IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Artificial Neural network : Training- I

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Learning of neural networks: Topics

- ✓ Concept of learning
- Learning in
    - ✓ Single layer feed forward neural network
    - Multilayer feed forward neural network
    - Recurrent neural network
- Types of learning in neural networks

Concept of Learning

The concept of learning

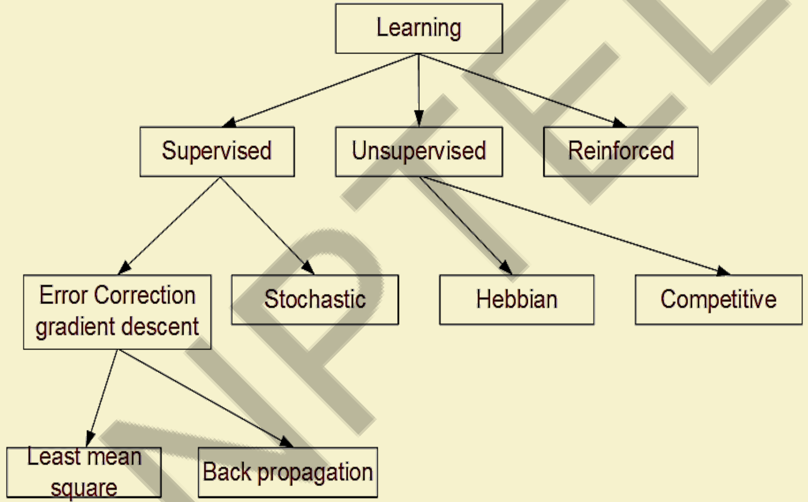
- The learning is an important feature of human computational ability.
- Learning may be viewed as the change in behaviour acquired due to practice or experience, and it lasts for relatively long time.
- As it occurs, the effective coupling between the neuron is modified.

The concept of learning

- In case of artificial neural networks, it is a process of modifying neural network by updating its weights, biases and other parameters, if any.
- During the learning, the parameters of the networks are optimized and as a result process of curve fitting.
- It is then said that the network has passed through a learning Phase.

Types of learning

- There are several learning techniques.
- A taxonomy of well known learning techniques are shown in the following.



- In the following, we discuss in brief about these learning techniques.

Supervised learning

- In this learning, every input pattern that is used to train the network is associated with an output pattern.
- This is called training set of data. Thus, in this form of learning, the input-output relationship of the training scenarios are available.
- Here, the output of a network is compared with the corresponding target value and the error is determined.
- It is then feed back to the network for updating the same. This results in an improvement.
- This type of training is called learning with the help of teacher.

Unsupervised learning

- If the target output is not available, then the error in prediction can not be determined and in such a situation, the system learns of its own by discovering and adapting to structural features in the input patterns.
- This type of training is called learning without a teacher.

Reinforced learning

- In this techniques, although a teacher is available, it does not tell the expected answer, but only tells if the computed output is correct or incorrect. A reward is given for a correct answer computed and a penalty for a wrong answer. This information helps the network in its learning process.

- **Note:** Supervised and unsupervised learnings are the most popular forms of learning. Unsupervised learning is very common in biological systems.

- It is also important for artificial neural networks: training data are not always available for the intended application of the neural network.

Gradient descent learning

- This learning technique is based on the minimization of error E defined in terms of weights and the activation function of the network.
  - Also, it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.

Gradient descent learning

- Thus, if ΔW_{ij} denoted the weight update of the link connecting the i-th and j-th neuron of the two neighbouring layers then

$$\Delta W_{ij} = \eta \frac{\partial E}{\partial W_{ij}}$$

where η is the **learning rate parameter** and $\frac{\partial E}{\partial Wij}$ is the **error gradient** with reference to the weight W_{ij}

- The least mean square and back propagation are two variations of this learning technique.

Stochastic learning

- **Stochastic learning**

  In this method, weights are adjusted in a probabilistic fashion.

  Simulated annealing is an example of such learning (proposed by Boltzmann and Cauch)

Hebbian learning

Hebbian learning:

- This learning is based on correlative weight adjustment. This is, in fact, the learning technique inspired by biology.
- Here, the input-output pattern pairs $(X_i ; Y_i)$ are associated with the weight matrix W. W is also known as the correlation matrix.
- This matrix is computed as follows.

$$W = \sum_{i=1}^{n} X_i Y_i^T$$

where Y_i^T is the transpose of the associated vector y_i

Competitive learning

Competitive learning:

- In this learning method, those neurons which responds strongly to input stimuli have their weights updated.
- When an input pattern is presented, all neurons in the layer compete and the winning neuron undergoes weight adjustment.
- This is why it is called a Winner-takes-all strategy.

In this course, we discuss a generalized approach of supervised learning to train different type of neural network architectures.

Learning ANNs

In this course, we discuss a generalized approach of supervised learning to train different type of neural network architectures.

Training SLFFNNs

Single layer feed forward NN (SLFFNN) training

- We know that, several neurons are arranged in one layer with inputs and weights connect to every neuron.
- Learning in such a network occurs by adjusting the weights associated with the inputs so that the network can classify the input patterns.
- A single neuron in such a neural network is called perceptron.

Single layer feed forward NN training

The algorithm to **train a perceptron** is stated below.

- Let there is a perceptron with $(n + 1)$ inputs $x_0, x_1, x_2, \dots, x_n$ where $x_0 = 1$ is the bias input.
- Let f denotes the transfer function of the neuron. Suppose, $\bar{X}$ and $\bar{Y}$ denotes the input-output vectors as a training data set. $\bar{W}$ denotes the weight matrix.

With this input-output relationship pattern and configuration of a perceptron, the algorithm **Training Perceptron** to train the perceptron is stated in the following slide.

Single layer feed forward NN training

1. Initialize $W = w_0, w_1, \ldots, w_n$ to some random weights.
2. For each input pattern $x \in \bar{X}$ do   Here, $x = \{x_0, x_1, \ldots, x_n\}$
   - Compute $I = \sum_{i=0}^{n} w_i x_i$
   - Compute observed output y

$$y = f(I) = \begin{cases} 1 & ,if\ I > 0 \\ 0 & ,if\ I \leq 0 \end{cases}$$

$\bar{Y}' = \bar{Y}' + y$ Add y to $\bar{Y}'$, which is initially empty

Single layer feed forward NN training

3. If the desired output $\bar{Y}$ matches the observed output $\bar{Y'}$ then output $\bar{W}$ and exit.

4. Otherwise, update the weight matrix $\bar{W}$ as follows :
   - For each output $y \in \bar{Y'}$ do
   - If the observed out y is 1 instead of 0, then
     $$w_i = w_i - \alpha x_i, (i = 0, 1, 2, \dots, n)$$
   - Else, if the observed out y is 0 instead of 1, then
     $$w_i = w_i + \alpha x_i, (i = 0, 1, 2, \dots, n)$$

5. Go to step 2.

Single layer feed forward NN training

In the above algorithm, α is the learning parameter and is a constant decided by some empirical studies.

Note :

- The algorithm Training Perceptron is based on the supervised learning technique.
- ADALINE: Adaptive Linear Network Element is also an alternative term to perceptron.
- If there are 10 number of neurons in the single layer feed forward neural network to be trained, then we have to iterate the algorithm 10 times for each perceptron in the network.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Artificial Neural Network: Training-II

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Learning of neural networks: Topics

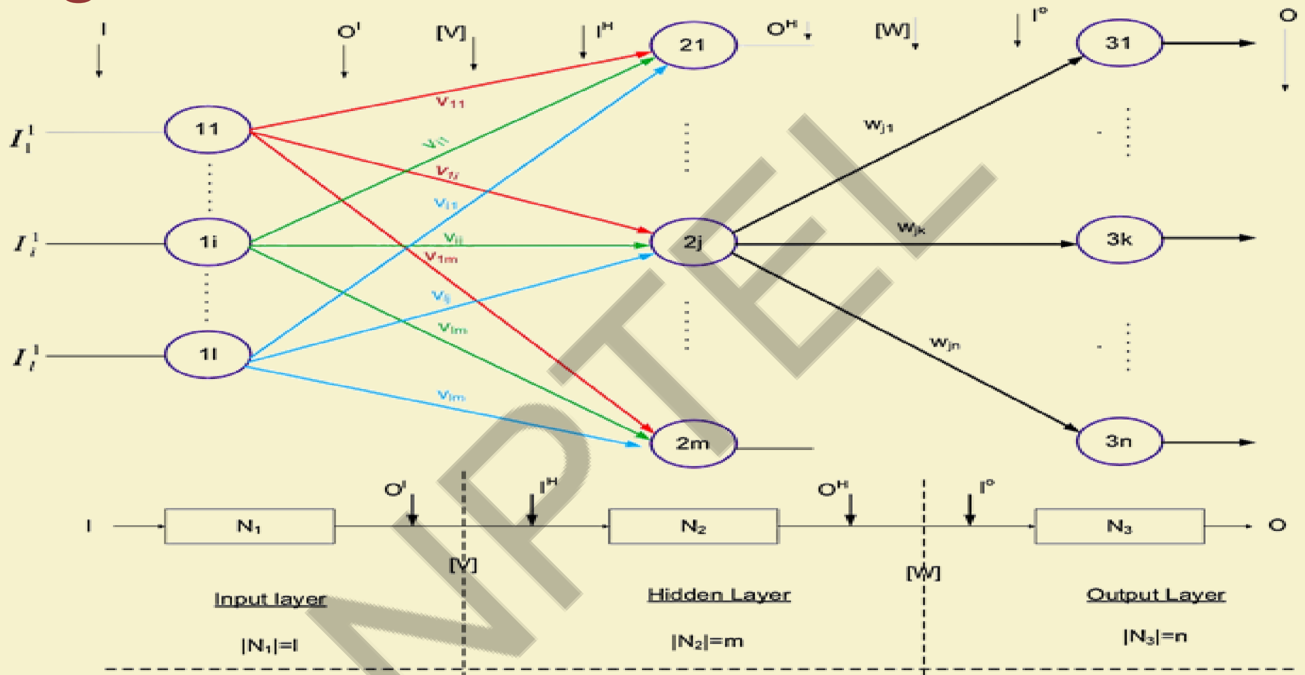
- ✓ Concept of learning
- ✓ Learning in
  - ✓ Single layer feed forward neural network
  - Multilayer feed forward neural network
  - Recurrent neural network

Training MLFFNNs

Training multilayer feed forward neural network

- Like single layer feed forward neural network, supervisory training methodology is followed to train a multilayer feed forward neural network.
- Before going to understand the training of such a neural network, we redefine some terms involved in it.
- A block diagram and its configuration for a three layer multilayer FF NN of type $l - m - n$ is shown in the next slide.

