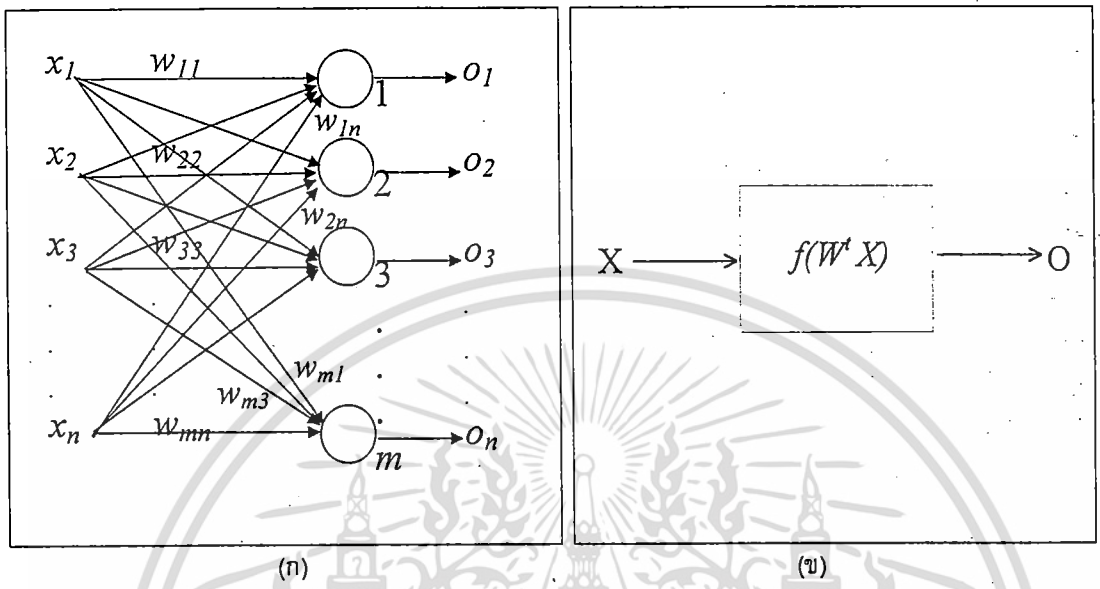
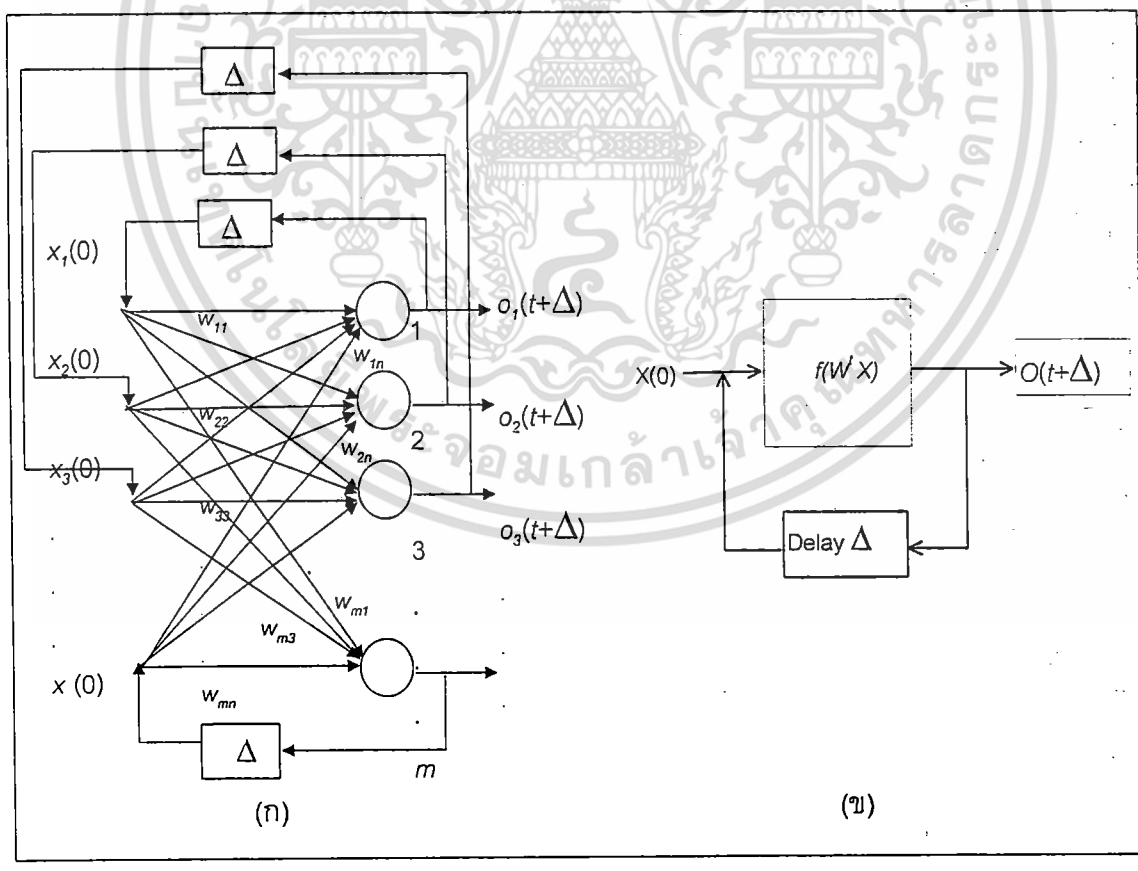
การจดจำตัวอักษรแบบลายมือเขียนโดยใช้นิวโรแอนนีลลิ่ง

รศ. ดร. กิตติ ไพฑูรย์วัฒนกิจ
สาธิต อินทจักร์

เป็นการสอนโครงข่ายด้วยแบบจำลองการแอนนีลลิ่ง หัวข้อที่ 5. เป็นการสอนโครงข่ายแบบแบ็กค์โพรพาเกชันแอนนีลลิ่งหรือนิวโรแอนนีลลิ่ง หัวข้อที่ 6 เป็นการทดสอบโครงข่ายที่พัฒนาขึ้นมาเพื่อใช้ในการรู้จำตัวเลขที่เป็นลายมือเขียน และหัวข้อสุดท้ายเป็นการสรุปพร้อมทั้งข้อเสนอแนะเพื่อเป็นแนวทางสำหรับงานวิจัยที่จะพัฒนาต่อไป



รูปที่ 1. โครงข่ายฟีดฟอร์เวิร์ด (ก) ลักษณะการเชื่อมโยง (ข) บล็อกไดอะแกรม



รูปที่ 2. โครงข่ายป้อนกลับแบบเวลาไม่ต่อเนื่องชั้นเดียว (ก) ลักษณะการเชื่อมโยง (ข) บล็อกไดอะแกรม

2. โครงข่ายฟีดฟอร์เวิร์ด

โครงข่ายเซลล์ประสาทหรือนิวรอลเน็ตเวิร์ค ทั้งที่เป็นแบบจำลองและเป็นโครงข่ายของเซลล์ประสาทจริงของมนุษย์ จะมีการเชื่อมต่อกันของโหนด (Nodes) ในลักษณะของโครงข่ายอย่างหนาแน่น เพื่อให้โครงข่ายสามารถเรียนรู้และสามารถจดจำสิ่งที่เรียนรู้มาแล้วได้ ซึ่งการเชื่อมโยงของโครงข่ายจะมีสองลักษณะด้วยกันคือ

1. โครงข่ายฟีดฟอร์เวิร์ดหรือโครงข่ายที่ส่งสัญญาณไปข้างหน้า โครงข่ายชนิดนี้จะประกอบด้วยชั้นต่าง ๆ ของโครงข่ายโดยชั้นแรกจะเป็นอินพุทและชั้นสุดท้ายเป็นชั้นของเอาท์พุท ส่วนระหว่างชั้นอินพุทกับเอาท์พุท อาจจะมีหรือไม่มีก็ได้ ซึ่งจะขึ้นอยู่กับอัลกอริทึมที่ใช้ในการสอนโครงข่าย เช่น ถ้าเป็นโครงข่าย Perceptron แบบหลายชั้น (Multilayer Perceptron) [1] ก็จะมีชั้นที่อยู่ระหว่างอินพุทกับเอาท์พุท ซึ่งอาจจะมีมากกว่าหนึ่งชั้นก็ได้, ส่วนโครงข่าย Self-Organizing Map ของ Kohonen [2] จะมีเพียงชั้นของอินพุทกับเอาท์พุทเท่านั้น การเชื่อมต่อระหว่างชั้นของโครงข่ายแบบฟีดฟอร์เวิร์ด จะมีค่าน้ำหนักเป็นตัวเชื่อม และสัญญาณอินพุทที่เข้ามาจะถูกส่งไปตามทิศทางของลูกศรจนถึงชั้นของเอาท์พุทโดยไม่มีการป้อนกลับดังรูปที่ 1.
2. โครงข่ายที่มีการป้อนกลับ (Feedback Networks) ในส่วนแรกของโครงข่ายนี้จะเป็นโครงข่ายฟีดฟอร์เวิร์ดเหมือนกับแบบแรก และส่วนที่เพิ่มเข้ามาคือส่วนของการป้อนกลับดังแสดงในรูปที่ 2. และการป้อนกลับจะมีการหน่วงเวลาไปจากเวลาเดิมเท่ากับ Δ ซึ่งโครงข่ายในรูปที่ 2. จะเรียกว่า โครงข่ายรีเคอร์เรนท์ (Recurrent Networks) [1]

3. การสอนโครงข่าย

3.1 การเรียนรู้ของโครงข่าย

การเรียนรู้ของระบบโครงข่ายนิวรอลเน็ตเวิร์คจะมีประสิทธิภาพเพียงใดนั้น จะขึ้นอยู่กับค่าถ่วงน้ำหนักของโครงข่าย ซึ่งการสอน (Training) โครงข่ายก็คือการหาค่าถ่วงน้ำหนัก (Weights) ที่เหมาะสมให้แก่โครงข่ายนั้น ๆ วิธีการสอนแบบจำลองระบบโครงข่ายนิวรอลเน็ตเวิร์คมีอยู่สองแบบด้วยกันคือ

1. การสอนแบบชี้นำ (Supervised Learning) การสอนโดยวิธีนี้ จะกำหนดเซตของการสอนให้กับโครงข่าย ซึ่งเซตนี้ประกอบด้วยอินพุทและเอาท์พุทที่ต้องการ (Output Desired) เมื่อป้อนอินพุทให้กับโครงข่าย ๆ ก็จะทำการประมวลผลจนได้คำตอบและค่าถ่วงน้ำหนักออกมาชุดหนึ่ง สำหรับคำตอบที่ได้จากโครงข่ายจะถูกนำมาคำนวณค่าความผิดพลาด โดยวัดเป็นระยะทางว่ามีความห่างจากคำตอบที่ต้องการของอินพุทในชุดเดียวกันมากน้อยเพียงใด ถ้ายังมีความผิดพลาดสูงอยู่ก็จะมีการปรับค่าถ่วงน้ำหนัก และทำการสอนต่อไปจนกว่าค่าความผิดพลาด ระหว่างคำตอบของโครงข่ายกับเอาท์พุทที่ต้องการมีค่าน้อยพอที่จะยอมรับได้จึงจะหยุดการสอน และค่าถ่วงน้ำหนักที่ได้ก็จะเป็นเหมือนฟังก์ชันที่ใช้ในการแปลงข้อมูล
2. การสอนแบบไม่มีการชี้นำ (Unsupervised Learning) การสอนโดยวิธีนี้ จะป้อนอินพุทเข้าสู่โครงข่าย และภายในโครงข่ายจะมีโหนดเอาท์พุทอยู่หลายโหนดด้วยกัน โดยแต่ละโหนดจะแทนกลุ่มของข้อมูลที่มีคุณสมบัติเหมือนกัน เมื่อป้อนอินพุทเข้าสู่โครงข่าย ๆ จะคำนวณค่าความสัมพันธ์ที่มีอยู่ภายในเซตของอินพุท โดยอาศัยค่าถ่วงน้ำหนักเป็นตัวแยกความแตกต่างของอินพุทไปเก็บไว้ในโหนดเอาท์พุทของโครงข่าย การสอนโดยวิธีนี้จะไม่สามารถระบุได้ว่าเอาท์พุทโหนดใดเป็นของ

