Chapter 4

Neuro-Fuzzy Logic Controller

The fusion of ideas from fuzzy control and neural networks had acknowledged a significant role in improving controller performances. Fuzzy logic has proven effective for complex, non-linear and imprecisely defined systems. The common bottleneck in fuzzy logic is the derivation of fuzzy rules and the parameter tuning for the controller. The neural networks have powerful learning abilities, optimization abilities and adaptation. The fuzzy logic and neural networks can be integrated to form a connectionist adaptive network based fuzzy logic controller. This integrated adaptive system modifies the characteristics of the rules, topology of fuzzy sets and/or the structure of control system.

4.1 Fuzzy Logic Controller

Fuzzy Logic control (FLC) has proven effective for complex, non-linear and imprecisely defined processes for which standard model based control techniques are impractical or impossible [95]. Fuzzy Logic, unlike boolean or crisp logic, deals with problems that have vagueness, uncertainty and use membership functions with values varying between 0 and 1. Fuzzy Logic tends to mimic human thinking that is often fuzzy in nature.

In fuzzy logic a particular object has a degree of membership in a given set, which is in the range of 0 to 1. The essence of fuzzy control algorithms is a conditional statement between a fuzzy input variable A and a fuzzy output variable B. This is expressed by a linguistic implication statement such as

\[ \text{IF } A \text{ THEN } B. \]

In general a fuzzy variable is expressed through a fuzzy set, which in turn is defined by a membership function \( \mu \).
4.2 Configuration of FLC

The basic configuration of an FLC developed by Lofti A. Zadeh (1984) [95] is given in Figure 4.1. It comprises of four principal components:

1. A fuzzification interface
2. A knowledge base
3. A decision-making logic and
4. A defuzzification interface.

Fig. 4.1. Basic Configuration of Fuzzy Logic Controller

1. The fuzzification interface involves the following functions.

(a) measures the values of input variable.

(b) performs a scale mapping that transfers the range of values of input variable into corresponding universe of discourse.
(c) performs the function of fuzzification that converts input data into suitable linguistic values.

2. The knowledge base consists of database and a linguistic control rule base.

(a) The database provides necessary definitions, which are used to define linguistic control rules.

(b) The rule base characterized the control goals and control policy of the domain experts by means of a set of linguistic control rules.

3. The decision-making logic is the kernel of an FLC. It has the capability of simulating human decision-making based on fuzzy concepts and of inferring fuzzy control actions employing fuzzy implication and the rules of inference in fuzzy logic.

4. The defuzzification interface performs the following functions.

(a) A scale mapping, that converts the range of values of output variables into corresponding universe of discourse.

(b) Defuzzification, which yields a non-fuzzy control action from an inferred fuzzy control action.

Thus the idea behind the FLC is to fuzzify the controller inputs, and then infer the proper fuzzy control decision based on defined rules. The output is then produced by defuzzifying this inferred control decision.

4.2.1. Fuzzification and membership functions

Fuzzification is a process of transferring the crisp (real) control variables to corresponding fuzzy variables. Selection of the control variables relies on the nature of the system and its desired output. The FLC input and output signals are interpreted into a number of linguistic variables. The number of linguistic variables varies according to the application. Increasing the number of linguistic variables results in a corresponding
increase in the number of rules. Each linguistic variable has its fuzzy membership function. The membership function maps the crisp values into fuzzy variables.

A fuzzy set $A$ in $X$ is defined as

$$A = \{(x, \mu_A(x)) / x \in X\}$$

where $\mu_A(x)$ is called membership function (MF). The membership grade of each element of $X$ is in the range 0-1.

Some of the one-dimensional membership functions commonly used are:

1. Triangular membership functions.
2. Gaussian membership functions.
3. Trapezoidal membership functions.

The general description of fuzzy classes of Triangular MF's is shown in Figure 4.2(a). It is specified by three parameter $\{a, b, c\}$ with $(a < b < c)$ are the x-coordinates of the three corners of the MF.

