การตรวจเอกสาร

รายงานผลงานที่มีผู้ทำมาแล้ว

จากการค้นคว้าและศึกษาข้อมูลการหาแบบจำลองของระบบ พบว่า สำหรับระบบที่มีคุณสมบัติเชิงเส้น (linear system) จะมีวิธีการหาแบบจำลองของระบบที่มีประสิทธิภาพและถูกนำมาใช้งานอย่างกว้างขวางมากมายหลายวิธี เช่น least-square estimate, stochastic least-square (Lennart, 1999) เป็นต้น แต่ในความเป็นจริงระบบต่าง ๆ จะมีสมบัติความไม่เชิงเส้น (nonlinear) ซึ่งไม่สามารถใช้วิธีดังที่กล่าวมาข้างต้นมาหาแบบจำลองของระบบได้ จึงมีผู้คิดค้นและทำวิจัยเกี่ยวกับการหาแบบจำลองของระบบที่มีคุณสมบัติไม่เชิงเส้นขึ้นมา ดังเช่น

ในปี ค.ศ. 1990 Kumpati S. Narenda และ Kannan Parthasarathy ได้เสนอเทคนิคสำหรับหาแบบจำลองของระบบพลวัตที่มีคุณสมบัติไม่เชิงเส้น โดยใช้ neural network

ในปี ค.ศ. 1990 S. Reynold Chu, Rahmat Shoureshi และ Manoel Tenorio ได้เสนอวิธีการหาแบบจำลองของระบบ โดยใช้ hopfield neural network ในการทำ least-square estimation นอกจากนี้ยังเสนอวิธีการหาแบบจำลองของระบบโดยใช้ neural network ในการทำ Fourier Analysis เพื่อหา Fourier coefficients ในการหาแบบจำลองของระบบในเทอมของความถี่

ในปี ค.ศ. 1991 J.R. Chen และ P. Mars ได้เสนอบทความเกี่ยวกับความเป็นไปได้ในการใช้ multi layer perceptron neural network (MLP) มาหาแบบจำลองของระบบ โดยได้สรุปว่า คุณสมบัติ universality ของ MLP ไม่ได้เป็นข้อได้เปรียบในการประยุกต์ใช้กับงานการหาแบบจำลองของระบบ

ในปี ค.ศ. 1993 Jyh-Shing Roger Jang ได้เสนอโครงสร้างและกระบวนการการเรียนรู้ของระบบ ANFIS (adaptive neuro-fuzzy inference system) ซึ่งเป็นการนำเอาหลักการของ fuzzy inference system มาปรับให้เป็นโครงสร้างของ adaptive neurel network เพื่อที่จะสามารถใช้กระบวนการเรียนรู้แบบ hybrid learning algorithm ได้ ต่อมาในปี ค.ศ. 1996 ยังได้เสนอวิธีการหาแบบจำลองของระบบพลวัต โดยใช้ neuro-fuzzy model และได้ทำการทดลองหาแบบจำลองของเครื่องเป่าผมโดยใช้ adaptive neuro-fuzzy inference system (ANFIS) และนำผลการทดลองมา

เปรียบเทียบกับการหาแบบจำลองของระบบโดยใช้วิธี ARX และในปีเดียวกันยังได้เสนอวิธีการเลือกว่าจะใช้ตัวแปรใดบ้าง มาเป็นอินพุตที่จะป้อนให้กับ ANFIS

ในปี ค.ศ. 1996 Mizutani E. และ Jyh-Shing Roger Jang ได้เสนอวิธีการประยุกต์ใช้ Levenberg-Marquardt method มาใช้ใน hybrid learning algorithm สำหรับนำไปใช้ในกระบวนการเรียนรู้ของ ANFIS

ในปี ค.ศ. 2000 M.Onder Efe, O. Kaynak ได้เสนอผลการทดลองการหาแบบจำลองของระบบโดยใช้วิธีต่าง ๆ ได้แก่ FNN , RBFNN , RKNN และ ANFIS (with backpropagation learning method) มาเปรียบเทียบกัน โดยทำการทดลองด้วยวิธี simulation กับระบบ robotic manipulator

ในปี ค.ศ. 2002 Jeen-Shing Wang และ C.S. George Lee ได้นำเสนอโครงสร้างและกระบวนการการเรียนรู้ของระบบ SANFIS (self-adaptive neuro-fuzzy inference system) โดยได้มีการเพิ่ม MCA clustering algorithms ซึ่งเป็นกระบวนการสร้าง fuzzy set ของแต่ละอินพุตตามลักษณะของข้อมูลที่จะป้อนให้กับANFIS

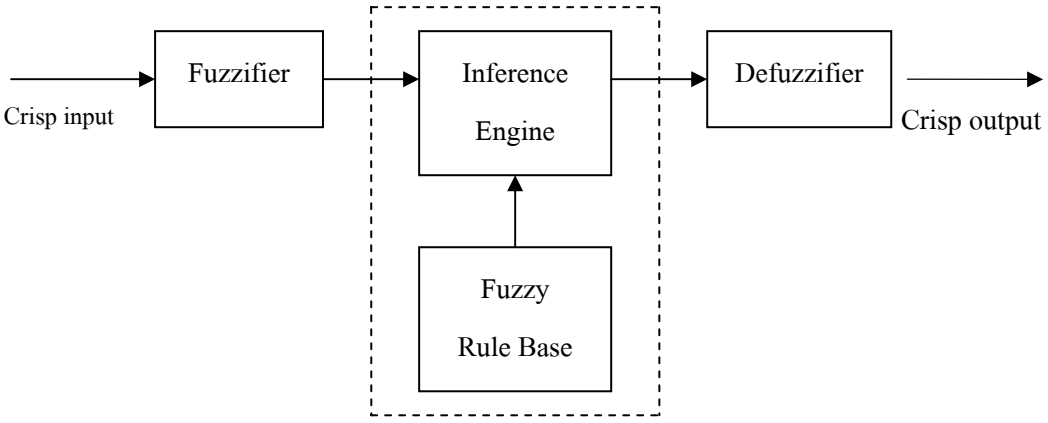
ในปี ค.ศ. 2003 Zhixiang Hou, Quantai Shen และ Heqing Li ได้เสนอบทความ โดยกล่าวถึงข้อด้อยของการหาแบบจำลองของระบบโดยใช้ fuzzy model หรือ neural network model เพียงอย่างเดียว และได้เสนอวิธีการหาแบบจำลองของระบบโดยใช้ adaptive neuro-fuzzy inference system (ANFIS)

Fuzzy Inference System

Jyh-Shing et al. (1997) fuzzy inference system ถูกนำมาใช้เป็นแบบจำลองของระบบใด ๆ โดยใช้หลักการของ fuzzy set theory, fuzzy if-then rule และ fuzzy reasoning และเป็นที่นิยมอย่างแพร่หลายในการนำมาประยุกต์ใช้งานในหลาย ๆ ด้าน เช่น automatic control, data classification, decision analysis, robotics, pattern recognition เป็นต้น

1. <u>โครงสร้างของ Fuzzy Inference System</u>

โครงสร้างของ fuzzy inference system จะประกอบด้วย 3 ส่วนหลัก ๆ ดังภาพที่ 1



<u>ภาพที่ 1</u> โครงสร้างของ Fuzzy Inference System

1.1 Fuzzifier ประกอบด้วยหลาย ๆ fuzzy set ของแต่ละอินพุต โดยจะทำหน้าที่แปลง crisp input ให้เป็น fuzzy input กล่าวคือ แปลงให้เป็นค่าความเป็นสมาชิกของแต่ละเซต

1.2 Fuzzy reasoning unit ประกอบด้วย inference engine, fuzzy rule base และ หลายๆ fuzzy set ของเอาท์พุต โดยในส่วนนี้จะทำหน้าที่ ในการหาความสัมพันธ์ระหว่างค่าความเป็นสมาชิกของอินพุต และค่าความเป็นสมาชิกของเอาท์พุต โดยในการหาความสัมพันธ์จะขึ้นอยู่กับ Fuzzy rule base ที่ถูกกำหนดขึ้นมา

1.3 Defuzzifier ทำหน้าที่ในการแปลง ค่าความเป็นสมาชิกของเอาท์พุต ให้เป็นค่า crisp output

2. <u>ประเภทของ Fuzzy Inference System</u>

Fuzzy inference system ที่ถูกนำมาประยุกต์ใช้งานอย่างกว้างขวางมีอยู่ 3 ประเภท โดยทั้ง 3 ประเภทจะแตกต่างกันในส่วนของ implication ของ if-then rule, ในส่วนของการรวม ผลของแต่ละ if-then rule เข้าด้วยกัน (aggregation) และในส่วนของการทำ defuzzification

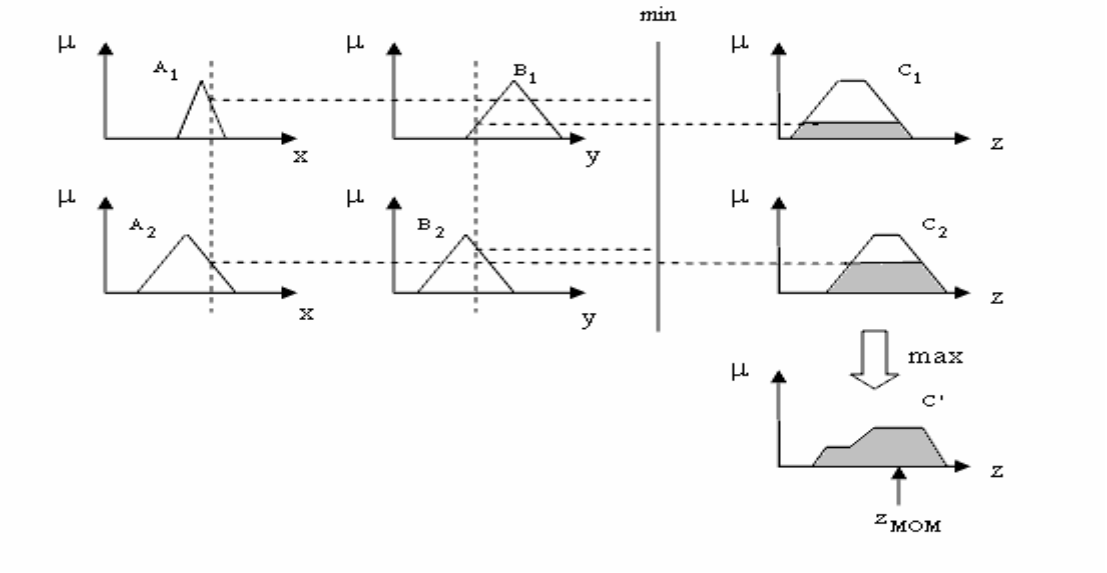
2.1 Mamdani fuzzy inference system เป็น fuzzy inference system ที่เป็นที่นิยมใช้งานอย่างกว้างขวางในระบบควบคุม สมมุติให้ Mamdani fuzzy inference system มี 2 crisp input คือ $x = x_0$ และ $y = y_0$, มี 1 crisp output คือ z และกำหนดให้มีกฎจำนวน n ข้อ แล้วจะสามารถหา fuzzy reasoning ได้ดังทฤษฎีบทต่อไปนี้

ทฤษฎีบทที่ 1 กำหนดให้ fuzzy if-then rule ข้อที่ i คือ $R_i = A_i \times B_i \rightarrow C_i$ สำหรับการทำ implication ใน if-then rule แบบ minimum และ การรวมกฎแบบ maximum (max-min composition) แล้วจะได้ฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎ (aggregated output membership function) ดังนี้

$$\mu_{c'}(z) = \max_{i=1}^{n}\{\min(\omega_i, \mu_c(z))\} \tag{1}$$

โดยที่ $\omega_i = \min\{\mu_{Ai}(x_0), \mu_{Bi}(y_0)\}$

ตัวอย่างการทำ Mamdani fuzzy reasoning ของระบบที่มี 2 อินพุท-2 กฎ แบบ max-min composition สามารถแสดงได้ดังภาพที่ 2



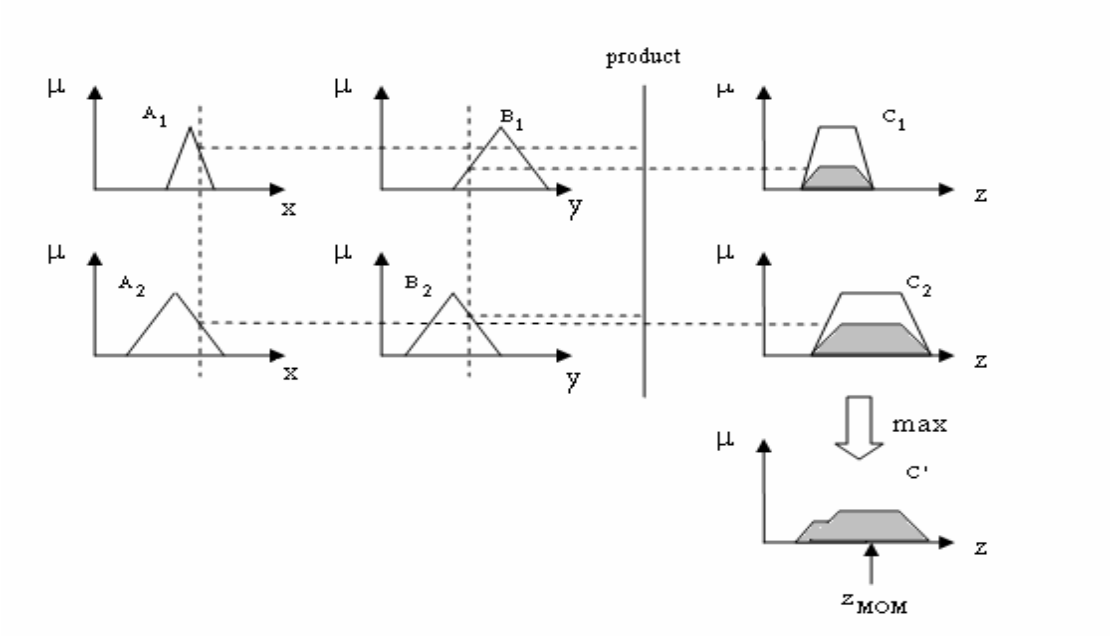
ภาพที่ 2 โครงสร้างของ Mamdani Fuzzy Inference System (max-min composition)

