2 Fundamentals of Fuzzy Logic Systems

i. A and B are equal iff their membership functions are identical.

ii. A's compliment, written as A' is defined thus $f_{A'} = 1 - f_A$.

iii. A is contained in B (a.k.a. subset, smaller than) iff $f_A \leq f_B$

2.1 Fuzzy Logic Operations

i. $A \cup B = f_A \vee f_B$

ii. $A \cap B = f_A \wedge f_B$

iii. $AB = f_A f_B \subset A \cap B$

iv. $A + B = f_{A+B} = f_A + f_B$ *only meaningful when* < 1

v. $A \oplus B = (A'B')' = A + B - AB$

vi. $|A - B| = f_{|A-B|} = |f_A - f_B|$

i. Boundary conditions, $C(\emptyset) = X, C(X) = \emptyset$

ii. Non-increasing.

iii. Involutive: $C(C(A)) = A$

T-Norm - Generalized Intersection

The T-norm is non-decreasing in each argument, commutative, associative, and satisfies the boundary condition $aT0 = 0$. An S-norm is the opposite.

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x))$$

S-Norm - T-Conorm, Generalized Union

The S-norm is non-decreasing in each argument, commutative, associative, and satisfies the boundary condition $aS0 = a, aS1 = 1$.

Set Inclusion

In the context of fuzzy logic, a set may be partially included in another set. In that case it is convenient to define a grade of inclusion of a fuzzy set A in another fuzzy set B.

$$\mu_{A \subset B}(x) = \begin{cases} 1, & \text{if } \mu_A(x) < \mu_B(x) \\ \mu_A(x) T \mu_B(x), & \text{otherwise} \end{cases} \quad (1)$$

$$\mu_{A \subseteq B}(x) = \begin{cases} 1, & \text{if } \mu_A(x) \leq \mu_B(x) \\ \mu_A(x) T \mu_B(x), & \text{otherwise} \end{cases} \quad (2)$$

2.2 Implication

Consider two fuzzy sets: A in X and B in Y.

Method 1:

$$\mu_{A \to B}(x, y) = min[\mu_A(x), \mu_B(y)] \quad (3)$$

Method 1:

$$\mu_{A \to B}(x, y) = min[1, \{1 - \mu_A(x) + \mu_B(y)\}] \quad (4)$$

2.3 Definitions

Height of a Fuzzy Set The maximum height of its membership function.

$$hgt(A) = \sup \mu_A(x) \quad (5)$$

Support Set Crisp set containing all elements whose membership grade is greater than 0.

$$S = \{x \in X | \mu_A(x) > 0\} \quad (6)$$

α-cut of a Fuzzy Set Crisp set formed by those elements whose membership grade is greater than or equal to the threshold specified by α.

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}, \alpha \in [0, 1] \quad (7)$$

Representation Theorem A fuzzy set can be recomposed or *represented*, as the largest α-cuts of the set for all x.

$$\mu_A(x) = \sup[\alpha \mu_{A_\alpha}(x)] \quad (8)$$

This provides a method for reconstructing a fuzzy set using a crisp set, although sequences of crisp sets are not equal to fuzzy sets because the value of α is itself a membership grade. Also called *resolution identity* because the fuzzy set can be resolved into an infinite set of α-cuts.

Supremum The maximum of a function over a continuous interval.

2.4 Fuzziness and Fuzzy Resolution

Fuzzy Resolution Consider {COLD, MEDIUM, HOT} against {COLD, COOL, MEDIUM, HOT, SCALDING}. The latter has a higher resolution.

Degree of Fuzziness is the distance from where the membership function is 0.5 to 1, or 0.

2.5 Fuzzy Relations

Analytical Representation Any membership function represents a fuzzy relation. If crisp, we can shade in a 2D relation where the function is "true." With a fuzzy relation, we have to draw the relation in 3D.

