สปาร์ตันซิมพลิซิตี อัลกอริธึมที่มีประสิทธิภาพสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียม

SPARTAN SIMPLICITY: AN EFFICIENT ALGORITHM FOR PRUNING ARTIFICIAL NEURAL NETWORKS

เกียรติกูล เจียรนัยธนะกิจ

KIETIKUL JEARANAITANAKIJ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2551

KMITL-2008-EN-D-018-201

SPARTAN SIMPLICITY: AN EFFICIENT ALGORITHM FOR PRUNING ARTIFICIAL NEURAL NETWORKS

KIETIKUL JEARANAITANAKIJ

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF ENGINEERING IN ELECTRICAL ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2008

KMITL-2008-EN-D-018-201

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ สปาร์ตันซิมพลิซิตี อัลกอริธึมที่มีประสิทธิภาพสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียม

Thesis Title Spartan Simplicity : An Efficient Algorithm for Pruning Artificial Neural Networks

นักศึกษา นายเกียรติกูล เจียรนัยธนะกิจ

รหัสประจำตัว 48060052

ปริญญา วิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.ดร.บุญวัฒน์ อัตชู

หมายเลขวิทยานิพนธ์ KMITL-2008-EN-D-018-201

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.บุญธีร์	เครือตราชู	
ผศ.ดร.สมศักดิ์	วลัยรัชต์	
รศ.ดร.วีระศักดิ์	คุรุธัช	
รศ.ดร.เอื้อน	ปิ่นเงิน	
รศ.ดร.บุญวัฒน์	อัตชู	

วัน / เดือน/ ปี ที่สอบ วันพฤหัสบดีที่ 27 พฤศจิกายน พ.ศ. 2551 เวลา 09.00-11.00 น.

สถานที่สอบ ณ อาคาร A ชั้น 3 ห้องประชุม 1

คณะวิศวกรรมศาสตร์ รับรองแล้ว

(รองศาสตราจารย์ ดร.กอบชัย เดชหาญ)

คณบดี คณะวิศวกรรมศาสตร์

วันที่ 27 พฤศจิกายน พ.ศ. 2551

หัวข้อวิทยานิพนธ์	สปาร์ตันซิมพลิซิตี อัลกอริธึมที่มีประสิทธิภาพสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียม
นักศึกษา	นายเกียรติกูล เจียรนัยธนะกิจ
รหัสประจำตัว	48060052
ปริญญา	วิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2551
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ. ดร. บุญวัฒน์ อัตชู
อาจารย์ที่ปรึกษาวิทยานิพนธ์ร่วม	รศ. ดร. เอื้อน ปิ่นเงิน

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ นำเสนออัลกอริธึมใหม่สำหรับการกำจัดหน่วยซ่อนที่ไม่จำเป็นออกจากโครงข่ายประสาทเทียมแบบป้อนไปข้างหน้าที่มีชั้นซ่อนหนึ่งชั้น ซึ่งการมีจำนวนหน่วยซ่อนที่มากเกินความจำเป็นอาจทำให้โครงข่ายประสาทเทียมมีความสามารถในการทำนายข้อมูลไม่ดีนัก อัลกอริธึมนี้ถูกออกแบบมาเพื่อให้ง่ายในการใช้งานและมีประสิทธิภาพในการลดขนาดโครงสร้างของโครงข่ายประสาทเทียม อีกทั้งยังคงความสามารถในการทำนายข้อมูลได้ดี แนวคิดพื้นฐานของอัลกอริธึมคือการฝึกโครงข่ายจนกระทั่งโครงข่ายเริ่มสูญเสียความสามารถในการทำนายข้อมูล จากนั้นอัลกอริธึมจะวัดค่าความสำคัญของหน่วยซ่อนซึ่งก็คือผลต่างแอคติเวชั่นของหน่วยเอาท์พุทระหว่างค่าที่ต้องการและค่าแอคติเวชั่นเมื่อหน่วยซ่อนนั้นถูกกำจัดออกไป ค่าผลต่างแอคติเวชั่นนี้จะใช้ในการระบุความสำคัญของแต่ละหน่วยซ่อน อัลกอริธึมที่นำเสนอมีความแตกต่างจากวิธีการกำจัดหน่วยซ่อนอื่นๆตรงที่มีการคำนวณค่าความสำคัญของหน่วยซ่อนจากชุดข้อมูลแวลิเดชั่นแทนที่จะใช้ชุดข้อมูลสำหรับฝึก การคำนวณค่าความสำคัญของหน่วยซ่อนนี้ถูกออกแบบให้ไม่มีผลกระทบต่อความซับซ้อนของอัลกอริธึมแบ็คพรอพาเกชั่น ข้อดีของการใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณค่าความสำคัญของหน่วยซ่อนคืออัลกอริธึมสามารถเลือกหน่วยซ่อนที่มีความสำคัญน้อยที่สุดได้อย่างถูกต้อง นั่นคือ มีผลต่อค่าผิดพลาดของการทำนายชุดข้อมูลแวลิเดชั่นน้อยที่สุด เราได้ประยุกต์ใช้งานอัลกอริธึมนี้กับปัญหาสังเคราะห์สามปัญหาและปัญหาจริงอีกเจ็ดปัญหา ผลปรากฏว่า อัลกอริธึมที่นำเสนอสามารถสร้างโครงข่ายประสาทเทียมที่มีขนาดเล็กและมีประสิทธิภาพในการทำนายข้อมูลสูงเมื่อเทียบกับอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนแบบอื่นๆ ในเกือบทุกปัญหา