Specifying a MLFFNN



Linear transfer function, e

$$f_i^I = (I_i, \theta_i)$$

Log-Sigmoid transfer function

$$f_j^m = (I_j^H, \alpha_j) = \frac{1}{1+e^{-\alpha_j I_j^H}}$$

Tan-Sigmoid transfer function

$$f_k^n = (I_k^o, \alpha_k) = \frac{e^{+\alpha_o I_o} - e^{-\alpha_o I_o}}{e^{+\alpha_o I_o} + e^{-\alpha_o I_o}}$$

Specifying a MLFFNN

- For simplicity, we assume that all neurons in a particular layer follow the same transfer function and different layers follows their respective transfer functions as shown in the configuration.
- Let us consider a specific neuron in each layer say i-th, j-th and k-th neurons in the input, hidden and output layer, respectively.
- Also, let us denote the weight between i-th neuron $(i = 1, 2, \ldots, l)$ in input layer to j-th neuron $(j = 1, 2, \ldots, m)$ in the hidden layer is denoted by v_{ij} .

Specifying a MLFFNN

- The weight matrix between the input to hidden layer say V is denoted as follows.

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1j} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2j} & \cdots & v_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{i1} & v_{i2} & \cdots & v_{ij} & \cdots & v_{im} \\ v_{l1} & v_{l2} & \cdots & v_{lj} & \cdots & v_{lm} \end{bmatrix}$$

Specifying a MLFFNN

- Similarly, w_{jk} represents the connecting weights between $j-th$ neuron $(j = 1, 2, \ldots, m)$ in the hidden layer and $k-th$ neuron $(k = 1, 2, \ldots, n)$ in the output layer as follows.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2k} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{ik} & \cdots & w_{in} \\ w_{m1} & w_{m2} & \cdots & w_{mk} & \cdots & w_{mn} \end{bmatrix}$$

Learning a MLFFNN

- Whole learning method consists of the following three computations:
    1. **Input layer computation**
    2. **Hidden layer computation**
    3. **Output layer computation**
- In our computation, we assume that $\mathrm{T} = < T_o, T_I >$ be the training set of size $|T|$.

Input layer computation

- Let us consider an input training data at any instant be

  $I^I = [I_1^1, I_2^1, \dots, I_i^1, I_l^1]$ where $I^I \in T_I$

- Consider the outputs of the neurons lying on input layer are the same with the corresponding inputs to neurons in hidden layer. That is,

$$O^I = I^I$$
$$[l \times 1] = [l \times 1]$$

[Output of the input layer]

Input layer computation

- The input of the j-th neuron in the hidden layer can be calculated as follows.

$$I_j^H = [v_{1j}O_1^I + v_{2j}O_2^I + \cdots + v_{ij}O_j^I + v_{lj}O_l^I]$$

where $j = 1, 2, \ldots, m$.

[Calculation of input of each node in the hidden layer]

- In the matrix representation form, we can write

$$I^H = V^T . O^I$$

$$[m \times 1] = [m \times l][l \times 1]$$

Hidden layer computation

- Let us consider any j-th neuron in the hidden layer.
- Since the output of the input layer's neurons are the input to the j-th neuron and the j-th neuron follows the log-sigmoid transfer function, we have

$$O_j^H = \frac{1}{1 + e^{-\alpha_H . I_j^H}}$$

where $j = 1, 2, \ldots, m$ and α_H is the constant co-efficient of the transfer function.

Hidden layer computation

- Note that all output of the nodes in the hidden layer can be expressed as a one-dimensional column matrix.

$$O^H = \begin{bmatrix} \cdots \\ \cdots \\ \vdots \\ \dfrac{1}{1+e^{-\alpha_H . I_j^H}} \\ \vdots \\ \cdots \\ \cdots \end{bmatrix}_{m\times 1}$$

Output layer computation

Let us calculate the input to any k-th node in the output layer. Since, output of all nodes in the hidden layer go to the k-th layer with weights $w_{1k}, w_{2k}, \dots, w_{mk}$, we have

$$I_k^O = w_{1K}.o_1^H + w_{2k}.o_2^H + \cdots + w_{mk}.o_m^H$$

where $k = 1, 2, \dots, n$.

In the matrix representation form, we have

$$I^O = W^T.O^H$$

$$[n \times 1] = [n \times m][m \times 1]$$

Output layer computation

Now, we estimate the output of the k-th neuron in the output layer. We consider the tan-sigmoid transfer function.

$$O_k = \frac{e^{\alpha_O . I_k^O} - e^{-\alpha_O . I_k^O}}{e^{\alpha_O . I_k^O} + e^{-\alpha_O . I_k^O}}$$

where $k = 1, 2, \dots, \text{n}$

Output layer computation

- Hence, the output of output layer's neurons can be represented as

$$O = \begin{bmatrix} \cdots \\ \cdots \\ \vdots \\ \dfrac{e^{\alpha_O . I_k^O} - e^{-\alpha_O . I_k^O}}{e^{\alpha_O . I_k^O} + e^{-\alpha_O . I_k^O}} \\ \vdots \\ \cdots \\ \cdots \end{bmatrix}_{n\times 1}$$

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Artificial Neural Network: Training-III

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Back Propagation Algorithm

- The above discussion comprises how to calculate values of different parameters in $l - m - n$ multiple layer feed forward neural network.
- Next, we will discuss how to train such a neural network.
- We consider the most popular algorithm called Back- Propagation algorithm, which is a supervised learning.
- The principle of the Back-Propagation algorithm is based on the error-correction with Steepest-descent method.
- We first discuss the method of steepest descent followed by its use in the training algorithm.

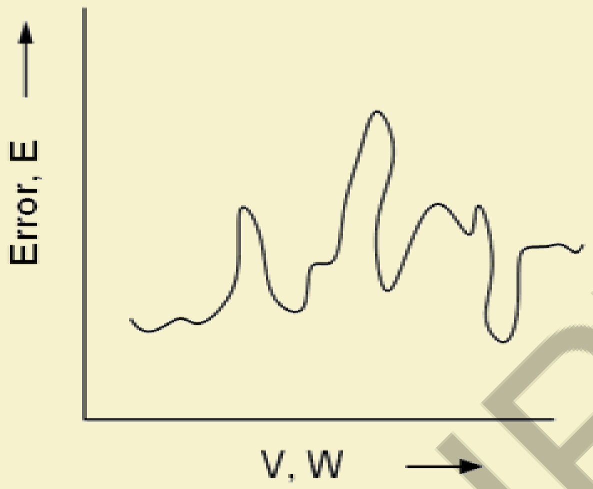
Method of Steepest Descent

- Supervised learning is, in fact, error-based learning.
- In other words, with reference to an external (teacher) signal (i.e. target output) it calculates error by comparing the target output and computed output.
- Based on the error signal, the neural network should modify its configuration, which includes synaptic connections, that is , the weight matrices.

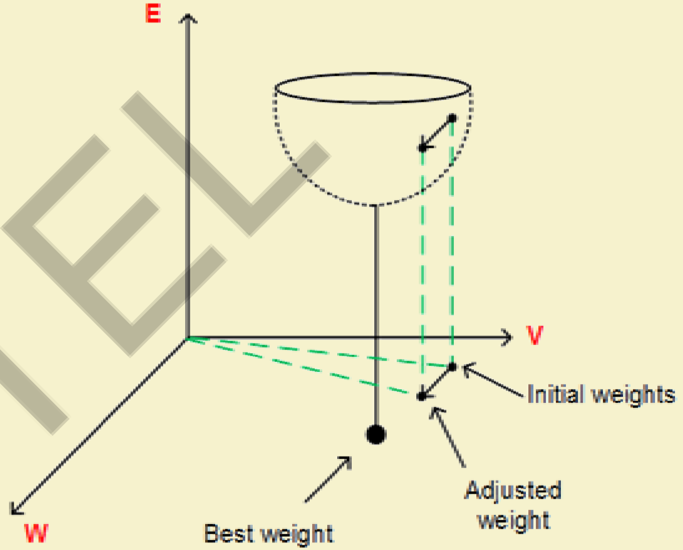
Method of Steepest Descent

- It should try to reach to a state, which yields minimum error.
- In other words, its searches for a suitable values of parameters minimizing error, given a training set.
- Note that, this problem turns out to be an optimization problem.

Method of Steepest Descent



(a) Searching for a minimum error



(b) Error surface with two parameters V and W

Method of Steepest Descent

- For simplicity, let us consider the connecting weights are the only design parameter.
- Suppose, V and W are the weights parameters to hidden and output layers, respectively.
- Thus, given a training set of size N, the error surface, E can be represented as

$$E = \sum_{i=1}^{N} e^{i}(V, W, I_i)$$

where I_i is the i-th input pattern in the training set and e^i (...) denotes the error computation of the i-th input.