ทฤษฎีบทที่ 2 กำหนดให้ fuzzy if-then rule ข้อที่ i คือ $R_i = A_i \times B_i \rightarrow C_i$ สำหรับการทำ implication ใน if-then rule แบบ product และ การรวมกฎแบบ maximum (max-product composition) แล้วจะได้ฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎดังนี้

$$\mu_{c'}(z) = \max_{i=1}^{n}\{\omega_i \cdot \mu_c(z)\} \tag{2}$$

โดยที่ $\omega_i = \min\{\mu_{Ai}(x_0), \mu_{Bi}(y_0)\}$

ตัวอย่างการทำ Mamdani fuzzy reasoning ของระบบที่มี 2 อินพุท-2 กฎ แบบ max-product composition สามารถแสดงได้ดังภาพที่ 3



ภาพที่ 3 โครงสร้างของ Mamdani Fuzzy Inference System (max-product composition)

หลังจากได้ฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎ, $\mu_{C'}(z)$ จาก fuzzy reasoning unit แล้ว ขั้นตอนต่อมาคือการทำ defuzzification กล่าวคือ จะต้องทำการเปลี่ยน fuzzy set ให้เป็น crisp output ซึ่งในขั้นตอน defuzzification จะทำได้หลายวิธี ดังเช่น

2.1.1 Centroid of area (z_{COA})

$$z_{COA} = \frac{\int_Z \mu_{C'}(z) \cdot z \; dz}{\int_Z \mu_{C'}(z) \; dz} \tag{3}$$

2.1.2 Bisector of area (z_{BOA})

$$\int_{\alpha}^{z_{BOA}} \mu_{C'}(z)\, dz \; = \; \int_{z_{BOA}}^{\beta} \mu_{C'}(z)\, dz \tag{4}$$

โดยที่ $\alpha = \min\{z / z \in Z\}$ *and* $\beta = \max\{z / z \in Z\}$

2.1.3 Mean of maximum (z_{MOM})

$$z_{MOM} = \frac{\int_{z'} z \; dz}{\int_{z'} dz} \tag{5}$$

โดยที่ $z' = \{z / \mu_{C'}(z) = \mu^*\}$ และ μ^* คือค่าที่มากที่สุดของฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎ

2.1.4 Smallest of maximum (z_{SOM}) : (z_{SOM}) คือ ค่า z ที่น้อยที่สุดที่ทำให้ $\mu_{C'}(z)$ เท่ากับ ค่ามากที่สุดของฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎ

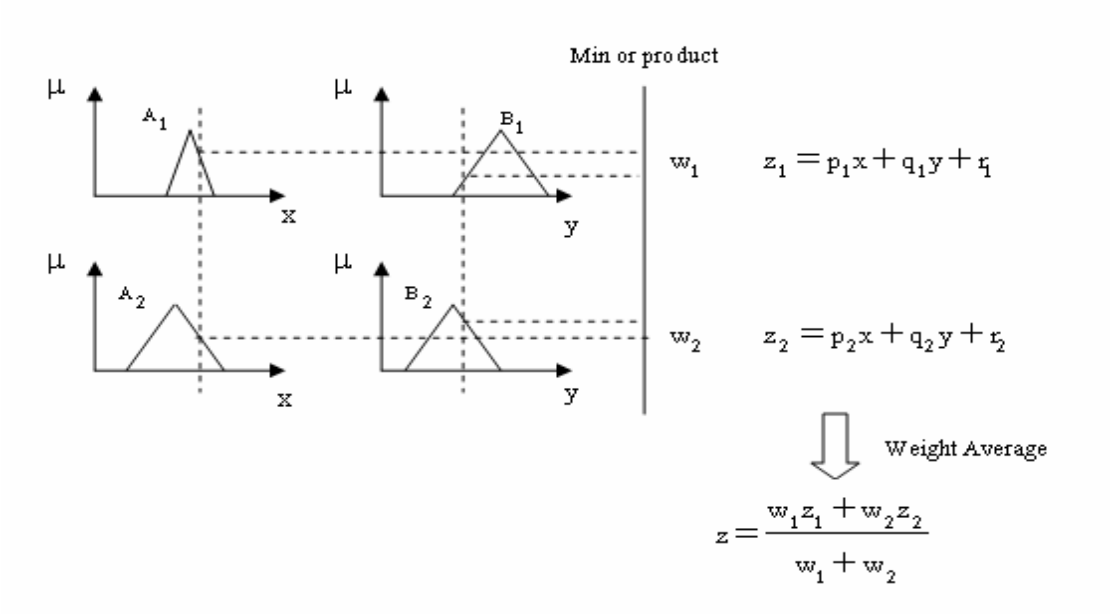
2.1.5 Largest of maximum (z_{LOM}) : (z_{LOM}) คือ ค่า z ที่มากที่สุดที่ทำให้ $\mu_{C'}(z)$ เท่ากับ ค่ามากที่สุดของฟังก์ชันค่าความเป็นสมาชิกจากการรวมกฎ

2.2 Sugeno fuzzy inference system ถูกสร้างขึ้นมาเพื่อพยายามที่จะพัฒนา fuzzy inference system โดยการสร้าง fuzzy rule จาก input-output data set โดยทั่วไป fuzzy rule ของ Sugeno fuzzy inference system จะมีรูปแบบ

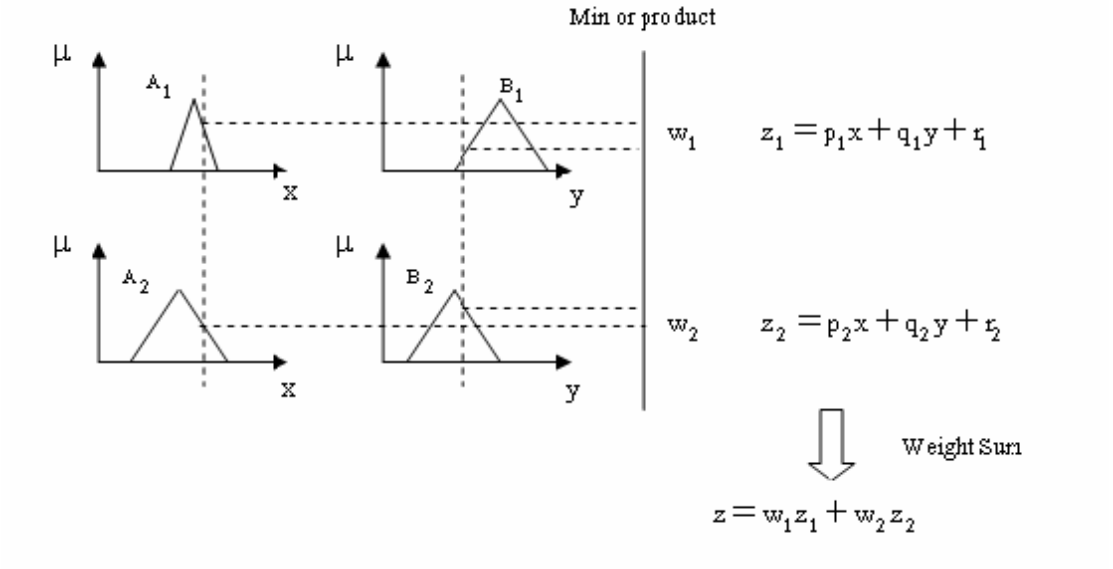
ถ้า x เป็น A และ y เป็น B แล้ว $z = f(x,y)$

โดยที่ A และ B เป็น fuzzy set ของอินพุต และ $z = f(x,y)$ เป็น crisp function ในส่วนของ implication ของ fuzzy if-then rule ซึ่งโดยทั่วไปจะกำหนดให้เป็นสมการโพลิโนเมียลของตัวแปรอินพุต x และ y ในส่วนของการรวมกฎและการ defuzzification จะทำได้ 2 วิธี คือ weight

average และ weight sum ภาพที่ 4 และ 5 แสดง Sugeno fuzzy reasoning สำหรับ 2 อินพุต- 2 กฎ โดยการรวมกฎและการ defuzzification เป็นแบบ weight average และแบบ weight sum ตามลำดับ



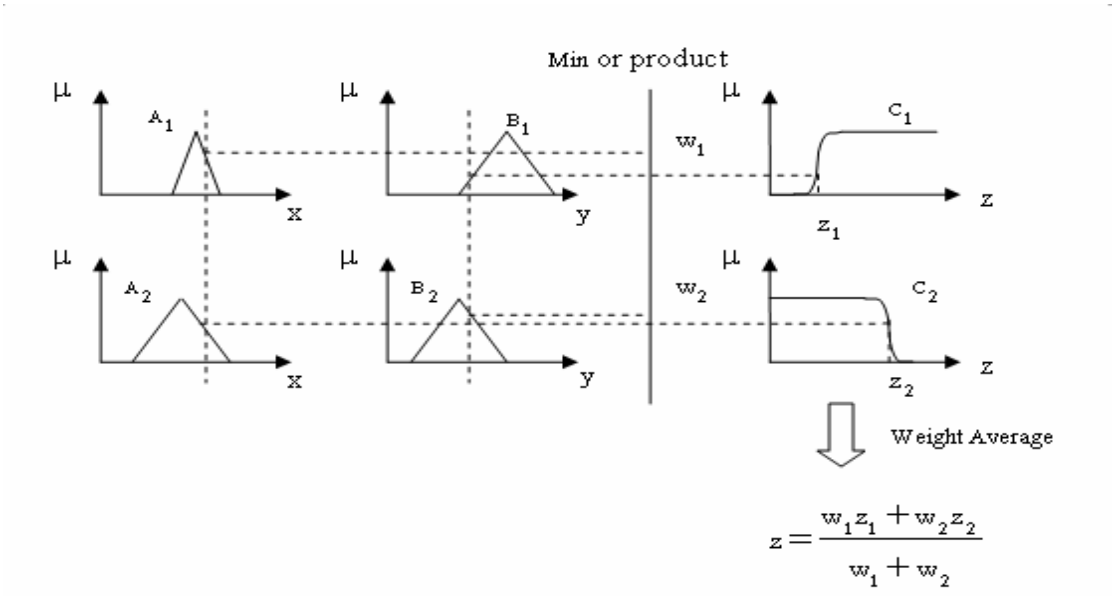
ภาพที่ 4 โครงสร้างของ Sugeno Fuzzy Inference System (weight average)



ภาพที่ 5 โครงสร้างของ Sugeno Fuzzy Inference System (weight sum)

2.3 Tsugamoto fuzzy inference system ในส่วน implication ของ fuzzy if-then rule จะอยู่ในรูปแบบ fuzzy set แบบ monotonical membership function ดังแสดงในภาพที่ 6 แล้วจะได้ crisp

output ของกฎแต่ละข้อ จากนั้นก็นำมารวมกฎและทำ defuzzification แบบ weight average ภาพที่ 6 แสดง Tsugamoto fuzzy reasoning สำหรับ 2 อินพุต- 2 กฎ และการรวมกฎและdefuzzification แบบ weight average



ภาพที่ 6 โครงสร้างของ Tsugamoto Fuzzy Inference System

Regression and Optimization

ก่อนที่เราจะกล่าวถึงส่วนหลักของการหาแบบจำลองของระบบโดยใช้ adaptive neuro-fuzzy model ในหัวข้อนี้จะกล่าวถึงกระบวนการทางคณิตศาสตร์ในการหาแบบจำลองทางคณิตศาสตร์เพื่อนำไปใช้ในการอธิบายคุณสมบัติของระบบให้ใกล้เคียงกับระบบจริง ๆ มากที่สุด

1. Least-Square Estimator

โดยทั่วไปเราจะใช้ least-square estimator ในการแก้ปัญหาการหาค่าตัวแปรของแบบจำลองทางคณิตศาสตร์ที่ระบบมีคุณสมบัติเชิงเส้น (Steven , 2002) โดยมีรูปแบบสมการแบบจำลองทางคณิตศาสตร์ดังนี้

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \cdots + \theta_n f_n(u) \qquad (6)$$

โดย $u = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}^T$ เป็นอินพุตเวกเตอร์ของแบบจำลองทางคณิตศาสตร์ และ $f_1, f_2, \ldots, f_n$ เป็นฟังก์ชันของอินพุต u ซึ่งสามารถหาค่าได้ และ $\theta_1, \theta_2, \ldots, \theta_n$ เป็นตัวแปรที่ไม่ทราบค่าซึ่งจะเราจะต้องทำการประมาณค่าของมัน

โดยทั่วไปกระบวนการประมาณค่าตัวแปร θ สามารถทำได้โดยทดลองป้อนค่าอินพุตให้กับระบบ เพื่อให้ได้ชุดข้อมูลสำหรับการเรียนรู้ (training data set) ซึ่งประกอบด้วยชุดข้อมูลจำนวน m คู่ดังนี้ $\{(u_i, y_i), i = 1, \cdots, m\}$ โดยชุดข้อมูลเหล่านี้จะเป็นคู่ข้อมูลของอินพุตและเอาท์พุตของระบบที่เราจะทำการหาแบบจำลองทางคณิตศาสตร์ จากชุดข้อมูลที่ได้เราสามารถนำมาเขียนในรูปแบบระบบสมการเชิงเส้นจำนวน m สมการได้ดังนี้

$$\begin{aligned} f_1(u_1)\theta_1 + f_2(u_1)\theta_2 + \cdots + f_n(u_1)\theta_n &= y_1 \\ f_1(u_2)\theta_1 + f_2(u_2)\theta_2 + \cdots + f_n(u_2)\theta_n &= y_2 \\ &\vdots \\ f_1(u_m)\theta_1 + f_2(u_m)\theta_2 + \cdots + f_n(u_m)\theta_n &= y_m \end{aligned} \tag{7}$$

จากระบบสมการเชิงเส้นดังกล่าว สามารถนำมาเขียนในรูปสมการเมตริกซ์ได้ดังนี้

$$A\theta = y \tag{8}$$

โดย A เป็นเมตริกซ์ขนาด m x n เรียกว่า design matrix

$$A = \begin{bmatrix} f_1(u_1) & f_2(u_1) & \cdots & f_n(u_1) \\ f_1(u_2) & f_2(u_2) & \cdots & f_n(u_2) \\ \vdots & & & \vdots \\ f_1(u_m) & f_2(u_m) & \cdots & f_n(u_m) \end{bmatrix} \tag{9}$$