Cartesian Product of Fuzzy Sets If we consider a fuzzy set A_1 defined in universe X_1 and a second set A_2 from X_2 then $A_1 \times A_2$ is a fuzzy subset of $X_1 \times X_2$. The membership function is below.

$$\mu_{A_1 \times A_2}(x_1, x_2) = min[\mu_{A_1}(x_1), \mu_{A_2}(x_2)], \tag{9}$$
$$\forall (x_1, x_2) \in (X_1 \times X_2) \tag{10}$$

Extension Principle If we have A as a fuzzy set on X then the Extension principle allows us to have function f which maps X to Y and extend A to Y as fuzzy set B.

$$B = f(A) = \frac{\mu_A(x_1)}{y_1} + \frac{\mu_A(x_2)}{y_2} + \text{ ... where} \tag{11}$$
$$y_1 = f(x_1), y_2 = f(x_2)... \tag{12}$$

2.6 Composition and Inference

Projections occur when we reduce dimension. For a fuzzy set over $X \times Y$, we obtain the first projection by projecting the relation onto X. The second projection is obtained by projecting the relation onto Y.

Cylindrical Extension We can extend into another dimension by essentially copying the values into the next dimension, like stretching a circle into a cylinder.

Compositional Rule of Inference A typical fuzzy logic system's knowledge base is composed of a series of if-then rules. When all individual rules are aggregated they result in one fuzzy relation represented by a fuzzy set and a multi-veriable membership function. The membership function of the inference is defined as follows.

Composition (sup-min)

$$\mu_I = \sup_y \{\min[\mu_D, \mu_K]\} \tag{13}$$

Composition (sup-product)

$$\mu_I = \sup\nolimits_y \{\mu_D . \mu_K\} \tag{14}$$

Properties of Composition

Sup-t Composition

$$\mu_{P \circ R}(x, y) = \sup\nolimits_{z \in Z} [\mu_P(x, z) T \mu_R(z, y)] \tag{15}$$

Inf-s Composition

$$\mu_{P \otimes R}(x, y) = \inf\nolimits_{z \in Z} [\mu_P(x, z) S \mu_R(z, y)] \tag{16}$$

Commutativity

$$P \circ R = R \circ P \tag{17}$$
$$P \otimes R = R \otimes P \tag{18}$$

Associativity

$$P \circ (Q \circ R) = P \circ Q) \circ R \tag{19}$$
$$P \otimes (Q \otimes R) = (P \otimes Q) \otimes R \tag{20}$$

Distributivity

$$(P \cup Q) \circ R = (P \circ R) \cup (Q \circ R) \tag{21}$$
$$\tag{22}$$

DeMorgan's Laws

$$\overline{P \circ R} = \overline{P} \otimes \overline{R} \tag{23}$$
$$\overline{P \otimes R} = \overline{P} \circ \overline{R} \tag{24}$$

Inclusion

$$R_1 \subset R_2 \Rightarrow P \circ R_1 \subset P \circ R_2 \tag{25}$$

Extension Principle

$$\mu_{P \circ R}(y) = \sup_{x = f'(y)} \min(\mu_P(x)) \tag{26}$$

3 Fuzzy Logic Control

3.1 Basics of Fuzzy Control

Steps of Fuzzy Logic Control

i. Develop a set of linguistic control rules.

ii. Obtain a set of membership functions.

iii. Obtain rule base R_i using the fuzzy AND operation and the fuzzy implication operation. (These are min operations.)

iv. Combine the relations using the fuzzy conectives ELSE to obatin the overall control surface. (This is a max operation.)

Then:

i. Fuzzify the measured process variables.

ii. Match the fuzzy requirements to the rules.

iii. Defuzzify the control inference obtained.