Method of Steepest Descent

- Now, we will discuss the steepest descent method of computing error, given a changes in V and W matrices.
- Suppose, A and B are two points on the error surface. The vector $\overrightarrow{AB}$ can be written as

$$\overrightarrow{AB} = (V_{i+1} - V_i).\bar{x} + (W_{i+1} - W_i).\bar{y} = \Delta V.\bar{x} + \Delta W.\bar{y}$$

The gradient of $\overrightarrow{AB}$ can be obtained as

$$e_{\overrightarrow{AB}} = \frac{\partial E}{\partial V}.\bar{x} + \frac{\partial E}{\partial W}.\bar{y}$$

Method of Steepest Descent

- Hence, the unit vector in the direction of gradient is

$$\bar{e}_{\overrightarrow{AB}} = \frac{1}{\left|e_{\overrightarrow{AB}}\right|}\left[\frac{\partial E}{\partial V}.\bar{x} + \frac{\partial E}{\partial W}.\bar{y}\right]$$

With this, we can alternatively represent the distance vector AB as

$$\overrightarrow{AB} = \eta\left[\frac{\partial E}{\partial V}.\bar{x} + \frac{\partial E}{\partial W}.\bar{y}\right]$$

where $\eta = \frac{k}{\left|e_{\overrightarrow{AB}}\right|}$ and k is a constant

Method of Steepest Descent

- So, comparing both, we have

$$\Delta V = \eta \frac{\partial E}{\partial V}$$

$$\Delta W = \eta \frac{\partial E}{\partial W}$$

This is also called as **delta rule** and is called **learning rate**.

Calculation of error in a neural network

- Let us consider any k-th neuron at the output layer. For an input pattern $I_i \in T_I$ (input in training) the target output T_{O_k} of the k -th neuron be T_{O_k} .
- Then, the error e_k of the k-th neuron is defined corresponding to the input I_i as

$$e_k = \frac{1}{2}\left(T_{O_k} - O_{O_k}\right)^2$$

where O_{O_k} denotes the observed output of the k-th neuron.

Calculation of error in a neural network

- For a training session with $I_i \in T_I$ the error in prediction considering all output neurons can be given as

$$e = \sum_{k=1}^{n} e_k = \frac{1}{2}\sum_{k=1}^{n}(T_{O_k} - O_{O_k})^2$$

where n denotes the number of neurons at the output layer.

- The total error in prediction for all output neurons can be determined considering all training session $< T_I, T_O >$ as

$$E = \sum_{\forall I_i \in T_I} e = \frac{1}{2}\sum_{\forall t \in <T_I, T_O>}\sum_{k=1}^{n}\left(T_{O_k} - O_{O_k}\right)^2$$

Supervised learning : Back-propagation algorithm

- The back-propagation algorithm can be followed to train a neural network to set its topology, connecting weights, bias values and many other parameters.
- In this present discussion, we will only consider updating weights. Thus, we can write the error E corresponding to a particular training scenario T as a function of the variable V and W. That is

$$E = f(V, W, T)$$

Supervised learning : Back-propagation algorithm

- In BP algorithm, this error E is to be minimized using the gradient descent method. We know that according to the gradient descent method, the changes in weight value can be given as

$$\Delta V = -\eta \frac{\partial E}{\partial V} \qquad (1)$$

$$\Delta W = -\eta \frac{\partial E}{\partial W} \qquad (2)$$

Supervised learning : Back-propagation algorithm

- Note that $-ve$ sign is used to signify the fact that if $\frac{\partial E}{\partial V}$ (or $\frac{\partial E}{\partial W}$) > 0, then we have to decrease V and vice-versa.

- Let v_{ij} (and w_{jk}) denotes the weights connecting i-th neuron (at the input layer) to j-th neuron(at the hidden layer) and connecting j-th neuron (at the hidden layer) to k-th neuron (at the output layer).

- Also, let e_k denotes the error at the k-th neuron with observed output as $O_{O_k^o}$ and target output $T_{O_k^o}$ as per a sample input $I \in T_I$.

Supervised learning : Back-propagation algorithm

- It follows logically therefore,

$$e_k = \frac{1}{2}\left(T_{O_k^O} - O_{O_k^O}\right)^2$$

and the weight components should be updated according to equation (1) and (2) as follows,

$\bar{w}_{jk} = w_{jk} + \Delta w_{jk}$ (3) where $\Delta w_{jk} = -\eta \frac{\partial e_k}{\partial w_{jk}}$

$\bar{v}_{ij} = v_{ij} + \Delta v_{ij}$ (4) where $\Delta v_{ij} = -\eta \frac{\partial e_k}{\partial v_{ij}}$

Supervised learning : Back-propagation algorithm

- Here, v_{ij} and w_{jk} denotes the previous weights and $\bar{v}_{ij}$ and $\overline{w}_{jk}$ denote the updated weights.
- Next, we will learn the calculation of $\bar{v}_{ij}$ and $\overline{w}_{jk}$.

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Artificial Neural Network: Training-IV

Prof. Debasis Samanta
Department of Computer Science & Engineering
IIT Kharagpur

Supervised learning : Back-propagation algorithm

- ✓ Note that $-ve$ sign is used to signify the fact that if $\frac{\partial E}{\partial V}$ (or $\frac{\partial E}{\partial W}$) > 0, then we have to decrease V and vice-versa.
- ✓ Let v_{ij} (and w_{jk}) denotes the weights connecting i-th neuron (at the input layer) to j-th neuron(at the hidden layer) and connecting j-th neuron (at the hidden layer) to k-th neuron (at the output layer).
- ✓ Also, let e_k denotes the error at the k-th neuron with observed output as $O_{O_k^o}$ and target output $T_{O_k^o}$ as per a sample input $I \in T_I$.

Supervised learning : Back-propagation algorithm

- ✓ It follows logically therefore,

$$e_k = \frac{1}{2}\left(T_{O_k^o} - O_{O_k^o}\right)^2$$

and the weight components should be updated according to equation (1) and (2) as follows,

$$\bar{w}_{jk} = w_{jk} + \Delta w_{jk} \quad (3) \qquad \text{where } \Delta w_{jk} = -\eta \frac{\partial e_k}{\partial w_{jk}}$$

$$\bar{v}_{ij} = v_{ij} + \Delta v_{ij} \quad (4) \qquad \text{where } \Delta v_{ij} = -\eta \frac{\partial e_k}{\partial v_{ij}}$$

Supervised learning : Back-propagation algorithm

- ✓ Here, v_{ij} and w_{jk} denotes the previous weights and $\bar{v}_{ij}$ and $\overline{w}_{jk}$ denote the updated weights.
- ✓ Now we will learn the calculation of $\bar{v}_{ij}$ and $\overline{w}_{jk}$.

Calculation of $\overline{w}_{jk}$

We can calculate $\frac{\partial e_k}{\partial w_{jk}}$ using the chain rule of differentiation as stated below.

$$\frac{\partial e_k}{\partial w_{jk}} = \frac{\partial e_k}{\partial O_{O_k^O}} \cdot \frac{\partial O_{O_k^O}}{\partial I_k^O} \cdot \frac{\partial I_k^O}{\partial w_{jk}} \qquad (5)$$

Now we have

$$e_k = \frac{1}{2}\left(T_{O_k^O} - O_{O_k^O}\right)^2 \qquad (6)$$

Calculation of $\overline{w}_{jk}$

$$O_{o_k^O} = \frac{e^{\theta_O . I_k^O} - e^{-\theta_O . I_k^O}}{e^{\theta_O . I_k^O} + e^{-\theta_O . I_k^O}} \quad (7)$$

$$I_k^O = w_{1k}.O_1^H + w_{2k}.O_2^H + \cdots + w_{mk}.O_m^H \quad (8)$$

Thus,

$$\frac{\partial e_k}{\partial O_{o_k^O}} = -\left(T_{o_k^O} - O_{o_k^O}\right) \quad (9)$$

$$\frac{\partial O_{o_k^O}}{\partial I_k^O} = \theta_O (1 + O_{o_k^O})(1 - O_{o_k^O}) \quad (10)$$

Calculation of $\overline{w}_{jk}$

$$\frac{\partial I_k^O}{\partial w_{jk}} = O_j^H \tag{11}$$

Substituting the value of $\frac{\partial e_k}{\partial O_{O_k^o}}$, $\frac{\partial O_{O_k^o}}{\partial I_k^o}$ and $\frac{\partial I_k^O}{\partial w_{jk}}$ we have

$$\frac{\partial e_k}{\partial w_{jk}} = -\left(T_{O_k^o} - O_{O_k^o}\right).\theta_O\left(1 + O_{O_k^o}\right)\left(1 - O_{O_k^o}\right).O_j^H \tag{12}$$

Again, substituting the value of $\frac{\partial e_k}{\partial w_{jk}}$ from Eq. (12) in Eq.(3), we have

Calculation of $\overline{w}_{jk}$

$$\Delta w_{jk} = \eta . \theta_O \left(T_{O_k^O} - O_{O_k^O}\right) . \left(1 + O_{O_k^O}\right)\left(1 - O_{O_k^O}\right) . O_j^H \qquad (13)$$

Therefore, the updated value of w_{jk} can be obtained using Eq. (3)

$$\overline{w}_{jk} = w_{jk} + \Delta w_{jk}$$

$$\overline{w}_{jk} = w_{jk} + \eta . \theta_O \left(T_{O_k^O} - O_{O_k^O}\right) . \left(1 + O_{O_k^O}\right)\left(1 - O_{O_k^O}\right) . O_j^H \qquad (14)$$

Calculation of $\bar{v}_{ij}$

Like $\frac{\partial e_k}{\partial w_{jk}}$, we can calculate $\frac{\partial e_k}{\partial v_{ij}}$ using the chain rule of differentiation as follows.