ข้อมูลกลุ่มไหน ผู้ใช้จะต้องกำหนดเอง ซึ่งต่างจากการสอนแบบชี้นำที่โครงข่ายสามารถระบุกลุ่มของเอาท์พุทได้อย่างแน่นอน

การสอนโครงข่ายแบบจำลองเซลล์ประสาท เป็นการหาฟังก์ชันการแปลง (Transfer function) โดยฟังก์ชันการแปลงที่ได้จะมีคุณสมบัติไม่เป็นเชิงเส้น ซึ่งฟังก์ชันการแปลงของนิวรอลเน็ตเวิร์ค ในที่นี้คือเซตของค่าถ่วงน้ำหนักของโครงข่าย ดังนั้นฟังก์ชันการแปลงจะมีศักยภาพมากน้อยเพียงใดนั้น จะขึ้นอยู่กับค่าถ่วงน้ำหนักของโครงข่ายนั้นๆ ว่ามีความเสถียรมากน้อยเพียงใด และค่าถ่วงน้ำหนักคำนวณได้จากการสอนโครงข่าย ซึ่งการสอนโครงข่ายมีหลายแบบด้วยกันเช่น กฎการสอนของ Hebb, กฎการสอนแบบ Perceptron ของ Rosenblatt, กฎการสอนแบบเดลต้า, กฎการสอนของ Widrow-Hoff, กฎการสอนโดยใช้สหสัมพันธ์, กฎการสอนแบบ Winner-Take-All, และกฎการสอนแบบ Outstar ของ Grossberg ซึ่งกฎการสอนเหล่านี้ได้กล่าวไว้อย่างละเอียดใน [1, 3, 4] และในตารางที่ 1. ได้สรุปกฎการสอนโครงข่ายของวิธีต่างๆ ไว้

ตารางที่ 1. แสดงการปรับค่าถ่วงน้ำหนักสำหรับกฎการสอนแบบต่างๆ

กฎการสอน	การปรับค่าถ่วงน้ำหนัก	ค่าเริ่มต้นของค่าถ่วงน้ำหนัก	การสอน	Activation Function
Hebb	$\Delta w_{ij} = cf(W_i^t X)x_j$	0	U	Any
Preceptron	$\Delta w_{ij} = c\left(d_i - f(W_i^t X)\right)x_j$	จำนวนจริงใดๆ	S	Binary Bipolar or Binary Unipolar
Delta	$\Delta w_i = c\left(d_i - o_i\right)f'(W_i^t X)X$	จำนวนจริงใดๆ	S	ฟังก์ชันต่อเนื่อง
Widrow-Hoff	$\Delta w_{ij} = c\left(d_i - W_i^t X\right)x_j$	จำนวนจริงใดๆ	S	Any
Correlation	$\Delta w_{ij} = cd_i x_j$	0	S	Any
Winner-Take-All	$\Delta w_{mj} = c\left(x_j - w_{mj}\right)$ m : โหนดที่เป็น Winner	Random Normalized	U	ฟังก์ชันต่อเนื่อง
Outstar	$\Delta w_{ij} = c\left(d_i - w_{ij}\right)$	0	S	ฟังก์ชันต่อเนื่อง

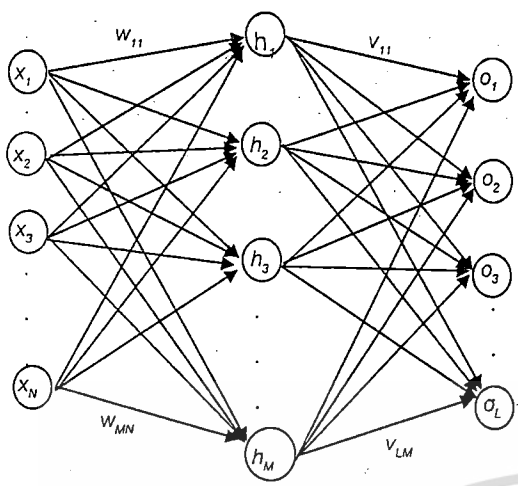
c เป็นค่าคงที่ของอัตราการเรียนรู้ [3-5]

S : การสอนแบบชี้นำ, U : การสอนแบบไม่มีการชี้นำ

Δw_{ij} : ค่าถ่วงน้ำหนักที่ถูกปรับค่าและมีการเชื่อมต่อระหว่างโหนดอินพุทที่ i และ โหนดเอาท์พุทที่ j

3.2 การสอนโครงข่ายแบบแพร่กระจายกลับ

การแพร่กระจายกลับหรือแบกค์โพรพาเกชัน (Back-propagation) เป็นขั้นตอนที่ใช้สอนโครงข่ายแบบฟีดฟอร์เวิร์ด ซึ่งเป็นแบบจำลองโครงข่ายเซลล์ประสาทที่มีการเชื่อมโยงกันเป็นโครงข่ายแบบเป็นชั้นๆ ดังในรูปที่ 3.

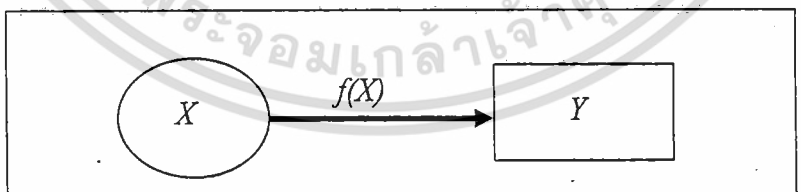


$X = \{x_1, x_2, ..., x_N\}$;
เซตของอินพุท (Input Layer)
$H = \{h_1, h_2, ..., h_M\}$;
เซตของฮิดเดน (Hidden Layer)
$O = \{o_1, o_2, ..., o_L\}$;
เซตของเอาท์พุท (Output Layer)
W : เมตริกซ์ของค่าถ่วงน้ำหนักที่เชื่อมต่อระหว่างชั้นอินพุทกับฮิดเดน
V : เมตริกซ์ของค่าถ่วงน้ำหนักที่เชื่อมต่อระหว่างชั้นฮิดเดนกับชั้นเอาท์พุท

รูปที่ 3. แสดงโครงข่ายฟีดฟอร์เวิร์ดที่มีการเชื่อมต่อกัน 3 ชั้น

ซึ่งโครงข่ายนี้มีการเชื่อมโยงกัน 3 ชั้น ซึ่งประกอบด้วยชั้นของอินพุทที่มีเซลล์ประสาทอยู่ N โหนด ถัดมาเป็นชั้นของฮิดเดนหรือชั้นภายใน (Hidden Layer) ซึ่งประกอบด้วยโหนดต่างๆ จำนวน M โหนด และสุดท้ายคือชั้นของเอาท์พุทที่มีโหนดต่างๆ อยู่ L โหนด โครงข่ายแบบฟีดฟอร์เวิร์ดในรูปที่ 3. แต่ละโหนดในชั้นเดียวกันจะไม่มีการเชื่อมต่อกัน การเชื่อมโยงกันจะมีเฉพาะระหว่างชั้นเท่านั้น และการเชื่อมโยงนี้จะต่อถึงกันทุกโหนด โครงข่ายแบบฟีดฟอร์เวิร์ด ไม่จำเป็นต้องมีสามชั้นเหมือนในรูปที่ 3. เสมอไป อาจจะมีจำนวนชั้นมากกว่านี้ก็ได้ซึ่งอาจมีสี่ชั้น [6] โดยการเพิ่มชั้นฮิดเดนเข้าไปอีกหนึ่งชั้น หรือถ้าต้องการจำนวนชั้นมากกว่านี้ก็สามารถทำได้โดยการเพิ่มชั้นของฮิดเดน

ชั้นฮิดเดน [1, 2, 4] เป็นตัวเพิ่มความสามารถให้แก่โครงข่ายเซลล์ประสาท โครงข่ายฟีดฟอร์เวิร์ด ถ้าไม่มีชั้นฮิดเดนก็จะกลายเป็นโครงข่ายแบบ Perceptron อย่างเช่นโครงข่ายแบบ Perceptron ที่เสนอโดย Rosenblatt ซึ่ง Minsky และ Papert ได้แสดงให้เห็นว่าโครงข่ายเซลล์ประสาทจำลองนี้ เป็นโครงข่ายที่มีความสัมพันธ์ระหว่างอินพุทกับเอาท์พุทในลักษณะเชิงเส้น ซึ่งไม่สามารถจะแก้ปัญหาเอ็กคลูซีฮออร์ (Exclusive OR: XOR) ได้ [2] ในกรณีที่ไม่มีชั้นฮิดเดน ความสัมพันธ์ของฟังก์ชันการแปลงสามารถแสดงในตัวอย่างข้างล่างนี้

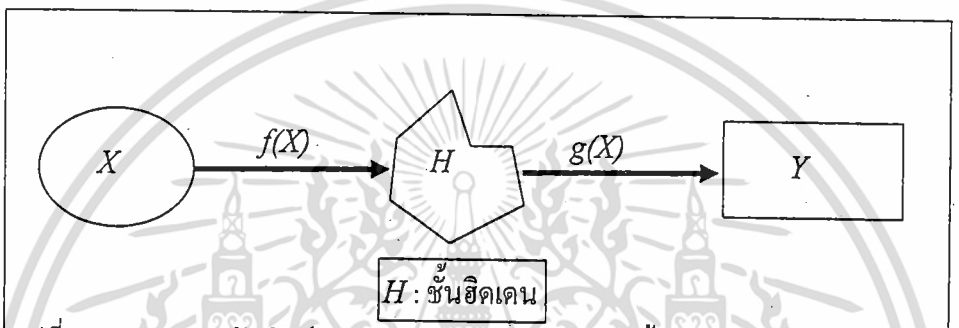


รูปที่ 4. แสดงความสัมพันธ์ระหว่างตัวแปร X กับ Y

จากรูปที่ 4. เมื่อ X เป็นตัวแปรอิสระ Y เป็นตัวแปรตามและ $f(X)$ เป็นฟังก์ชันการแปลงจาก X ไปยัง Y ในกรณีที่ X เป็นอิสระต่อกันแบบเชิงเส้น (Linearly Independent) เราสามารถคำนวณฟังก์ชันการแปลง $f(X)$ ได้ ตัวอย่างเช่น ถ้า $\alpha_1 = \{0, 1\}$ และ $\alpha_2 = \{1, 0\}$ เมื่อ $X = [\alpha_1, \alpha_2]$ และ $Y^t = [1, 1]$ ดังนั้นเราสามารถคำนวณฟังก์ชันการแปลงได้ดังนี้

$$X \cdot f(X) = Y \rightarrow \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

เมื่อฟังก์ชันการแปลง $f(X) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ แต่ถ้าตัวแปร X ไม่เป็นอิสระต่อกันแบบเชิงเส้น (Linearly Dependent) เราไม่สามารถจะกำหนดฟังก์ชันการแปลง $f(X)$ ได้ เช่นถ้า α_1 = {0, 0} และ α_2 = {1, 1} เมื่อ $X = [\alpha_1, \alpha_2]$ และ Y^t = [1, 1] จะได้ว่า $X = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ ซึ่งในกรณีนี้เราไม่สามารถคำนวณฟังก์ชัน $f(X)$ ที่สอดคล้องกับ Y ได้ และในกรณีนี้จะเหมือนกับปัญหา XOR แต่ถ้าเพิ่มชั้นฮิดเดนเข้าไปในรูปที่ 4. จะทำให้สามารถแก้ปัญหาที่เป็น Linearly Dependent นี้ได้ [1, 2] (ดูรูปที่ 5.)