![Figure 4.2(a). Triangular Membership functions](image)

The class of Gaussian membership functions is shown in Figure 4.2 (b). It is given by

$$\mu_{a_i}(x_i) = \frac{-1}{\sigma_i^2} (c_i - x_i)^2$$

where $c_i$ and $\sigma_i$ are the center and width of the $i^{th}$ fuzzy set $A_i$ respectively.
Figure 4.2(b). Gaussian membership functions

The Trapezoidal membership function is specified by four parameters \(a, b, c, d\). The parameters \(a, b, c, d\) with \(a < b < c < d\) determine the x-coordinates of the four corners of the underlying trapezoidal MF. The membership function description of trapezoidal family is described in Figure 4.2 (c).

![Diagram of trapezoidal membership functions]

Figure 4.2(c). Trapezoidal membership functions

4.2.2. Rules Creation and Inference

In general, fuzzy systems map input fuzzy sets to output sets. Fuzzy rules are relations between input/output fuzzy sets. The modes of deriving fuzzy rules are based either of the following.

- Expert experience and control engineering knowledge.
- Operator’s control actions.
- Learning from the training examples.
The general form of the fuzzy control rules in learning from training examples case is

\[
\text{IF } x \text{ is } A, \text{ AND } y \text{ is } B, \text{ THEN } z = f_i(x, y)
\]

where \( x \), \( y \) and \( z \) are linguistic variables representing the process state variables and the control variable respectively. \( A, B \) are the linguistic values of the linguistic variables, \( f_i(x, y) \) is a function of the process state variables \( x, y \) and the resulting fuzzy inference system (FIS) is called a first order sugeno fuzzy model.

4.2.3. Defuzzification

The function of the inference engine is to calculate the overall value of the control output variable based on the individual contributions of each rule in the rule base. (i.e.) the defuzzification process. There is no systematic procedure for choosing defuzzification. In first-order sugeno fuzzy model each rule has a crisp output and overall output is obtained as weighted average thus avoiding the time consuming process of defuzzification required in a conventional FLC.

4.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are systems that are deliberately constructed to make use of some organizational principles resembling those of human brain [92,94]. The key factors that distinguish Artificial Neural Networks from other computational techniques are

- ANN’s are nonlinear; able to classify patterns and capture complex interactions among the input variables in the system.
- ANN’s are adaptive: they can take data and learn from it. (i.e.) online training.
- ANN’s can generalize: they can correctly process data that broadly resembles the data they were trained originally.
- ANN is a parallel-distributed information processing structure.
4.4 Artificial Neuron model

The model of an artificial neuron is shown in Figure 4.3. In this model [98] the processing elements (neuron) compute the weighted sum of its inputs. Accordingly the output is computed with this weighted input sum is above or below a certain threshold \( \theta_k \). The externally applied bias has the effect of lowering the net input of the activation function.

\[
y_k = f(u_k - \theta_k)
\]

where, \( u_k = \sum w_i x_i \quad i = 1, \ldots, p \)  

Here \( x_1, x_2, \ldots, x_p \) are input signals and \( w_1, w_2, \ldots, w_p \) are interconnection weights of the neuron. \( u_k \) is the linearly combined output.

Some commonly used activation functions are

- **Step function** \( f(x) = \begin{cases} 1 & \text{for } x \geq 0, 0 & \text{otherwise} \end{cases} \)  

- **Ramp function** \( f(x) = \begin{cases} 1 & \text{for } x > 1, x & \text{for } 0 \leq x \leq 1, 0 & \text{for } x < 0 \end{cases} \)  

- **Sigmoid function** \( f(x) = \frac{1}{1 + e^{-ax}} \)

where ‘ \( a \) ’ is the slope parameter of the function.
4.5 Neural Network Connections

ANN’s are weighted directed graphs in which neurons are nodes and directed edges (with weights) are connected between neuron outputs and neuron inputs.

Based on connection patterns ANN’s are grouped as

- Feed-forward networks in which graphs have no loops.
- Recurrent or feedback networks in which loops occurs.