$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_n \end{bmatrix}^T$ เป็นเวกเตอร์ขนาด n x 1 ของตัวแปรที่จะทำการประมาณค่า และ $y = [y_1 \quad y_2 \quad \cdots \quad y_m]^T$ เป็นเวกเตอร์ขนาด m x 1 ของเอาท์พุต

ในกรณีที่ระบบสมการเชิงเส้น มีจำนวนสมการเท่ากับจำนวนตัวแปร (m = n) และพบว่า A เป็น nonsingular matrix (เมตริกซ์ไม่เอกฐาน) เราพบว่าสามารถหาค่าตัวแปร θ ได้จากสมการ

$$\theta = A^{-1}y \tag{10}$$

แต่โดยทั่วไปแล้ว ระบบสมการเชิงเส้นจะมีจำนวนสมการมากกว่าจำนวนตัวแปร (m > n) ดังนั้น คำตอบของระบบสมการเชิงเส้นไม่สามารถหาคำตอบได้ทุกครั้งไป ซึ่งเกิดได้จากการที่ข้อมูลที่ได้จากการทดลองอาจมีสัญญาณรบกวนรวมอยู่ด้วย หรืออาจเกิดจากแบบจำลองทางคณิตศาสตร์ที่เราสร้างขึ้นมาไม่สามารถอธิบายระบบได้ทั้งหมด ดังนั้นจากสมการ (8) เราจะทำการเพิ่มส่วนที่เกิดจากความผิดพลาดเข้าไปด้วย โดยมีสมการใหม่ดังนี้

$$A\theta + e = y \tag{11}$$

โดย e เรียกว่า เวกเตอร์ความผิดพลาด ดังนั้นแทนที่เราจะทำการหาค่าคำตอบของระบบสมการเชิงเส้น เราจะทำการหาค่าคำตอบ $\theta = \hat{\theta}$ ที่ทำให้ผลรวมของค่าความผิดพลาดกำลังสองมีค่าน้อยที่สุด ซึ่งอธิบายโดยสมการดังนี้

$$\begin{aligned} E(\theta) &= e^T e \\ &= (y - A\theta)^T (y - A\theta) \\ &= (y^T - \theta^T A^T)(y - A\theta) \\ &= \theta^T A^T A\theta - y^T A\theta - \theta^T A^T y + y^T y \\ &= \theta^T A^T A\theta - 2y^T A\theta + y^T y \end{aligned} \tag{12}$$

หาค่า least-square estimator $\theta = \hat{\theta}$ ที่ทำให้ $E(\theta)$ มีค่าน้อยที่สุดได้โดยพิจารณา

$$\left.\frac{\partial E(\theta)}{\partial \theta}\right|_{\theta=\hat{\theta}} = 0 = 2A^T A\hat{\theta} - 2A^T y$$

$$A^T A\hat{\theta} = A^T y \tag{13}$$

ถ้าพบว่า $A^T A$ เป็นเมตริกซ์ไม่เอกฐาน (nonsingular) แล้วจะได้ว่าสามารถหาค่า $\hat{\theta}$ ได้เพียงคำตอบเดียวจากสมการ

$$\hat{\theta} = (A^T A)^{-1} A^T y \tag{14}$$

แต่ถ้าพบว่า $A^T A$ เป็นเมตริกซ์เอกฐาน (singular matrix) จะพบว่า $\hat{\theta}$ จะมีได้หลายคำตอบ โดยเราจะทำการแก้ปัญหาโดยใช้ QR-decomposition นอกจากนี้สำหรับกรณีที่ $A^T A$ เป็นเมตริกซ์ที่ใกล้กับเมตริกซ์เอกฐานจะทำให้การหาค่า $(A^T A)^{-1}$ นั้นให้ผลลัพธ์ไม่ดี เราก็สามารถใช้ QR-decomposition ในการแก้ปัญหาได้เช่นกัน

ทฤษฎีบทที่ 3 ถ้า A เป็นเมตริกซ์ขนาด m x n โดยมี rank เท่ากับ k แล้วจะพบว่าสามารถเขียนเมตริกซ์ A ในรูป normalized QR-decomposition $A = QR$ โดยที่ Q เป็นเมตริกซ์ขนาด m x k และแต่ละคอลัมน์จะเป็น orthonormal ที่ span column space ของ A (กล่าวคือ $Q^T Q = I$) และ R เป็นเมตริกซ์สามเหลี่ยมบน (upper triangular) ที่มีขนาด k x n และมีค่าrank เท่ากับ k

จากสมการที่ (13) เราสามารถเขียนใหม่ได้ว่า

$$\begin{aligned} A^T A\hat{\theta} &= A^T y \\ (R^T Q^T)(QR)\hat{\theta} &= R^T Q^T y \\ R^T Q^T QR\hat{\theta} &= R^T Q^T y \\ R^T R\hat{\theta} &= R^T Q^T y \end{aligned} \tag{15}$$

โดยจะพบว่า R และ R^T มีค่า rank เท่ากันคือเท่ากับ k ซึ่งสามารถหาเมตริกซ์ L ซึ่งเป็นอินเวิร์สทางซ้ายของ R^T ได้ กล่าวคือ $LR^T = I_K$ แล้วเอา L คูณทั้งสองข้างของสมการ จะได้ว่า

$$\begin{aligned} LR^T R\hat{\theta} &= LR^T Q^T y \\ I_K R\hat{\theta} &= I_K Q^T y \\ R\hat{\theta} &= Q^T y \end{aligned} \tag{16}$$

ดังนั้นสรุปได้ว่าสามารถคำตอบของระบบสมการ $A\hat{\theta} = y$ ด้วยการแก้ปัญหากำลังสองน้อยสุดโดยการหาคำตอบของสมการที่ (16)

2. Derivative-Based Optimization

ในหัวข้อนี้จะกล่าวถึงการหาค่า $\theta = \hat{\theta}$ ที่ทำให้ฟังก์ชัน $E(\theta)$ มีค่าน้อยที่สุดซึ่งโดยทั่วไปแล้วฟังก์ชัน E จะอยู่ในรูปแบบของฟังก์ชันไม่เชิงเส้นที่ขึ้นอยู่กับตัวแปร θ ซึ่งสามารถปรับค่าได้ (Martin et al., 1996)

2.1 Steepest Descent Algorithms จะมีหลักการคือจะทำการค่อยๆปรับค่าตัวแปร θ ในแต่ละครั้ง โดยจะมีสมการปรับดังนี้

$$\theta_{k+1} = \theta_k + \alpha_k P_k \triangleq \theta_k + \Delta\theta_k \tag{17}$$

โดย k หมายถึงจำนวนครั้งของการปรับค่าที่เวลาปัจจุบัน เวกเตอร์ P เรียกว่า search direction และค่าสเกลาร์ α เรียกว่า learning rate โดยในแต่ละครั้งของการปรับค่า θ จะมีเงื่อนไขให้ค่าฟังก์ชัน $E(\theta)$ ของการปรับครั้งต่อไปต้องมีค่าน้อยกว่าค่าในปัจจุบัน สามารถเขียนในรูปอสมการดังต่อไปนี้

$$E(\theta_{k+1}) < E(\theta_k) \tag{18}$$

ดังนั้นสิ่งที่จำเป็นต้องหาคือ α_k และ P_k เพื่อนำไปใช้ในสมการที่ (17) พิจารณา first-order Taylor series expansion ของฟังก์ชัน $E(\theta)$ ดังสมการ

$$\begin{aligned} E(\theta_{k+1}) &= E(\theta_k + \Delta\theta_k) \\ &\approx E(\theta_k) + g_k^T \Delta\theta_k \end{aligned} \tag{19}$$

โดย $g_k = \nabla E(\theta)\big|_{\theta=\theta_k} = \left(\frac{\partial E(\theta)}{\partial \theta}\right)^T \Big|_{\theta=\theta_k}$ เรียกว่า gradient matrix และเพื่อให้เป็นไปตามเงื่อนไขของอสมการที่ (18) จะได้ว่า

$$g_k^T \Delta\theta_k = g_k^T \alpha_k P_k < 0 \tag{20}$$

เราจะทำการเลือกค่า α_k ให้มีค่าน้อยๆ แต่ให้มากกว่า 0 ดังนั้นจะได้ว่า

$$g_k^T P_k < 0 \tag{21}$$

โดยเวกเตอร์ P_k ใด ๆ ที่เป็นไปตามอสมการที่ (21) จะเรียกว่า descent direction ซึ่งโดยทั่วไปแล้วจะกำหนดให้มีทิศทางตรงกันข้ามกับ gradient เวกเตอร์ กล่าวคือ

$$P_k = -g_k \tag{22}$$

สรุปได้ว่า การปรับค่าตัวแปรตามหลักการของ steepest-descent algorithm มีสมการการปรับค่าดังนี้

$$\theta_{k+1} = \theta_k - \alpha_k g_k \tag{23}$$

โดย $g_k = \nabla E(\theta)\big|_{\theta=\theta_k} = (\frac{\partial E(\theta)}{\partial \theta})^T\Big|_{\theta=\theta_k}$

2.2 Newton‘s Method จะอาศัยหลักการการปรับค่าตัวแปร θ เหมือนวิธี steepest-descent algorithm แต่จะแตกต่างกันในกระบวนการหาค่า $\Delta\theta_k$ โดยวิธี steepest-descent algorithm จะใช้ first-order Taylor series expansion ในการประมาณค่าฟังก์ชัน $E(\theta_{k+1})$ เพื่อนำไปใช้หาค่า $\Delta\theta_k$ แต่วิธี Newton‘s method จะประมาณฟังก์ชัน $E(\theta_{k+1})$ โดยใช้ second-order Taylor series expansion ซึ่งมีสมการดังนี้

$$E(\theta_{k+1}) = E(\theta_k) + g_k^T \Delta\theta_k + \frac{1}{2}\Delta\theta_k^T H_k \Delta\theta_k \tag{24}$$

โดย $g_k = \nabla E(\theta)\big|_{\theta=\theta_k} = (\frac{\partial E(\theta)}{\partial \theta})^T\Big|_{\theta=\theta_k}$ เรียกว่า gradient matrix และ

$H_k = \nabla^2 E(\theta)\big|_{\theta=\theta_k} = \frac{\partial^2 E(\theta)}{\partial \theta^2}\Big|_{\theta=\theta_k}$ เรียกว่า Hessian matrix

สามารถหาค่า $\hat{\theta}$ ที่ทำให้ $E(\hat{\theta})$ มีค่าน้อยที่สุด โดยการหาอนุพันธ์ของสมการที่ (24) เทียบกับตัวแปร $\Delta\theta_k$ แล้วให้ค่าเท่ากับศูนย์ จะได้ว่า

$$g_k + H_k \Delta\theta_k = 0 \quad \rightarrow \quad \Delta\theta_k = -H_k^{-1} g_k \tag{25}$$

และจะได้สมการการปรับค่าตัวแปร ดังนี้

$$\theta_{k+1} = \theta_k - H_k^{-1} g_k \tag{26}$$

โดย $\Delta\theta_k = -H_k^{-1} g_k$ เรียกว่า Newton step และทิศทางของมันจะเรียกว่า Newton direction

ถ้า Hessian matrix H เป็น positive definite matrix และฟังก์ชัน $E(\theta)$ อยู่ในรูปแบบ quadratic แล้ว จะพบว่าวิธี Newton's Method จะให้ค่าฟังก์ชัน $E(\theta)$ ที่ให้ค่าน้อยที่สุด ด้วยการปรับค่าตัวแปร θ เพียงครั้งเดียว แต่ถ้าพบว่า Hessian matrix H ไม่ได้เป็น positive definite matrix จะพบว่า Newton direction อาจจะชี้ไปในทิศทางที่ให้ค่าฟังก์ชัน $E(\theta)$ มากที่สุด ดังนั้นสรุปได้ว่าวิธี Newton's method ไม่สามารถมั่นใจได้ว่าจะสามารถหาค่า θ ที่ทำให้ค่าฟังก์ชัน $E(\theta)$ มีค่าน้อยที่สุดได้ ทั้งนี้ทั้งนั้นขึ้นอยู่กับฟังก์ชันและค่าตัวแปรเริ่มต้น

2.3 Levenberg-Marquardt Method จะมีหลักการพื้นฐานมาจาก Newton's method โดยจะพิจารณาในกรณีที่ Hessian matrix H ไม่สามารถหาอินเวิร์สเมตริกซ์ได้ ดังนั้นวิธี Levenberg-Marquardt method จะทำการแก้ไขด้วยการเพิ่มเมตริกซ์ $P = \mu I$ ให้กับเมตริกซ์ H (I เป็นเมตริกซ์เอกลักษณ์ หรือ identity matrix) โดยเราจะพบว่าเมตริกซ์ $(H + \mu I)$ สามารถหาอินเวิร์สได้ นอกจากนี้เราจะทำให้เมตริกซ์ $(H + \mu I)$ เป็น positive definite matrix เพื่อที่จะมั่นใจได้ว่าสามารถหาค่า θ ที่ทำให้ค่าฟังก์ชัน $E(\theta)$ มีค่าน้อยที่สุดได้ โดยสมมุติให้ Hessian matrix H มีค่าเจาะจง(eigenvalues) เป็น $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ และจะได้ว่าเมตริกซ์ $(H + \mu I)$ มีค่าเจาะจงเป็น $\{\lambda_1 + \mu, \lambda_2 + \mu, \cdots, \lambda_n + \mu\}$ ดังนั้นเราจะทำการเลือกค่า μ ที่ทำให้ทุก ๆ ค่าเจาะจงมีค่ามากกว่าศูนย์ และจะได้ว่า เมตริกซ์ $(H + \mu I)$ เป็น positive definite matrix สรุปได้ว่า Levenberg-Marquardt method มีสมการการปรับค่าตัวแปรดังนี้

$$\theta_{k+1} = \theta_k - (H_k + \mu_k I)^{-1} g_k \tag{27}$$

โดย $g_k = \nabla E(\theta)\big|_{\theta=\theta_k} = \left(\frac{\partial E(\theta)}{\partial \theta}\right)^T \Big|_{\theta=\theta_k}$ และ $H_k = \nabla^2 E(\theta)\big|_{\theta=\theta_k} = \left.\frac{\partial^2 E(\theta)}{\partial \theta^2}\right|_{\theta=\theta_k}$