รูปที่ 5. แสดงความสัมพันธ์ระหว่างตัวแปร X กับ Y โดยมีชั้นฮิดเดนเป็นตัวเชื่อม

ในการหาความสัมพันธ์จาก X ไปยัง Y โดยใช้ชั้นฮิดเดน ในแบบจำลองโครงข่ายนิวรอลเน็ตเวิร์ค จะทำให้ฟังก์ชัน $f(X)$ และ $g(X)$ ไม่เป็นเชิงเส้นซึ่ง $f(X)$ และ $g(X)$ ในที่นี้ก็คือ ค่าถ่วงน้ำหนักที่ได้จากการสอนโครงข่ายนั่นเอง การที่จะกำหนดชั้นของฮิดเดนว่าในโครงข่ายหนึ่งๆ ควรมีฮิดเดนกี่ชั้น และแต่และชั้นประกอบด้วยกี่โหนดนั้น ไม่มีกฎเกณฑ์หรือทฤษฎีมาสนับสนุน ดังนั้นการกำหนดจำนวนชั้นและจำนวนโหนดของแต่ละชั้นของฮิดเดนสามารถทำได้โดยการทดลอง

3.3 วิธีการของกฎเดลต้า

กฎเดลต้า (Delta Rule) ถูกพัฒนาขึ้นมาเพื่อใช้สอนโครงข่ายแบบจำลองเซลประสาท [4] การพัฒนาขั้นมาครั้งแรกจะใช้สอนโครงข่ายเซลล์ประสาทจำลอง Perceptron ซึ่งจะเป็นกฎการสอน Perceptron แบบต่อเนื่อง และต่อมาได้พัฒนากฎการสอนเดลต้าให้สามารถใช้ได้กับโครงข่ายที่มีการเชื่อมต่อกันหลายชั้น อย่างฟีดฟอร์เวิร์ด จึงเรียกกฎเดลต้าที่ถูกพัฒนาขึ้นใหม่นี้ว่า กฎเดลต้าเอนกประสงค์ (Generalized Delta Rule: GDR)

ขั้นตอนการสอนโครงข่ายฟีดฟอร์เวิร์ดจะเรียกว่า "ขั้นตอนการสอนแบบค่าความผิดพลาดแพร่กระจายกลับหรือ Error Back-propagation" ซึ่งเป็นการแพร่กระจายกลับของค่าความผิดพลาดที่เกิดขึ้นในชั้นเอาท์พุท ที่ต้องการกับเอาท์พุทที่คำนวณได้ โดยคำนวณย้อนกลับจากชั้นเอาท์พุทผ่านชั้นฮิดเดนตลอดมาจนถึงชั้นอินพุท เพื่อทำการปรับค่าถ่วงน้ำหนัก จากรูปที่ 3. เมื่อทำการสอนโครงข่ายนี้ด้วยการสอนแบบชี้นำ การสอนจะต้องป้อนเซตของข้อมูลที่จะใช้สอน ซึ่งประกอบด้วยเซตของอินพุทและเซตของเอาท์พุทที่ต้องการ กำหนดให้ P เป็นจำนวนเซตทั้งหมดของข้อมูลที่ใช้ในการสอนโครงข่าย ดังนั้นเซตอินพุทและเอาท์พุทที่ต้องการทั้งหมดจะ

เป็นเซตคู่ลำดับที่มีอยู่ P เซต, ถ้าให้ D เป็นเซตของเอาท์พุทที่ต้องการจะได้ว่า $D_p = \{d_1, d_2, d_3, ..., d_L\}$, เมื่อ p = 1, 2, 3, ..., P

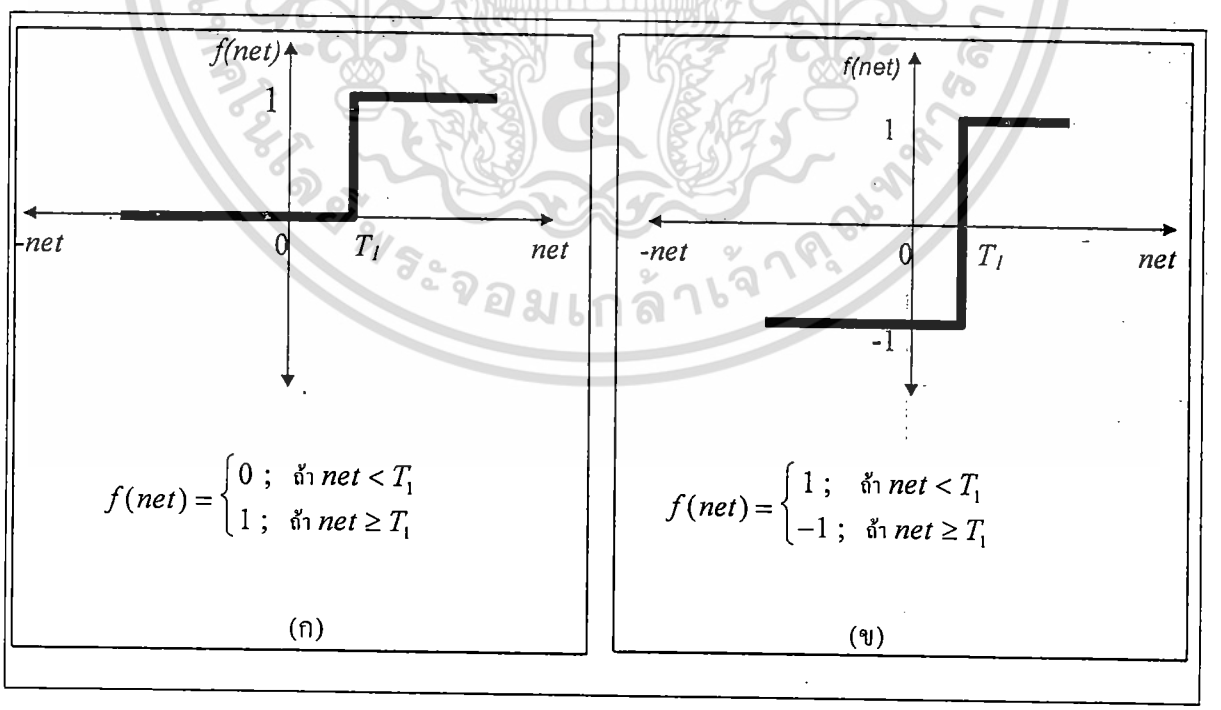
W เป็นเมตริกซ์ของค่าถ่วงน้ำหนักที่เชื่อมโยงระหว่างชั้นของอินพุทกับชั้นฮิดเดนซึ่ง W มีขนาด M แถว และ N คอลัมน์

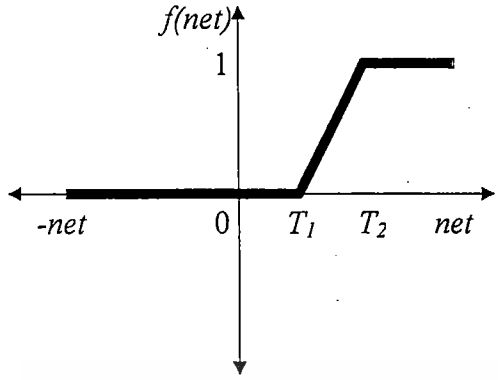
$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3N} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ w_{M1} & w_{M2} & w_{M3} & \cdots & w_{MN} \end{bmatrix}$$

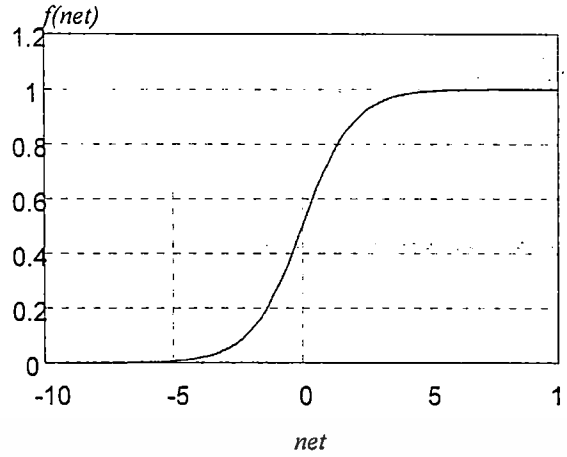
V เป็นเมตริกซ์ของค่าถ่วงน้ำหนักที่เชื่อมโยงระหว่างชั้นฮิดเดนไปยังชั้นเอาท์พุท ซึ่งมีขนาด L แถว และ M คอลัมน์

$$V = \begin{bmatrix} v_{11} & v_{12} & v_{13} & \cdots & v_{1M} \\ v_{21} & v_{22} & v_{23} & \cdots & v_{2M} \\ v_{31} & v_{32} & v_{33} & \cdots & v_{3M} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ v_{L1} & v_{L2} & v_{L3} & \cdots & v_{LM} \end{bmatrix}$$

และ *f(net)* เป็น Activation Function ใดๆ ดังที่แสดงในรูปที่ 6.