Composition Using Individual Rules

Defuzzification Centroid Method The centriod of the fuzzy control surface is its centre of gravity.

$$\hat{c} = \frac{\sum_{c_i \in s} c_i \mu_C(c_i)}{\sum_{c_i \in s} \mu_C(c_i)} \tag{27}$$

Mean of Maxima Method This is simply the average location of your max values.

$$\hat{c} = c_{max} \text{ such that } \mu_C(c_{max}) = \max_{c \in S} \mu_C(c) \tag{28}$$

Threshold Method Eliminate everything below a certain α and then repeat with one of the above methods. The new inference set is reduced to:

$$S_\alpha = \{c | \mu_C(c) \geq \alpha\} \tag{29}$$

Comparison of the Defuzzification Methods The centroid method uses the whole surface and is the most complete representation. The mean of maxima is really quick and very dirty, resulting in sharper control. Thresholding makes the centroid more sensitive, less robust, and less dependent on input. It hardly effects the mean of maxima.

Fuzzification Fuzzification the process of turning real input into an abstract fuzzy value such as LOW, MEDIUM, or HIGH, etc.

Singleton Method

$$\mu_F(y) = \mu_{Y_j}(y)\delta(y - y_0) \tag{30}$$

Triangular Function Method

$$\mu_F(y) = \mu_{Y_j}(y_0)\mu_{A_j}(y) \text{ where} \tag{31}$$

$$\mu_{A_j}(y) = 1 - \frac{|y - y_o|}{s_j} \text{ for } |y - y_0| \leq s_j \text{ or 0 otherwise} \tag{32}$$

Gaussian Function Method

$$\mu_F(y) = \mu_{Y_j}(y_0)\mu_{A_j}(y) \text{ where} \tag{33}$$

$$\mu_{A_j}(y) = \exp\left[\frac{y - y_0}{s_j}\right]^2 \tag{34}$$

Discrete Case of Fuzzification

$$F = \frac{\frac{y_{j-1} - y_{j-2}}{y_0 - y_{j-2}}, \frac{y_{j+1} - y_j}{y_{j+1} - y_0}}{y_{j-1}} \tag{35}$$

Fuzzy Control Surface A fuzzy controller is a nonlinear controller.

Extensions of Mamdani Fuzzy Control One version is the Segueno or TSK model. Here the knowledge base has fuzzy rules with crisp functions as inferences, of the form:

$$\text{IF } x \text{ is } A_i, \text{ AND IF } y \text{ is } B_i \text{ THEN } c_i = f(x, y) \tag{36}$$

Then the crisp control inference is determined as a weighted average of the individual rule inferences. See textbook pg. 162.

4 Fundamentals of Artificial Neural Networks

4.1 Features of Artificial Neural Networks

Massive parallelism and computationally intensive learning through examples make ANNs suitable for application in nonlinear functional mapping, pattern recognition, categorization, and data compression.

Topology: the way the nodes and interconnections are arranged within the layers of an ANN.

Feedforward Topology A system of nodes with the input layer, followed by the hidden layers (computational nodes), followed by the output layer. Each layer is connected by a series of unidirectional paths so that the outputs of a given layer are the inputs to the next layer. Occasionally called *open-loop* networks because they don't have feedback. Back Propagation Learning (BPL) is associated with feedforward networks.

Recurrent Topology Networks that allow feedback connections, this allows storage of information in output nodes through dynamic states. Instead of mapping input to output regardless of network state (like FF), recurrent networks maintain a state and are useful for modeling dynamic systems.

Neural Network Activation Functions Neurons are just simple processors which take the weighted sum of their inputs and apply an activation function before delivering output to next neuron.

$$o_k = f(\sum_l w_{ik}x_i - \theta_k) \tag{37}$$

θ_k is the node's threshold value. The following are some sample functions, these functions act like real neurons, the neurons either fire or they don't. See page 230 for the graphs of these functions.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{38}$$

$$\text{sigmoid}'(x) = x(1 - x) \tag{39}$$

$$\text{signum}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{40}$$

$$\text{step}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

Supervised Learning

- External teacher provides network with set of input for which the ideal output is already known
- During training, output is continuously compared with the targets
- A learning rule (like gradient descent) uses the error between the actual output and the target output to adjust the weights to obtain the closest possible match between target output and actual output for all of the examples given.