Thesis	Spartan Simplicity: An Efficient Algorithm for Pruning Artificial Neural Networks
Student	Mr. Kietikul Jearanaitanakij
Student ID	48060052
Degree	Doctor of Engineering
Program	Electrical Engineering
Year	2008
Thesis Advisor	Assoc. Prof. Dr. Boonwat Attachoo
Thesis Co-Advisor	Assoc. Prof. Dr. Ouen Pinngern

ABSTRACT

This thesis proposes a new algorithm for pruning unnecessary hidden units away from the single-hidden layer feedforward neural network. A neural network having more hidden units than necessary may not generalize to the data set. The proposed algorithm is simple and easy to implement, yet produces a very good result. The idea is to train the network until it begins to lose its generalization. Then the algorithm measures the sensitivity and automatically prunes away the most irrelevant unit. The sensitivity is defined as the absolute difference between the desirable output and the output of the pruned network. Unlike other pruning methods, the proposed algorithm is distinct in calculating the sensitivity from the validation set, instead of the training set, without increasing the asymptotic time complexity of the back-propagation algorithm. Three artificial and seven standard benchmarks are applied to test the algorithm. In most problems, the algorithm can produce a compact-sized network with high generalization ability in comparison with other pruning algorithms.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จได้ด้วยความกรุณาและความช่วยเหลือจากอาจารย์ที่ปรึกษา รศ.ดร. เอื้อน ปิ่นเงิน และ รศ. ดร. บุญวัฒน์ อัตชู ผู้ซึ่งให้คำแนะนำอันเป็นประโยชน์อย่างยิ่งต่องานวิจัยและการตีพิมพ์บทความวิจัย

ขอขอบพระคุณ รศ. ดร. บุญธีร์ เครือตราชู ดร. วัชระ ฉัตรวิริยะ รศ.ดร. ครรชิต ไมตรี รศ.ดร. ศุภมิตร จิตตะยโศธร ผศ.ดร. สมศักดิ์ วลัยรัชต์ ผศ.ดร. อรฉัตร จิตต์โสภักตร์ และ รศ.ดร. วีระศักดิ์ คุรุธัช (กรรมการผู้ทรงคุณวุฒิภายนอกสถาบัน) ซึ่งเป็นกรรมการสอบวิทยานิพนธ์, กรรมการสอบหัวข้อและโครงร่างวิทยานิพนธ์, กรรมการสอบวัดคุณสมบัติและกรรมการสอบวิชาสัมมนาที่ได้กรุณาให้คำแนะนำอันเป็นประโยชน์เพื่อปรับปรุงคุณภาพของงานวิจัยจนสำเร็จลุล่วงด้วยดี และขอขอบพระคุณ ดร. ชุติเมษฎ์ ศรีนิลทา ที่กรุณาเซ็นรับรองในสัญญาการศึกษาในระดับปริญญาเอกของข้าพเจ้า