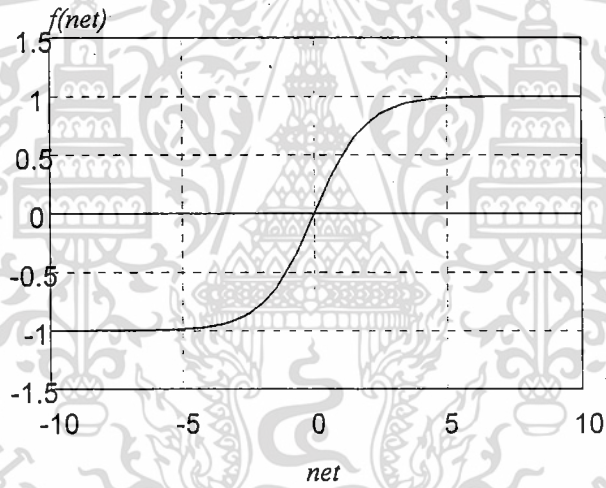




$$f(net) = \begin{cases} 1; & \text{ถ้า } net \geq T_2 \\ \frac{1}{T_2 - T_1}(T_2 - net); & \text{ถ้า } T_1 \leq net < T_2 \\ 0; & \text{ถ้า } net < T_1 \end{cases}$$

$$f(net) = \frac{1}{1+e^{-net}}$$

(ง)



(ง)

$$f(net) = \frac{2}{1+e^{-net}} - 1$$

(จ)

รูปที่ 6. Activation Function แบบชนิดต่างๆ (ก) Threshold Logic (ข) Bipolar Binary (ค) Linear Threshold (ง) Sigmoid (จ) Bipolar Continuous

เมื่อป้อนเซตของข้อมูลที่ใช้สอนเข้าสู่โครงข่ายจะสามารถคำนวณโหนดของฮิดเดนที่ m ได้ดังนี้

$$net_m = \sum_{n=1}^{N} w_{mn} x_n$$
$$h_m = f(net_m) \qquad (1)$$

และสามารถคำนวณเอาท์พุทโหนดที่ l ได้ดังนี้

$$net_l = \sum_{m=1}^{M} v_{lm} x_m$$
$$o_l = f(net_l) \quad (2)$$

เมื่อคำนวณชั้นของเอาท์พุทครบทุกโหนดแล้ว ขั้นต่อไปเป็นการปรับค่าถ่วงน้ำหนักของโครงข่าย โดยใช้ค่าผลรวมของค่าความคลาดเคลื่อนกำลังสอง ระหว่างค่าของผลลัพธ์ที่ได้จากโครงข่ายกับค่าเอาท์พุทที่ต้องการของแพทเทิร์นที่ p ซึ่งสามารถคำนวณได้ดังนี้

$$E_p = \frac{1}{2} \sum_{l=1}^{L} (o_{pl} - d_{pl})^2 \quad (3)$$

ดังนั้นในการปรับค่าถ่วงน้ำหนักที่เชื่อมต่อระหว่างชั้นเอาท์พุทกับชั้นฮิดเดนสำหรับแพทเทิร์นที่ p (H_p) สามารถคำนวณได้ดังนี้

$$\Delta V = -\eta \frac{\partial E_p}{\partial V}$$
$$= \alpha V + \eta \delta_p^V H_p \quad (4)$$

โดยที่ $\delta_p^V = (D_p - O_p) O_p (1 - O_p)$ ซึ่งเป็นความคลาดเคลื่อนภายในชั้นของเอาท์พุท

α เป็นค่าโมเมนตัม [1, 4]

η เป็นค่าอัตราการเรียนรู้ (Learning Rate) ซึ่งเป็นค่าคงที่ [1, 4]

และการปรับค่าถ่วงน้ำหนักที่เชื่อมต่อระหว่างชั้นฮิดเดนกับชั้นอินพุทสามารถคำนวณได้ดังนี้

$$\Delta W = \alpha W + \eta \delta_p^W X_p \quad (5)$$

โดยที่ $\delta_p^W = H_p (1 - H_p)(\delta_p^V V)$ ซึ่งเป็นค่าความคลาดเคลื่อนภายในชั้นฮิดเดน สำหรับ α และ η จะมีคุณสมบัติเหมือนกับสมการที่ (4)

ในกรณีที่มีโครงข่ายมี I ชั้นสามารถปรับค่าถ่วงน้ำหนักโดยวิธีค่าความผิดพลาดแพร่กระจายกลับได้ดังนี้

$$\Delta W^i = \alpha W^i + \eta \delta_p^i X_p^{i-1} \quad (6)$$

เมื่อ W^i เป็นเมตริกซ์ค่าถ่วงน้ำหนักที่อยู่ระหว่างชั้นที่ i กับ i-1 (i = 1, 2, 3, ..., I), X_p^{i-1} เป็นเซตของอินพุทแพทเทิร์นที่ p จากชั้นที่ i-1 และ δ_p^i เป็นเวกเตอร์ของความคลาดเคลื่อนสำหรับชั้นที่ i สำหรับชั้นเอาท์พุทหรือชั้นที่ I สามารถคำนวณได้ดังนี้

$$\delta_p^I = (D_p - O_p^I)O_p^I(1 - O_p^I) \quad (7)$$

โดยที่ O_p^I คือชั้นของเอาท์พุท และสำหรับความคลาดเคลื่อนของชั้นที่ i ใดๆ โดยที่ $i \neq I$ สามารถคำนวณได้ดังนี้

$$\delta_p^i = O_p^i(1 - O_p^i)(\delta_p^{i+1} W^{i+1}) \quad (8)$$

โดยที่ O_p^i คือเอาท์พุทชั้นที่ i แพทเทิร์นที่ p และสำหรับ δ_p^{i+1} และ W^{i+1} เป็นค่าความคลาดเคลื่อนและเมตริกซ์ค่าถ่วงน้ำหนักของชั้นถัดไป

3.4 ปัจจัยที่ทำให้ประสิทธิภาพการสอนโครงข่ายด้วยวิธี Back-propagation เพิ่มมากขึ้นมีดังนี้

- การกำหนดค่าเริ่มต้นให้กับเมตริกซ์ถ่วงน้ำหนัก

ก่อนที่จะทำการสอนโครงข่ายฟีดฟอร์เวิร์ดโดยวิธี Error Back-propagation จำเป็นต้องกำหนดค่าเริ่มต้นให้กับเมตริกซ์ค่าถ่วงน้ำหนักที่เชื่อมโยงระหว่างชั้นทุกชั้น โดยค่านี้จะเป็นเลขจำนวนจริงที่มีค่าน้อยๆ ซึ่งได้มาจากการสุ่ม ค่าเริ่มต้นของเมตริกซ์เชื่อมโยงจะมีผลต่อเวลาที่ใช้ในการสอน และอาจจะส่งผลถึงค่าถ่วงน้ำหนักหลังจากที่ได้สอนโครงข่ายไปแล้ว ว่าจะมีศักยภาพมากน้อยเพียงใดในการแปลงข้อมูลอินพุทไปสู่เอาท์พุท ในอัลกอริทึมที่ 1 แสดงขั้นตอนการกำหนดค่าเริ่มต้นให้กับเมตริกซ์ค่าถ่วงน้ำหนัก

อัลกอริทึมที่ 1 ขั้นตอนการกำหนดค่าเริ่มต้นให้กับเมตริกซ์ค่าถ่วงน้ำหนัก

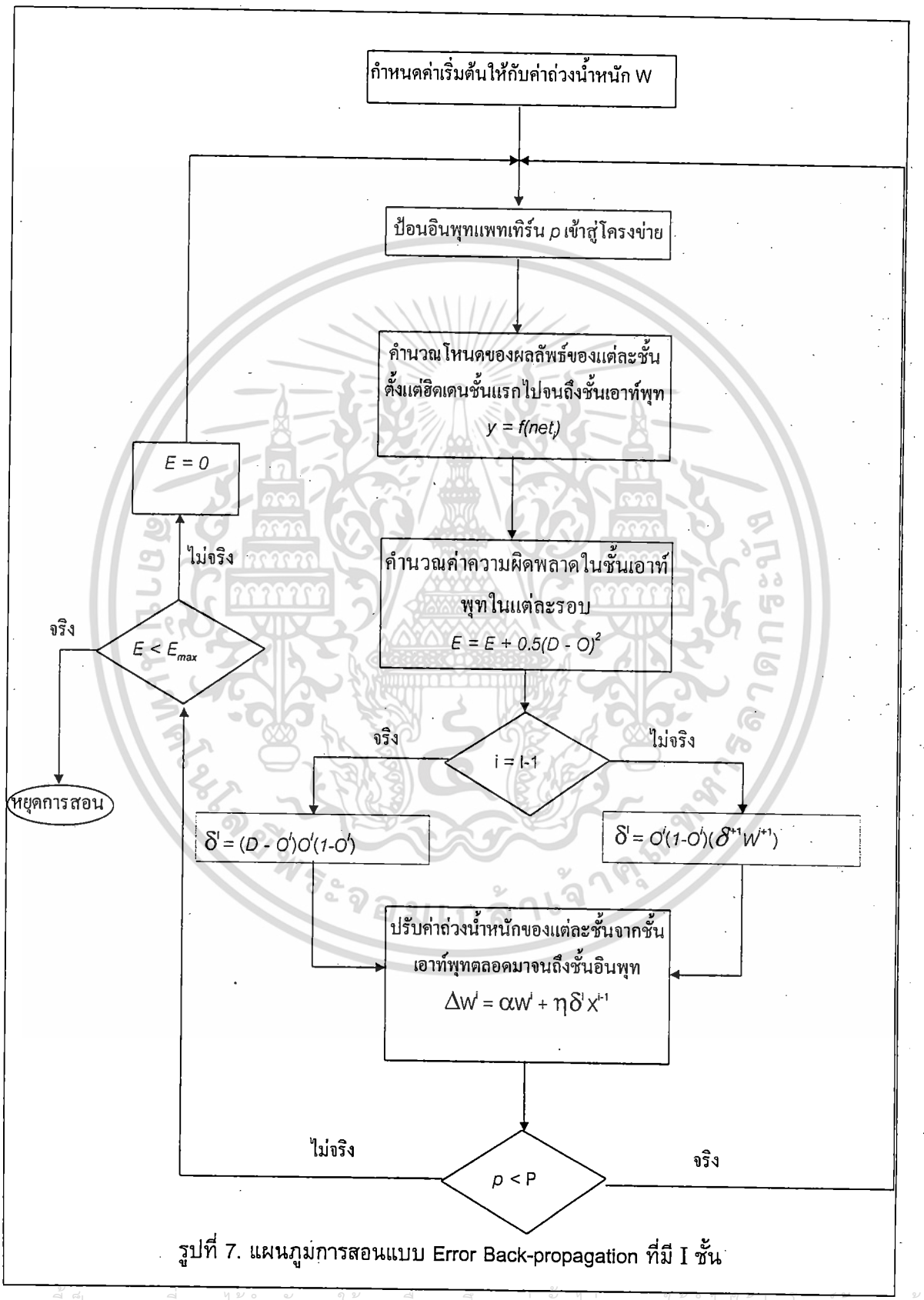
กำหนดให้ ROW เป็นจำนวนแถวของเมตริกซ์ถ่วงน้ำหนัก
COL เป็นจำนวนคอลัมน์ของเมตริกซ์ถ่วงน้ำหนัก
W เป็นเมตริกซ์ของค่าถ่วงน้ำหนักที่เชื่อมโยงระหว่างชั้นใดๆ

```
void    Initial_Weights (Matrix W, int ROW, int COL)
{
        for (int i = 0;  i < ROW;  i++)
                for (int j = 0; j < COL; j++)
                        W[i][j] = 1 - random (101) / 53.0;
}
```

- อัตราเร็วในการเรียนรู้ (Learn Rate : η) ซึ่งโดยทั่วไปแล้วค่านี้จะเป็นค่าคงที่ที่มีค่าอยู่ระหว่าง 0.05 - 0.25 [2] แต่มีงานวิจัยบางส่วนที่ค่าอัตราเร็วในการเรียนรู้มีการปรับค่าได้เช่น กำหนดให้อัตราเร็วในการเรียนรู้ผันแปรตามจำนวนรอบที่สอน [7], อัตราเร็วในการเรียนรู้ผันแปรตามค่าความผิดพลาดเฉลี่ยของระบบ [8] และ ค่าอัตราเร็วในการเรียนรู้ผันแปรตามค่าความแปรปรวนของแต่ละแพทเทิร์น [6]