Unsupervised Learning

- Training data and input patterns are presented to system
- System organizes data into clusters according to predefined properties
- Competition scheme occurs among output nodes to decide a winning node
- The weights in the path between the input and the winning output node are then strengthened (as well as neighbouring nodes at the output layer)
- Neighbourliness Parameter: allows groups of nodes to behave as single entity

Reinforcement Learning

- Mimics human learning (feedback from environment)
- Input is presented to network and propagates to output (like normal)
- Environment gives simple feedback to the network - "good" or "bad"
- If good response, corresponding connections are strengthened and others are weakened
- Reward/Penalty system mimics biological learning system

4.2 Fundamentals of Connectionist Modeling

McCulloch-Pitts Models

- first serious attempt to model computing process of biological neuron (1940s)
- single neuron with no learning but sparked interest in the field
- activation function is the **step function**

$$o = f(\sum_{i=1}^{l} w_i x_i - \theta) \tag{42}$$

The bias θ is usually expressed as another input to the neuron with value = 1 and weight = $-\theta$.

Perceptron

- Rosenblatt @ Cornell University (1960s)
- supervised learning
- designed for pattern classification of linearly separable sets
- input layer - corresponding to sensor or retina
- second layer - feature detector unit with fixed connection weights and thresholds
- output layer - a single node with adjustable connection weights
- activation function is the **step** function
- uses perceptron learning rule
- Criticism 1: cannot deal with nonlinearly separable sets
- Criticism 2: cannot adapt weights to new sets of training data

$$\Delta w_i = \eta[t - f(\sum_i w_i x_i - \theta)]x_i \tag{43}$$

η is the learning rate [0,1]

The steps for training a single layer perceptron are as follows:

i. Initialize weights and thresholds to small random values

ii. Choose input-output pattern from set of training data

iii. Compute actual output $f(\sum_i w_i x_i - \theta)$

iv. Adjust weights according to perceptron learning rule

v. if $\Delta w_i \neq 0$, repeat from Step 2 with different training pattern

Adaline

- ADaptive LInear NEuron by Widrow in 1962
- consists of linear combiner, activation function, and set of trainable weights
- weights adjusted according to Least Mean Square algorithm also known as the Widrow-Hoff learning rule
- adjusts weights by incrementing them every iteration by an amount proportional to gradient of cumulative error of the network $E(w)$
- very easy to implement and has ability for generalization - once it has been trained for a set of patterns it can classify a noisy version of the same set
- cannot separate nonlinearly separable sets

The gradient descent rule basically adjusts weights according to the cumulative error of the network. So for a given pattern we have the LMS rule:

$$\Delta w_i = \eta[t - (\sum_i w_i x_i - \theta)]x_i \tag{44}$$

The steps for training an Adaline network are as follows:

i. Initialize weights and thresholds to small random values

ii. Choose an input-output pattern from set of training data

iii. Compute the output before activation $r = (\sum_{i=1}^{l} w_i x_i - \theta)$

iv. Adjust the weights according to LMS rule

v. if $\Delta w_i \neq 0$, repeat from Step 2 with different training pattern

Madaline

Simply a set of parallel adaline units...not much else in the text

5 Major Classes of Artificial Neural Networks

5.1 Multilayer Perceptron

Topology

- belongs to class of feedforward neural networks (FF NN)
- structure: input layer, hidden layer(s), output layer
- number of hidden layers depend on type of problem being addressed

Backpropogation Learning Algorithm

Step 1: Initialize weights and thresholds to small random values.

Step 2: Choose an input-output pattern from the training input-output data set $(x(k), t(k))$.

Step 3: Propagate the k-th signal forward through the network and compute the output values for all i neurons at every layer (l) using $o_i^l(k) = f(\sum_{p=0}^{n_{l-1}} w_{ip}^l o_p^{l-1})$.

Step 4: Compute the total error value $E = E(k) + E$ and the error signal $\sigma_i^{(L)}$ using the formulae $\delta_i^{(L)} = [t_i - o_i^L][f'(\text{tot}_i^{(L)})]$.