Feed-forward networks are static as they produce only one set of output values for a given input. These networks are memory-less in the sense that their response to an input is independent of the previous network state. In the simplest form of single layer feed-forward networks there is an input layer of source codes that project into an output layer of neurons. The simple layer feed forward networks are shown in Figure 4.4 (a). The interconnection of several layers forms a multilayer feed-forward networks as shown in Figure 4.4(b). The layer between the input and output layer is called a hidden layer and its function is to intervene between the external input and output. The hidden layer had no direct contact with the external environment. The radial basis function network is a special class of multilayer feed-forward networks. The hidden layer in this employs a radial basis function, such as a Gaussian kernel as the activation function.

![Figure 4.4(a) Single layer feed forward network](image-url)
Feedback networks that have closed loops are called recurrent networks. In single layer recurrent network the processing element output is feedback to itself or to other processing element or to both. In the multiplayer recurrent network the neuron output can be directed back to the nodes in the preceding layers.

4.6 Learning in ANN

Learning is not a unique process; there are different learning processes, each suitable to different species. In ANN the concepts of learning processes have been borrowed from the behaviorist’s lab and implemented in electronic circuitry.

Learning is a process by which neural network adapts itself to stimulus and eventually (after making the proper parameter adjustments to itself) it produces a desired response. Learning is also a continuous classification process of input stimuli. During the process of learning the network adjusts its parameters, the synaptic weights in response to and input stimulus so that its actual output response converges to the derived output response. The back propagation algorithm is most widely used for training the neural network.
4.7. Back propagation Learning Algorithm

Based on this algorithm [93, 98], the networks learns a distributed associative map between the input and output layers. What makes this algorithm different than the others is the process by which the weights are calculated during the learning phase of the network. In general, difficulty with multilayer perceptions is calculating the weights of the hidden layers in an efficient way that results in the least (or zero) output error; the more hidden layers there are; the more difficult it becomes. To update the weights, one must calculate an error. At the output layer this error is easily measured; this is the difference between the actual and desired (target) outputs. At the hidden layers however, there is no direct observation of the error, hence some other technique must be used to calculate error, as this is the ultimate goal.

4.7.1 Training with Back propagation Algorithm

During the training session of the network, a pair of patterns is presented \((x_k, d_k)\), where \(x_k\) is the input pattern and \(d_k\) is the target or desired pattern. The \(x_k\) pattern causes output responses at each neuron in each layer and, hence actual output \(O_k\) at the output layer. At the output layer, the difference between the actual and target outputs yields an error signal. This error signal depends on the values of the weights of the neurons in each layer. This error is minimized, and during this process new values for the weights are obtained. The speed and accuracy of the learning process (i.e.), the process of updating the weights also depends on a factor known as the learning rate.

The basis for this weight update algorithm is simply the gradient – descent method as used for simple perceptron with differentiable units. For a given input – output training pair \((x_k', d_k')\) the back – propagation algorithm performs two phases of data flow. First, the input pattern is propagated from the input layer to the output layer and, as a result of this forward flow of data, it produces an actual output \(y_k\). Then the error signals resulting from the difference between \(d_k\) and \(y_k\) are back - propagated from the output layer to the previous layers for them to update their weights.
Let us consider \( m \) PEs (processing elements) in the input layer, \( l \) PEs in the hidden layer and \( n \) PEs in the output layer as shown in Figure 4.5.

Consider an input–output training pair \( (x, d) \) a PE \( q \) in the hidden layer receives a net input of

\[
\text{net } q = \sum v_{qj} x_j \quad j = 1, \ldots, m \tag{4.6}
\]

and produces an output of

\[
z_q = a(\text{net } q) = a(\sum v_{qj} x_j) \tag{4.7}
\]

The net input for a PE \( i \) in the output layer is then

\[
\text{net } i = \sum w_{iq} z_q = \sum w_{iq} a(\sum v_{qj} x_j) \quad q = 1, \ldots, l \tag{4.8}
\]

and it produces an output of

\[
y_i = a(\text{net } i) = a(\sum w_{iq} z_q)
\]

**Figure 4.5. Back propagation Network**
= a(\sum w_q a(\sum v_q x_j)) \tag{4.9}

The above equations indicate the forward propagation of input signals through the layers of neurons. Next, we shall consider the error signals and their back propagation.