- โมเมนต์ตัม (Momentum : α) สำหรับกฎการสอนแบบเจเนอรอลไลซ์เดลต้า โดยทั่วไปค่าโมเมนต์ตัมอาจจะใช้หรือไม่ใช้ก็ได้ แต่ค่านี้จะมีคุณสมบัติช่วยป้องกันการแกว่ง (Oscillate) ของระบบ [3-4]

โดยค่านี้จะสัมพันธ์กับค่าอัตราเร็วในการเรียนรู้ คือถ้าอัตราเร็วในการเรียนรู้มีค่ามากแต่ค่าโมเมนต์ดัมมีค่าน้อย จะทำให้ระบบโครงข่ายเกิดการแกว่ง ส่วนกรณีอื่นๆ โครงข่ายจะไม่เกิดการแกว่งแต่จะมีผลต่อเวลาที่ใช้ในการสอน



รูปที่ 7. แผนภูมิการสอนแบบ Error Back-propagation ที่มี I ชั้น

- ไบแอส (Bias) โหนดไบแอสสำหรับโครงข่ายโดยทั่วไปอาจจะมีหรือไม่มีก็ได้ ในกรณีของโครงข่ายที่ใช้ในระบบที่เป็น Real-Time แล้วจะไม่ค่อยมีการใช้โหนดไบแอสเนื่องจากจะเสียเวลาในการคำนวณ และถ้ามีการออกแบบโครงข่ายอย่างเหมาะสมแล้วเทอมไบแอสไม่จำเป็นต้องมี แต่ถ้าโครงข่ายมีจำนวนโหนดไม่เหมาะสมคือมีจำนวนโหนดน้อยไป การเพิ่มโหนดไบแอสเข้าไปจะทำให้โครงข่ายสามารถจดจำได้ดีขึ้น โหนดไบแอสจะถูกเพิ่มเข้าไปเป็นโหนดๆ หนึ่งในแต่ละชั้น ยกเว้นในชั้นของผลลัพธ์และโหนดไบแอสที่เพิ่มเข้าไปมักจะเป็นค่าคงที่ ผลของการเพิ่มโหนดไบแอสเข้าไปในโครงข่ายจะทำให้ประสิทธิภาพการสอนดีขึ้น [9]

3.5 ขั้นตอนการสอน

การสอนโครงข่ายฟีดฟอร์เวิร์ดด้วยวิธี Back-propagation มีขั้นตอนการสอนดังในรูปที่ 7. ซึ่งเป็นแผนภูมิของขั้นการสอนของอัลกอริทึมนี้ โดยที่ E_{max} เป็นค่าความผิดพลาดต่ำสุดที่พอจะยอมรับได้ ซึ่งในการสอนโครงข่ายในรอบหนึ่งๆ (Epoch) ก็คือการสอนชุดของแพทเทิร์นทั้งหมด P แพทเทิร์น ถ้าค่าความผิดพลาดของระบบ E ยังมีค่ามากกว่าค่าความผิดพลาดที่ตั้งไว้ E_{max} ก็จะนำข้อมูลชุดเดิมมาทำการสอนใหม่จนกว่าค่าความผิดพลาด E จะน้อยกว่าค่า E_{max}

4. การจำลองการแอนนีลลิ่ง

ในหัวข้อ 3.4 เราได้เห็นแล้วว่ามีปัจจัยหลายตัวที่มีผลต่อประสิทธิภาพของโครงข่ายฟีดฟอร์เวิร์ด ที่ใช้อัลกอริทึมแบ็กค์โพรพาเกชัน ซึ่งปัจจัยเหล่านี้ไม่ได้มีค่าตายตัวสำหรับระบบทั่วๆ ไป คือเมื่อเราใช้แบ็กค์โพรพาเกชันกับระบบงานหนึ่ง ค่าเหล่านี้ก็จะใช้ได้ดีกับระบบนี้เท่านั้น แต่เมื่อนำแบ็กค์โพรพาเกชันไปใช้กับข้อมูลของระบบอื่น เราก็ต้องทดลองปรับเปลี่ยนค่าปัจจัยเหล่านี้กันใหม่ เพื่อให้ได้ค่าที่ต้องการ ดังนั้นในงานวิจัยของเราจึงพยายามแก้ปัญหาที่เกิดขึ้น โดยการพัฒนาการสอนโครงข่ายฟีดฟอร์เวิร์ดขึ้นมาใหม่ คือใช้การจำลองการแอนนีลลิ่งเข้ามาช่วยในกระบวนการสอนด้วยอัลกอริทึมแบ็กค์โพรพาเกชัน ซึ่งเราเรียกอัลกอริทึมที่พัฒนาขึ้นมาใหม่นี้ว่า "แบ็กค์โพรพาเกชันแอนนีลลิ่ง" แต่ก่อนที่จะไปดูอัลกอริทึมที่พัฒนาขึ้นมา เรามาทำความเข้าใจแนวคิดพื้นฐานของการจำลองการแอนนีลลิ่งกันก่อน เพื่อให้สามารถเข้าใจระบบที่พัฒนาขึ้นมาได้ง่ายขึ้น

การจำลองการแอนนีลลิ่งเป็นวิธีการค้นหาคำตอบแบบสุ่ม ที่นำหลักการคำนวณมาจากกลศาสตร์สถิติ (Statistical Mechanics) สำหรับคำตอบแบบสุ่มที่ระบบแอนนีลลิ่งค้นหาได้ จะเป็นคำตอบที่ดีที่สุด (Global Optimum) หรือไม่ก็เป็นคำตอบที่ใกล้เคียงกับคำตอบจริงมากที่สุด ซึ่งเมื่อนำการจำลองการแอนนีลลิ่งมาใช้ร่วมกับการสอนโครงข่ายฟีดฟอร์เวิร์ด ด้วยอัลกอริทึมแบ็กค์โพรพาเกชัน กระบวนการแอนนีลลิ่งจะค้นหาคำตอบที่ดีที่สุดให้กับโครงข่าย ซึ่งคำตอบที่ดีที่สุดในที่นี้คือ ค่าถ่วงน้ำหนักที่ทำให้ค่าความผิดพลาดระหว่างผลลัพธ์ที่ได้จากโครงข่าย กับผลลัพธ์ที่เราต้องการมีค่าต่ำที่สุด

4.1 รู้จักกับการจำลองการแอนนีลลิ่ง

แอนนีลลิ่งเป็นแบบจำลองที่เทียบเคียงได้กับกระบวนการผสมโลหะ คือเมื่อโลหะที่ถูกหลอมละลายถูกทำให้เย็นตัวลง อะตอมของโลหะก็จะจัดเรียงตัวของมันเอง เพื่อจะลดพลังงานศักย์ในอะตอมลง ถ้าโลหะที่ถูกหลอมละลายถูกทำให้เย็นตัวลงอย่างรวดเร็ว อะตอมจะจัดเรียงตัวเองอย่างไม่เป็นรูปแบบที่แน่นอน ทำให้โลหะนั้นๆ ขาดความแข็งแรงทนทาน ในทางกลับกันถ้าโลหะที่ถูกหลอมละลายมีการเย็นตัวลงอย่างช้าๆ แต่ละระดับของอุณหภูมิที่ลดลงมา อะตอมของโลหะก็จะค่อยๆ เข้าสู่สภาวะสมดุลย์ทางความร้อน และโลหะที่ได้ก็จะมีโครง

สร้างผลึกที่แข็งแรงและมีพลังงานศักย์ต่ำที่สุด ซึ่งกระบวนการเย็นตัวลงอย่างช้าๆ ของอุณหภูมิของโลหะที่หลอมละลายจนกลายเป็นโลหะที่อยู่ในสภาวะเสถียรที่อุณหภูมิห้องนี้จะเรียกว่า "การแอนนีลลิ่ง"

การจำลองระบบแอนนีลลิ่งขึ้นมา เราจะกำหนดให้ T เป็นค่าอุณหภูมิ และถ้ากำหนดให้ตัวแปรสุ่ม (Stochastic variable) X เป็นระดับพลังงานของสถานะปัจจุบันของโลหะ ดังนั้นค่าความน่าจะเป็นที่สถานะของพลังงานจะเท่ากับ E จะมีฟังก์ชันการแจกแจงเป็นแบบโบสต์มาน (Boltzman distribution) [10] ซึ่งสามารถกำหนดได้ดังนี้

$$\Pr(X = E) = \frac{1}{Z(T)} \exp\left(-\frac{E}{K_b T}\right) \qquad (9)$$

เมื่อ Z(T) เป็นฟังก์ชันจำแนก (Partition Function) ที่เป็นค่าถ่วงน้ำหนัก ซึ่งกำหนดได้โดย

$$Z(T) = \sum \exp\left(-\frac{E}{K_b T}\right) \qquad (10)$$

ส่วนค่า K_b เป็นค่าคงที่โบสต์มาน (Boltzman Constant)

4.2 อัลกอริทึมเมโทรโปลีส (The Metropolis Algorithm)

ในทางฟิสิกส์ การแอนนีลลิ่งเป็นกระบวนการทางความร้อน ที่ใช้หาสถานะที่มีพลังงานศักย์ต่ำสุดของของแข็ง ซึ่งมีอยู่ด้วยกันสองขั้นตอนคือ

- ขั้นตอนการเพิ่มอุณหภูมิให้กับสารในสถานะของแข็ง จนของแข็งร้อนมากพอที่จะทำให้เปลี่ยนสถานะเป็นของเหลว
- ขั้นตอนการลดอุณหภูมิของสารลงอย่างช้าๆ จนกระทั่งอะตอมของสารนั้นมีการจัดเรียงตัวเข้าสู่สถานะเดิมของมัน (Ground State) หรืออีกนัยหนึ่งก็คืออยู่ในสภาพที่เป็นผลึกนั่นเอง

ในสภาวะที่สารเป็นของเหลว อะตอมของของเหลวจะมีการจัดเรียงตัวในลักษณะที่ไม่เป็นระเบียบ แต่เมื่อสารกลับสู่สถานะเดิมของมัน พลังงานศักย์ที่แขวงตัวอยู่ในอะตอมก็จะลดลงจนมีค่าต่ำสุด และอะตอมก็จะจัดเรียงตัวกันอย่างเป็นระเบียบตามโครงสร้างเดิมของสารนั้นๆ การที่จะได้สถานะเดิมเช่นนี้ทำได้โดยการเพิ่มอุณหภูมิให้มากพอ และทำให้มีการเย็นตัวลงอย่างช้าๆ ถ้าถูกทำให้เย็นตัวลงเร็วเกินไปสารนั้นก็จะอยู่ในสถานะกึ่งเสถียร (Meta-Stable state) หรือผลึกจะจัดเรียงตัวกันไม่เป็นระเบียบ (Amorphous)

