and } \sigma_{1i} = \sigma_{\text{init}}, \text{for } i = 1 \ldots n \]

where \(\sigma_{\text{init}}\) is a constant that determines the initial width of the first cluster.

For the succeeding inputs \(x(t)\), find

\[ I = \arg \max_{1 \leq r(t) \leq r} F^i(x) \]

where \(r(t)\) is the number rules at time \(t\). If \(F_1 \leq F_{\text{init}}\), then new rule is generated, where \(F_{\text{init}} \in (0,1)\) is a pre-specified threshold that decays during learning process. The center and width of the new rule can be set according to the first-nearest-neighbor heuristic as

\[ m_{(r(t)+1)i} = x_i(t) \text{ and } \]
\[ \sigma_{(r(t)+1)i} = \beta \sum_{j=1}^{n} \frac{(x_j - m_{ij})^2}{\sigma_{ij}^2} \]
where, i varies from 1 to n and $\beta \geq 0$ decides the overlap degree between the two clusters. Both parameters $F_{\text{init}}$ and $\beta$ decides the number of rules to be generated.

After the generation of a new rule for the input data $<x(t), y(t)>$, the consequent nodes in layer 4 and context node in layer 5 are computed. The initial value of parameter $a_{00}$ is set to $y(t)$ and the other $a_{ij}$ parameters are set to small random values in between $[-0.05, 0.05]$. To make initial values of $h$ lies in the sensitive region of sigmoid function, the initial link weights are set as random values in the range $[-1,1]$. As a result, quick parameter learning can be reached at the beginning. The output, $h$, of the new context node is fed back as input in the precondition part of the newly generated rule. With this setting, each rule has its own memory elements for memorizing the temporal firing strength history. By repeating the above process for every incoming training data, a new rule is generated, one after another, and a whole TRNFN is constructed finally.

The parameter learning process is performed concurrently with the structural learning phase based on the same training pattern. The free parameters such as $a$, $m$, $\sigma$ and $w$ are tuned using the real time recurrent learning algorithm [155, 156].

The network structure of TRNFN is such that it encodes knowledge learned in the form of fuzzy if-then rules and takes into account the dynamic aspects of gene regulation through its recurrent structure. The main features of TRNFN include:
1. The learning algorithm automatically produces an adaptive number of fuzzy rules that describe the relationships between the input (regulating) genes and the output (regulated) gene.

2. It overcomes the need for prior data discretization.

3. The recurrent property, achieved by feeding the internal variables in the form of feedback connections, increases the learning ability of TRNFN.

When applied to the problem of reconstruction of gene regulatory network from microarray data, TRNFN was provided good results from the accuracy and speed points of view.