นอกจากนี้ ข้าพเจ้าขอขอบคุณ Dr. Matthew Daily (Asian Institute of Technology) Dr. Vasin Punyakanok (University of Illinois Urbana-Champaign) และ Samarth Swarup (PhD candidate at University of Illinois Urbana-Champaign) ที่ช่วยให้ความคิดเห็นและตรวจสอบความถูกต้องของบทความวิจัยฉบับเต็มก่อนที่จะส่งไปเพื่อพิจารณาตีพิมพ์ในวารสาร JCSC และขอขอบคุณคณะกรรมการผู้พิจารณาบทความวิจัยฉบับเต็มของ "Journal of Circuit, Systems, and Computer" ที่ได้ช่วยให้คำแนะนำที่สร้างสรรค์และเป็นประโยชน์

สุดท้ายขอขอบคุณภรรยาของข้าพเจ้า คุณนพวรรณ เจียรนัยธนะกิจ ที่คอยเป็นกำลังใจและช่วยตรวจสอบความถูกต้องของการใช้ภาษาอังกฤษในบทความที่ตีพิมพ์ระดับนานาชาติ และขอบใจ เด็กชายภูรี เจียรนัยธนะกิจ บุตรชายของข้าพเจ้าที่เป็นเสมือนพลังน้อยๆที่คอยผลักดันให้ข้าพเจ้าผ่านอุปสรรคต่างๆมาได้ตลอด คุณค่าของวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดา และครูอาจารย์ที่ได้อบรมสั่งสอน เป็นผลให้ข้าพเจ้าสามารถประยุกต์ความรู้ทั้งหมดกับงานวิจัยในวิทยานิพนธ์นี้จนสำเร็จลุล่วงด้วยดี

เกียรติกูล เจียรนัยธนะกิจ

สารบัญ

หน้า

สารบัญตาราง

สารบัญรูป

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

โครงข่ายประสาทเทียมมีความสามารถในการเลียนแบบการทำงานของสมองของมนุษย์ ซึ่งใช้ประสบการณ์ในการปรับปรุงประสิทธิภาพ เมื่อได้รับการฝึกฝนด้วยตัวอย่างข้อมูลที่เพียงพอ และแอ็คติเวชั่นฟังก์ชั่นที่เหมาะสม โครงข่ายจะสามารถทำนายข้อมูลที่ยังไม่เคยเห็นมาก่อนได้อย่างถูกต้อง อย่างไรก็ตาม จำนวนของหน่วยซ่อนซึ่งมากเกินความจำเป็นถือเป็นอุปสรรคที่ทำให้โครงข่ายประสาทเทียมมีประสิทธิภาพในการทำนายข้อมูลต่ำ เนื่องจากโครงข่ายจะจดจำทุกๆ รูปแบบของข้อมูลที่ใช้ในการฝึกโดยปราศจากการเรียนรู้ (Overfitting) นอกจากนี้จำนวนหน่วยซ่อนที่เกินมายังทำให้โครงข่ายใช้เวลานานในการเรียนรู้ชุดข้อมูลอีกด้วย ดังนั้น อัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียมจึงเป็นปัญหาที่ถูกเลือกขึ้นมาเพื่อทำการวิจัย

การลดจำนวนของหน่วยซ่อนภายในโครงข่ายประสาทเทียมเป็นประเด็นที่น่าสนใจ เนื่องจาก อัลกอริธึมในปัจจุบันที่สามารถลดจำนวนหน่วยซ่อนมีความหลากหลายและมีความซับซ้อนในการทำงาน นอกจากนี้ อัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนที่มีอยู่ในปัจจุบันอาจคาดเดาความสำคัญของหน่วยซ่อนผิด ทำให้อัลกอริธึมสำหรับกำจัดหน่วยซ่อนขาดความแม่นยำในการประเมินความสำคัญของหน่วยซ่อนแต่ละหน่วย ดังนั้นจึงเป็นเรื่องที่น่าสนใจหากเราสามารถสร้างอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนโดยใช้วิธีที่เรียบง่ายและอยู่บนหลักการที่เป็นที่น่าเชื่อถือแต่กลับให้ประสิทธิภาพที่ดีในการประเมินความสำคัญของหน่วยซ่อน