จากขบวนการแอนนีลลิ่งทางฟิสิกส์ข้างต้น เมโทรโปลิสได้สร้างอัลกอริทึมที่ประยุกต์มาจากการจำลองการที่ของแข็ง มีการเปลี่ยนแปลงทางอุณหภูมิจนกระทั่งอยู่ในสภาวะสมดุล โดยอัลกอริทึมนี้จะอาศัยวิธีการมอนติคาโล (Monte Carlo Technique) มาสร้างลำดับสถานะขึ้นมาดังนี้

ให้สถานะปัจจุบันของพลังงานที่อยู่ในสารเป็นสถานะที่ i ซึ่งเราแทนด้วย E_i และสถานะต่อมาเป็นสถานะที่ j ที่เกิดจากกลไกการรบกวน (Perturbation Mechanism) ซึ่งสถานะนี้สารก็จะมีการเปลี่ยนแปลงทางพลังงานไปเล็กน้อยไปเป็น E_j ตัวอย่างกลไกการรบกวน เช่นการทำให้อนุภาคของสารมีการเคลื่อนที่ ถ้าผลต่างของพลังงาน $E_j - E_i$ มีค่าน้อยกว่าศูนย์ สถานะที่ j จะถูกยอมรับเป็นสถานะปัจจุบัน แต่ถ้าผลต่างมีค่ามากกว่าศูนย์ สถานะที่ j จะถูกยอมรับเป็นสถานะปัจจุบันด้วยค่าความน่าจะเป็นค่าหนึ่ง ซึ่งกำหนดได้ดังนี้

$$\Pr(\text{การยอมรับสถานะที่ } j) = \exp\left(\frac{E_i - E_j}{K_b T}\right) \qquad (11)$$

เงื่อนไขในการยอมรับสถานะทางพลังงานที่ j ที่อธิบายมานี้เรียกว่า เกณฑ์เมโทรโปลิส (Metropolis Criterion) และอัลกอริทึมที่ใช้ในวิธีการนี้เรียกว่า "เมโทรโปลิสอัลกอริทึม (Metropolis Algorithm)"

4.3 อัลกอริทึมการจำลองการแอนนีลลิ่ง

เราสามารถนำวิธีการเมโทรโพลิสมาใช้สร้างลำดับของคำตอบของปัญหาการเรียงลำดับที่เหมาะสม (Combinatorial optimization) ในขั้นตอนต่างๆ ของกระบวนการนี้ สามารถเปรียบเทียบระบบหลายอนุภาคกับปัญหาการเรียงลำดับที่เหมาะสม โดยอาศัยสิ่งที่เหมือนกันดังนี้

- คำตอบของปัญหาการหาคำตอบที่เหมาะสมของปัญหาการเรียงลำดับ จะเปรียบได้กับสถานะทางกายภาพของระบบที่มีหลายอนุภาค
- ฟังก์ชันค่าใช้จ่าย (Cost Function) จะสอดคล้องกับพลังงานของแต่ละสถานะ

ตัวแปรที่มีบทบาทสำคัณต่ออุณหภูมิ หรืออีกนัยหนึ่งคือจำนวนรอบการวนซ้ำ (Iteration) ของการคำนวณ ที่เรียกว่าตัวแปรควบคุม (Control Parameter) ดังนั้นวิธีการจำลองการแอนนีลลิ่งสามารถจัดได้ว่าเป็นการวนซ้ำ ของอัลกอริทึมเมโทรโพลิสที่มีกลไกการยอมรับดังนี้

กำหนดให้ (S, f) เป็นการหาคำตอบที่เหมาะสมของปัญหาใดๆ เมื่อ i และ j เป็นคำตอบสองค่าที่มีฟังก์ชันค่าใช้จ่ายเป็น $f(i)$ และ $f(j)$ ตามลำดับ ฟังก์ชันความน่าจะเป็นที่ใช้ในการกำหนดเกณฑ์ในการยอมรับค่า j เป็นคำตอบสำหรับการหาของปัญหาคือ

$$P_c(\text{ยอมรับค่า } j) = \begin{cases} 1 & \text{ถ้า } f(j) \le f(i) \\ \exp\left(\dfrac{f(i)-f(j)}{c}\right) & \text{ถ้า } f(j) > f(i) \end{cases} \quad (12)$$

เมื่อ c เป็นสมาชิกของเลขจำนวนจริงที่มีค่าเป็นบวก ซึ่งในที่นี้มันทำหน้าที่เป็นตัวแปรควบคุม

จะเห็นว่ากลไกในการยอมรับค่า j จะขึ้นอยู่กับกลไกการรบกวนของอัลกอริทึมเมโทรโพลิส ด้วยเหตุที่เกณฑ์ในการยอมรับค่า j จะมีลักษณะเช่นเดียวกันกับเกณฑ์ของเมโทรโพลิส

ในอัลกอริทึมที่ 2. เป็นอัลกอริทึมของการจำลองการแอนนีลลิ่ง เมื่อ c_k เป็นค่าตัวแปรควบคุมของการวนซ้ำครั้งที่ k และ L_k เป็นการเปลี่ยน (Transition) ที่เกิดขึ้นในการวนซ้ำครั้งที่ k ของอัลกอริทึมเมโทรโพลีส โดยการเปลี่ยนนี้จะเป็นการรวมการกระทำต่างๆ ที่เกิดขึ้นกับผลลัพธ์ ที่มีการเปลี่ยนรูปไปของผลลัพธ์ปัจจุบันเข้าไปเก็บไว้ในลำดับย่อย โดยการกระทำที่เกิดขึ้นกับผลลัพธ์ที่ทำในมีการเปลี่ยน จะประกอบไปด้วยการทำงานสองขั้นตอนคือ

- การทำงานของกลไกการสร้าง
- การทำงานของเกณฑ์การยอมรับ

อัลกอริทึมที่ 2. แสดงขั้นตอนต่างๆ ของการจำลองการแอนนีลลิ่ง [10]

```
void SIMULATED_ANNEALING()
{
        INITIALIZE(i_start, c_0, L_0);
        k = 0;
        i = i_start;
        do
        {
                for( l = 1; l<=L_k; l++)
```

```
        {
            GENERATE(j from S_i );
            If ( f(j) ≤ f(i) )
                i = j;
            else
                if ( exp( (f(i) − f(j)) / c_k ) > random[0,1) )
                    i = j;
        }
        k = k+1;
        CALCULATE_LENGTH(L_k);
        CALCULATE_CONTROL(c_k);
    }while stop_criterion_is_not_TRUE;
}
```

4.4 การสอนโครงข่ายฟีดฟอร์เวิร์ดด้วยการจำลองการแอนนีลลิ่ง

จากอัลกอริทึมการจำลองการแอนนีลลิ่ง เราได้พัฒนาให้สามารถนำมาสอนโครงข่ายฟีดฟอร์เวิร์ดได้ สำหรับแนวทางในการใช้แอนนีลลิ่งสอนโครงข่ายคือ จะใช้ค่าความผิดพลาดของโครงข่ายเป็นฟังก์ชันค่าใช้จ่าย เพื่อหาค่าถ่วงน้ำหนักที่เหมาะสมให้กับโครงข่าย ซึ่งอัลกอริทึมการจำลองการแอนนีลลิ่งจะทำหน้าที่ค้นหาค่าความผิดพลาดต่ำสุด (Global minimum) เมื่อเราได้ค่าความผิดพลาดต่ำสุด ก็จะได้ค่าถ่วงน้ำหนักที่เหมาะสมกับระบบมากที่สุดเช่นกัน ซึ่งในอัลกอริทึมที่ 3. ได้แสดงการทำงานของการจำลองการแอนนีลลิ่งเพื่อหาค่าถ่วงน้ำหนักที่เหมาะสมให้กับโครงข่ายฟีดฟอร์เวิร์ด

อัลกอริทึมที่ 3. แสดงขั้นตอนการหาค่าถ่วงน้ำหนักที่เหมาะสมให้กับโครงข่ายฟีดฟอร์เวิร์ดด้วยการจำลองแอนนีลลิ่ง

```
// Feedforward Simulated Annealing;
/* 1  */ void FEEDFORWARD_SIMULATED_ANNEALING()
/* 2  */ {
/* 3  */       Wi = GetWeightFromBackpropagation();
/* 4  */       Ei = ErrorFromFeedforward(Wi);
/* 5  */       Eopt = Ei;
/* 6  */       Esub_opt = Ei;
/* 7  */       Wopt = Wi;
/* 8  */       Wsub_opt = Wi;
/* 9  */       l = 1;
/* 10*/        k = 0;
/* 11*/        while(Eopt > Ethreshold)
/* 12*/        {
/* 13*/              l++;
/* 14*/              Wj = Perturbation(Wi);
/* 15*/              Ej = ErrorFromFeedforward(Wj);
/* 16*/              if(Ej <= Ei)
/* 17*/              {
/* 18*/                    AcceptNewState(Ei, Ej, Wi, Wj);
/* 19*/                    if(Ej < Esub_opt)
/* 20*/                          AcceptSub_optimalState(Esub_opt, Ej, Wsub_opt, Wj);
/* 21*/                    if(Esub_opt < Eopt)
```

```
/* 22*/                        AcceptOptimalState(Eopt, Esub_opt, Wopt, Wsup_opt);
/* 23*/              }
/* 24*/              else    //Metropolis Algorithm
/* 25*/              {
/* 26*/                      Pc = exp((Ej-Ei)/Ck);
/* 27*/                      Pr = UniformDistribution();
/* 28*/                      if(Pr < Pc)
/* 29*/                              AcceptNewState(Ei, Ej, Wi, Wj);
/* 30*/                      else
/* 31*/                              ResetWeightState(Wi, Wsub_opt, Ei, Esub_opt);
/* 32*/              }
/* 33*/      }       // End While
/* 34*/ }
```

การทำงานของอัลกอริทึมที่ 3. เริ่มด้วยการอ่านไฟล์ค่าถ่วงน้ำหนักของโครงข่ายฟีดฟอร์เวิร์ดที่สอนด้วยแบ็กค์โพรพาเกชันเข้ามา โดยที่ไฟล์นี้ได้สอนโครงข่ายไปแล้วแต่ค่าความผิดพลาดยังไม่ลดลงมาถึงค่าที่ต้องการ ต่อมาในบรรทัดที่ 4 จะส่งค่าถ่วงน้ำหนักที่อ่านขึ้นมาแล้วให้กับฟังก์ชันฟีดฟอร์เวิร์ด เพื่อหาค่าความผิดพลาดแล้วส่งค่าให้กับ Ei ซึ่งเป็นค่าความผิดพลาด ณ ปัจจุบัน

ส่วนบรรทัดที่ 5-10 เป็นการกำหนดค่าเริ่มต้นให้กับตัวแปรต่างๆ

ในบรรทัดที่ 11 เป็นคำสั่ง while เพื่อวนซ้ำจนค่า Eopt น้อยกว่าค่าความผิดพลาดที่ตั้งไว้ (Ethreshold)