Step 5: Update the weights, and proceeding backward.

Step 6: Repeat the process starting from Step 2 using another exemplar. Once all exemplars have been used, we then reach what is known as one-epoch training.

Step 7: Check if the cumulative error E in the output layer has become less than a predetermined value. If so, we conclude the network has been trained. If not, repeat the whole process for one more epoch.

Momentum

- small values of learning parameter η leads to very slow convergence rate of algorithm
- larger learning parameters have been known to lead to unwanted oscillations in weight space and may even cause divergence of algorithm
- $\Delta w^{(l)}(t+1) = -\eta \frac{\partial E_c(t)}{\partial w^{(l)}} + \gamma \Delta w^{(l)}(t)$

5.2 Radial Basis Function Networks

Topology

- developed for mapping nonlinear behavior of static processes and for function approximation purposes
- basic structure: input layer, single hidden layer with radial activation function, output layer
- uses nonlinear transformations at its hidden layer (typically Gaussian curves), but uses linear transformations between the hidden and output layers
- rationale: input spaces (cast nonlinearly into high-dimensional domains) are more likely to be linearly separable than those cast into low-dimensional ones
- unlike most FF NNs, the connection weights between input layer and neuron units of hidden layer are all equal to unity
- nonlinear transformations at hidden layer have the main characteristic of being symmetrical - attain their maximum at the function center and generate positive values that decrease rapidly with distance from center
- produces radial activation signals that are bounded and localized
- two parameters that characterize each activation function: center and width (normalization parameter)
- learns its parameters during training process, similar to other FF NNs
- optimal performance - hidden layer nodes span training data input space - too sparse or too overlapping functions may cause degradation of network performance
- have undesirably high number of hidden nodes, but dimension of space can be reduced by careful planning of network

Learning Algorithm

Hybrid approach, two stage learning strategy.

Step 1: Train the RBFN layer to get the adaptation of centers and scaling parameters using the unsupervised training.

Step 2: Adapt the weights of the output layer using the supervised training algorithm.

RBF Function:

$$g_i(x) = r_i\left(\frac{||x - v_i||}{\sigma_i}\right) \tag{45}$$

Gaussian Kernel Function:

$$g_i(x) = \exp\left(\frac{-(||x - v_i||)^2}{2\sigma_i^2}\right) \tag{46}$$

Logistic function:

$$g_i(x) = \frac{1}{1 + \exp\left(\frac{(||x - v_i||)^2}{2\sigma_i^2}\right)} \tag{47}$$

Output: (n units in the hidden layer, and r output nodes)

$$o_j(x) = \sum_{i=1}^{n} w_{ij} g_j(x)\ j = 1, ..., r \tag{48}$$

Finally:

$$G = [\{g_{ij}\}] \tag{49}$$

$$D = GW \tag{50}$$

$$W = G^+ D \tag{51}$$

$$G^+ = (G^T G)^{-1} G^T \tag{52}$$

5.3 Kohonen's Self-Organizing Network

Topology

- known as KSON or self-organizing map (SOM)
- belongs to class of unsupervised learning networks
- updates its weighting parameters without the need for a performance feedback from a teacher or network trainer
- Major feature - nodes distribute themselves across input space to recognize groups of similar input vectors, while output nodes compete among themselves to be fired one at a time in response to a particular input vector - process known as competitive learning - winner-take-all strategy
- produces low-dimension representation of input space that preserves ordering of original structure of network when suitably trained
- implies that two input vectors with similar pattern characteristics excite two physically close layer nodes
- nodes of KSON recognize groups of similar input vectors
- generates topographic mapping of input vectors to output layer, which depends primarily on pattern of input vectors and results in dimensionality reduction of input space

Learning Algorithm (Theory)

Step 1: Initialize all weights to small random values. Set a value for the initial learning rate α and a value for the neighborhood N_c.

Step 2: Choose an input pattern x from the input data set.

