

## บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

### 2.1 ระบบโครงข่ายประสาทเทียม (Neural Networks)

หลายกิจกรรมในชีวิตประจำวันของเราเกี่ยวข้องกับงานปัญญาประดิษฐ์ หรือการรู้จำ ซึ่งปัญหาการรู้จำเป็นปัญหาที่มีความยากและมีความซับซ้อนมากถ้าจะพัฒนาให้เป็นระบบอัตโนมัติในเครื่องคอมพิวเตอร์ แต่ในทางตรงกันข้ามงานดังกล่าวนี้กลับสามารถดำเนินการได้โดยง่ายโดยมนุษย์ กล่าวคือ มนุษย์สามารถรู้จำหรือแยกแยะวัตถุต่างๆ ได้อย่างมากมาย ทั้งที่วัตถุดังกล่าวนั้นกำลังอยู่ในสภาวะแวดล้อมที่มีความหลากหลาย ยกตัวอย่างเช่น นายขาวจะมีความสามารถในการจำเสียงของนายดำ และไม่ว่าในคำจะโทรศัพท์มาคุยกับนายขาวจากสภาพแวดล้อมใดๆ ก็ตาม นายขาวยังคงจำเสียงนายดำได้เสมอ เป็นต้น ดังนั้น จึงเป็นเหตุผลที่สำคัญที่เราต้องพัฒนาระบบการคำนวณ (Computing system) ที่สามารถเข้าใจและเลียนแบบการทำงานของมนุษย์ระบบดังกล่าวนี้ เราเรียกว่าระบบโครงข่ายประสาท (Neural Networks)

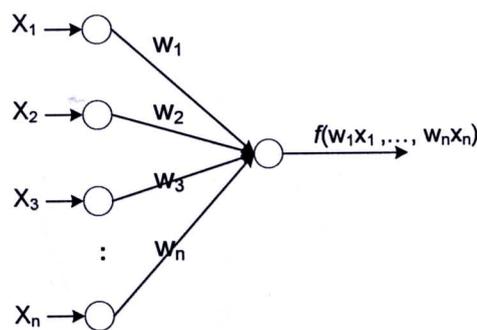
ระบบโครงข่ายประสาทเทียม (Artificial Neural Networks: ANN) หรือ เรียกสั้น ๆ ว่าระบบโครงข่ายประสาท (Neural Networks) เกิดมาจากแรงบันดาลใจเกี่ยวกับความต้องการเลียนแบบการทำงานของสมองมนุษย์ด้วยเครื่องคอมพิวเตอร์ เพื่อสร้างระบบคอมพิวเตอร์แบบใหม่ที่มีความสามารถในการเรียนรู้ด้วยตนเอง ซึ่งแตกต่างจากระบบคอมพิวเตอร์ในปัจจุบัน ที่ต้องทำงานตามชุดคำสั่งที่มนุษย์เขียนสั่งไว้ล่วงหน้า (Software/Program) เท่านั้น และ ก็เป็นที่ทราบทั่วไปว่าการประมวลผลของสมองมนุษย์เรานั้น มีความซับซ้อน มีความไม่เป็นเชิงเส้น และ เป็นแบบขนานอย่างมาก นอกจากนี้ สมองมนุษย์เรานั้นยังมีความสามารถทางการคำนวณ (เช่น การรู้จำ การรับรู้ และ การสั่งการในการควบคุมเครื่องจักร) ได้อย่างรวดเร็วกว่าเครื่องคอมพิวเตอร์ที่เรามีใช้อยู่ในปัจจุบันเราเป็นอย่างมาก ยกตัวอย่างเช่น การมองเห็นของมนุษย์ ซึ่งถือว่าเป็นการประมวลผลสารสนเทศแบบหนึ่ง ในระบบการมองเห็นของมนุษย์เรานั้น จะประกอบด้วยขั้นตอนการแทนข้อมูลสภาพแวดล้อมต่างๆ ที่ได้รับ (Data Representation) และ รวมไปถึงความสามารถในการติดต่อกับสภาพแวดล้อมต่างๆ เหล่านั้นด้วย กล่าวคือ ทันทีที่เรามองเห็น เราจะสามารถแทนข้อมูลที่เรามองเห็นได้แทบจะทันที และหลังจากนั้นเราจะยังสามารถจะรู้จำสภาพแวดล้อมต่างๆ นั้นได้ (การรู้จำ หมายถึง ความสามารถในการจดจำ และ บรรยายสิ่งต่างๆ ที่มองเห็นให้กับผู้อื่นได้ทราบ) กล่าวกันว่า สมองมนุษย์เรานั้น มีความสามารถในการรู้จำสิ่งต่างๆ ที่มองเห็นภายในเวลาประมาณ 100 - 200 มิลลิวินาที ซึ่งไม่มีเครื่องคอมพิวเตอร์ใดสามารถทำได้