นอกจากนี้ในทางปฏิบัติ เราจะเริ่มต้นด้วยการเลือกค่า μ_k มีค่าน้อย ๆ ถ้าพบว่าในครั้งนี้ค่าฟังก์ชัน $E(\theta)$ ไม่น้อยว่าครั้งที่ผ่านมา ให้เราทำซ้ำขั้นนี้อีกครั้งโดยการเพิ่มค่า μ_k (คูณด้วย 10) ซึ่งค่าฟังก์ชัน $E(\theta)$ ควรจะลดลงแล้วเนื่องจากเมื่อเพิ่มค่า μ_k มันจะเข้าใกล้วิธี steepest descent algorithm ซึ่งมั่นใจได้ว่ามันจะลู่เข้าสู่ค่า $E(\theta)$ ที่น้อยที่สุด และเมื่อพบว่าในขั้นตอนใดให้ค่าฟังก์ชัน $E(\theta)$ ลดลงจากครั้งที่แล้ว เราจะทำการลดค่า μ_k (หารด้วย10)ในขั้นตอนต่อไป เนื่องจากเมื่อลดค่า μ_k มันจะเข้าใกล้วิธี Newton's method ซึ่งจะทำให้ลู่เข้าหาค่า $E(\theta)$ ที่น้อยที่สุดเร็วขึ้น จะเห็นได้ว่าวิธี Levenberg-Marquardt method เป็นการรวมเอาข้อดีระหว่างความเร็วของ Newton's method และความมั่นใจได้ว่าลู่เข้าหาค่า $E(\theta)$ ที่น้อยที่สุดแน่นอนของวิธี steepest descent algorithm

Adaptive Neural Network

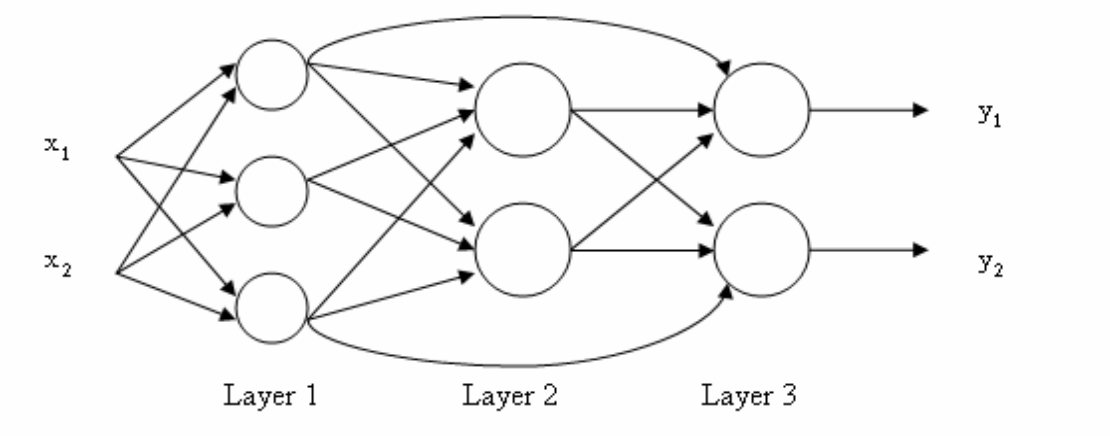
Adaptive neural network เป็นโครงสร้างแบบโครงข่ายชนิดหนึ่งที่มีโครงสร้างและกระบวนการการทำงานจำลองมาจากระบบประสาทของมนุษย์ (Martin *et al.*, 1996)

1. โครงสร้างของ Adaptive Neural Network

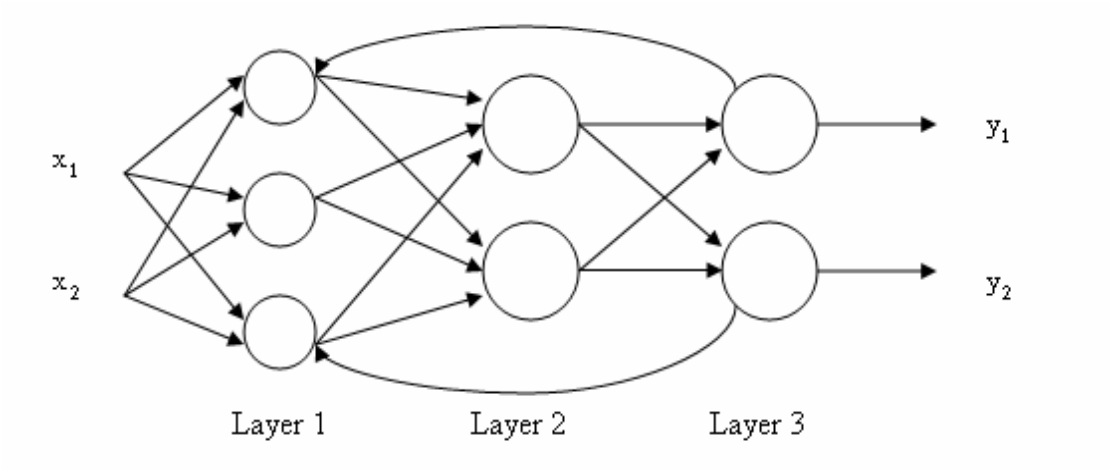
Adaptive neural network เป็นโครงสร้างแบบโครงข่าย ที่ประกอบด้วยนิวรอนหลาย ๆ นิวรอนที่เชื่อมต่อกัน โดยแต่ละนิวรอนจะแทนด้วยกระบวนการการทำงานใด ๆ และเส้นเชื่อมต่อระหว่างนิวรอนจะแทนความสัมพันธ์ระหว่างนิวรอนที่เชื่อมต่อกัน

โครงสร้างภายในของแต่ละนิวรอน จะประกอบด้วยสัญญาณอินพุตจำนวน 1 หรือมากกว่า 1 สัญญาณ, สัญญาณเอาท์พุตเพียง 1 สัญญาณ และ ฟังก์ชันสำหรับสร้างสัญญาณเอาท์พุตจากสัญญาณอินพุต หรือที่เรียกว่า node function โดย node function อาจเป็นได้ทั้งแบบฟังก์ชันเชิงเส้นหรือแบบฟังก์ชันไม่เชิงเส้น ขึ้นอยู่กับโครงสร้างของโครงข่าย และความต้องการที่จะให้ได้สัญญาณเอาท์พุตตามที่กำหนด โดยกระบวนการภายในของแต่ละนิวรอน สัญญาณอินพุตที่เข้ามาจะต้องเข้ามาถึงพร้อมกันและยังคงอยู่จนกระทั่งมีสัญญาณเอาท์พุตออกไป นอกจากนี้ใน adaptive neural network แต่ละนิวรอนอาจจะมี node function ที่แตกต่างกันก็ได้

Adaptive neural network สามารถจำแนกออกเป็นประเภทตามลักษณะการเชื่อมต่อของแต่ละนิวรอน ได้ 2 ประเภท คือ feedforward และ recurrent โดยภาพที่ 7 แสดงโครงสร้าง feedforward adaptive neural network จะเห็นได้ว่าสัญญาณเอาท์พุตของแต่ละนิวรอนจะถูกส่งผ่านจากด้านอินพุต (ทางซ้าย) ไปทางด้านเอาท์พุต (ทางขวา) เท่านั้น ภาพที่ 8 แสดงโครงสร้าง recurrent adaptive neural network จะเห็นได้ว่าสัญญาณเอาท์พุตของแต่ละนิวรอน ไม่จำเป็นจะต้องถูกส่งผ่านจากด้านอินพุต (ทางซ้าย) ไปด้านเอาท์พุต (ทางขวา) เท่านั้น

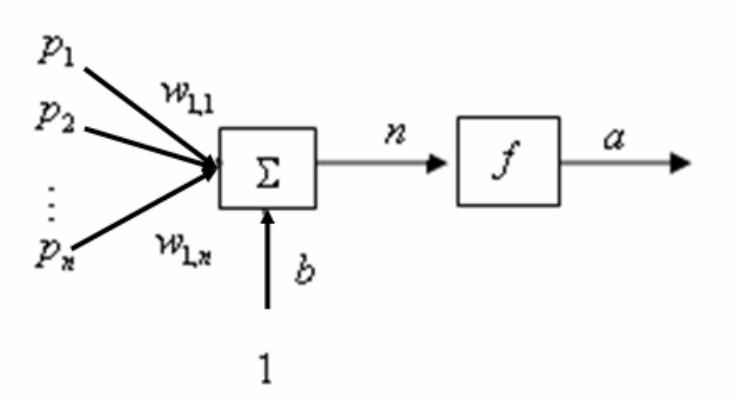


ภาพที่ 7 โครงสร้าง Feedforward Adaptive Neural Network



ภาพที่ 8 โครงสร้าง Recurrent Adaptive Neural Network

ในงานวิทยานิพนธ์นี้จะกล่าวเน้นเฉพาะ feedforward adaptive neural network พิจารณาโครงสร้างภายนิวรอนที่ประกอบด้วยอินพุตจำนวน n ตัว ดังภาพที่ 9



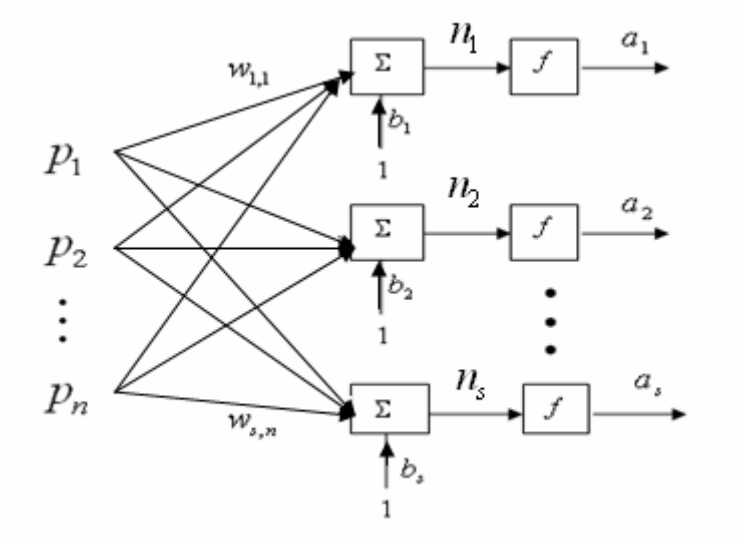
<u>ภาพที่ 9</u> โครงสร้างภายนิวรอนที่ประกอบด้วยอินพุตจำนวน n ตัว

จากภาพที่ 9 จะเห็นได้ว่าอินพุตแต่ละตัว $p_1, p_2, \ldots, p_n$ จะถูกคูณด้วยค่าน้ำหนัก $w_{1,1}, w_{1,2}, \ldots, w_{1,n}$ ตามลำดับ ก่อนที่เข้านิวรอน โดยเอาท์พุตของนิวรอนจะเขียนในรูปสมการได้ดังนี้

$$a = f(Wp + b) \tag{28}$$

โดย $W = [w_{1,1} \quad w_{1,2} \quad \cdots \quad w_{1,n}]$ เรียกว่า weight matrix, $p = [p_1 \quad p_1 \quad \cdots \quad p_1]^T$ เป็นเวกเตอร์ของอินพุตที่เข้านิวรอน และ b เรียกว่า bias

สำหรับในบางระบบที่มีจำนวนอินพุตหลาย ๆ อินพุต นิวรอนเพียงนิวรอนเดียวอาจจะไม่เพียงพอ ดังนั้นเราอาจใช้หลาย ๆ นิวรอนทำงานขนานกันไป โดยเรียกนิวรอนที่ขนานกันไปว่าเลเยอร์ ตัวอย่างโครงสร้างของระบบ adaptive neural network ที่มี 1 เลเยอร์ โดยในเลเยอร์ประกอบด้วย s นิวรอน แสดงดังภาพที่ 10



ภาพที่ 10 โครงสร้างของระบบ Adaptive Neural Network ที่มี 1 เลเยอร์ s นิวรอน

สามารถเขียนเอาท์พุตของเลเยอร์ด้วยสมการดังนี้

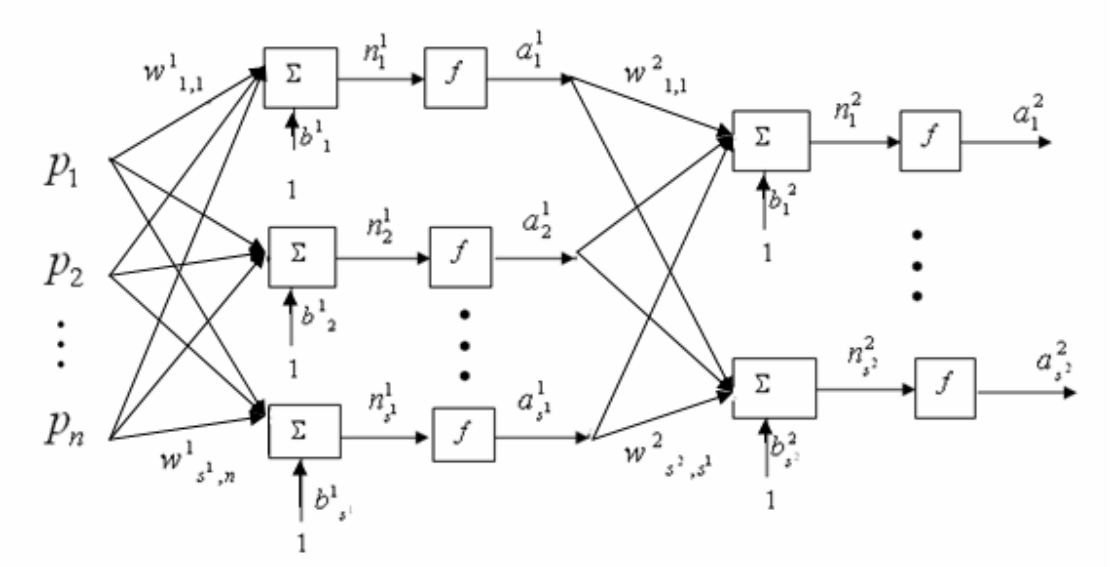
$$a = f(Wp + b) \tag{29}$$

โดย W เป็นเมตริกซ์ของค่าน้ำหนัก โดยมีสมาชิก $w_{i,j}$ เป็นค่าน้ำหนักของอินพุตตัวที่ j ที่ป้อนเข้านิวรอนที่ i ของเลเยอร์นี้