\[ E(w) = \frac{1}{2} \sum (d_i - y_i)^2 \quad i = 1, \ldots, n \tag{4.10} \]

Then according to the gradient – decent method the weights in the hidden to output connections are updated by

\[
\Delta w_{eq} = -\eta \frac{\partial E}{\partial w_{eq}}
\]

\[
= -\eta \left[ \frac{\partial E}{\partial y_i} \right] \left[ \frac{\partial y_i}{\partial \text{net } i} \right] \left[ \frac{\partial \text{net } i}{\partial w_{eq}} \right]
\]

\[
= -\eta [d_i - y_i] [a'(\text{net } i)] [z_q]
\]

\[
= \eta \delta_{oq} z_q \tag{4.10a}
\]

where \( \delta_{oi} \) is the error signal and its double subscript indicates the \( i \)th node in the output layer. The error signal is defined by

\[
\delta_{oi} = -\partial E / \partial \text{net } i = -[\partial E / \partial y_i] [\partial y_i / \partial \text{net } i]
\]

For the weight update on the input to hidden connections the chain rule with the gradient descent method is used and the weight update on the link weight connecting PE \( j \) in the input layer to PE \( q \) in the hidden layer is obtained.

\[
\Delta v_{qj} = -\eta \left[ \frac{\partial E}{\partial v_{qj}} \right]
\]

\[
= \eta \delta_{m} x_j
\]

where \( \delta_{mq} \) is the error signal of PE \( q \) in the hidden layer and is defined as

\[
\delta_{mq} = -\partial E / \partial \text{net } q
\]
where $net_q$ is the net input to the hidden PE $q$. The error signal of PE in a hidden layer is different from the error signal of a PE in the output layer. Because of this difference, the above weight update procedure is called the generalized delta-learning rule.

The process of computing the gradient and adjusting the weights is repeated until a minimum error is found. In practice, one develops on algorithm termination criterion so that the algorithm does not continue this iterative process forever.

In summary the error back–propagation algorithm can be outlined as

Step 1: Initialize all weights to small random values.

Step 2: Choose an input-output training pair.

Step 3: Calculate the actual output from each neuron in a layer by propagating the signal forward through the network layer by layer (forward propagation).

Step 4: Compute the error value and error signals for output layer.

Step 5: Propagate the errors backward to update the weights and compute the error signals for the preceding layers.

Step 6: Check whether the whole set of training data has been cycled once, yes – go to step 7; otherwise go to step 2.

Step 7: Check whether the current total error is acceptable; yes- terminate the training process and output the field weights, otherwise initiate a new training epoch by going to step 2.
4.8. Adaptive Neuro-Fuzzy Controllers

The basic concept of Neuro-Fuzzy control models proposed by Jang (1997) [93] is first to use structure-learning algorithms to find appropriate fuzzy logic rules and then use parameter-learning algorithms to fine-tune the membership functions and other parameters. The fuzzy adaptive learning control Network (FALCON) is a feed forward multilayer network, which integrates the basic elements, and functions of a traditional fuzzy logic controller into a connectionist structure thus had distributed learning abilities [91,92,93]. In this connectionist structure, the input and output nodes represent the input states and output control or decision signals respectively, and in hidden layers there are nodes functioning as membership functions and fuzzy logic rules.

The FALCON can be constructed from training examples by neural learning techniques and the connectionist structure can be trained to develop fuzzy logic rules and determine proper input-output membership function. Expert knowledge can also be incorporated into the FALCON. The rule base of connectionist structure contains fuzzy IF-THEN rules of sugeno’s first order type in which the output of each rule is a linear combination of input variables plus a constant term. For a first-order sugeno fuzzy model the common rule set with two fuzzy IF-THEN rules is the following. The final output is the weighted average of each rule’s output.