แบบจำลองอย่างง่ายของระบบโครงข่ายประสาทเทียมจะมีความคล้ายกับโครงสร้างทางชีววิทยาทางสมองของมนุษย์อย่างมาก ดังแสดงในตารางที่ 2-1 และ รูปที่ 2-1 โดยมีข้อกำหนดเบื้องต้น ดังนี้

1. ตำแหน่งของค่าน้ำหนักบนโครงข่ายไม่มีความสัมพันธ์กัน
2. คำตอบของแต่ละโหนดจะมีเพียงค่าเดียว ซึ่งจะกระจายไปยังโหนดต่างๆ ที่มีการเชื่อมโยงถึง โดยตำแหน่งของการเชื่อมก็ไม่ให้มีผลเช่นเดียวกัน
3. ข้อมูลที่เข้ามาถึงแต่ละโหนดในเวลาเดียวกันนั้นจะต้องคงสถานะเดิมไปจนกว่าการคำนวณของฟังก์ชัน  $f(w_1x_1, \dots, w_nx_n)$  จะเสร็จสิ้นลง

ตารางที่ 2-1 คำศัพท์เฉพาะเพื่อเทียบเคียง (ที่มา : K. Mehrotra et al., Element of Artificial Neural Network)

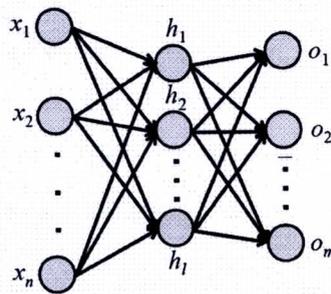
คำศัพท์เฉพาะชีววิทยาทางสมองของมนุษย์	คำศัพท์เฉพาะของโครงข่ายประสาทเทียม
Neuron	Node/Unit/Cell/Neurode
Synapse	Connection/Edge/Link
Synaptic Efficiency	Connection Strength/Weight
Firing Frequency	Node Output



รูปที่ 2-1 ระบบโครงข่ายประสาทเทียมอย่างง่าย

### 2.1.1 ระบบโครงข่ายประสาทเทียมแบบวิธีการแพร่กระจายย้อนกลับ

ขั้นตอนวิธีการแพร่กระจายย้อนกลับ (Back Propagation Algorithm) เป็นขั้นตอนวิธีที่ใช้ในการเรียนรู้ของเครือข่ายประสาทเทียมวิธีหนึ่งที่นิยมใช้ในโครงข่ายประสาทเทียมหลายชั้น (Multilayer Neural Networks) เพื่อใช้ในการปรับค่าน้ำหนักในเส้นเชื่อมต่อระหว่างโหนดให้เหมาะสม โดยการปรับค่านี้อาจขึ้นกับความแตกต่างของค่าเอาต์พุตที่คำนวณได้กับค่าเอาต์พุตที่ต้องการ พิจารณารูปต่อไปนี้ประกอบ



รูปที่ 2-2 ตัวอย่างข่ายงานประสาทเทียมแบบหลายชั้น

ตัวอย่างในรูปด้านบนแสดงข่ายงานป้อนไปหน้าแบบหลายชั้นซึ่งประกอบไปด้วยชั้นอินพุต ชั้นฮิดเดนหรือชั้นซ่อน และชั้นเอาต์พุต ในรูปแสดงชั้นฮิดเดนเพียงชั้นเดียวแต่อาจมีมากกว่าหนึ่งชั้นก็ได้ เส้นเชื่อมจะเชื่อมต่อเป็นชั้น ๆ ไม่ข้ามชั้นจากชั้นอินพุตไปชั้นฮิดเดน ถ้ามีชั้นฮิดเดนมากกว่าหนึ่งชั้นก็เชื่อมต่อกันไป และสุดท้ายจากชั้นฮิดเดนไปชั้นเอาต์พุต

ในการปรับค่าน้ำหนักโดยขั้นตอนวิธีการแพร่กระจายย้อนกลับนั้น เราต้องนิยามค่าผิดพลาดสำหรับการเรียนรู้ของข่ายงาน  $MSE(\vec{w})$  จากนั้นจะหาค่าน้ำหนักที่ให้ค่าผิดพลาดต่ำสุด นิยามค่าผิดพลาดดังนี้

$$MSE(\vec{w}) = \frac{1}{2} \sum_{p \in P} \sum_{k \in \text{outputs}} (d_{p,k} - o_{p,k})^2 \quad \dots(1)$$