$$\frac{\partial e_k}{\partial v_{ij}} = \frac{\partial e_k}{\partial O_{O_k^O}} \cdot \frac{\partial O_{O_k^O}}{\partial I_k^O} \cdot \frac{\partial I_k^O}{\partial O_j^H} \cdot \frac{\partial O_j^H}{\partial I_j^H} \cdot \frac{\partial I_j^H}{\partial v_{ij}} \qquad (15)$$

Now we have

$$e_k = \frac{1}{2}\left(T_{O_k^O} - O_{O_k^O}\right)^2 \qquad (16)$$

Calculation of $\bar{v}_{ij}$

$$O_k^O = \frac{e^{\theta_O . I_k^O} - e^{-\theta_O . I_k^O}}{e^{\theta_O . I_k^O} + e^{-\theta_O . I_k^O}} \tag{17}$$

$$I_k^O = w_{1K} . O_1^H + w_{2k} . O_2^H + \cdots , + w_{mk} . O_m^H \tag{18}$$

$$O_j^H = \frac{1}{1 + e^{-\alpha_H . I_j^H}} \tag{19}$$

$$I_j^H = v_{1j} . O_1^H + v_{2j} . O_2^H + \cdots + v_{ij} . O_j^I + v_{ij} . O_l^I \tag{20}$$

Calculation of $\bar{v}_{ij}$

Thus,

$$\frac{\partial e_k}{\partial O_{o_k^O}} = -\left(T_{o_k^O} - O_{o_k^O}\right) \tag{21}$$

$$\frac{\partial O_{o_k^O}}{\partial I_k^O} = \theta_O(1 + O_{o_k^O})(1 - O_{o_k^O}) \tag{22}$$

$$\frac{\partial I_k^O}{\partial O_j^H} = w_{ik} \tag{23}$$

Calculation of $\bar{v}_{ij}$

$$\frac{\partial O_j^H}{\partial I_j^H} = \theta_H.\left(1 - O_j^H\right).O_j^H \tag{24}$$

$$\frac{\partial I_j^H}{\partial v_{ij}} = O_i^H = I_i^H \tag{25}$$

From the above equations, we get

$$\frac{\partial e_k}{\partial v_{ij}} = -\theta_H.\theta_O\left(T_{O_k^O} - O_{O_k^O}\right).\left(1 - O^2_{O_k^O}\right).O_j^H.I_i^H.w_{jk} \tag{26}$$

Calculation of $\bar{v}_{ij}$

Again, substituting the value of $\frac{\partial e_k}{\partial v_{ij}}$ in Eq.(4), we have

$$\Delta v_{ij} = \eta . \theta_H . \theta_O \left(T_{O_k^O} - O_{O_k^O} \right) . \left(1 - O^2_{O_k^O} \right) . O_j^H . I_i^H . w_{jk} \qquad (27)$$

Therefore, the updated value of v_{ij} can be obtained using Eq.(4)

$$\bar{v}_{ij} = v_{ij} + \eta . \theta_H . \theta_O \left(T_{O_k^O} - O_{O_k^O} \right) . \left(1 - O^2_{O_k^O} \right) . O_j^H . I_i^H . w_{jk} \qquad (28)$$

Writing in matrix form for the calculation of $\overline{V}$ and $\overline{W}$

we have

$$\Delta w_{jk} = \eta \left| \theta_O . \left(T_{O_k^O} - O_{O_k^O} \right) . \left(1 + O_{O_k^O} \right) \left(1 - O_{O_k^O} \right) \right| . O_j^H \quad (29)$$

is the update for k-th neuron receiving signal from j-th neuron at hidden layer.

$$\Delta v_{ij} = \eta . \theta_H . \theta_O \left(T_{O_k^O} - O_{O_k^O} \right) . \left(1 - O_{O_k^O}^2 \right) . (1 - O_j^H) . O_j^H . I_i^H . w_{jk} \quad (30)$$

is the update for j-th neuron at the hidden layer for the i-th input at the i-th neuron at input level.

calculation of $\overline{W}$

Hence

$$[\Delta W]_{m\times n} = \eta.[O^H]_{m\times 1}.[N]_{1\times n} \tag{31}$$

Where

$$[N]_{1\times n} = \left\{\theta_O\left(T_{O_{O_k}^O} - O_{O_{O_k}^O}\right).\left(1 - O^2_{O_{O_k}^O}\right)\right\} \tag{32}$$

where $k = 1,2, \ldots, n$

Thus, the updated weight matrix for a sample input can be written as

$$[\overline{W}]_{m\times n} = [W]_{m\times n} + [\Delta W]_{m\times n} \tag{33}$$

calculation of $\bar{V}$

Similarly, for $[\bar{V}]$ matrix, we can write

$$\Delta v_{ij} = \eta . \left| \theta_O \left(T_{O_k^O} - O_{O_k^O} \right) . \left(1 - O_{O_k^O}^2 \right) . w_{jk} \right| . \left| \theta_H \left(1 - O_j^H \right) . O_j^H \right| . \left| I_i^H \right| \quad (34)$$

$$= \eta . w_j . \theta_H . \left(1 - O_j^H \right) . O_j^H \quad (35)$$

Thus,

$$\Delta V = [I^I]_{l \times 1} \times [M^T]_{1 \times m} \quad (36)$$

or

$$[\bar{V}]_{l \times m} = [V]_{l \times m} + [I^I]_{l \times 1} \times [M^T]_{1 \times m} \quad (37)$$

calculation of $\overline{V}$

This calculation of Eq. (32) and (36) for one training data $t =< T_O, T_I >$. We can apply it in incremental mode (i.e. one sample after another) and after each training data, we update the networks V and W matrix.

Batch mode of training

- ✓ A batch mode of training is generally implemented through the minimization of mean square error (MSE) in error calculation. The MSE for k-th neuron at output level is given by

$$\bar{E} = \frac{1}{2} \cdot \frac{1}{|T|} \sum_{t=1}^{|T|} \left(T^{t}_{O^{O}_{k}} - O^{t}_{O^{O}_{k}} \right)^2$$

where $|T|$ denotes the total number of training scenarios and t denotes a training scenario, i.e. $t = < T_O, T_I >$.

Batch mode of training

In this case, Δw_{jk} and Δv_{ij} can be calculated as follows

$$\Delta w_{jk} = \frac{1}{|T|} \sum_{\forall t \in T} \frac{\partial \bar{E}}{\partial W}$$

and

$$\Delta v_{ij} = \frac{1}{|T|} \sum_{\forall t \in T} \frac{\partial \bar{E}}{\partial V}$$

Once Δw_{jk} and Δv_{ij} are calculated, we will be able to obtain $\bar{w}_{jk}$ and $\bar{v}_{ij}$

Thank You!!

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Introduction to Soft Computing

Soft Computing Tools and Hybrid Computing

Debasis Samanta
Department of Computer Science and Engineering
IIT KHARAGPUR

Tools for Fuzzy Logic

Some of the popular tools of Fuzzy Logic are tabulated below

Tool	Link	Comment
Fuzzy Logic Toolbox	https://in.mathworks.com/products/fuzzy-logic.html	Matlab Toolbox
MLF	http://www.unisoftwareplus.com/products/mlf/index.html	Commercial
LFLC	http://irafm.osu.cz/en/c49_linguistic-fuzzy-logic-controller-lflc/	Commercial
FisPro	http://www7.inra.fr/mia/M/fispro/fispro2013_en.html	Open Source
Kappalab	https://cran.r-project.org/web/packages/kappalab /index.html	Open Source
GUAJE FUZZY	https://sourceforge.net/projects/guajefuzzy/	Open Source

Matlab Fuzzy Toolbox

- ✓ Matlab Fuzzy Toolbox consist of two useful tools:

  FIS Editor: This Editor in combination with 4 other editors provides a powerful environment to define and modify Fuzzy Inference System (FIS) variable

  Fuzzy Controller: This is a block in fuzzy Toolbox Library in Simulink environment. This block admits FIS variable produced by FIS Editor and implements the desirable rules

To start the MATLAB fuzzy toolbox type "fuzzy" in the command line of MATLAB

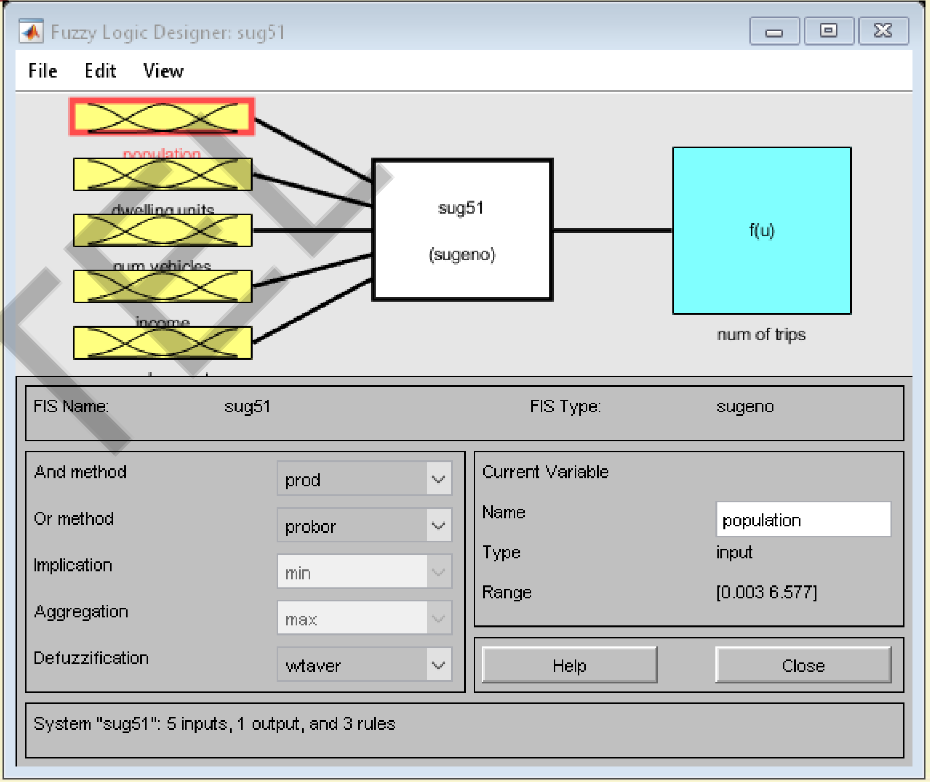
Example: Understanding Traffic Patterns

In this example we attempt to understand the relationship between the number of automobile trips generated from an area and the area's demographics. Demographic and trip data were collected from traffic analysis zones in New Castle County, Delaware. Five demographic factors are considered: population, number of dwelling units, vehicle ownership, median household income and total employment.