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{s,1} & w_{s,2} & \cdots & w_{s,n} \end{bmatrix} \tag{30}$$

$b = \begin{bmatrix} b_1 & b_2 & \cdots & b_s \end{bmatrix}^T$ เรียกเวกเตอร์ bias และ $a = \begin{bmatrix} a_1 & a_2 & \cdots & a_s \end{bmatrix}^T$ เป็นเวกเตอร์เอาท์พุตของเลเยอร์

นอกจากนี้โครงสร้างของ adaptive neural network อาจจะประกอบด้วยหลาย ๆ เลเยอร์ ยกตัวอย่างเช่น adaptive neural network ที่ประกอบด้วย 2 เลเยอร์ โดยในเลเยอร์ที่หนึ่งประกอบด้วย s^1 นิวรอน และเลเยอร์ที่สองประกอบด้วย s^2 นิวรอน มีโครงสร้างดังภาพที่ 11



ภาพที่ 11 โครงสร้างของระบบ Adaptive Neural Network R อินพุต $s^1 - s^2$ นิวรอน

สามารถเขียนเอาท์พุตของระบบ adaptive neural network R อินพุต $s^1 - s^2$ นิวรอน ได้ดังสมการ

$$a^2 = f(W^2 \cdot f(W^1 p + b^1) + b^2) \tag{31}$$

โดย W^1 เป็นเมตริกซ์ขนาด $s^1 \times n$ ซึ่งเป็นเมตริกซ์ของค่าน้ำหนักของเลเยอร์ที่ 1, W^2 เป็นเมตริกซ์ขนาด $s^2 \times s^1$ ซึ่งเป็นเมตริกซ์ของค่าน้ำหนักของเลเยอร์ที่ 2, b^1 เป็นเวกเตอร์ขนาด $s^1 \times 1$ ซึ่งเป็นเวกเตอร์ bias ของเลเยอร์ที่ 1 และ b^2 เป็นเวกเตอร์ขนาด $s^2 \times 1$ ซึ่งเป็นเวกเตอร์ bias ของเลเยอร์ที่ 2 นอกจากนี้โดยทั่วไปในเลเยอร์สุดท้ายของ adaptive neural network จะเรียกว่า เอาท์พุตเลเยอร์

2. Adaptive Neural Network Learning Algorithm

ในงานวิทยานิพนธ์นี้จะกล่าวถึงเฉพาะกระบวนการการเรียนรู้ของ feedforward adaptive neural network ซึ่งโดยทั่วไปแล้วกระบวนการการทำงานของ feedforward adaptive neural network เป็นการหาความสัมพันธ์ระหว่างข้อมูลอินพุตและเอาท์พุตของมัน ซึ่งความสัมพันธ์นี้อาจจะเป็นไปได้ทั้งในแบบเชิงเส้น หรือแบบไม่เชิงเส้น ซึ่งขึ้นอยู่กับโครงสร้างของโครงข่าย และฟังก์ชันภายในของแต่ละนิวรอน ซึ่งจุดประสงค์หลักของ adaptive neural network คือให้มันทำ

การจำลองระบบใดระบบหนึ่งเพื่อให้ได้ผลตอบสนองเหมือนกับระบบนั้นมากที่สุด ดังนั้นใน adaptive neural network จะต้องมีการกำหนดว่า ในทุก ๆ นิวรอน หรือบางนิวรอนจะต้องมีการถูกปรับเปลี่ยนค่าตัวแปรภายในเพื่อให้ได้เอาท์พุตตามที่ต้องการ โดยกระบวนการการปรับค่าตัวแปรภายในเรียกว่ากระบวนการเรียนรู้หรือ adaptive neural network learning algorithm

กระบวนการการเรียนรู้ของ adaptive neural network จะใช้ชุดของข้อมูลอินพุตและเอาท์พุตที่มีความสัมพันธ์กัน มาผ่านกระบวนการต่าง ๆ เพื่อทำการปรับค่าตัวแปรภายในของ adaptive neural network โดยชุดข้อมูลอินพุตและเอาท์พุตนั้นจะเรียกว่า training data set โดยทั่วไปคุณสมบัติของ adaptive neural network ที่ผ่านกระบวนการเรียนรู้มาแล้ว จะถูกวัดจากค่าความแตกต่างระหว่างข้อมูลเอาท์พุตที่ต้องการ กับข้อมูลเอาท์พุตที่ได้จาก adaptive neural network นั้น ภายใต้เงื่อนไขของการป้อนข้อมูลอินพุตชุดเดียวกัน

2.1 Backpropagation with Steepest-Descent Learning Algorithm สำหรับ adaptive neural network กระบวนการเรียนรู้ที่ง่ายที่สุดคือ steepest descent method หรือ gradient method กล่าวคือจะใช้ steepest descent มาประยุกต์ใช้ในการปรับค่าตัวแปรต่าง ๆ ในโครงสร้างของ adaptive neural network โดยจะต้องทำการหา gradient vector ซึ่งมีทิศทางตรงกันข้ามกับทิศทางของสัญญาณเอาท์พุตที่ออกมาจากแต่ละนิวรอน โดยจะเริ่มหาจากเอาท์พุตเลเยอร์แล้วย้อนกลับจากเลเยอร์สู่เลเยอร์จนกระทั่งถึงอินพุตเลเยอร์ จึงเรียกกระบวนการหา gradient vector ว่า backpropagation

สำหรับ adaptive neural network ที่มีเพียง 1 เลเยอร์และ node function ในแต่ละนิวรอนเป็นฟังก์ชันเชิงเส้น จะพบว่าฟังก์ชันค่าความผิดพลาดระหว่างเอาท์พุตที่ได้กับเอาท์พุตที่เราต้องการ เป็นฟังก์ชันเชิงเส้นของตัวแปรค่าน้ำหนักภายใน adaptive neural network นอกจากนี้ค่าอนุพันธ์ของฟังก์ชันค่าความผิดพลาดเทียบกับตัวแปรค่าน้ำหนักสามารถหาได้โดยง่าย แต่สำหรับ adaptive neural network ที่มีหลาย ๆ เลเยอร์จะพบว่าความสัมพันธ์ระหว่างฟังก์ชันค่าความผิดพลาดกับตัวแปรค่าน้ำหนักจะค่อนข้างสลับซับซ้อน อีกทั้งในการหาอนุพันธ์ของฟังก์ชันค่าความผิดพลาดต้องอาศัยกฎลูกโซ่ในทางแคลคูลัสมาช่วยคำนวณ

พิจารณา adaptive neural network ที่มีหลาย ๆ เลเยอร์ จะพบว่า เอาท์พุตของเลเยอร์ใด ๆ จะเป็นอินพุตของเลเยอร์ถัดไป โดยสามารถเขียนสมการอธิบายความสัมพันธ์ได้ดังนี้

$$a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1}) \quad ; m = 0,1,\cdots,M-1. \tag{32}$$

โดย M เป็นจำนวนเลเยอร์ใน adaptive neural network และกำหนดให้ $a^0 = p$ เป็นอินพุตของ adaptive neural network สำหรับกระบวนการการเรียนรู้โดยวิธี backpropagation จะกำหนดให้มีชุดข้อมูลที่ใช้สำหรับกระบวนการเรียนรู้ดังนี้

$$\{p_1, t_1\}, \{p_2, t_2\}, \cdots, \{p_Q, t_Q\} \tag{33}$$

โดย p_q เป็นอินพุตที่จะทำการป้อนให้กับ adaptive neural network และ t_q เป็นเอาท์พุตที่ต้องการโดยสัมพันธ์กับอินพุตที่ป้อนเข้าไป โดยแต่ละอินพุตถูกป้อนเข้าไปใน adaptive neural network แล้วเอาท์พุตที่ได้ออกมาจะถูกนำไปเปรียบเทียบกับเอาท์พุตที่ต้องการ ดังนั้นหลักการของกระบวนการเรียนรู้วิธี backpropagation คือจะทำการปรับค่าตัวแปรภายใน adaptive neural network เพื่อที่จะทำให้ค่าเฉลี่ยค่าความผิดพลาดกำลังสองมีค่าน้อยที่สุด ซึ่งมีสมการดังนี้

$$F(x) = E[e^2] = E[(t-a)^2] \tag{34}$$

โดย x เป็นเวกเตอร์ที่บรรจุตัวแปรค่าน้ำหนัก และตัวแปรค่า bias ของ adaptive neural network แต่ถ้า adaptive neural network มีหลาย ๆ เอาท์พุต สามารถเขียนสมการใหม่ได้ว่า

$$F(x) = E[e^T e] = E[(t-a)^T (t-a)] \tag{35}$$

และจาก Least-square estimator algorithm จะทำการประมาณค่าฟังก์ชันใหม่ โดยจะพิจารณาในแต่ละครั้งที่ k ของการเรียนรู้ จะได้ว่า

$$\hat{F}(x) = e^T(k)e(k) = (t(k)-a(k))^T(t(k)-a(k)) \tag{36}$$

จาก steepest-descent algorithm จะมีสมการการปรับค่าดังสมการที่ (23) ดังนั้นสมการการปรับค่าตัวแปรน้ำหนักและค่า bias ของ adaptive neural network มีดังนี้

$$\begin{aligned} w^m_{i,j}(k+1) &= w^m_{i,j}(k) - \alpha \frac{\partial \hat{F}}{\partial w^m_{i,j}} \\ b^m_i(k+1) &= b^m_i(k) - \alpha \frac{\partial \hat{F}}{\partial b^m_i} \end{aligned} \tag{37}$$

จะเห็นได้ว่า สิ่งที่เราต้องการคือหาอนุพันธ์ของฟังก์ชัน F เทียบกับตัวแปรค่าน้ำหนัก w โดยใช้กฎลูกโซ่ จะได้ว่า

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}$$
$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m} \tag{38}$$

พบว่าในเทอมที่สอง สามารถหาอนุพันธ์ได้ง่ายเนื่องจาก n เป็นฟังก์ชันที่ขึ้นอยู่กับ w และ b อยู่แล้ว ซึ่งสามารถเขียนในรูปสมการของแต่ละนิวรอนแต่ละเลเยอร์ได้ดังนี้

$$n_i^m = \sum_{j=1}^{s^{m-1}} w_{i,j}^m \cdot a_j^{m-1} + b_i^m \tag{39}$$

ดังนั้นจะได้ว่า

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \quad , \quad \frac{\partial n_i^m}{\partial b_i^m} = 1 \tag{40}$$

พิจารณาในเทอมแรกของสมการที่ (38) โดยกำหนดตัวแปรใหม่เป็น $s_i^m = \frac{\partial \hat{F}}{\partial n_i^m}$ เรียกว่าค่า sensitivity ของ $\hat{F}$ ต่อการเปลี่ยนแปลงของตัวแปร n ของนิวรอนที่ i ในเลเยอร์ที่ m และเขียนเป็นเวกเตอร์รวมของค่า sensitivity ของลเยอร์ที่ m ได้ดังนี้

$$s^m = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{s^m}^m} \end{bmatrix} \tag{41}$$

ในการหาค่า sensitivity s^m เราจะต้องใช้กระบวนการของ backpropagation เนื่องจากค่า sensitivity ของเลเยอร์ที่ m จะถูกคำนวนมาจากค่า sensitivity ของเลเยอร์ที่ $m+1$ โดยในการหาความสัมพันธ์ย้อนกลับของค่า sensitivity จะใช้ Jacobian matrix

$$\frac{\partial n^{m+1}}{\partial n^m} = \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{s^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{s^m}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_{s^m}^m} \end{bmatrix} \tag{42}$$

พิจารณาแต่ละสมาชิกของเมตริกซ์

$$\begin{aligned} \frac{\partial n_i^{m+1}}{\partial n_j^m} &= \frac{\partial \left\langle \sum_{l=1}^{s^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right\rangle}{\partial n_j^m} \\ &= w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} \\ &\triangleq w_{i,j}^{m+1} \dot{f}^m(n_j^m) \end{aligned} \tag{43}$$

ดังนั้น Jacobian matrix สามารถเขียนได้ในรูป

$$\frac{\partial n^{m+1}}{\partial n^m} = W^{m+1} \dot{F}^m(n^m) \tag{44}$$

โดย

$$\dot{F}^m(n^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(n_2^m) \end{bmatrix} \tag{45}$$

สามารถหา Sensitivity ได้จากกฎลูกโซ่ ดังนี้

$$\begin{aligned} s^m &= \frac{\partial \hat{F}}{\partial n^m} = \left(\frac{\partial n^{m+1}}{\partial n^m} \right)^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= \dot{F}^m(n^m)(W^{m+1})^T \frac{\partial \hat{F}}{\partial n^{m+1}} \\ &= \dot{F}^m(n^m)(W^{m+1})^T s^{m+1} \end{aligned} \tag{46}$$

โดยกระบวนการหา sensitivity ของแต่ละเลเยอร์จะทำในทิศทางย้อนกลับ กล่าวคือจะเริ่มจากเลเยอร์สุดท้ายแล้วย้อนกลับที่ละเลเยอร์จนถึงเลเยอร์แรก

$$s^M \rightarrow s^{M-1} \rightarrow \cdots \rightarrow s^2 \rightarrow s^1 \tag{47}$$

พิจารณาที่เลเยอร์สุดท้าย จะได้ค่า sensitivity ดังนี้

$$\begin{aligned} s_i^M &= \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (t-a)^T (t-a)}{\partial n_i^M} \\ &= \frac{\partial \sum_{j=1}^{s^M} (t_j - a_j)^2}{\partial n_i^M} \\ &= -2(t_j - a_j) \frac{\partial a_i}{\partial n_i^M} \\ &= -2(t_j - a_j) \dot{f}^M(n_i^M) \end{aligned} \tag{48}$$