โดยที่  $outputs$  คือ เซตของเอาต์พุตโหนดในข่ายงานประสาทเทียม  $d_{p,k}$  และ  $o_{p,k}$  เป็นค่าเอาต์พุตเป้าหมายและเอาต์พุตที่ได้จากข่ายงานประสาทเทียมตามลำดับของเอาต์พุตโหนดที่  $k$  ของตัวอย่างที่  $p$  ขั้นตอนการแพร่กระจายย้อนกลับจะค้นหาค่าน้ำหนักที่ให้ค่าผิดพลาดกำลังสองเฉลี่ยต่ำสุด

ขั้นตอนของ Back-propagation Algorithm มีดังนี้

**Algorithm Backpropagation;**

Start with randomly chosen weights;

**while** MSE is unsatisfactory

and computational bounds are not exceeded, **do**

for each input pattern  $x_p$ ,  $1 \leq p \leq P$ ,

    Compute hidden node inputs ( $net_{p,j}^{(1)}$ );

    Compute hidden node outputs ( $x_{p,j}^{(1)}$ );

    Compute inputs to the output nodes ( $net_{p,k}^{(2)}$ );

    Compute the network outputs ( $o_{p,k}$ );

    Modify outer layer weights:

$$\Delta w_{k,j}^{(2,1)} = \eta (d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) x_{p,j}^{(1)}$$

    Modify weights between input & hidden nodes:

$$\Delta w_{j,i}^{(1,0)} = \eta \sum_k \left( (d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) w_{k,j}^{(2,1)} \right) \mathcal{S}'(net_{p,j}^{(1)}) x_{p,i}$$

**end-for**

**end-while.**

Note: if  $S$  is a logistic function, then

$$S'(x) = S(x)(1 - S(x))$$

## 2.2 การเรียนรู้ด้วยระบบโครงข่ายประสาทเทียมแบบ

การเรียนรู้ด้วยระบบโครงข่ายประสาทเทียมแบบผสม ผสม (Neural Network Ensemble Learning) เป็นวิธีการเรียนรู้ที่นำผลลัพธ์จากการใช้วิธีการเรียนรู้หลายๆ แบบมารวมกัน เพื่อเพิ่มความแม่นยำของผลลัพธ์ ซึ่งมักจะนำมาใช้ในการเพิ่มประสิทธิภาพของการจำแนกข้อมูล การพยากรณ์ผลลัพธ์ การประมาณค่าของฟังก์ชัน เป็นต้น การรวมผลลัพธ์ที่ได้มาจากขั้นตอนวิธีการเรียนรู้ที่มากกว่า 1 แบบ ทำให้เพิ่มความถูกต้องว่าผลการตัดสินใจจากขั้นตอนวิธีเหล่านั้นให้ความมั่นใจได้มากขึ้น ตัวอย่างเช่น ในปัญหาการจำแนกข้อมูล มีงานวิจัยจำนวนมากที่นำเสนอขั้นตอนวิธีในการแก้ปัญหา ในการใช้อัลกอริทึมที่ต่างกัน ผลลัพธ์ที่ได้จากการเรียนรู้ในแต่ละครั้งก็มักจะให้ผลลัพธ์ที่แตกต่างกันด้วย ทำให้เกิดปัญหาตามมาว่า ถ้าในกรณีที่ตัวเรียนรู้แต่ละตัวให้ผลไม่ตรงกัน ควรจะเลือกผลลัพธ์ใดไปเป็นคำตอบ

จากปัญหาดังกล่าว แนวความคิดในการสร้างวิธีเรียนรู้แบบ Ensemble จึงเกิดขึ้นเพื่อเพิ่มประสิทธิภาพของคำตอบด้วยการรวมผลลัพธ์จากหลายๆ การเรียนรู้ ซึ่งขั้นตอนหลักของการเรียนรู้แบบ ensemble จะแบ่งเป็น 2 ขั้นตอนคือ

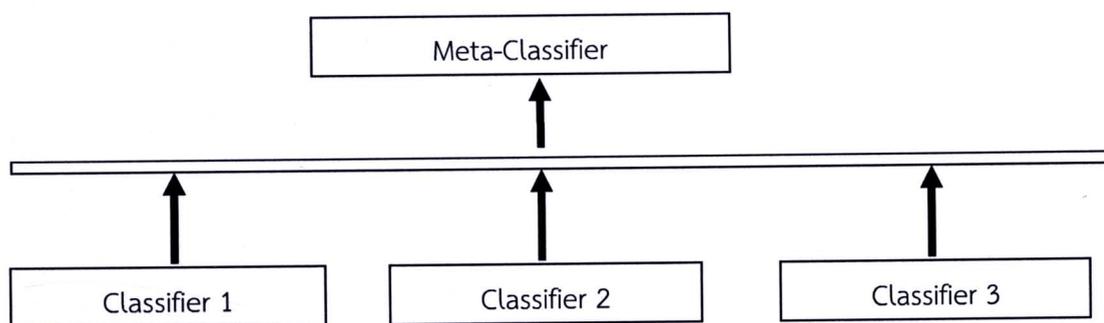
1. ทำการเรียนรู้โดยใช้ตัวเรียนรู้หลายๆ ตัว ซึ่งตัวเรียนรู้เหล่านี้อาจจะเป็นอัลกอริทึมเดียวกันที่ใช้ parameter ต่างกัน หรือว่าจะเป็นคนละอัลกอริทึมกันก็ได้
2. นำผลลัพธ์ที่ได้จากการเรียนรู้ที่มาจากหลายๆ แบบมารวมกัน แล้วทำให้จากหลายผลลัพธ์กลายเป็นคำตอบเดียวที่เหมาะสมที่สุด ซึ่งวิธีการรวมก็สามารถทำได้หลายวิธี เช่น อาจจะใช้การเฉลี่ยหรือการหาคะแนนนิยม เป็นต้น