Example: Understanding Traffic Patterns

- ✓ Define the inputs and output. For the given problem we have five input and one output.
- ✓ Assign names to inputs and outputs

- ✓ As can be seen, the FIS has 5 inputs and 1 output with the inputs mapped to the outputs through a rule-base (white box in the figure).



FIS Name: sug51
FIS Type: sugeno

And method: prod
Or method: probor
Implication: min
Aggregation: max
Defuzzification: wtaver

Current Variable
Name: population
Type: input
Range: [0.003 6.577]

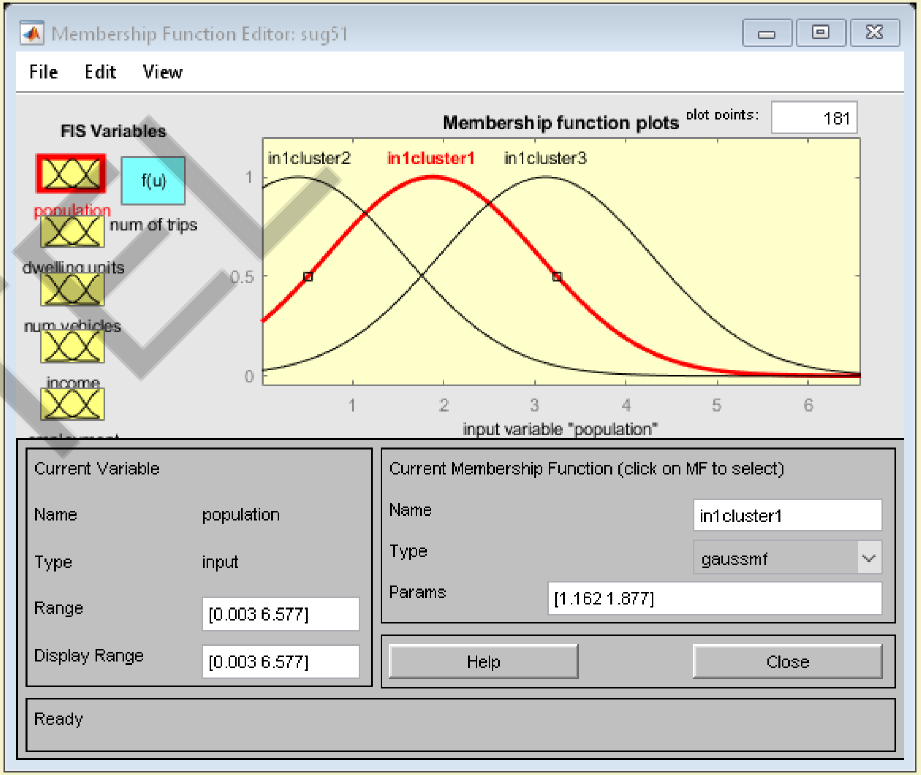
System "sug51": 5 inputs, 1 output, and 3 rules

Example: Understanding Traffic Patterns

- ✓ Notice that all the inputs and outputs have exactly 3 membership functions. The 3 membership functions represent the 3 clusters

- ✓ Each input in the FIS represents an input variable in the input dataset and each output in the FIS represents an output variable in the output dataset

- ✓ Notice that the membership function type is "gaussmf" (Gaussian function) and the parameters of the membership function are [1.162 1.877], where 1.162 represents the spread coefficient of the Gaussian curve and 1.877 represents the centre of the Gaussian curve



Current Variable

Name	population
Type	input
Range	[0.003 6.577]
Display Range	[0.003 6.577]

Current Membership Function (click on MF to select)

Name	in1cluster1
Type	gaussmf
Params	[1.162 1.877]

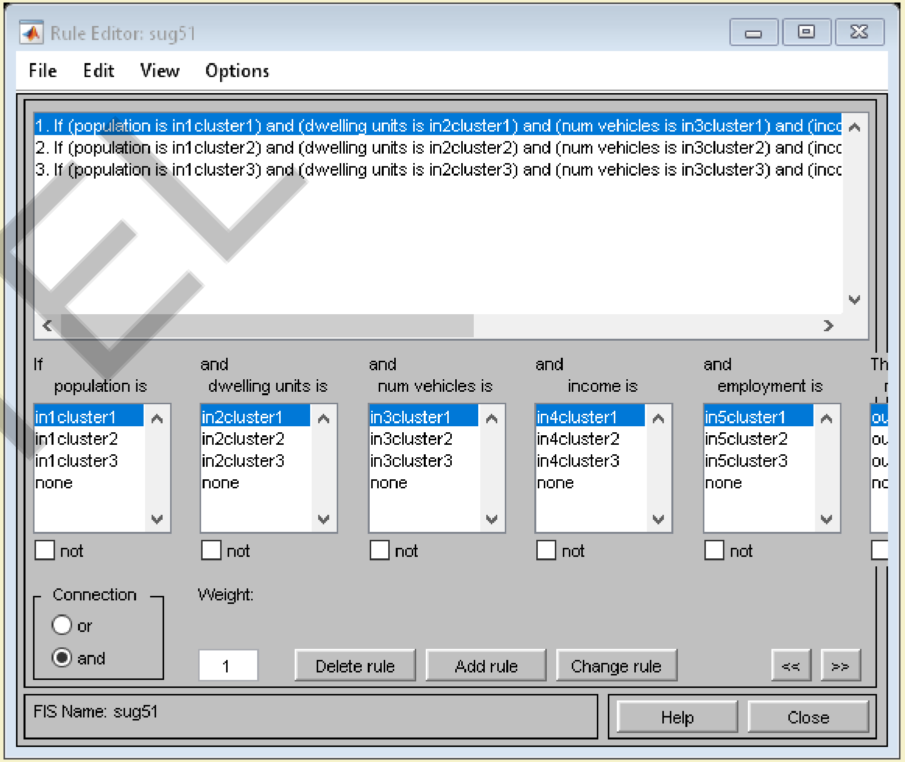
Ready

Fuzzy rules

- ✓ The first rule can be explained simply as follows.

If the inputs to the FIS, population, dwelling units, num vehicles, income, and employment, strongly belong to their respective cluster1 membership functions then the output, num of trips, must strongly belong to its cluster1 membership function.

- ✓ Similarly other rules can be defined



1. If (population is in1cluster1) and (dwelling units is in2cluster1) and (num vehicles is in3cluster1) and (inco
2. If (population is in1cluster2) and (dwelling units is in2cluster2) and (num vehicles is in3cluster2) and (inco
3. If (population is in1cluster3) and (dwelling units is in2cluster3) and (num vehicles is in3cluster3) and (inco

If population is	and dwelling units is	and num vehicles is	and income is	and employment is	Th
in1cluster1	in2cluster1	in3cluster1	in4cluster1	in5cluster1	ou
in1cluster2	in2cluster2	in3cluster2	in4cluster2	in5cluster2	ou
in1cluster3	in2cluster3	in3cluster3	in4cluster3	in5cluster3	ou
none	none	none	none	none	nc
☐ not	☐ not	☐ not	☐ not	☐ not	☐

Connection: ○ or [✓ ● and]

Weight: 1

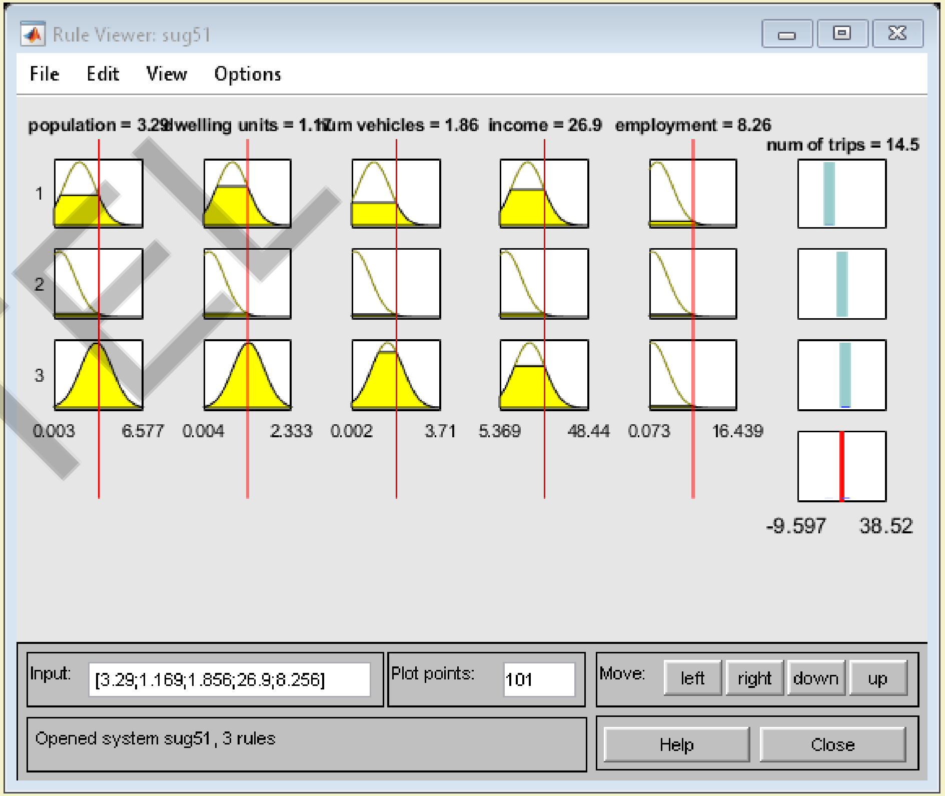
FIS Name: sug51

Output Evaluation

- ✓ After defining all possible fuzzy rules the output can be analyzed by providing all the input value to the fuzzy system
- ✓ In our case, three fuzzy rules are defined.
- ✓ For the given input