Rule 1: If \( x \) is \( A_1 \) and \( y \) is \( B_1 \) then \( f_1 = (p_1 \times x + q_1 \times y + r_1) \)

Rule 2: If \( x \) is \( A_2 \) and \( y \) is \( B_2 \) then \( f_2 = (p_2 \times x + q_2 \times y + r_2) \)


The architecture of the Adaptive Neuro-fuzzy Inference System (ANFIS) [91] is shown in Figure 4.6. Consider a first-order sugeno fuzzy model with two input \( x \) and \( y \) and one output with above-mentioned fuzzy IF-THEN rules.
The system has a total of five layers.

**Layer 1:**

Every node $i$ in this layer is an adaptive node performing membership function.

$$O_{i,t} = \mu_{A_i}(x_i) \quad i = 1, \ldots, n$$

(4.11)

where $x_i$ is the input to node $i$.

The membership function can be any appropriate parameterized membership function. Parameters in this layer are referred to as premise parameters.
Layer 2:

Every node in this layer is a fixed node whose output is the product of all the incoming signals. (i.e.) these nodes perform the fuzzy AND operation.

\[ O_{2i} = w_i = \mu_{A_i}(x_i) \ast \mu_{B_i}(y_i) \quad i = 1, \ldots, n \quad (4.12) \]

Each node output represents the firing strength of a rule.

Layer 3:

The nodes of this layer calculate the normalized firing strength of each rule.

\[ O_{3i} = \overline{w_i} = \frac{w_i}{\sum w_i} \quad i = 1, \ldots, n \quad (4.13) \]

\( w_i \) – firing strength of a rule.

Layer 4:

Every node \( i \) in this layer is an adaptive node with a node function.

\[ O_{4i} = \overline{w_i f_i} = \overline{w_i (p_i \ast x + q_i \ast y + r_i)} \quad i = 1, \ldots, n \quad (4.14) \]

where \( \overline{w_i} \) a normalized firing strength from layer 3 and \( \{p_i, q_i, r_i\} \) the parameter set. Parameters in this layer are referred to as consequent parameters.

Layer 5:

The single node in this layer computes the overall output as the summation of all incoming signals.

\[ O_{5i} = \sum \overline{w_i f_i} \quad i = 1, \ldots, n \quad (4.15) \]
where \( O_{5,i} \) denote the output of the \( i^{th} \) node in layer 5. This structure can update membership functions and rule base parameters accordingly to the gradient descent update procedure.

### 4.10 Hybrid Learning Algorithm

The learning algorithm for the connectionist network structure [91] consists of two separate stages of a learning strategy, which combines unsupervised learning and supervised gradient-descent learning procedure. In phase one a self-organized learning scheme is used to locate initial membership functions and to find the presence of fuzzy logic rules. In phase two a supervised learning scheme is used to optimally adjust the parameters of membership functions for desired output. The back-propagation algorithm is used for the supervised learning.

![Flowchart of Hybrid Learning Procedure](image)

To initiate the learning scheme, training data and the desired or guessed coarse of fuzzy partition (i.e., the size of the term set of each input/output linguistic variable) must be provided from the outside world. The hybrid learning procedure is summarized by the flow chart shown in Figure 4.7.
Before this network is trained, an initial structure of the network is first constructed. Then during the learning process, some nodes and links of this initial network are deleted or combined to form the final structure of the network. In its initial form there are \( \prod_i |T(x_i)| \) rule nodes with the inputs of each rule node coming from one possible combination of the terms of input linguistic variables under the constraint that only one term in a term set can be rule node’s input. Here \( |T(x_i)| \) denotes the number of fuzzy partitions of input state linguistic variable \( x_i \). The state space is initially divided into \( |T(x_1)| \times |T(x_2)| \times \ldots \times |T(x_i)| \) linguistically defined nodes, which represent the preconditions of fuzzy rules. Also the links between the rule nodes and the output term nodes are initially fully connected, meaning that the consequents of the rule nodes are not yet decided, only a suitable term in each linguistic variable’s term set will be chosen after the learning process.

4.11 Summary

The basic concepts of the Adaptive Neuro-fuzzy Inference system have been discussed briefly in this chapter. The hybrid-learning algorithm and the procedure of learning from the set of training data have also been presented. The Power System Stabilizer designed based on this architecture is dealt in the following chapter.