### ตัวอย่างของวิธีการเรียนรู้แบบผสม

1. Boosting (Schapire, 1990) เป็นการทำ ensemble โดยใช้ผลลัพธ์จากตัวเรียนรู้ตัวที่ให้ผลลัพธ์ยังไม่ค่อยดีเพื่อสร้างตัวเรียนรู้ที่ดีกว่า โดยการให้น้ำหนักของผลลัพธ์ที่เรียนรู้ได้ แล้วส่งให้ตัวเรียนรู้ตัวใหม่ใช้ข้อมูลนั้นในการเพิ่มประสิทธิภาพในการเรียนรู้ต่อไป โดยค่าน้ำหนักนี้จะเพิ่มขึ้นถ้าตัวเรียนรู้ไม่สามารถให้ผลลัพธ์ที่ถูกต้องได้ และค่าน้ำหนักจะลดลงถ้าตัวเรียนรู้สามารถเรียนรู้ข้อมูลที่ถูกต้องได้ถูกต้อง สาเหตุที่ต้องมีการกำหนดค่าน้ำหนักเช่นนี้เนื่องจากวิธีนี้ต้องการมุ่งเน้นไปที่การเรียนรู้ตัวอย่างยากที่ไม่สามารถให้ผลลัพธ์ที่ถูกต้องได้จากการเรียนรู้ในครั้งก่อนหน้า ขั้นตอนการเรียนรู้แบบนี้จะทำซ้ำหลายๆ รอบจนได้ตัวเรียนรู้ที่ให้ผลลัพธ์ที่ดีที่สุดออกมา วิธีการนี้เป็นวิธีการที่เสนอมาในยุคแรกๆ และเป็นที่ยอมรับใช้มาจนถึงทุกวันนี้ โดยอัลกอริทึมที่ถูกพัฒนาขึ้นมาในภายหลังและเป็นที่ยอมรับใช้กันในวงกว้างคือ อัลกอริทึม AdaBoost หรือ Adaptive boosting (Freund and Schapire, 1996)

2. Bagging (Breiman, 1996) วิธีการ Bagging หรือ Bootstrap Aggregating จะใช้ข้อมูลสอนหลายๆ ชุดที่มาจากการสุ่มแบบใส่กลับคืน เพื่อสร้างชุดข้อมูลสอนที่แตกต่างกัน และข้อมูลสอนเหล่านี้จะถูกนำมาใช้ในการเรียนรู้ของตัวเรียนรู้แต่ละตัวแล้วจะนำผลลัพธ์ที่ได้มาจากตัวเรียนรู้แต่ละตัว ซึ่งผลลัพธ์อาจจะได้ออกมาเหมือนหรือต่างกัน มาทำการเฉลี่ยหรือ vote โดยกำหนดค่าน้ำหนักของผลลัพธ์ที่มาจากแต่ละตัวเรียนรู้ด้วยน้ำหนักที่เท่ากัน

3. Stacked generalization (Wolpert, 1992) เป็นวิธีการที่นำหลักการของ Meta Learner เข้ามาใช้ โดยการแบ่งตัวเรียนรู้ออกเป็นลำดับชั้น ตัวเรียนรู้ที่เป็น Master Model จะทำหน้าที่เป็นตัวตัดสินใจเพื่อให้คำตอบสุดท้ายออกมา ซึ่งการตัดสินใจนี้จะขึ้นอยู่กับผลลัพธ์จากตัวเรียนรู้ที่เหลือ ซึ่งลักษณะการทำงานแสดงดังรูปที่ 2-3