  [3.29, 1.169,1.656,26.9,8.256]
- ✓ The output is

  14.5



population = 3.29 dwelling units = 1.17 num vehicles = 1.86 income = 26.9 employment = 8.26

num of trips = 14.5

Input: [3.29;1.169;1.856;26.9;8.256]

Plot points: 101

Move:

Opened system sug51 , 3 rules

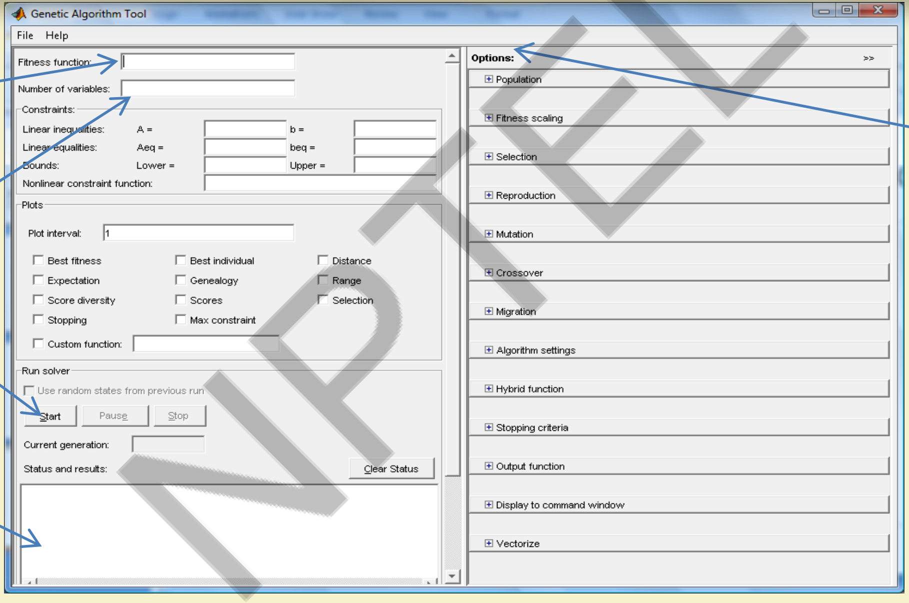
Tools for Genetic Algorithm

Some of the popular tools of GA are tabulated below

Tool	Link	Comment
Genetic Algorithm Toolbox	https://in.mathworks.com/help/gads/genetic-algorithm.html	Toolbox for Matlab
GEATbx	http://www.geatbx.com/	Toolbox for Matlab
ECJ	https://cs.gmu.edu/~eclab/projects/ecj/	Open Source
Evolver	http://www.palisade.com/evolver/	Commercial
GeneHunter	http://www.wardsystems.com/genehunter.asp	Commercial

Using the Genetic Algorithm Tool (Matlab)

To start the MATLAB ANN toolbox type “gatool” in the command line of MATLAB



Genetic Algorithm Tool

File Help

Fitness function:

Number of variables:

Constraints:

Linear inequalities: A = b =

Linear equalities: Aeq = beq =

Bounds: Lower = Upper =

Nonlinear constraint function:

Plots

Plot interval: 1

☐ Best fitness ☐ Best individual ☐ Distance

☐ Expectation ☐ Genealogy ☐ Range

☐ Score diversity ☐ Scores ☐ Selection

☐ Stopping ☐ Max constraint

☐ Custom function:

Run solver

☐ Use random states from previous run

Start Pause Stop

Current generation:

Status and results: Clear Status

Options:

⊞ Population

⊞ Fitness scaling

⊞ Selection

⊞ Reproduction

⊞ Mutation

⊞ Crossover

⊞ Migration

⊞ Algorithm settings

⊞ Hybrid function

⊞ Stopping criteria

⊞ Output function

⊞ Display to command window

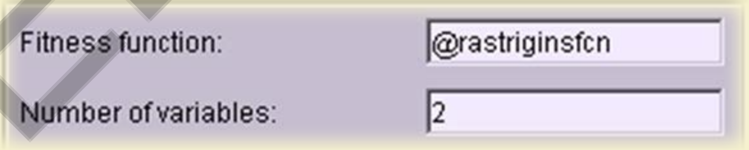
⊞ Vectorize

Example: Rastrigin's Function

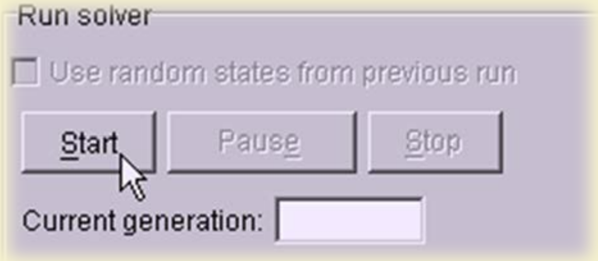
Finding the Minimum of Rastrigin's Function

$$Ras(x) = 20 + {x_1}^2 + {x_2}^2 - 10(cos2\pi x_1 + cos2\pi x_2)$$

1. Enter **gatool** at the command line to open the Genetic Algorithm Tool.
2. Enter the following in the Genetic Algorithm Tool: In the Fitness function field, enter **@rastriginsfcn.**
3. In the Number of variables field, enter 2, the number of independent variables for Rastrigin's function.
4. Click the Start button in the Run solver pane


Fitness function: @rastriginsfcn

Number of variables: 2

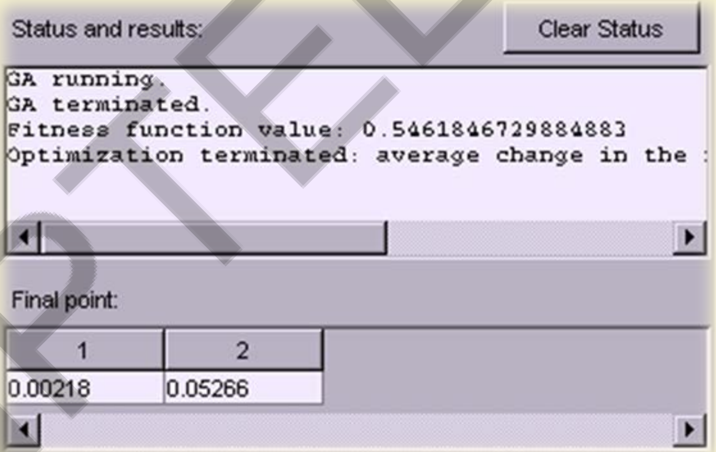


Example: Rastrigin's Function

Finding the Minimum of Rastrigin's Function

$$Ras(x) = 20 + {x_1}^2 + {x_2}^2 - 10(cos2\pi x_1 + cos2\pi x_2)$$

The final value of the fitness function when the algorithm terminated: Function value: **0.5461846729884883**


Status and results:

```
GA running.
GA terminated.
Fitness function value: 0.5461846729884883
Optimization terminated: average change in the
```

Final point:

1	2
0.00218	0.05266

Tools for ANN

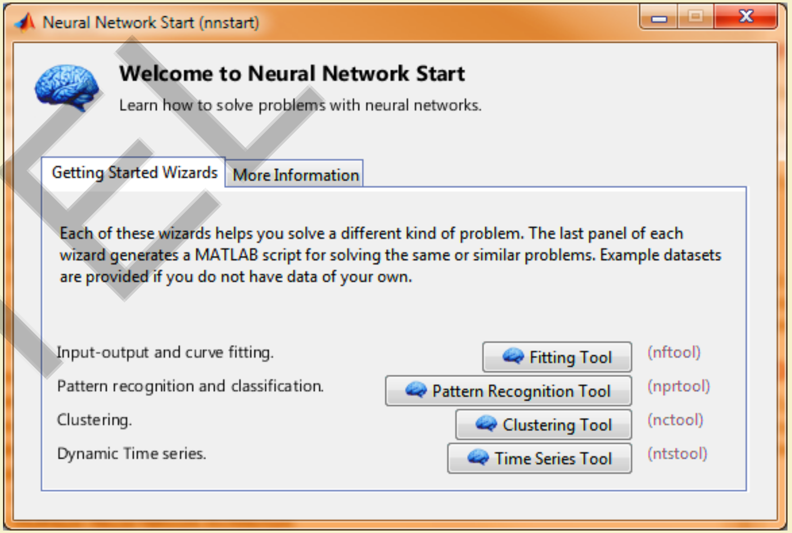
Some of the popular tools of ANN are tabulated below

Tool	Link	Comment
Neural Network Toolbox	https://in.mathworks.com/products/neural-network.html	Toolbox for MATLAB
FANN	http://leenissen.dk/fann/wp/	Open Source
Neuro Modeler	https://sourceforge.net/projects/neuromodeler/	Open Source
WEKA	https://www.cs.waikato.ac.nz/ml/weka/	Open Source
EasyNN	http://www.easynn.com/	Commercial
Encog Machine Learning Framework	http://www.heatonresearch.com/encog/	Commercial
Statistica	http://statistica.io/	Commercial

Using the ANN Toolbox (Matlab)

MATLAB ANN toolbox is used for the following tasks

- ✓ Function fitting
- ✓ Pattern recognition
- ✓ Data clustering
- ✓ Time series analysis



Neural Network Start (nnstart)

Welcome to Neural Network Start

Learn how to solve problems with neural networks.