ในบรรทัดที่ 14 เป็นการสร้างค่าถ่วงน้ำหนักชุดใหม่จากค่าถ่วงน้ำหนักชุดเดิม โดยการรบกวนค่าถ่วงน้ำหนักชุดเดิม และในบรรทัดที่ 15 จะคำนวณค่าความผิดพลาดใหม่จากค่าถ่วงน้ำหนักชุดใหม่ ส่วนในบรรทัดที่ 16 เป็นการเปรียบเทียบค่าความผิดพลาดตัวใหม่ (Ej) กับค่าความผิดพลาดเก่า (Ei) ถ้าค่าความผิดพลาดใหม่น้อยกว่า ก็จะยอมรับค่าใหม่โดยการเซตค่า Ei = Ej; และ Wi = Wj ในฟังก์ชัน AcceptNewState ถ้าเงื่อนไขในบรรทัดที่ 16 ไม่เป็นจริงก็จะข้ามมาทำงานในบรรทัดที่ 24 คือเข้าสู่อัลกอริทึมเมโทรโพลิส เพื่อคำนวณค่าเกณฑ์ในการยอมรับค่าความผิดพลาดใหม่ ซึ่งในขั้นตอนนี้จะเริ่มจากบรรทัดที่ 24-32

5. การสอนโครงข่ายด้วยแบ็กค์โพรพาเกชันแอนนีลลิ่ง

ในอัลกอริทึมที่ 3. สามารถหาค่าถ่วงน้ำหนักที่เหมาะสมได้ แต่ยังมีปัญหาเรื่องเวลาที่ใช้ในการค้นหาค่าถ่วงน้ำหนักที่เหมาะสมซึ่งต้องใช้เวลานาน เราจึงได้พัฒนาต่อมาเป็นอัลกอริทึมแบ็กค์โพรพาเกชันแอนนีลลิ่งหรือนิวโรแอนนีลลิ่ง ที่ใช้เวลาในการหาคำตอบน้อยลง สำหรับสิ่งที่เราได้พัฒนาเพิ่มเข้าไปจากอัลกอริทึมที่ 3 คือเพิ่มฟังก์ชันแบกค์โพรพาเกชันเข้าไปที่ฟังก์ชัน ErrorFromFeedForward คือฟังก์ชันนี้จะมีการทำงานเหมือนกับขั้นตอนการสอนแบบ Error Back-propagation ในรูปที่ 7. และเพิ่มฟังก์ชันการรบกวนค่าถ่วงน้ำหนักโมเมนตัมเข้าไปอีกชุดหนึ่ง เพื่อส่งให้กับฟังก์ชันแบ็กค์โพรพาเกชัน ซึ่งขั้นตอนต่างๆ ได้แสดงไว้ในอัลกอริทึมที่ 4.

อัลกอริทึมที่ 4. แสดงขั้นตอนการหาค่าถ่วงน้ำหนักที่เหมาะสมให้กับโครงข่ายฟีดฟอร์เวิร์ดด้วยแบกค์โพรพาเกชันแอนนีลลิ่ง

```
// Feedforward Back-propagation Simulated Annealing;
/* 1  */ void FEEDFORWARD_BACKPROPAGATION_ANNEALING()
/* 2  */ {
```

```
/* 3  */    Wi = GetWeightFromBackpropagation();
/* 4 */     Initial_Weights(momentum, Wi.noOfRows, Wi.noOfCols);
/* 5  */    Ei = ErrorFromFeedforward(Wi, momentum);
/* 6  */    Eopt = Ei;
/* 7  */    Esub_opt = Ei;
/* 8  */    Wopt = Wi;
/* 9  */    Wsub_opt = Wi;
/* 10*/     l = 1;
/* 11*/     k = 0;
/* 12*/     while(Eopt > Ethreshold)
/* 13*/     {
/* 14*/             l++;
/* 15*/             Wj = Perturbation(Wi);
/* 16*/             if(Ei < Eopt)           //Perturbation momentum.
/* 17*/                     momentum = Wi;
/* 18*/             Ej = ErrorFromFeedforward(Wj, momentum);
/* 19*/             if(Ej <= Ei)
/* 20*/             {
/* 21*/                     AcceptNewState(Ei, Ej, Wi, Wj);
/* 22*/                     if(Ej < Esub_opt)
/* 23*/                             AcceptSub_optimalState(Esub_opt, Ej, Wsub_opt, Wj);
/* 24*/                     if(Esub_opt < Eopt)
/* 25*/                             AcceptOptimalState(Eopt, Esub_opt, Wopt, Wsup_opt);
/* 26*/             }
/* 27*/             else    //Metropolis Algorithm
/* 28*/             {
/* 29*/                     Pc = exp((Ej-Ei)/Ck);
/* 30*/                     Pr = UniformDistribution();
/* 31*/                     if(Pr < Pc)
/* 32*/                             AcceptNewState(Ei, Ej, Wi, Wj);
/* 33*/                     else
/* 34*/                             ResetWeightState(Wi, Wsub_opt, Ei, Esub_opt);
/* 35*/             }
/* 36*/     }       // End While
/* 37*/ }
```

6. การทดลองและวิจารณ์

การทดลองที่จะเสนอในหัวข้อนี้ จะแสดงให้เห็นถึงประสิทธิภาพของอัลกอริทึมที่เราได้พัฒนาขึ้น ซึ่งจุดประสงค์หลักของแบ็กค์โพรพาเกชันแอนนีลลิ่งคือ หาคำตอบจริง (Global Optimization) ให้กับระบบ ส่วนผลพลอยได้ในส่วนอื่นๆ เช่น ความเร็วในการสอน, การกำหนดจำนวนโหนดในหรือตัวแปรต่างๆ ที่ใช้ในสอนสามารถทำได้ง่ายขึ้น แต่อย่างไรก็ตามการกำหนดค่าตัวแปรต่างๆ ยังต้องอาศัยประสบการณ์และความชำนาญในระบบนั้นๆ อยู่บ้างพอสมควร แต่สำหรับระบบที่พัฒนาขึ้น เราเพียงกำหนดค่าตัวแปรขึ้นมาโดยดูจากคุณสมบัติของตัวแปรนั้นๆ ไม่ต้องใช้เวลาในการลองผิดลองถูกมากนัก เพราะแบ็กค์โพรพาเกชันแอนนีลลิ่งจะคอยปรับค่าถ่วงน้ำหนักของมันเองเพื่อหลีกเลี่ยง Local minimum อีกต่อหนึ่ง

สำหรับในการทดลองนี้เราจะเปรียบเทียบเรื่องเวลาที่ใช้ในการสอนโครงข่ายฟีดฟอร์เวิร์ด. แสดงกราฟค่าความผิดพลาดที่ลดลงในจำนวนรอบการสอนที่เท่ากัน และสุดท้ายจะเป็นการเปรียบเทียบประสิทธิภาพการรู้

จำของโครงข่าย ส่วนโครงข่ายที่ใช้ในการรู้จำลายมือเขียนที่เป็นตัวเลข ทุกอัลกอริทึมที่ใช้ในการทดลองจะกำหนดตัวแปรต่างๆ ของโครงข่ายเท่ากันหมด คือเป็นโครงข่ายฟีดฟอร์เวิร์ดที่มี 3 ชั้น ส่วนข้อกำหนดอื่นๆ ได้แสดงไว้ในตารางที่ 2. สำหรับการเปรียบเทียบเวลาที่ใช้สอนโครงข่ายได้แสดงไว้ในตารางที่ 3.

ตารางที่ 2. แสดงข้อกำหนดต่างๆ ของโครงข่ายที่ใช้ในการรู้จำตัวเลขที่เป็นลายมือเขียน

ตัวแปร	ค่าของตัวแปร
จำนวนโหนดในชั้นอินพุท	100
จำนวนโหนดในชั้นฮิดเดน	20
จำนวนโหนดในชั้นเอาท์พุท	10
โมเมนตัม	0.8
อัตราเร็วในการเรียนรู้	1.5

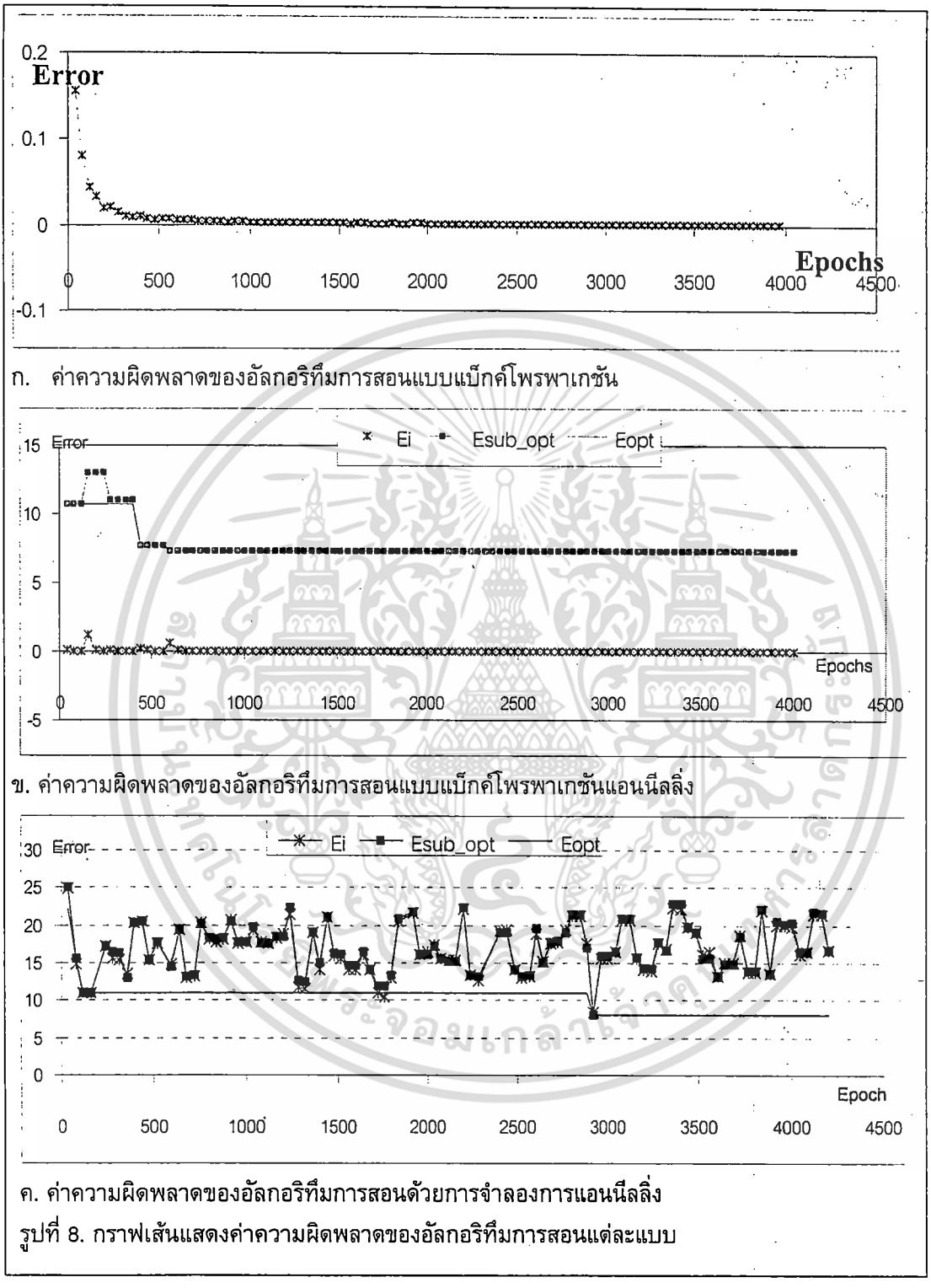
ตารางที่ 3. แสดงเวลาที่ใช้สอนโครงข่ายด้วยอัลกอริทึมต่างๆ