สามารถเขียนในรูปเมตริกซ์ของค่า sensitivity ได้ดังนี้

$$s^M = -2\dot{F}^M(n^M)(t-a) \tag{49}$$

สรุปกระบวนการปรับค่าตัวแปรของ adaptive neural network โดยใช้วิธี backpropagation steepest descent algorithm ดังนี้ ขั้นแรกป้อนอินพุตให้กับ adaptive neural network แล้วจะได้เอาท์พุตเป็น

ค่าของตัวแปร a จากนั้นเริ่มคำนวณค่า sensitivity ของเลเยอร์สุดท้ายจากสมการที่ (49) แล้วคำนวณค่า sensitivity ของเลเยอร์อื่น ๆ ย้อนกลับจนกระทั่งถึงเลเยอร์แรกจากสมการที่ (46) แล้วจะได้สมการการปรับค่าน้ำหนักและ bias ในแต่ล่ะเลเยอร์ดังนี้

$$\begin{aligned} W^m(k+1) &= W^m(k) - \alpha s^m (a^{m-1})^T \\ b^m(k+1) &= b^m(k) - \alpha s^m \end{aligned} \tag{50}$$

2.2 Backpropagation with Levenberg-Marquardt Learning Algorithm เป็นวิธีกระบวนการเรียนรู้อีกวิธีหนึ่งของ adaptive neural network โดยมีพื้นฐานของการปรับค่าตัวแปรตามวิธีของของ Levenberg-Marquardt algorithm ซึ่งมีสมการการปรับค่าตัวแปรดังนี้

$$x_{k+1} = x_k - (H_k + \mu_k I)^{-1} g_k \tag{51}$$

โดย $g_k = \nabla F(x)\big|_{x=x_k} = (\frac{\partial F(x)}{\partial x})^T\Big|_{x=x_k}$ และ $H_k = \nabla^2 F(x)\big|_{x=x_k} = \frac{\partial^2 F(x)}{\partial x^2}\Big|_{x=x_k}$

ถ้ากำหนดให้ $F(x)$ อยู่ในรูปแบบผลรวมกำลังสอง

$$F(x) = \sum_{i=1}^{N} v_i^2(x) = v^T(x)v(x) \tag{52}$$

จะได้ว่า gradient matrix $\nabla F(x) = 2J^T(x)e(x)$ โดยที่ Jacobian matrix $J(x)$ มีสมการดังนี้

$$J(x) = \begin{bmatrix} \frac{\partial v_1(x)}{\partial x_1} & \frac{\partial v_1(x)}{\partial x_2} & \cdots & \frac{\partial v_1(x)}{\partial x_n} \\ \frac{\partial v_2(x)}{\partial x_1} & \frac{\partial v_2(x)}{\partial x_2} & \cdots & \frac{\partial v_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial v_N(x)}{\partial x_1} & \frac{\partial v_N(x)}{\partial x_2} & \cdots & \frac{\partial v_N(x)}{\partial x_n} \end{bmatrix} \tag{53}$$

และ Hessian matrix $\nabla^2 F(x) = 2J^T(x)J(x) + 2S(x)$ โดย $S(x) = \sum_{i=1}^{Q} e_i(x)\nabla^2 e_i(x)$

กำหนดให้มีค่าน้อยมาก ๆ ดังนั้นสามารถประมาณ Hessian matrix ใหม่ได้ว่า

$$\nabla^2 F(x) \cong 2J^T(x)J(x) \tag{54}$$

แทนค่าสมการที่ (54) ลงในสมการที่ (51) จะได้สมการการปรับค่าตัวแปร ตามหลักการ Levenberg-Marquardt algorithm ดังนี้

$$\begin{aligned} x_{k+1} &= x_k - [2J^T(x_k)J(x_k) + \mu_k I]^{-1} 2J^T(x_k)e(x_k) \\ &= x_k - [J^T(x_k)J(x_k) + \mu_k I]^{-1} J^T(x_k)e(x_k) \end{aligned} \tag{55}$$

กำหนดให้กระบวนการเรียนรู้ของ adaptive neural network มี training data set ดังนี้

$$\{p_1, t_1\}, \{p_2, t_2\}, \cdots, \{p_Q, t_Q\} \tag{56}$$

และกำหนดค่าฟังก์ชันความผิดพลาดของกระบวนการเรียนรู้ มีสมการดังนี้

$$\begin{aligned} F(x) &= \sum_{q=1}^{Q} (t_q - a_q)^T (t_q - a_q) = \sum_{q=1}^{Q} e_q^T e_q \\ &= \sum_{q=1}^{Q} \sum_{j=1}^{s^M} (e_{j,q})^2 = \sum_{i=1}^{N} (v_i)^2 \end{aligned} \tag{57}$$

โดย $e_{j,q}$ เป็นสมาชิกตัวที่ j ของเวกเตอร์ค่าความผิดพลาดสำหรับคู่ข้อมูลอินพุตเอาท์พุตที่ q v เป็นเวกเตอร์ค่าความผิดพลาด สามารถเขียนในรูปเวกเตอร์ได้ดังนี้

$$\begin{aligned} v &= \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix}^T \\ &= \begin{bmatrix} e_{1,1} & e_{2,1} & \cdots & e_{s^M,1} & e_{1,2} & \cdots & e_{s^M,Q} \end{bmatrix}^T \end{aligned} \tag{58}$$

และ x เป็นเวกเตอร์ตัวแปรค่าน้ำหนักและ bias ของ adaptive neural network สามารถเขียนในรูปเวกเตอร์ได้ดังนี้

$$\begin{aligned} x &= \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \\ &= \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{s^1,R} & b^1_1 & \cdots & b^1_{s^1} & w^2_{1,1} & \cdots & b^M_{s^M} \end{bmatrix}^T \end{aligned} \tag{59}$$

โดย $N = Q \times s^M$ และ $n = s^1(R+1) + s^2(s^1+1) + \cdots + s^M(s^{M-1}+1)$ โดย R เป็นจำนวนอินพุตของ adaptive neural network แทนค่าลงในสมการที่ (53) จะได้ Jacobian matrix สำหรับกระบวนการเรียนรู้ของ adaptive neural network ที่มีหลายเลเยอร์ ดังนี้

$$J(x) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w^1_{s^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \cdots & \frac{\partial e_{1,1}}{\partial b^1_{s^1}} & \frac{\partial e_{1,1}}{\partial w^2_{1,1}} & \cdots & \frac{\partial e_{1,1}}{\partial b^M_{s^M}} \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w^1_{s^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \cdots & \frac{\partial e_{2,1}}{\partial b^1_{s^1}} & \frac{\partial e_{2,1}}{\partial w^2_{1,1}} & \cdots & \frac{\partial e_{2,1}}{\partial b^M_{s^M}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{s^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{s^M,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{s^M,1}}{\partial w^1_{s^1,R}} & \frac{\partial e_{s^M,1}}{\partial b^1_1} & \cdots & \frac{\partial e_{s^M,1}}{\partial b^1_{s^1}} & \frac{\partial e_{s^M,1}}{\partial w^2_{1,1}} & \cdots & \frac{\partial e_{s^M,1}}{\partial b^M_{s^M}} \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w^1_{s^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \cdots & \frac{\partial e_{1,2}}{\partial b^1_{s^1}} & \frac{\partial e_{1,2}}{\partial w^2_{1,1}} & \cdots & \frac{\partial e_{1,2}}{\partial b^M_{s^M}} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{s^M,Q}}{\partial w^1_{1,1}} & \frac{\partial e_{s^M,Q}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{s^M,Q}}{\partial w^1_{s^1,R}} & \frac{\partial e_{s^M,Q}}{\partial b^1_1} & \cdots & \frac{\partial e_{s^M,Q}}{\partial b^1_{s^1}} & \frac{\partial e_{s^M,Q}}{\partial w^2_{1,1}} & \cdots & \frac{\partial e_{s^M,Q}}{\partial b^M_{s^M}} \end{bmatrix} \tag{60}$$

สำหรับการนำไปใช้ในสมการการปรับค่าตัวแปรของ Levenberg-Marquardt algorithm แต่ละสมาชิกของ Jacobian matrix $J(x)$ จะสามารถคำนวณโดยใช้ backpropagation algorithm ตามสมการที่ (38) สำหรับในเทอมที่เทียบกับตัวแปรค่าน้ำหนัก จะได้ว่า

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times a^{m-1}_{j,q} \triangleq \tilde{s}^m_{i,h} \times a^{m-1}_{j,q} \tag{61}$$

และสำหรับในเทอมที่เทียบกับตัวแปรค่าbias จะได้ว่า

$$[J]_{h,l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times 1 \triangleq \tilde{s}^m_{i,h} \tag{62}$$

โดย $h=(q-1)s^M+k$ และ $\tilde{s}_{i,h}^m=\dfrac{\partial e_{k,q}}{\partial n_{i,q}^m}$ เรียกว่า Marquardt sensitivity ซึ่งจะเริ่มคำนวณจากเลเยอร์สุดท้าย แล้วคำนวณย้อนกลับจนถึงเลเยอร์แรก พิจารณาที่เลเยอร์สุดท้ายจะได้ Marquardt sensitivity ดังนี้

$$\begin{aligned}\tilde{s}_{i,h}^M &= \frac{\partial e_{k,q}}{\partial n_{i,q}^M}=\frac{\partial(t_{k,q}-a_{k,q}^M)}{\partial n_{i,q}^M}=-\frac{\partial a_{k,q}^M}{\partial n_{i,q}^M}\\ &=\begin{cases}-\dot{f}^M(n_{i,q}^M) & \text{for } i=k\\ 0 & \text{for } i\neq k\end{cases}\end{aligned} \tag{63}$$

สรุปได้ว่า เมื่ออินพุต p_q ถูกป้อนให้กับ adaptive neural network จะได้เอาท์พุต a_q^M ออกมา แล้วใช้กระบวนการ backpropagation จะให้ค่า Marquardt sensitivity เริ่มต้นเป็น

$$\tilde{S}_q^M=-\dot{F}^M(n_q^M) \tag{64}$$

โดย $\dot{F}^M(n^M)$ เป็นไปตามสมการที่ (45) และจะได้ค่า Marquardt sensitivity ในเลเยอร์ก่อนหน้าเป็นไปตามสมการที่ (46) ดังนี้

$$\tilde{S}_q^m=-\dot{F}^m(n_q^m)(W^{m+1})^T\tilde{S}_q^{m+1} \tag{65}$$

ค่า Marquardt sensitivity ทั้งหมดจะถูกนำไปคำนวณหา Jacobian matrix ตามสมการที่ (61) และ (62) จากนั้นนำไปแทนค่าในสมการการปรับค่าตัวแปรตามสมการที่ (55)

Adaptive Neuro-Fuzzy Inference Systems (ANFIS)

ANFIS เป็น adaptive neural network ชนิดหนึ่งที่เป็นการรวมเอาหลักการของ first-order Sugeno fuzzy inference system และหลักการการปรับค่าตัวแปรของ adaptive neural network มาไว้ในระบบเดียวกัน (Jyh-Shing, 1993)

1. <u>โครงสร้างของ ANFIS</u>

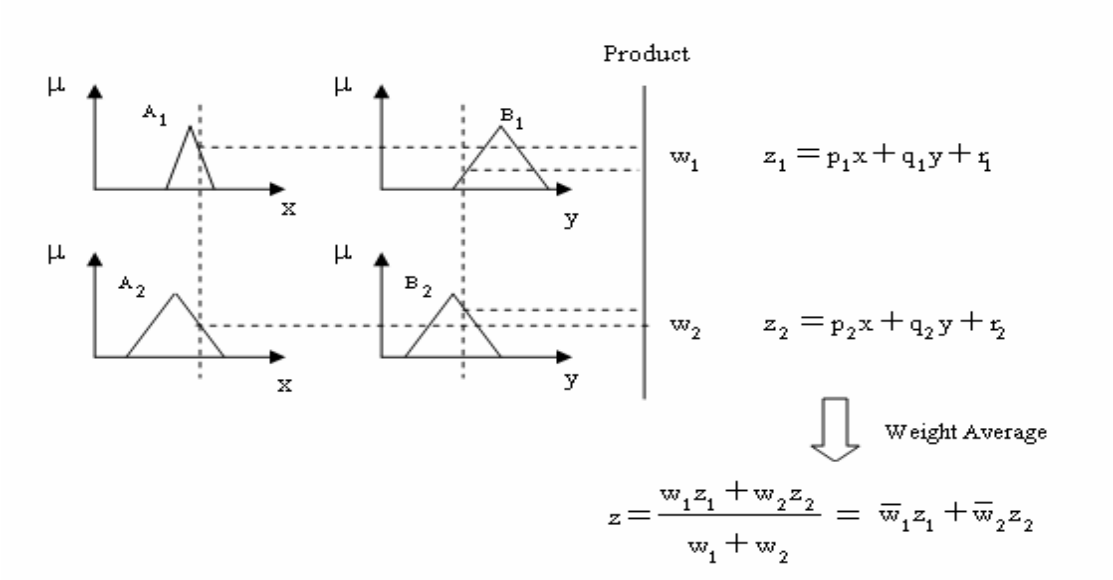
พิจารณาหลักเกณฑ์การสร้างชุดกฎของ first-order Sugeno fuzzy if-then rule โดยมีรูปแบบดังนี้

$$\text{if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = p \cdot x + q \cdot y + r \tag{66}$$

ยกตัวอย่าง ระบบ first-order Sugeno fuzzy model ที่ประกอบด้วย 2 อินพุต คือตัวแปร x, y เอาท์พุตคือตัวแปร z และกำหนดให้มีกฎ 2 ข้อ ดังนี้

ถ้า x เป็น A_1 และ y เป็น B_1 แล้ว $z_1 = p_1 x + q_1 y + r_1$
ถ้า x เป็น A_2 และ y เป็น B_2 แล้ว $z_2 = p_2 x + q_2 y + r_2$