Getting Started Wizards | More Information

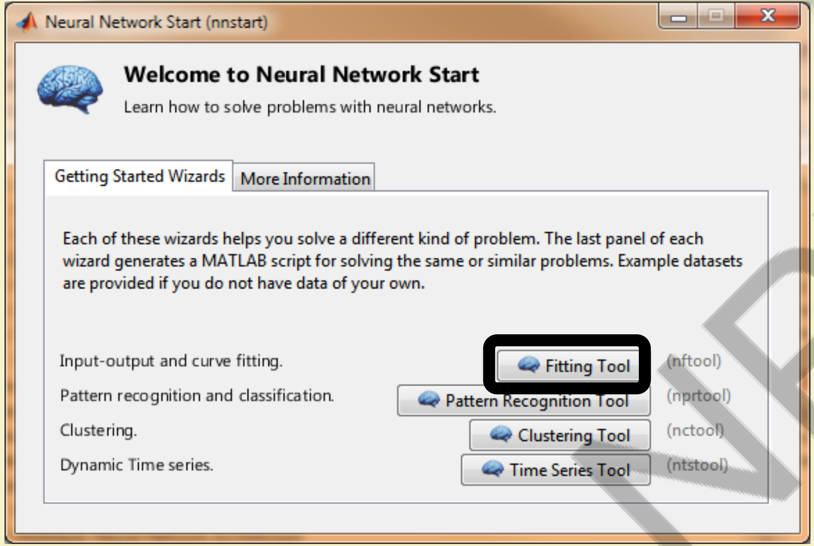
Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	Fitting Tool	(nftool)
Pattern recognition and classification.	Pattern Recognition Tool	(nprtool)
Clustering.	Clustering Tool	(nctool)
Dynamic Time series.	Time Series Tool	(ntstool)

To start the MATLAB ANN toolbox type "nnstart" in the command line of MATLAB

Fitting Tool

Click Fitting Tool to open the Neural Network Fitting Tool.



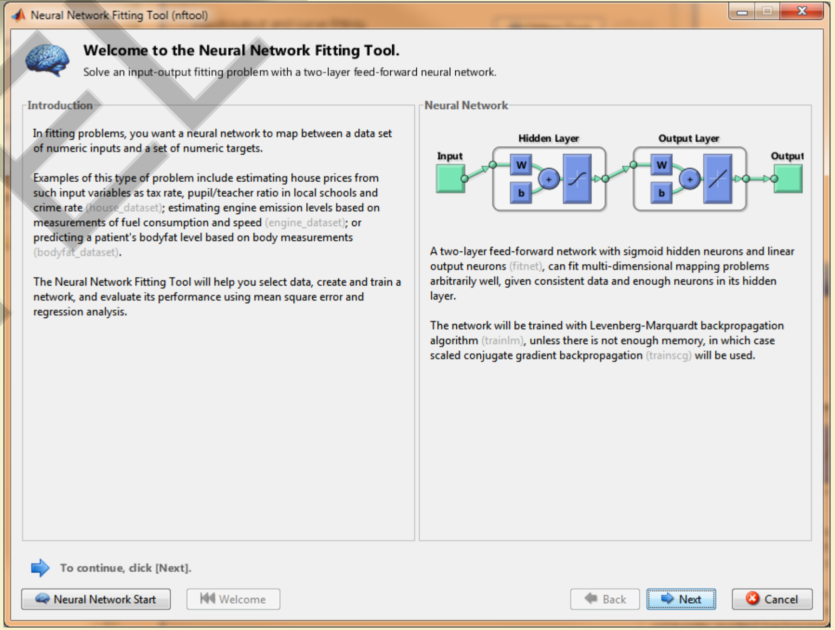
Neural Network Start (nnstart)

Welcome to Neural Network Start

Learn how to solve problems with neural networks.

Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	Fitting Tool	(nftool)
Pattern recognition and classification.	Pattern Recognition Tool	(nprtool)
Clustering.	Clustering Tool	(nctool)
Dynamic Time series.	Time Series Tool	(ntstool)



Neural Network Fitting Tool (nftool)

Welcome to the Neural Network Fitting Tool.

Solve an input-output fitting problem with a two-layer feed-forward neural network.

Introduction

In fitting problems, you want a neural network to map between a data set of numeric inputs and a set of numeric targets.

Examples of this type of problem include estimating house prices from such input variables as tax rate, pupil/teacher ratio in local schools and crime rate (house_dataset); estimating engine emission levels based on measurements of fuel consumption and speed (engine_dataset); or predicting a patient's bodyfat level based on body measurements (bodyfat_dataset).

The Neural Network Fitting Tool will help you select data, create and train a network, and evaluate its performance using mean square error and regression analysis.

Neural Network

A two-layer feed-forward network with sigmoid hidden neurons and linear output neurons (fitnet), can fit multi-dimensional mapping problems arbitrarily well, given consistent data and enough neurons in its hidden layer.

The network will be trained with Levenberg-Marquardt backpropagation algorithm (trainlm), unless there is not enough memory, in which case scaled conjugate gradient backpropagation (trainscg) will be used.

To continue, click [Next].

Fitting Tool

Click Fitting Tool to open the Neural Network Fitting Tool.

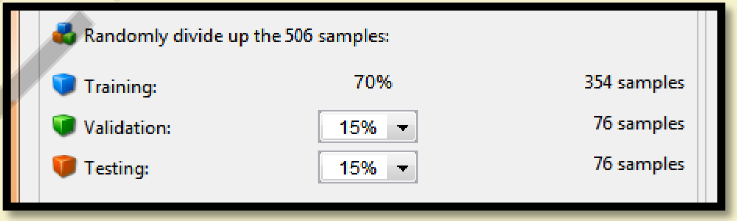
Problem Statement

Suppose, for instance, that you have data from a housing application. You want to design a network that can predict the value of a house (in $1000s), given 13 pieces of geographical and real estate information. You have a total of 506 example homes for which you have those 13 items of data and their associated market values

Fitting Tool

- ✓ Click Load Example Data Set in the Select Data window. The Fitting Data Set Chooser window opens
- ✓ Select House Pricing, and click Import. This returns you to the Select Data window.
- ✓ Click Next to display the Validation and Test Data window, shown in the following figure.

The validation and test data sets are each set to 15% of the original data.


Randomly divide up the 506 samples:

Training:	70%	354 samples
Validation:	15%	76 samples
Testing:	15%	76 samples

- ✓ The standard network that is used for function fitting is a two-layer feedforward network, with a sigmoid transfer function in the hidden layer and a linear transfer function in the output layer. The default number of hidden neurons is set to 10
- ✓ Click Next to reach the training window and then click Train.
- ✓ The training continued until the validation error failed to decrease for six iterations

Hybrid Computing

Hybridization

- ✓ Hybrid systems employ more than one technology to solve a problem.
- ✓ Hybridization of technologies can have pitfalls and therefore need to be done with care.
- ✓ If one technology can solve a problem then a hybrid technology ought to be used only if its application results in a better solution.

Hybrid Systems

- ✓ ***<u>Sequential hybrid system</u>***:
  The technologies are used in pipelining fashion;

- ✓ ***<u>Auxiliary hybrid system</u>*****:**
  The one technology calls the other technology as subroutine;

- ✓ ***<u>Embedded hybrid system</u>*****:**
  The technologies participating appear to be fused totally.

Sequential Hybrid System

In Sequential hybrid system, the technologies are used in pipelining fashion. Thus, one technology's output becomes another technology's input and it goes on. However, this is one of the weakest form of hybridization since an integrated combination of technologies is not present.

Example: A Genetic algorithm pre-processor obtains the optimal parameters for different instances of a problem and hands over the pre-processed data to a neural network for further processing.

Auxiliary Hybrid System

In Auxiliary hybrid system, one technology calls the other technology as subroutine to process or manipulate information needed. The second technology processes the information provided by the first and hands it over for further use. This type of hybridization is better than the sequential hybrids.

Example: A neuron-genetic system in which a neural network employs a genetic algorithm to optimize its structural parameters that defines its architecture.

Embedded Hybrid System

In Embedded hybrid system, the technologies participating are integrated in such a manner that they appear intertwined. The fusion is so complete that it would appear that no technology can be used without the others for solving the problem.

Example: A NN-FL hybrid system may have an NN which receives fuzzy inputs, processes it and extracts fuzzy outputs as well.