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ความมุ่งหมายและวัตถุประสงค์ของการศึกษาในวิทยานิพนธ์นี้เพื่อต้องการสร้างอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียมที่มีสามชั้น คือ ชั้นอินพุท ชั้นซ่อน และชั้นเอาท์พุท เนื่องจากจำนวนของหน่วยซ่อนซึ่งมากเกินความจำเป็นถือเป็นอุปสรรคที่ทำให้โครงข่ายประสาทเทียมมีประสิทธิภาพในการทำนายข้อมูลต่ำ อัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนนี้จะต้องถูกออกแบบให้มีความง่ายต่อการเข้าใจและใช้เวลาน้อยในการประยุกต์เข้ากับปัญหาต่างๆ นอกจากนี้ยังต้องลดจำนวนหน่วยซ่อนให้เหลือน้อยโดยที่ความสามารถในการทำนายข้อมูลยังคงมีประสิทธิภาพที่ดี ผลการทดลองของอัลกอริธึมที่นำเสนอจะต้องถูกเปรียบเทียบกับผลการทดลองของอัลกอริธึมในการกำจัดหน่วยซ่อนอื่นๆทั้งในแง่ของจำนวนหน่วยซ่อนสุดท้ายและความผิดพลาดในการทำนายข้อมูลเพื่อเป็นการยืนยันความสำเร็จของอัลกอริธึมที่นำเสนอ สุดท้าย

จะต้องมีการวิเคราะห์ผลการทดลองเพื่ออธิบายถึงเหตุผลของปัญหาหรือปรากฏการณ์ต่างๆที่เกิดขึ้นในการทดลอง

1.3 สมมุติฐานของการศึกษา

เป็นที่ทราบกันดีในวงการปัญญาประดิษฐ์ว่าชุดข้อมูลที่ใช้ในการฝึก (Training set) จักรกลให้เกิดการเรียนรู้เป็นชุดข้อมูลที่จักรกลนั้นเคยเห็นมาแล้วในช่วงเวลาของการเรียนรู้ หากเราต้องการทราบประสิทธิภาพที่แท้จริงของจักรกล เราจะต้องใช้ชุดข้อมูลที่จักรกลนั้นไม่เคยเห็นมาก่อนในช่วงของการเรียนรู้ เช่น การทดสอบจักรกลโดยใช้ชุดข้อมูลแวลิเดชั่น (Validation set) และชุดข้อมูลทดสอบ (Test set) สำหรับการกำจัดหน่วยซ่อนของโครงข่ายประสาทเทียม หากเรานำชุดข้อมูลฝึกมาใช้คำนวณค่าความไม่สำคัญของหน่วยซ่อนอาจทำให้ค่าที่ได้ไม่สอดคล้องกับความไม่สำคัญที่แท้จริงของหน่วยซ่อนนัก ดังนั้น แนวคิดของอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนที่นำเสนอจึงนำชุดข้อมูลแวลิเดชั่นมาใช้ในการคำนวณความไม่สำคัญของหน่วยซ่อนเพื่อให้ทราบค่าความไม่สำคัญที่แท้จริงของหน่วยซ่อนแต่ละหน่วยภายในโครงข่ายประสาทเทียม ดังนั้นสมมุติฐานของการศึกษาของวิทยานิพนธ์นี้คือ เราจะต้องมีชุดข้อมูลแวลิเดชั่นที่มากพอ (ประมาณร้อยละยี่สิบห้าของชุดข้อมูลทั้งหมด) และชุดข้อมูลแวลิเดชั่นนี้จะต้องเป็นชุดข้อมูลที่สามารถแทนการกระจายตัวของชุดข้อมูลทั้งหมดได้เป็นอย่างดี

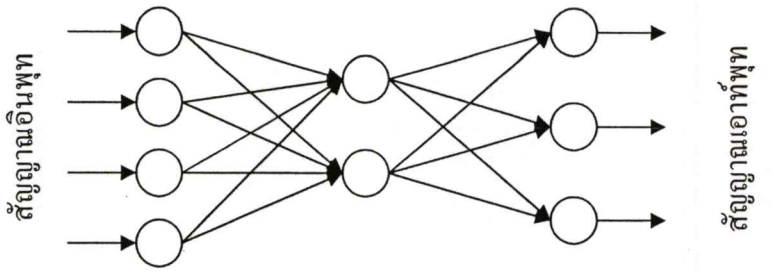
1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