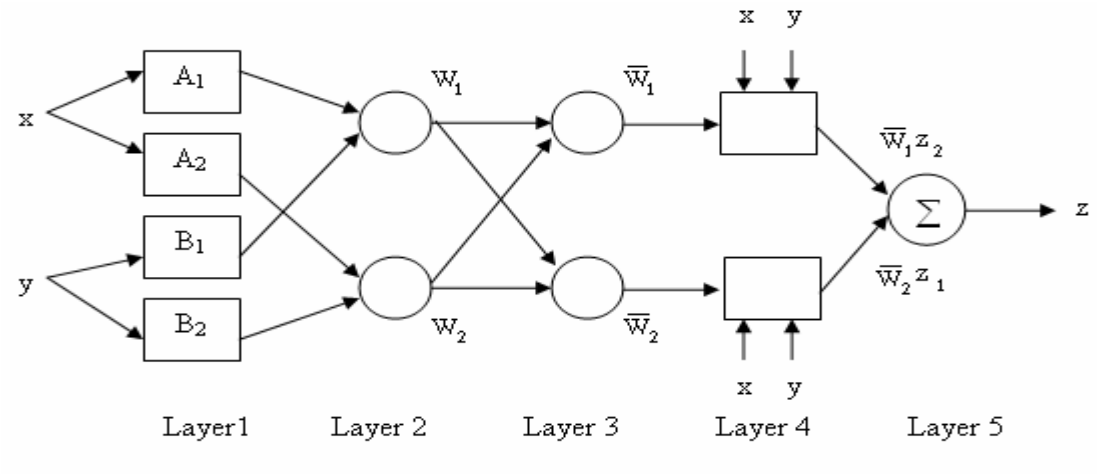
สามารถนำมาเขียนในรูปโครงสร้าง Sugeno fuzzy inference system ได้ดังภาพที่ 12



<u>ภาพที่ 12</u> โครงสร้าง Sugeno Fuzzy Inference System สำหรับ ANFIS

เพื่อความสะดวกในการปรับค่าตัวแปรต่าง ๆ ภายในโครงสร้าง Sugeno fuzzy inference system เราจะทำการปรับเปลี่ยนให้อยู่ในรูปของโครงสร้างแบบ adaptive neural network ดังแสดงในภาพที่ 13

เพื่อให้สามารถใช้กระบวนการเรียนรู้ของ adaptive neural network ในการปรับค่าตัวแปรต่าง ๆ ภายในระบบได้ โดยเรียกระบบที่มีโครงสร้างดังภาพที่ 13 นี้ว่า adaptive neuro-fuzzy inference system (ANFIS)



ภาพที่ 13 โครงสร้างของ Adaptive Neuro-Fuzzy Inference System

จากโครงสร้างของ ANFIS จะเห็นได้ว่าเป็น network ที่ประกอบด้วยเลเยอร์ 5 เลเยอร์ ในแต่ละเลเยอร์ก็จะประกอบด้วยหลาย ๆ นิวรอน โดยในเลเยอร์เดียวกันแต่ละนิวรอน จะมีฟังก์ชันการทำงานเหมือนกัน สามารถอธิบายได้ดังนี้ (กำหนดให้เอาท์พุตของนิวรอนที่ i ของเลเยอร์ที่ l เขียนแทนด้วยตัวแปร O_i^l)

เลเยอร์ที่ 1: แต่ละนิวรอนของเลเยอร์นี้จะทำการคำนวนหาค่าความเป็นสมาชิกของฟังก์ชันของแต่ละอินพุตในแต่ละ fuzzy set โดยทั่วไปแล้วฟังก์ชันค่าความเป็นสมาชิกของแต่ละนิวรอนในเลเยอร์นี้จะนิยมใช้เป็น bell-shape function ซึ่งมีสมการดังนี้

$$O_i^1 = \mu_{A_i}(x) = \frac{1}{1+\left|\frac{x-c_i}{a_i}\right|^{2b_i}} \tag{67}$$

โดยที่ x เป็นค่าอินพุตที่ถูกป้อนเข้าไปในนิวรอนที่ i ส่วน A_i เป็น fuzzy set ที่สัมพันธ์กับนิวรอนตัวที่ i และ $\{a_i, b_i, c_i\}$ เป็นเซตของตัวแปรสำหรับการเปลี่ยนรูปร่างของฟังก์ชันค่าความเป็นสมาชิก โดยตัวแปรในเลเยอร์นี้จะเรียกว่า premise parameters

เลเยอร์ที่ 2: แต่ละนิวรอน ในเลเยอร์นี้จะทำหน้าที่คำนวณหาค่า firing strength ของกฎในแต่ล่ะข้อ โดยการนำเอาค่าอินพุตทุกอันที่เข้ามายังนิวรอนมาคูณกัน

$$O_i^2 = w_i = \mu_{A_i}(x) \cdot \mu_{B_i}(y) \quad , i = 1, 2. \tag{68}$$

เลเยอร์ที่ 3: นิวรอนที่ i ในเลเยอร์นี้จะทำการคำนวณหาค่าอัตราส่วนระหว่างค่า firing strength ของกฎที่ i ส่วนด้วยผลรวมของค่า firing strength ของทุก ๆ กฎ

$$O_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2} \quad , \quad i = 1, 2. \tag{69}$$

เลเยอร์ที่ 4: นิวรอนที่ i ในเลเยอร์นี้จะทำการคำนวณหาค่าผลของกฎข้อที่ i โดยแต่ละนิวรอนจะมี node function ดังนี้

$$O_i^4 = \overline{w}_i z_i = \overline{w}_i (p_i x + q_i y + r_i) \tag{70}$$

โดยที่ $\overline{w}_i$ เป็นค่าเอาท์พุตของเลเยอร์ที่ 3 และ $\{p_i, q_i, r_i\}$ เป็นเซตของค่าตัวแปร โดยตัวแปรในเลเยอร์นี้จะเรียกว่า consequence parameters

เลเยอร์ที่ 5: ในเลเยอร์นี้จะมีเพียงแค่หนึ่งนิวรอน โดยจะทำหน้าที่คำนวณหาเอาท์พุตรวมของระบบโดยการนำเอาผลจากทุก ๆ นิวรอนในเลเยอร์ที่ 4 มารวมกัน

$$O_1^5 = z = \sum_i \overline{w}_i z_i = \frac{\sum_i w_i z_i}{\sum_i w_i} \tag{71}$$

2. <u>ANFIS Learning Algorithm</u>

เนื่องจาก ANFIS เป็นระบบที่มีโครงสร้างเหมือน adaptive neural network ดังนั้นเราสามารถใช้กระบวนการเรียนรู้ของ adaptive neural network มาใช้ในการปรับค่าตัวแปรต่าง ๆ ใน ANFIS ได้ จากกระบวนการเรียนรู้ของ adaptive neural network เราพบว่าวิธี backpropagation with steepest descent algorithm เป็นวิธีที่ง่ายที่สุด และก็มั่นใจได้ว่าสามารถปรับค่าตัวแปรให้ค่าความผิดพลาดมีค่าน้อยที่สุดได้ แต่ก็ใช้เวลานานมาก แต่จากการสังเกตจะพบว่าเอาท์พุตของ ANFIS จะมีคุณสมบัติความเป็นเชิงเส้นในบางตัวแปร พิจารณา ANFIS ที่มีโครงสร้างดังภาพที่ 13 เราจะพบว่า เมื่อค่าตัวแปรในส่วนของ premise parameters ถูกกำหนดให้เป็นค่าใดค่าหนึ่ง เอาท์พุตรวมของ ANFIS สามารถเขียนได้ในรูปสมการผลรวมเชิงเส้นของตัวแปรในส่วน consequence parameters ได้ดังนี้

$$\begin{aligned} z &= \frac{w_1}{w_1+w_2}z_1 + \frac{w_2}{w_1+w_2}z_2 \\ &= \bar{w}_1 z_1 + \bar{w}_2 z_2 \\ &= \bar{w}_1(p_1x+q_1y+r_1) + \bar{w}_2(p_2x+q_2y+r_2) \\ &= (\bar{w}_1x)p_1 + (\bar{w}_1y)q_1 + \bar{w}_1r_1 + (\bar{w}_2x)p_2 + (\bar{w}_2y)q_2 + \bar{w}_2r_2 \end{aligned} \tag{72}$$

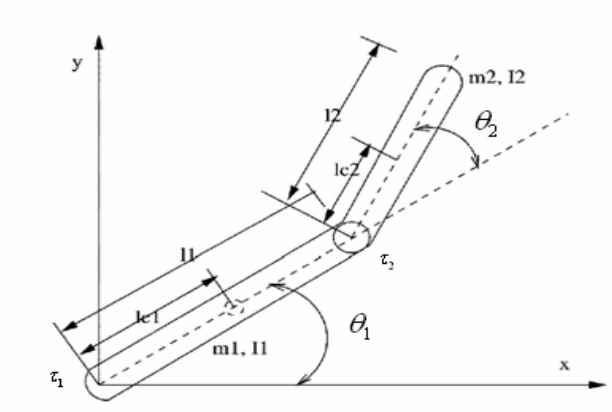
จะเห็นได้ว่า $\{p_1, q_1, r_1, p_2, q_2, r_2\}$ เป็นตัวแปรเชิงเส้น ดังนั้นในส่วนที่เป็นตัวแปรเชิงเส้น เราสามารถที่จะใช้ least-square method ในการปรับค่าตัวแปรนั้นได้ จึงนำไปสู่กระบวนการเรียนรู้แบบ hybrid learning algorithm ซึ่งเป็นการรวมเอากระบวนการเรียนรู้แบบวิธี backpropagation กับวิธี least-square estimator เข้าไว้ด้วยกัน

hybrid learning algorithm จะแบ่งกระบวนการออกเป็น 2 ส่วนคือ ส่วนแรก forward pass ในส่วนนี้สัญญาณอินพุตที่ถูกใส่เข้าไป จะผ่านกระบวนการต่าง ๆ ตั้งแต่เลเยอร์แรกจนถึงเลเยอร์ที่ 4 จากนั้นค่าตัวแปรในส่วน consequence parameters จะถูกหาค่าโดยใช้วิธี least-square estimator และส่วนที่สอง backward pass จะนำค่าความผิดพลาดของเอาท์พุตมาทำ backpropagation เพื่อทำการปรับค่าตัวแปรในส่วน premise parameters ด้วย Levenberg-Marquardt algorithm

ระบบแขนหุ่นยนต์

1. โครงสร้างของระบบแขนหุ่นยนต์

ในงานวิทยานิพนธ์นี่จะทำการศึกษาการหาแบบจำลองของระบบแขนหุ่นยนต์ที่ประกอบด้วย 2 แขน หรือที่เรียกว่า two-link planar robot arms (Andrea, 1999) ซึ่งมีโครงสร้างดังภาพที่ 14



ภาพที่ 14 โครงสร้างของระบบแขนหุ่นยนต์

เมื่อ m_1, m_2 แทน มวลของแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (Kg.)

I_1, I_2 แทน ความเฉื่อย (inertia) ของแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (Kg. m^2)

l_1, l_2 แทน ความยาวของแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (m.)

l_{c1}, l_{c2} แทน ความยาวของศูนย์กลางมวลแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (m.)

τ_1, τ_2 แทน แรงบิด (torque) ที่จ่ายให้กับแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (N-m.)

θ_1, θ_2 แทน ตำแหน่งของแขนหุ่นยนต์แขนที่ 1 และ 2 ตามลำดับ (radian)

2. สมการการเคลื่อนที่ของระบบแขนหุ่นยนต์

สมการการเคลื่อนที่ของแขนหุ่นยนต์สามารถหาได้โดยการใช้สมการของ Euler-Lagrange โดยจะต้องทำการคำนวณหา Lagrangian ซึ่งถูกกำหนดให้อยู่ในเทอมของพลังงานศักย์และพลังงานจลน์ของระบบแขนหุ่นยนต์

ในส่วนของพลังงานจลน์ จะมีวิธีหลาย ๆ วิธีที่ใช้ในการหาค่าพลังงานจลน์ แต่ที่นิยมใช้กันมากจะคำนวณจากค่าความเร็วเชิงมุมในแต่ละส่วนการเคลื่อนที่ของแขนหุ่นยนต์ ซึ่งจะสามารถแบ่งพลังงานจลน์ได้ออกเป็น 2 ส่วนคือ พลังงานจลน์ที่เกิดจากการเลื่อนที่ (translational energy ; Kt) และ พลังงานจลน์ที่เกิดจากการหมุน(rotational energy ; Kr) สำหรับพลังงานจลน์ที่เกิดจากการเลื่อนที่ มีสมการดังนี้

$$Kt = \frac{1}{2}\dot{\theta}^T (J_1^T m_1 J_1 + J_2^T m_2 J_2)\dot{\theta} \tag{73}$$

โดยกำหนดให้ $\dot{\theta} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$

$$J_1 = \begin{bmatrix} -l_{c1}\sin(\theta_1) & 0 \\ l_{c1}\cos(\theta_1) & 0 \end{bmatrix}$$

และ $$J_2 = \begin{bmatrix} -l_1\sin(\theta_1) - l_{c2}\sin(\theta_1+\theta_2) & -l_{c2}\sin(\theta_1+\theta_2) \\ l_1\cos(\theta_1) + l_{c2}\cos(\theta_1+\theta_2) & l_{c2}\cos(\theta_1+\theta_2) \end{bmatrix}$$

สำหรับพลังงานจลน์ที่เกิดจากการหมุน มีสมการดังนี้

$$Kr = \frac{1}{2}\dot{\theta}^T \begin{bmatrix} I_1 + I_2 & I_2 \\ I_2 & I_2 \end{bmatrix} \dot{\theta} \tag{74}$$

ในส่วนของพลังงานศักย์จะมีสมการดังนี้

$$V = m_1 l_{c1} g \sin(\theta_1) + m_2 g (l_1 \sin(\theta_1) + l_{c2}\sin(\theta_1+\theta_2)) \tag{75}$$

กำหนดให้ Lagrangian ของสมการการเคลื่อนที่ของแขนหุ่นยนต์เป็น $L = Kt + Kr - V$ โดยจะสามารถหาสมการการเคลื่อนที่ของแขนหุ่นยนต์ได้จากความสัมพันธ์ ดังนี้

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i} = \tau \qquad i = 1, 2 \tag{76}$$