รูปที่ 2-3 การทำงานของการเรียนรู้แบบผสมแบบ Stacked generalization

### การประมาณต้นทุนการพัฒนาซอฟต์แวร์ (Estimating Software Development Costs)

วิธีการในการประมาณมูลค่าซอฟต์แวร์แบ่งออกเป็นสองชนิดหลักได้แก่ วิธีการแบบ Non-algorithmic ซึ่งหมายถึงการวิธีการคิดที่ไม่สามารถระบุเป็นขั้นตอนวิธีได้อย่างชัดเจน และ วิธีการแบบ Algorithmic ซึ่งสามารถระบุขั้นตอนวิธีได้อย่างชัดเจน โดยสร้างเป็นตัวแบบอัลกอริทึมที่มีพื้นฐานในการใช้สูตรทางคณิตศาสตร์ บางตัวแบบใช้วิธีการคำนวณเชิงสถิติ เช่น การคำนวณหาค่าเฉลี่ยและค่าเบี่ยงเบนมาตรฐาน และบางตัวแบบมีพื้นฐานมาจากตัวแบบการถดถอย สมการอนุพันธ์ เป็นต้น ตัวแบบเหล่านี้ไม่สามารถนำมาใช้งานได้ทันทีแต่จำเป็นต้องมีการปรับค่าและประยุกต์ให้เข้ากับสถานการณ์เฉพาะที่เพื่อเพิ่มความเที่ยงตรงของตัวแบบ

#### Non-algorithmic Methods

วิธีนี้ใช้ข้อมูลและความรู้จากผู้เชี่ยวชาญในการประมาณมูลค่าซอฟต์แวร์ ตัวแบบการประมาณมูลค่าซอฟต์แวร์ โดยไม่มีการนำตัวแบบเชิงคณิตศาสตร์มาเป็นพื้นฐานแต่อย่างใด วิธีการประมาณมูลค่าซอฟต์แวร์ที่จัดอยู่ในกลุ่มนี้ได้แก่

**วิธี Analogy Costing** วิธีนี้จำเป็นต้องมีโครงการที่สมบูรณ์ก่อนหน้าอย่างน้อย 1 โครงการที่มีลักษณะคล้ายคลึงกับโครงการใหม่เพื่อใช้ในการหาค่าใช้จ่ายที่เกิดขึ้นจริง การประมาณค่าใช้จ่ายโดยอาศัยความคล้ายคลึงสามารถทำได้ทั้งในระดับภาพรวมหรือส่วนใดส่วนหนึ่งของโครงการ ระดับภาพรวมมีประโยชน์ คือ องค์กรประกอบทุกด้านของระบบถูกนำเข้ามาพิจารณา ขณะที่ระดับย่อยมีประโยชน์ในการให้รายละเอียดในความคล้ายคลึงและความแตกต่างระหว่างโครงการใหม่กับโครงการเก่าได้ดีกว่า จุดเด่นของวิธีนี้คือ การประมาณอยู่บนพื้นฐานของประสบการณ์ที่เคยเกิดขึ้นจริง อย่างไรก็ตาม โครงการที่เดิมเคยทำมาแล้วมักจะไม่ชัดเจนพอที่จะใช้เป็นตัวแทนของเงื่อนไข สภาพแวดล้อม และการทำงานต่าง ๆ ของโครงการใหม่ทั้งหมด

**วิธี Expert Judgment** วิธีนี้เกี่ยวข้องกับผู้เชี่ยวชาญอย่างน้อยหนึ่งคนขึ้นไป ผู้เชี่ยวชาญจะใช้วิธีและประสบการณ์ของตนเองในการประมาณค่า กลไกในการดำเนินการของวิธีการนี้ไม่ว่าจะเป็นวิธีการแบบ Delphi หรือ หลักการแบบ PERT จะใช้ในการแก้ปัญหาความไม่แน่นอนในการประมาณค่า

วิธีการ Delphi มีขั้นตอนโดยย่อดังนี้

1. ผู้ประสานงานส่งมอบข้อมูลทางเทคนิคและแบบฟอร์มให้แก่ผู้เชี่ยวชาญแต่ละคนเพื่อบันทึกการประมาณค่า
2. ผู้เชี่ยวชาญแต่ละคนกรอกข้อมูลลงในแบบฟอร์มโดยอิสระหากมีข้อสงสัยผู้เชี่ยวชาญสามารถถามรายละเอียดกับผู้ประสานงานได้
3. ผู้ประสานงานทำการสรุปข้อมูลการประมาณทั้งหมดที่ได้มาจากผู้เชี่ยวชาญ (ด้วยวิธีหาค่ากลาง เช่น ค่าเฉลี่ย หรือ ค่ามัธยฐาน) ลงบนแบบฟอร์มเพื่อให้ผู้เชี่ยวชาญแต่ละคนประมาณค่าอีกครั้งและให้เหตุผลประกอบ
4. ทำซ้ำขั้นตอนที่ 2 และ 3 หลาย ๆ ครั้งตามความเหมาะสม กล่าวคือ ค่าที่ได้ค่อนข้างคงที่ หรือมีการเปลี่ยนแปลงค่าเพียงเล็กน้อย