Typical Hybrid Systems

S. No.	Hybrid System	Description
1	Neuro – Fuzzy Hybrid with Multilayer Feed forward Network as the host architecture	Fuzzy back propagation network
2	Neuro – Fuzzy Hybrid with Recurrent Network as the host architecture	Simplified Fuzzy ARTMAP
3	Neuro – Fuzzy Hybrid with single layer Feed forward architecture	Fuzzy Associative Memory
4	Neuro Genetic Hybrid	Genetic algorithm based back propagation network
5	Fuzzy – Genetic Hybrid	Fuzzy logic controlled Genetic algorithm

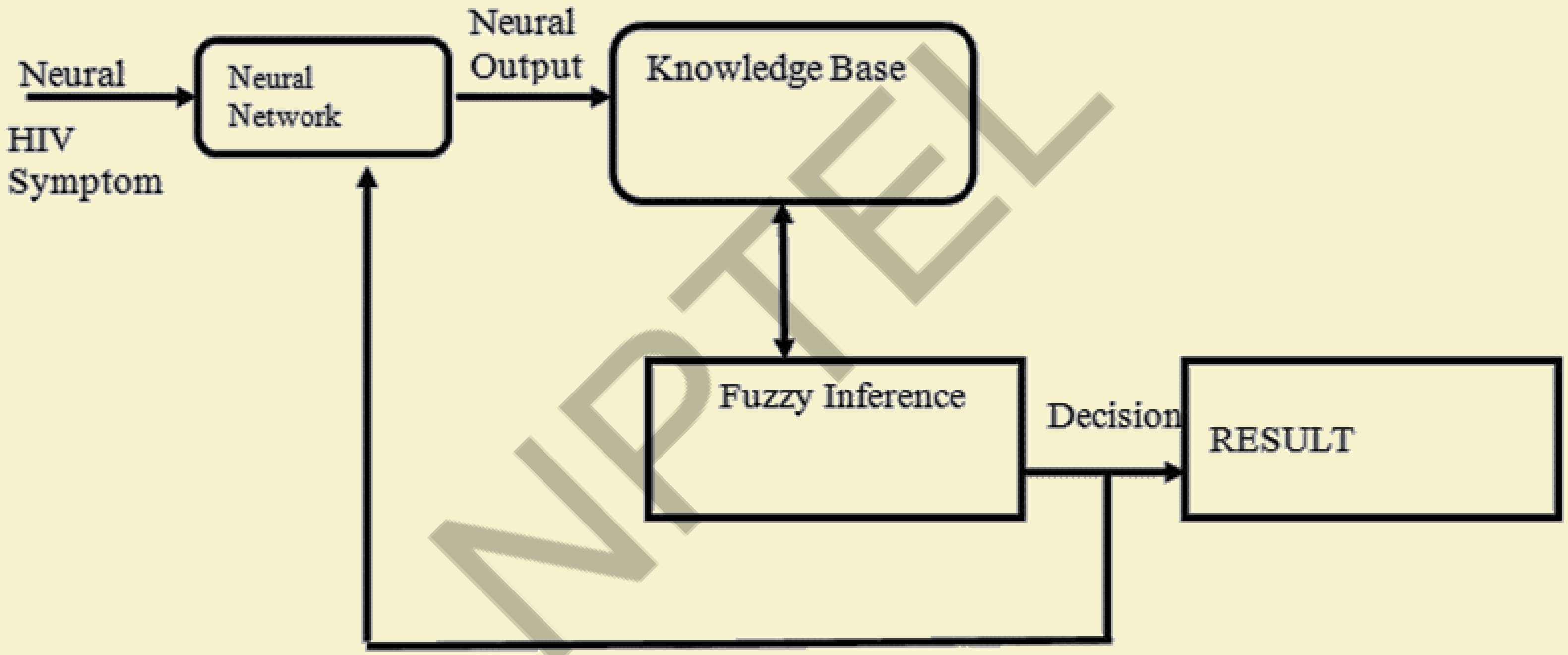
Neuro-Fuzzy Systems

- ✓ Fuzzy logic and neural networks are natural complementary tools in building intelligent systems.

- ✓ While neural networks are low-level computational structures that perform well when dealing with raw data, fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts.

- ✓ However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment. On the other hand, although neural networks can learn, they are opaque to the user.

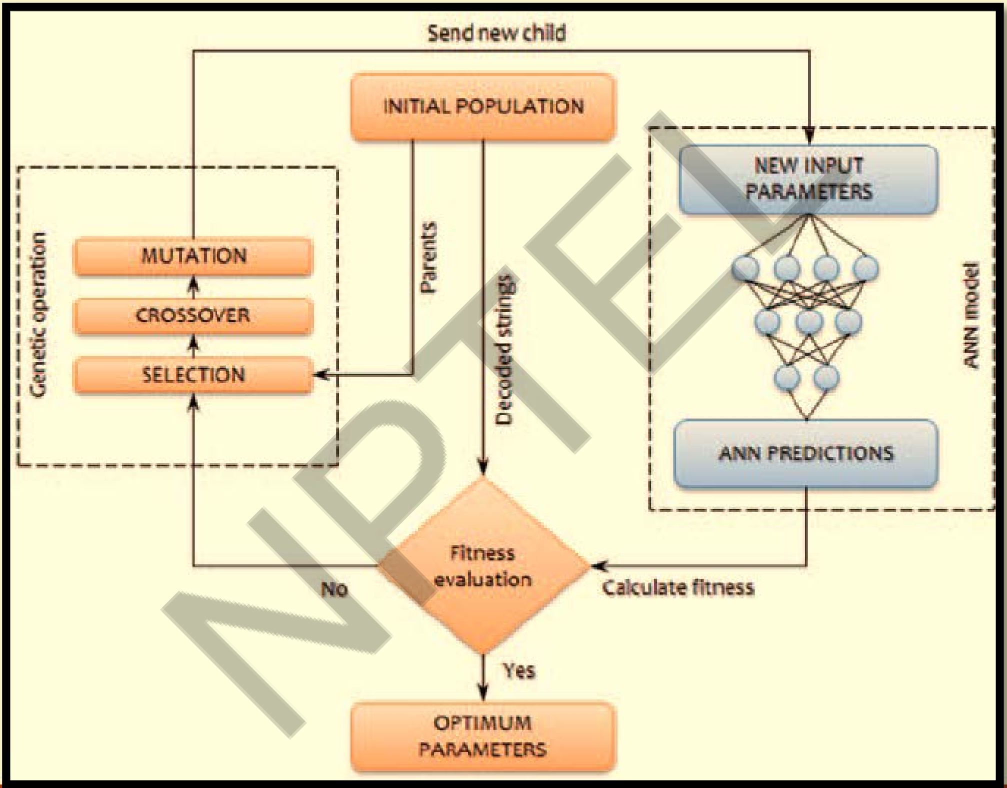
Neuro-Fuzzy Systems Example



Neuro-Genetic Systems

- ✓ In neuro-genetic systems neural network calls a genetic algorithm to optimize its structural parameters
- ✓ GA based algorithms have provided encouraging results especially with regard to face recognition, animal control, and others.
- ✓ GA-NN is also known as GANN have the ability to locate the neighbourhood of the optimal solution quicker than other conventional search strategies.
- ✓ **The drawbacks of GANN algorithms are :** large amount of memory required to handle and manipulate chromosomes for a given network; the question is whether this problem scales as the size of the networks become large.

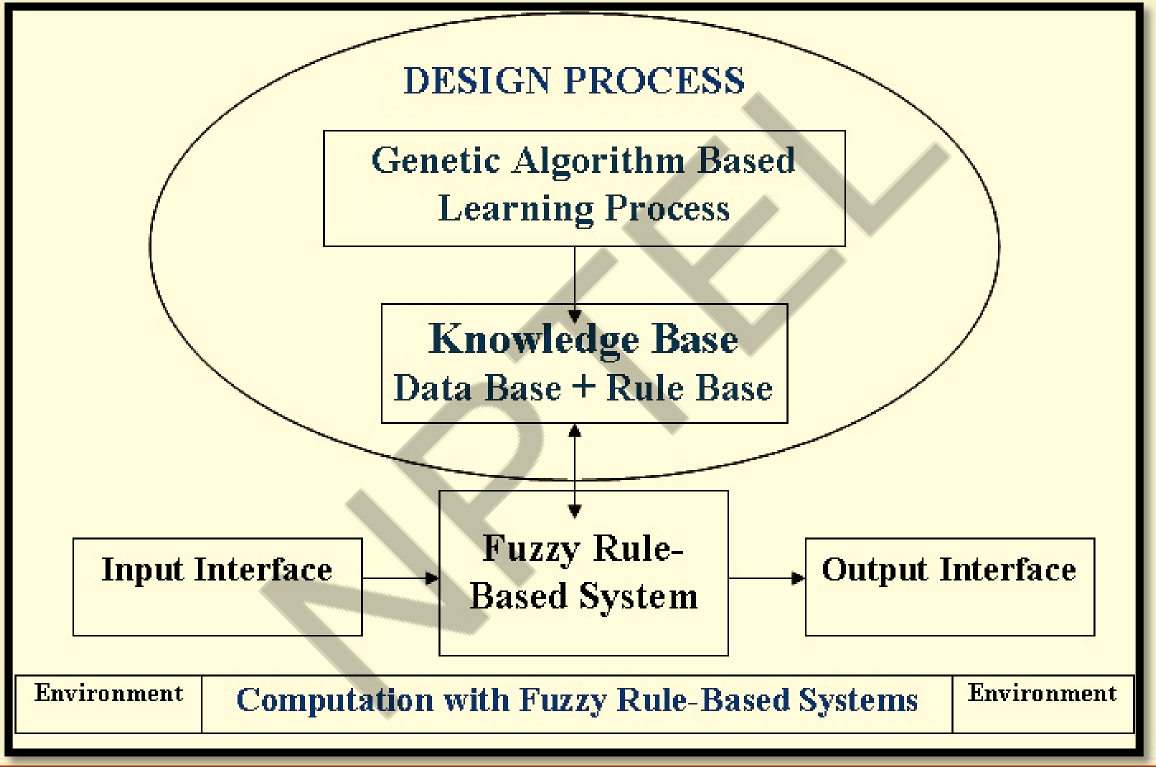
Neuro-Genetic Systems Example



Fuzzy-Genetic Systems

- ✓ The automatic definition of an *Fuzzy Rule Base* can be seen as an optimization or search problem
- ✓ GAs are a well known and widely used global search technique with the ability to explore a large search space for suitable solutions only requiring a performance measure.
- ✓ In addition to their ability to find near optimal solutions in complex search spaces, the generic code structure and independent performance features of GAs make them suitable candidates to incorporate a priori knowledge.
- ✓ In the case of FRBSs, this a priori knowledge may be in the form of linguistic variables, fuzzy membership function parameters, fuzzy rules, number of rules, etc. These capabilities extended the use of GAs in the development of a wide range of approaches for designing FRBSs over the last few years.

Fuzzy-Genetic Systems Example



Thank You!!