เมื่อจัดรูปสมการแล้วจะได้สมการการเคลื่อนที่ของหุ่นยนต์ดังนี้

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau \tag{77}$$

โดย $\tau = \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}^T$ เป็นเวกเตอร์ของค่าแรงบิดที่จะป้อนให้กับแขนหุ่นยนต์
$D(\theta)$ เป็นเมตริกซ์ของค่าความเฉื่อย มีค่าดังนี้

$$D(\theta) = \begin{bmatrix} I_1 + I_2 + m_1 l_{c1}^2 + m_2 (l_1^2 + l_{c2}^2 + 2 l_1 l_{c2} \cos(\theta_2)) & I_2 + m_2 (l_{c2}^2 + l_1 l_{c2} \cos(\theta_2)) \\ I_2 + m_2 (l_{c2}^2 + l_1 l_{c2} \cos(\theta_2)) & I_2 + m_2 l_{c2}^2 \end{bmatrix} \quad (78)$$

$C(\theta, \dot{\theta})$ เป็นเมตริกซ์ในเทอมของ Centripetal and Coriolis มีค่าดังนี้

$$C(\theta, \dot{\theta}) = \begin{bmatrix} -m_2 l_1 l_{c2} \sin(\theta_2) \dot{\theta}_2 & -m_2 l_1 l_{c2} \sin(\theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ m_2 l_1 l_{c2} \sin(\theta_2) \dot{\theta}_1 & 0 \end{bmatrix} \quad (79)$$

และ $G(\theta)$ เป็นเวกเตอร์ของแรงโน้มถ่วง มีค่าดังนี้

$$G(\theta) = \begin{bmatrix} (m_1 l_{c1} + m_2 l_1) g \cos(\theta_1) + m_2 l_{c2} g \cos(\theta_1 + \theta_2) \\ m_2 l_{c2} g \cos(\theta_1 + \theta_2) \end{bmatrix} \quad (80)$$

เมื่อนำสมการการเคลื่อนที่ของแขนหุ่นยนต์ มาเขียนในรูปแบบของสมการ state-space โดยกำหนดตัวแปร state ดังนี้ $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$ และกำหนดตัวแปรอินพุตดังนี้ $u = [\tau_1 \quad \tau_2]^T$ จะได้สมการ state-space ของการเคลื่อนที่ของแขนหุ่นยนต์ $\dot{x} = f(x, u)$ ดังนี้

$$\dot{x} = \begin{bmatrix} x_3 \\ x_4 \\ -D^{-1}(\theta)(C(\theta, \dot{\theta})\dot{\theta} + G(\theta)) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ D^{-1}(\theta)[1 \quad 1]^T \end{bmatrix} u \quad (81)$$

3. <u>Feedback Linearization for Closed-Loop Control</u>

จากสมการการเคลื่อนที่ของระบบแขนหุ่นยนต์ ตามสมการที่ (77) คือ

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) = \tau \quad (82)$$

สามารถนำมาเขียนในรูปสมการ state-output โดยกำหนดตัวแปร state เป็น $x=\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T$ เอาท์พุตของระบบเป็น $y=\begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix}^T$ และ $u=\begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}^T$ เป็นทอร์กอินพุตที่ป้อนให้กับระบบแขนหุ่นยนต์ ดังนั้นสามารถเขียนสมการได้ดังนี้

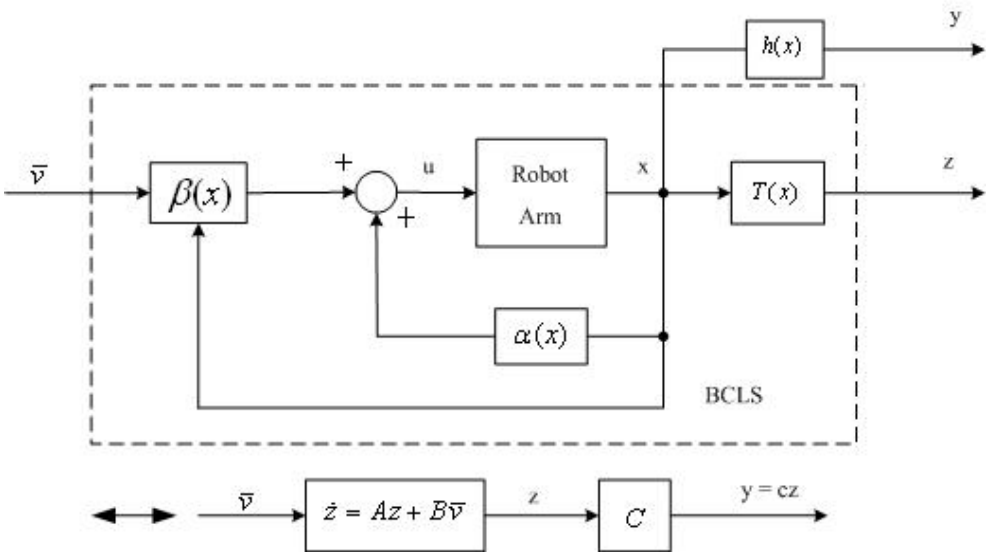
$$\begin{aligned} \dot{x} &= \begin{bmatrix} x_3 \\ x_4 \\ -D^{-1}(\theta)(C(\theta,\dot{\theta})\dot{\theta}+G(\theta)) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ D^{-1}(\theta)[1 \quad 1]^T \end{bmatrix} u \quad \triangleq f(x)+g(x)u \\ y &= \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad \triangleq h(x) \end{aligned} \tag{83}$$

พิจารณาการควบคุมแบบป้อนกลับแบบ feedback linearization ของระบบแขนหุ่นยนต์ ซึ่งมีสมการการควบคุมแบบป้อนกลับ ดังนี้

$$u=\alpha(x)+\beta(x)\overline{v} \tag{84}$$

โดยที่ $\alpha(x)=-D(x)J_h^{-1}\begin{bmatrix} L_f^2 h_1 \\ L_f^2 h_2 \end{bmatrix}$, $\beta(x)=D(x)J_h^{-1}$, $J_h=\begin{bmatrix} \frac{\partial h_1}{\partial \theta_1} & \frac{\partial h_1}{\partial \theta_2} \\ \frac{\partial h_2}{\partial \theta_1} & \frac{\partial h_2}{\partial \theta_2} \end{bmatrix}$

$L_f h_i$ คือ Lie derivative ของ h_i ตามเวกเตอร์ฟิลด์ f และ $L_f^2 h_i$ คือ 2nd Lie derivative ของ h_i ตามเวกเตอร์ฟิลด์ f จากนั้นนำมาเขียนในรูปโครงสร้างการควบคุมแบบ Brunovsky canonical (BCLS) ได้ดังภาพที่ 15



ภาพที่ 15 แผนภาพแสดง Brunovsky Canonical Linear System (BCLS)

โดย $T(x)=\begin{bmatrix} h_1 \\ L_f h_1 \\ h_2 \\ L_f h_2 \end{bmatrix}$ เรียกว่า diffeomorphic transformation และจะพบว่า

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},\quad B=\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix},\quad z=\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix},\quad C=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},\quad \bar{v}=\begin{bmatrix} \bar{v}_1 \\ \bar{v}_2 \end{bmatrix}$$

สังเกตได้ว่า เราสามารถแบ่งระบบออกเป็นระบบย่อย 2 ระบบที่เป็นอิสระต่อกัน ได้ดังนี้

$$\begin{aligned} \begin{bmatrix} \dot{z}_{2i-1} \\ \dot{z}_{2i} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z_{2i-1} \\ z_{2i} \end{bmatrix}+\begin{bmatrix} 0 \\ 1 \end{bmatrix}\bar{v}_i \\ y_i &= \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} z_{2i-1} \\ z_{2i} \end{bmatrix} \end{aligned} \quad ;i=1,2 \tag{85}$$

เพื่อที่จะทำให้ระบบมีเสถียรภาพ เราจะทำการเพิ่มตัวป้อนกลับเชิงเส้น โดยมีสมการดังนี้

$$\bar{v}_i = v_i - \begin{bmatrix} f_{i1} & f_{i2} \end{bmatrix}\begin{bmatrix} z_{2i-1} \\ z_{2i} \end{bmatrix} \tag{86}$$

โดยที่ f_{i1} และ f_{i2} คือค่าคงที่ที่ใช้ในการป้อนกลับ ดังนั้นเราจะได้ระบบย่อยใหม่เป็น ดังสมการ

$$\begin{bmatrix} \dot{z}_{2i-1} \\ \dot{z}_{2i} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -f_{i1} & -f_{i2} \end{bmatrix} \begin{bmatrix} z_{2i-1} \\ z_{2i} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v_i \triangleq \bar{A}_i \begin{bmatrix} z_{2i-1} \\ z_{2i} \end{bmatrix} + \bar{b}_i v_i \tag{87}$$

ซึ่งโพลของระบบเชิงเส้นข้างต้น สามารถหาได้จากสมการ

$$\begin{aligned} \left| \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -f_{i1} & -f_{i2} \end{bmatrix} \right| &= 0 \\ \begin{vmatrix} s & -1 \\ -f_{i1} & s+f_{i2} \end{vmatrix} &= 0 \\ s^2 + f_{i2}s + f_{i1} &= 0 \end{aligned} \tag{88}$$

แก้สมการจะได้ โพลของระบบ คือ $\dfrac{-f_{i2} \pm \sqrt{{f_{i2}}^2 - 4f_{i1}}}{2}$ สมมุติให้ระบบที่เราต้องการมีโพลอยู่ที่ตำแหน่ง $s_{1,2} = -\xi\omega_n \pm j\omega_n\sqrt{1-\xi^2}$ เราก็สามารถหาค่า f_{i1}, f_{i2} เพื่อให้ได้ตำแหน่งโพลของระบบตามที่ต้องการได้ โดยการเทียบสัมประสิทธิ์

พิจารณาเลือกแบบจำลองทางคณิตศาสตร์ของระบบเชิงเส้น

$$\begin{aligned} \begin{bmatrix} \dot{z}^d_{2i-1} \\ \dot{z}^d_{2i} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -f_{i1} & -f_{i2} \end{bmatrix} \begin{bmatrix} z^d_{2i-1} \\ z^d_{2i} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v_i^d \\ y_i^d &= \dot{z}^d_{2i-1} \end{aligned} \tag{89}$$

จากการกำหนด y_i^d พิจารณาอนุพันธ์อันดับ 1 และ 2 ได้ดังสมการ

$$\begin{aligned} y_i^d &= z^d_{2i-1} \\ \dot{y}_i^d &= \dot{z}^d_{2i-1} = z^d_{2i} \\ \ddot{y}_i^d &= \dot{z}^d_{2i} = -f_{i1}z^d_{2i-1} - f_{i2}z^d_{2i} + v_i^d \\ &= -f_{i1}y_i^d - f_{i2}\dot{y}_i^d + v_i^d \end{aligned} \tag{90}$$

ดังนั้นสามารถเขียนสมการการหาค่า v_i^d ได้ดังนี้

$$v_i^d = \ddot{y}_i^d + f_{i2}\dot{y}_i^d + f_{i1}y_i^d \tag{91}$$

เพื่อที่จะกำจัดค่าความผิดพลาดของสัญญาณเอาท์พุต $\left(y_i - y_i^d\right)$ เราจะกำหนด cost function ดังนี้

$$J(\Delta v_i) = \int_0^{t_f} \left[\left(v_i - v_i^d\right)^T R \left(v_i - v_i^d\right) + \left(\overline{y}_i - \overline{y}_i^d\right)^T Q \left(\overline{y}_i - \overline{y}_i^d\right) \right] dt \tag{92}$$

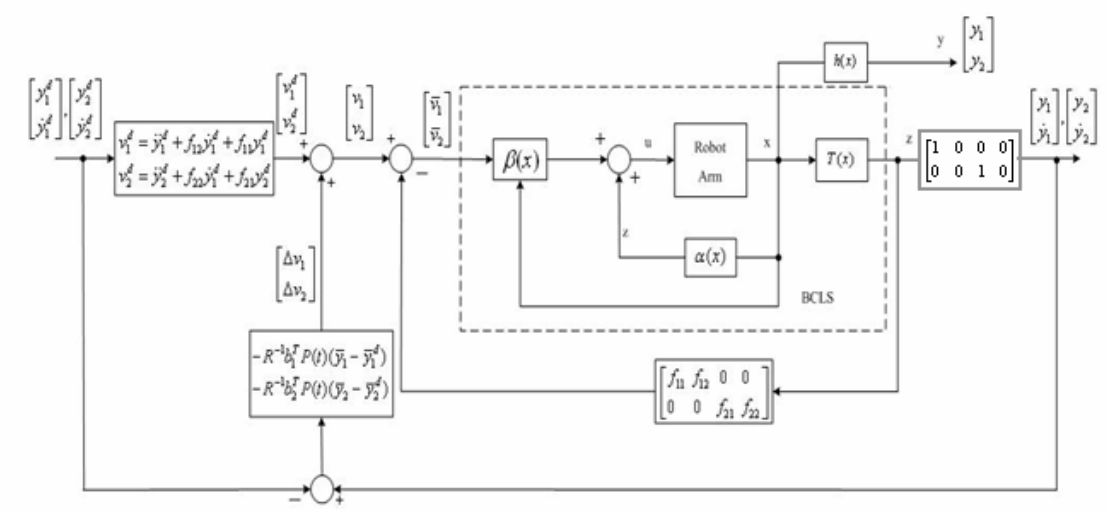
โดย $\overline{y}_i = [y_i \;\; \dot{y}_i]^T$ และ $\overline{y}_i^d = [y_i^d \;\; \dot{y}_i^d]^T$ แล้วจะได้ว่า ค่า optimal correction ที่สอดคล้องกับเงื่อนไขสมการที่ (92) คือ

$$\Delta v_i = -R^{-1}\overline{b}_i^T P(t)\left(\overline{y}_i - \overline{y}_i^d\right) \tag{93}$$

เมื่อ $P(t)$ สามารถหาค่าได้จากสมการ Riccati ตามสมการดังนี้

$$P(t)\overline{A}_i + \overline{A}_i^T P(t) - P(t)\overline{b}_i R^{-1}\overline{b}_i^T P(t) + Q = 0 \tag{94}$$

โดยที่ R เป็นค่าคงที่จำนวนจริงบวก และ Q เป็น semi-positive definite matrix ดังนั้นสามารถนำมาเขียนเป็นแผนภูมิการควบคุมตามภาพที่ 16



<u>ภาพที่ 16</u> แผนภาพการควบคุมระบบแขนหุ่นยนต์แบบ Feedback Linearization