Boehm และ Fahquhar (Boehm, 1981) เป็นผู้เสนอให้มีการปรับปรุงแก้ไขวิธี Delphi ให้มีประสิทธิภาพมากขึ้น โดยก่อนการประมาณค่า ผู้ประสานงานและผู้เชี่ยวชาญต้องเข้าร่วมประชุมเพื่ออภิปรายประเด็นต่างๆของการประมาณค่า และในขั้นตอนที่ 3 ผู้เชี่ยวชาญไม่จำเป็นต้องให้เหตุผลของการประมาณ แต่จะมีการนัดประชุมเพื่อให้ผู้เชี่ยวชาญอภิปรายถึงประเด็นต่างๆในกรณีที่มีความแตกต่างของการประมาณค่าเกิดขึ้น

**วิธี Parkinson** ในวิธีนี้มีแนวคิดที่ว่า “งานจะถูกขยายออกไปจนเต็มความสามารถของทรัพยากรที่มีอยู่” มูลค่าจะถูกกำหนด (ไม่ใช่เกิดจากการประมาณค่า) โดยทรัพยากรที่มีอยู่ทั้งหมด เช่น ถ้าซอฟต์แวร์ต้องส่งมอบภายใน 12 เดือน และมีบุคลากร 5 คน แต่ผลที่ได้จะถูกประมาณค่าเป็น 60 คน-เดือน ซึ่งในกรณีนี้ไม่สามารถเกิดขึ้นได้จริง อย่างไรก็ตามแม้ว่าบางครั้งการประมาณค่าจะให้ผลการประมาณค่าที่ดี แต่ไม่แนะนำให้ใช้เพราะอาจจะทำให้การประมาณค่าไม่สมจริง และไม่เป็นวิธีการที่ดีในทางปฏิบัติในแง่ของวิศวกรรมซอฟต์แวร์

**วิธี Price-to-win** มูลค่าซอฟต์แวร์ถูกประมาณค่าด้วยราคาที่ดีที่สุด การประมาณค่าขึ้นอยู่กับประมาณของลูกค้ามากกว่าฟังก์ชันการทำงานของซอฟต์แวร์ ตัวอย่างเช่น ถ้าการประมาณค่าที่สมเหตุสมผลสำหรับโครงการหนึ่งมีมูลค่าเท่ากับ 100 คน-เดือนแต่ลูกค้าสามารถจ่ายได้ 60 คน-เดือน ผู้ประมาณค่าจะทำการปรับค่าการประมาณค่าให้เท่ากับ 60 คน-เดือน เพื่อให้ได้รับงานนี้ ในทาง

ปฏิบัติแล้ววิธีการนี้ไม่ใช่วิธีการที่ดีเนื่องจากอาจเป็นเหตุให้เกิดความล่าช้าในการส่งมอบหรือทำให้ทีมงานพัฒนาต้องทำงานล่วงเวลามากขึ้น

**วิธี Bottom-up** วิธีการนี้แต่ละองค์ประกอบของระบบซอฟต์แวร์จะถูกทำการประมาณค่าแยกกันและผลลัพธ์ของแต่ละส่วนจะถูกนำมารวมเพื่อทำการประมาณค่าระบบทั้งหมด หลักการที่สำคัญของวิธีการนี้ คือ จะต้องมีการออกแบบเบื้องต้นไว้แล้ว เพื่อเป็นตัวบ่งชี้ว่าระบบจะถูกแบ่งออกเป็นองค์ประกอบย่อยที่แตกต่างกันได้อย่างไรบ้าง

**วิธี Top-down** วิธีการนี้ตรงข้ามกับวิธี Bottom-up การประมาณมูลค่าทั้งหมดของระบบได้มาจากคุณสมบัติโดยรวม มูลค่ารวมสามารถแบ่งย่อยไปยังองค์ประกอบย่อยได้ วิธีการนี้เหมาะสมสำหรับการประมาณมูลค่าในระยะเริ่มต้นของโครงการ

### Algorithmic methods

วิธีนี้มีพื้นฐานจากตัวแบบเชิงคณิตศาสตร์ซึ่งทำการประมาณค่าด้วยฟังก์ชันของตัวแปรต่างๆ ที่เป็นปัจจัยหลักของมูลค่าที่จะเกิดขึ้น ตัวแบบอัลกอริทึมจะอยู่ในรูปของ