วิธีการสอน	เวลา(วินาที)	ค่าความผิดพลาด	จำนวนรอบที่สอน
แบ็กค์โพรพาเกชัน	1729.05	0.000066	4000
แบ็กค์โพรพาเกชันแอนนีลลิ่ง	161.53	0.000060	3245
การจำลองการแอนนีลลิ่ง	2547.01	0.081579	80000

จากตารางที่ 3. อัลกอริทึมที่สามารถตั้งค่าตัวแปรต่างๆ ให้สามารถเปรียบเทียบกันได้ในเรื่องเวลาคือวิธีการสอนแบบแบ็กค์โพรพาเกชันกับ แบ็กค์โพรพาเกชันแอนนีลลิ่ง คือสามารถกำหนดจำนวนรอบที่ใช้สอนและค่าความผิดพลาดที่ต้องการ ซึ่งในที่นี้เราให้ความสำคัญกับค่าความผิดพลาดมากกว่า คือเมื่ออัลกอริทึมการสอนสามารถลดค่าความผิดพลาดลงมาถึงค่าที่ต้องการก็จะให้หยุดการสอน และไปดูจำนวนรอบที่ใช้สอน ซึ่งเมื่อเปรียบเทียบเวลาที่ใช้ปรากฏว่า การสอนแบบแบ็กค์โพรพาเกชันแอนนีลลิ่งเร็วกว่ามาก คือโดยเฉลี่ยจากการทดลอง 4 ครั้ง วิธีการสอนแบบแบ็กค์โพรพาเกชันแอนนีลลิ่งจะเร็วกว่าการสอนแบบแบ็กค์โพรพาเกชันประมาณ 10 เท่า ที่จำนวนรอบการสอนใกล้เคียงกันแต่เวลาต่างกันมาก เนื่องจากแบ็กค์โพรพาเกชันแอนนีลลิ่งมีการคำนวณที่น้อยกว่า ส่วนการสอนด้วยอัลกอริทึมการแอนนีลลิ่งจะใช้เวลามาก เนื่องจากวิธีการของอัลกอริทึมเองที่ต้องการคำตอบที่เป็น Global แต่มันขาดตัวชี้นำ ซึ่งเมื่อเราใช้แบ็กค์โพรพาเกชันเป็นตัวชี้นำ มันสามารถลดเวลาการสอนลงได้เป็นอย่างมาก และยังสามารถค้นหาคำตอบที่เข้าใกล้ Global optimization ด้วย

ในรูปที่ 8. แสดงกราฟเส้นของค่าความผิดพลาดที่ลดลงมา เมื่อจำนวนรอบการสอนมีมากขึ้น ซึ่งในกราฟจะแสดงค่าความผิดพลาดที่สอนประมาณ 4000 รอบ อย่างในรูปที่ 8ก. เป็นค่าความผิดพลาดที่ลดลงมาเรื่องๆ จนถึงประมาณรอบที่ 500 ค่านี้ก็จะมีการเปลี่ยนแปลงน้อยมาก เนื่องจากไปติดอยู่ที่ค่า Local minimum สำหรับรูปที่ 8ข และ 8ค. เป็นการสอนด้วยนิวโรแอนนีลลิ่ง และการจำลองการแอนนีลลิ่ง ตามลำดับ ค่าความผิดพลาดของทั้งสองวิธีนี้ที่พอจะเปรียบเทียบกันได้กับค่าความผิดพลาดของแบ็กค์โพรพาเกชันคือ ค่าความผิดพลาด ณ ปัจจุบันหรือ Ei จะเห็นว่าค่า Ei ของนิวโรแอนนีลลิ่งมีจะค่าแกว่งในช่วงแรก และมีค่าเป็นศูนย์ไปในที่สุด ซึ่งตรงจุดนี้หมายความว่าโครงข่ายสามารถแม็ปอินพุทไปยังเอาท์พุทได้ถูกต้องทั้งหมด หรืออีกนัยหนึ่งก็คือ

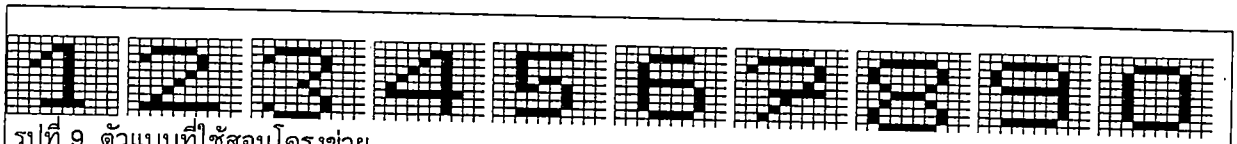
นิวโรแอนนีลลิ่งสามารถค้นหา Global minimum ได้นั่นเอง ส่วนการสอนโครงข่ายด้วยการจำลองการแอนนีลลิ่ง ค่า Ei จะแกว่งอยู่เป็นช่วงๆ แล้วค่อยๆ ลดลง ที่เป็นเช่นนี้เนื่องจากการรีเซตค่าถ่วงน้ำหนักในแต่ละครั้ง

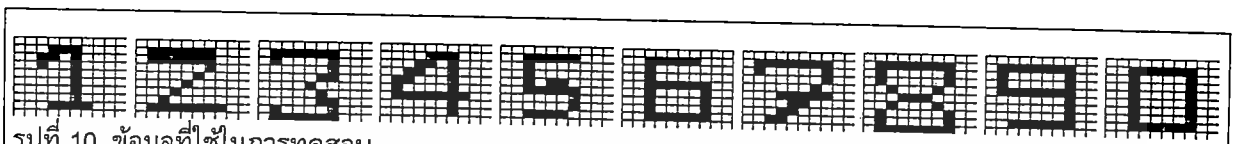


รูปที่ 8. กราฟเส้นแสดงค่าความผิดพลาดของอัลกอริทึมการสอนแต่ละแบบ

ในการเปรียบเทียบประสิทธิภาพของการรู้จำตัวเลขที่เป็นลายมือเขียน ของแต่ละอัลกอริทึมที่ใช้สอนโครงข่าย ในส่วนของนิวโรแอนนีลลิ่งกับแบ็กค์โพรพาเกชันจะมีศักย์ภาพในการรู้จำใกล้เคียงกัน เนื่องจากขั้นตอนการสอนและการปรับค่าถ่วงน้ำหนักมีส่วนที่คล้ายกัน แต่โครงข่ายที่สอนด้วยการจำลองแอนนีลลิ่งที่ใช้วิธี

การค้นหาค่าถ่วงน้ำหนักที่เหมาะแทนการปรับค่าถ่วงน้ำหนักอย่างวิธีแบ็กโพรพาเกชัน จะมีประสิทธิภาพในการรู้จำด้อยกว่าสองวิธีข้างต้น


รูปที่ 9. ตัวแบบที่ใช้สอนโครงข่าย


รูปที่ 10. ข้อมูลที่ใช้ในการทดสอบ

ในการทดลองเราได้สร้างข้อมูลสำหรับสอนโครงข่ายขึ้นมาชุดหนึ่งที่เป็นตัวเลข 0-9 ซึ่งแสดงในรูปที่ 9. เมื่อเราใช้ข้อมูลชุดนี้สอนโครงข่ายทั้งสามอัลกอริทึม แล้วใช้ข้อมูลชุดเดิมทดสอบ ปรากฏว่าแต่ละอัลกอริทึมสามารถรู้จำตัวอักษรได้ทั้งหมด แต่เมื่อนำข้อมูลชุดเดิมมาเพิ่มสัญญาณรบกวนเข้าไป ซึ่งข้อมูลชุดนี้ได้แสดงไว้ในรูปที่ 10. เมื่อนำข้อมูลที่มีสัญญาณรบกวนไปทดสอบปรากฏว่า นิวโรแอนนิวลิ่งกับแบ็กค์โพรพาเกชันยังสามารถรู้จำตัวเลยได้ทั้งหมด แต่โครงข่ายที่สอนด้วยการจำลองแอนนีลลิ่งไม่สามารถแยกแยะระหว่าง 6 กับ 9 และ 3 กับ 8 ได้

6. สรุป

อัลกอริทึมการสอนโครงข่ายนิวโรแอนนีลลิ่งนี้ สามารถรู้จำตัวเลขที่เป็นลายมือเขียนได้เป็นอย่างดี เช่นเดียวกันกับแบ็กค์โพรพาเกชัน แต่วิธีที่นำเสนอใช้เวลาในการสอนที่น้อยกว่า การกำหนดจำนวนโหนดในชั้นฮิดเดนและการกำหนดค่าตัวแปรต่างๆ ที่ใช้สอนโครงข่ายสามารถทำได้ง่ายขึ้น คือไม่ต้องใช้เวลาในการลองผิดลองถูกกับการกำหนดค่าเหล่านี้นานเกินไป เพราะในนิวโรแอนนีลลิ่งจะมีฟังก์ชันการรบกวนค่าถ่วงน้ำหนักเพื่อหลีกเลี่ยงปัญหา Local minimum อย่างได้ผล ที่สำคัญคือหลักการที่นำเสนอสามารถประยุกต์ใช้กับตัวอักษรชุดอื่นๆ ได้ โดยเปลี่ยนชุดอักษรใหม่ในการสอน ไม่ต้องเขียนโปรแกรมทั้งหมดใหม่

เอกสารอ้างอิง

[1]. J. M. Zurada, *Introduction to Artificial Neural Systems,* West Publishing Company, Singapore, 1992.

[2]. J. A. Freeman, and D. M. Skapara, *Neural Networks, Applications, and Programming Techniques,* Addison Wesley, USA., 1991.

[3]. D. E. Rumelhart, and J. L. McClelland, *Exploration in Parallel Distributed Processing; A Handbook of Model Programs; and Exercises*, MIT Press, Massachusette, USA., 1988.

[4]. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing; Exploration in the Microstructure of Cognition*, Volume 1: Foundation, MIT Press, Massachusette, USA., 1986.

[5]. A. A. Kempka, "Activating Neural Networks : Part II," AI Expert, Aug., 1994.

[6]. กิตติ ไพฑูรย์วัฒนกิจ สาธิต อินทจักร์ และ ไพลิน บุญเดช, "นิวรอลเน็ตเวิร์คบนทรานสพิวเตอร์กับการหาขอบภาพ," ประชุมใหญ่วิชาการทางวิศวกรรมประจำปี 2537, วิศวกรรมสถานแห่งประเทศไทยฯ, (2537) หน้า EE 75-90.

[7]. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, USA., 1989.

[8]. กิตติ ไพฑูรย์วัฒนกิจ และ เศรษฐพล ลิ้มปราชญา, "การใช้นิวรอลเน็ตเวิร์คเพื่อจดจำแพทเทิร์น," วารสารสำนักงานคณะกรรมการวิจัยแห่งชาติ, ปีที่ 25, เล่มที่ 1, (ม.ค.-มิ.ย. 2536) หน้า 45-67.

[9]. J. E. Dayhoff, *Neural Network Architectures an Introduction*, Van Nostrand Reinhold, 1990.

[10]. E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, Great Britain, 1989.