Step 3: Select the winning unit c (the index of the best matching output unit) such that the performance index I given by the Euclidian distance from x to w_{ij} is minimized:

$$I = |x - w_c| = min_{ij}|x - w_{ij}| \tag{53}$$

Step 4: Update the weights according to the global network updating phase from iteration k to iteration $k+1$ as:

$$w_{ij}(k+1) = w_{ij}(k) + \alpha(k)[x - w_{ij}(k)] \text{ if } (i,j) \in N_c(k) \tag{54}$$

$$w_{ij}(k+1) = w_{ij}(k) \text{ if } (i,j) \notin N_c(k) \tag{55}$$

where $\alpha(k)$ is the adaptive learning rate (strictly positive value smaller than unity) and $N_c(k)$ is the neighborhood of the unit c at iteration k.

Step 5: The learning rate and the neighborhood are decreased at every iteration according to an appropriate scheme.

Step 6: The learning scheme continues until a sufficient number of iterations has been reached or until each output reaches a threshold of sensitivity with respect to a portion of the input space.

Learning Algorithm (Applied)

For the purposes of this course, we can assume a neighborhood of 0 for all SOM networks. Weight vectors are normalized to unit length.

Step 0: Given initial weight matrix and learning rates.

Step 1: For each input vector v_i of the set of inputs V, where v_i is of the form $[x_1, x_2, ...x_n]$, calculate

$$I(v_i) = \sum (x_j - w_{ij})^2 \tag{56}$$

Step 2: Find the lowest $I(v_i)$ value.

Step 3: Adjust weight j and update weight matrix.

$$w_{ij}^{new} = w_{ij}^{old} + \alpha(x_j - w_{ij}^{old}) \tag{57}$$

Step 4: Repeat Steps 2-3 for every input vector $v_i \in V$.

Step 5: Change learning rate at the end of epoch. (Given)

$$\alpha(i+1) = (\text{decay}) \text{ times } \alpha(i) \tag{58}$$

Step 6: Repeat Steps 1-5.

Can change KSONs to be applied differently using parameters:

- neighborhood size
- shape (circular, square, diamond)
- learning rate decaying behavior
- dimensionality of neuron array (1-D, 2-D, or n-D)

5.4 Hopfield Network

Topology

Learning Algorithm

i. **Givens.**
   (a) v - vector of n elements, where n is the number of nodes
   (b) W - $n * n$ matrix of weights

ii. **Enumerate all possible inputs (column 1).** Give labels to the 2^n possible inputs and enumerate the values. If there are 4 inputs, we would have 16 inputs, A = 1 1 1 1, B = 1 1 1 -1, C = 1 1 -1 -1, etc.. Store these in column 1 of table. These represent the initial states of the nodes in the network.

iii. **Initialize nodes with the input.** Initialize network with an input. $o(0) = u$. Ex: $o_1 = u = A = [1111]$.

iv. **Calculate transition l for the input.** Input goes through network; output is same as input. Now, output is fedback to one input (not all). Calculate this new output ($o_{1,2,3,4}(1)$), with only the ith node being affected by the feedback. i = random int, $1 \leq i \leq n$, l = is the transition/iteration number, starting at 0.

$o_i(l+1) = sgn\left(\sum_{j=1}^{n} w_{ij} o_j(l)\right)$

Remember, only the ith bit can change. Do this n times, but change i each time to a new random value between 1 and n, never repeating a previous value. (The book's "random" order is 1, 2, 3, 4.)

In summary, to calculate transition l, you pick a random i, multiply the ith row of W with the previous transition output, $o_{1,2,3,4}(l-1)$ (this notation if really $[o_1(l-1), o_2(l-1), o_3(l-1), o_4(l-1)]$).

v. **Calculate all n transitions for all 2^n possible inputs.** After the n transitions, for a given input, the value of the final output will be "settled" (jumping to lower and lower energies) to either the network's intended output or the complement (negative) of the intended output.

Applications