$$Effort = f(x_1, x_2, \dots, x_n)$$

โดยที่  $\{x_1, x_2, \dots, x_n\}$  ใช้แทน *cost factors* วิธีการอัลกอริทึมที่มีอยู่ในปัจจุบันมีข้อแตกต่างอยู่สองจุด คือ รูปแบบของฟังก์ชัน  $f$  และการเลือก *cost factors* ตลอดจนทศวรรษที่ผ่านมาตัวแบบการประมาณมูลค่าซอฟต์แวร์เชิงปริมาณได้มีการพัฒนาอย่างต่อเนื่อง เริ่มจากตัวแบบ Empirical ไปจนถึงตัวแบบ Analytical (Putnam, 1978; Parr, 1980, Cantone et al., 1986) โดยตัวแบบ Empirical จะใช้ข้อมูลจากโครงการก่อนหน้าเพื่อประเมินมูลค่าโครงการปัจจุบันและได้สูตรพื้นฐานมาจากการวิเคราะห์ระบบฐานข้อมูลเฉพาะทางที่มีอยู่ ตัวอย่างเช่น ,COCOMO ของ Boehm (Boehm, 1981) ในขณะที่ตัวแบบ Analytical นั้นจะใช้สูตรที่มีพื้นฐานมาจากสมมติฐานรวม เช่น อัตราการแก้ปัญหาของนักพัฒนาและจำนวนปัญหาที่มีอยู่

ตัวแบบการประมาณมูลค่าซอฟต์แวร์ด้วยวิธีอัลกอริทึม

- Linear models อยู่ในรูปแบบ

$$Effort = a_0 + \sum_{i=1}^n a_i x_i$$

โดยที่  $a_1, \dots, a_n$  ถูกเลือกให้เหมาะสมกับข้อมูลของโครงการที่สมบูรณ์ งานของ Nelson ได้ใช้ตัวแบบนี้

- Multiplicative Models อยู่ในรูปแบบ

$$Effort = a_0 \prod_{i=1}^n a_i^{x_i}$$



โดยที่  $a_1, \dots, a_n$  ถูกเลือกให้เหมาะสมกับข้อมูลของโครงการที่สมบูรณ์ งานของ Walston-Felix ได้ใช้ตัวแบบนี้ และ  $x_i$  มีค่าที่เป็นไปได้ 3 ค่าคือ -1, 0, +1 ตัวแบบ Doty จัดอยู่ในตัวแบบนี้ เช่นเดียวกันโดย  $x_i$  มีค่า 2 ค่าคือ 0 และ +1

- Power function models อยู่ในรูปแบบ

$$\text{Effort} = a \times S^b$$

โดยที่  $S$  เป็น code-size และ  $a, b$  เป็นฟังก์ชันของ cost factor อื่น ตัวแบบนี้มีชื่อเสียงและเป็นที่ยอมรับได้แก่ COCOMO และ Putnam & SLIM

- Model Calibration โดยการใช้ linear regression ตัวแบบที่กล่าวมาข้างต้นทั้งหมดไม่ได้พิจารณาถึงสถานการณ์ท้องถิ่น อย่างไรก็ตามสามารถปรับ cost factors โดยใช้ข้อมูลท้องถิ่นและวิธีการถดถอยเชิงเส้น
- Discrete Models อยู่ในรูปแบบของตารางซึ่งโดยปกติจะเกี่ยวข้องกับ ความพยายาม (Effort) ระยะเวลา ความยาก และ cost factors อื่นๆ ตัวแบบในกลุ่มนี้ได้แก่ตัวแบบ Aron (Aron, 1969) ตัวแบบ Wolverton (Wolverton, 1974) และ ตัวแบบ Boeing (Black et al., 1977) ตัวแบบเหล่านี้ได้รับความนิยมในยุคแรกๆเนื่องจากใช้งานง่าย
- Price-S เป็นตัวแบบที่ประมาณมูลค่าซอฟต์แวร์กรรมสิทธิ์พัฒนาและบำรุงรักษาโดย RCA, New Jersey (Park, 1988) เริ่มจากการประมาณขนาด ชนิดและความยากของโครงการ ตัวแบบคำนวณมูลค่าโครงการและตารางการทำงาน
- SoftCost ใช้ขนาด ความพยายาม และระยะเวลาเพื่อระบุความเสี่ยงโดยใช้รูปแบบของการกระจายความน่าจะเป็นของ Rayleigh (Tausworthe, 1981) ตัวแบบนี้มี Heuristics เพื่อใช้เป็นแนวทางให้กับผู้ประมาณในการจัดการกับเทคโนโลยีใหม่ๆและความสัมพันธ์ที่ซับซ้อนระหว่างพารามิเตอร์ต่างๆที่เกี่ยวข้อง

ตารางที่ 2-2 การแบ่งประเภทของตัวแบบอัลกอริทึม

ตัวแบบอัลกอริทึม					
	Linear	Multiplicative	Power function	Discrete	Others
Empirical	Nelson	Walston-Felix Herd et al.	COCOMO(s)	Aron Boeing Wolverton	Price-S
Analytical			Putnam		SoftCost

สำนักงานคณะกรรมการวิจัยแห่งชาติ  
 ห้องสมุดฯ วิจัย  
 วันที่..... 8 ส.ค. 2555  
 เลขทะเบียน..... 245676  
 เลขเรียกหนังสือ.....

นอกจากขนาดของซอฟต์แวร์ยังมี Cost Factors ประเภทอื่นๆ ซึ่งกลุ่มของ Cost Factors ที่ใช้กันอย่างกว้างขวางและถูกเสนอและใช้โดย Boehm et al. ในตัวแบบ COCOMO II (Boehm et al. ,1996) cost factors เหล่านี้แบ่งออกเป็น 4 ชนิด คือ ปัจจัยทางด้านผลิตภัณฑ์ (Product factors) ปัจจัยทางด้านคอมพิวเตอร์ (Computer factors) ปัจจัยส่วนตัว (Personal factors) และ ปัจจัยทางด้านโครงการ (Project factors)

## 2.3 การทบทวนวรรณกรรม/สารสนเทศที่เกี่ยวข้อง

B. Tifimula Rao (Rao, 2009) และคณะ นำเสนองานวิจัยเรื่อง A Novel Neural Network Approach for Software Cost Estimation Using Functional Link Artificial Neural Network (FLANN) โดยในงานวิจัยนี้นำเสนอระบบโครงข่ายประสาทเทียมสำหรับการพยากรณ์มูลค่าของซอฟต์แวร์ซึ่งช่วยลดความซับซ้อนในการคำนวณและสามารถพยากรณ์มูลค่าซอฟต์แวร์แบบออนไลน์ได้ สถาปัตยกรรมของโครงข่ายประสาทเทียมที่นำเสนอในงานวิจัยนี้เป็นแบบที่ไม่มีชั้นซ่อน (Hidden Layer) กระบวนการสอนจะไม่ได้ใช้แบบแพร่ย้อนกลับแบบสมบรูณ์ ค่าตอบที่ใช้ในการเปรียบเทียบของการสอนจะใช้ค่าที่ได้จากการประมาณมูลค่าซอฟต์แวร์ด้วยวิธี COCOMO (Constructive Cost Model) จากผลการทดลองพบว่า วิธีการที่นำเสนอให้ผลการทดลองที่มีประสิทธิภาพสูงกว่าวิธีการโครงข่ายประสาทเทียมทั่วไป

N. Tadayon (Tadayon, 2005) นำเสนองานวิจัยเรื่อง Neural Network Approach for Software Cost Estimation โดยใช้ระบบโครงข่ายประสาทเทียมแบบเปอร์เซปตรอนหลายชั้น (Multi Layer Perceptron) ซึ่งมีชั้นซ่อนได้มากกว่าหนึ่งชั้น โดยใช้การคำนวณของการสอนที่ได้จากวิธี COCOMO การปรับค่าน้ำหนักของโครงข่ายประสาทเทียมจะใช้วิธี Gradient Descent Learning Rule

K. Vinay Kumar และคณะ (Kumar, 2008) นำเสนองานวิจัยเรื่อง Software Development Cost Estimation using Wavelet Neural Networks งานวิจัยนี้เสนอการประยุกต์ใช้ Wavelet Neural Networks สำหรับการพยากรณ์มูลค่าซอฟต์แวร์ โดย K. Vinay Kumar แบ่งการทำงานออกเป็นสองขั้นตอนใหญ่ๆ คือ 1.) การใช้ฟังก์ชัน Morlet และ ฟังก์ชันเกาส์เป็นฟังก์ชันสำหรับการทราสเฟอร์และ 2.) นำเสนอขั้นตอนวิธีสำหรับการหาค่า Threshold ที่สามารถยอมรับได้สำหรับการสอนของ Wavelet Neural Networks (TAWNN) จากผลการศึกษาเปรียบเทียบผลระหว่าง WNN กับวิธี MLP, RBF, MLR, DENFIS และ SVM พบว่า WNN ที่นำเสนอให้ผลการทดลองที่ดีกว่าบางวิธีการที่กล่าวไว้ข้างต้น

Y. Kultur และ คณะ (Kultur, 2009) นำเสนองานวิจัยเรื่อง Ensemble of Neural Networks with Associative Memory (ENNA) for Estimating Software Development Costs ในงานวิจัยนี้แนะนำวิธีการเรียนรู้แบบผสมโดยใช้เทคนิค Associative Memory สำหรับการประมาณมูลค่าซอฟต์แวร์ ผลจากการนำเสนอในงานวิจัยนี้ทำให้ได้วิธีการที่สามารถเพิ่มประสิทธิภาพในการปรับเทียบค่าอย่างอัตโนมัติแทนการจัดการด้วยมือ นอกจากนี้ วิธีการที่นำเสนอยังให้ค่าความถูกต้องที่ดีกว่าวิธีโครงข่ายประสาทเทียมแบบมาตรฐาน