สำหรับแนวความคิดและทฤษฎีที่ใช้เป็นกรอบของการออกแบบการวิจัยในวิทยานิพนธ์นี้ สามารถอธิบายได้ตามหัวข้อย่อยได้ดังนี้

1.4.1 โครงข่ายประสาทเทียมที่มีสามชั้น

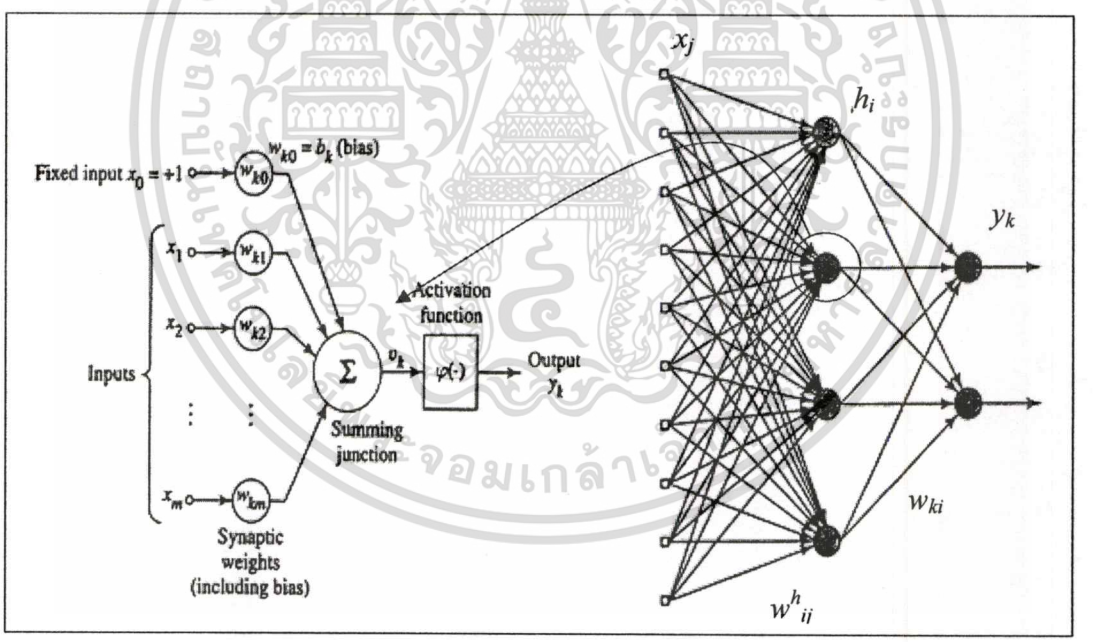
โครงข่ายประสาทเทียมเป็นเครื่องมือในการเรียนรู้ข้อมูล โดยอาศัยหลักการทางสถิติและโครงสร้างของสมองของมนุษย์ ซึ่งสมองสามารถทำการประมวลผลที่ซับซ้อนแบบไม่เชิงเส้นได้ เมื่อโครงข่ายประสาทเทียมได้รับการฝึกด้วยจำนวนข้อมูลที่เหมาะสม โครงข่ายสามารถเรียนรู้ถึงรูปแบบของข้อมูลและสามารถทำนายข้อมูลใหม่ที่ยังไม่เคยเห็นมาก่อนได้อย่างถูกต้องและแม่นยำ

โครงข่ายประสาทเทียมจำลองโครงสร้างของสมองมนุษย์โดยประกอบด้วยหน่วยประมวลผลอย่างง่ายๆซึ่งมีขนาดเล็กเป็นจำนวนมากเรียกว่า นิวรอน (Neuron) ซึ่งนิวรอนเหล่านี้ถูกเชื่อมกันด้วยเส้นเชื่อม (Link) ซึ่งมีน้ำหนัก (Weight) กำกับในแต่ละเส้นเชื่อม เส้นเชื่อมเหล่านี้ทำหน้าที่พาสัญญาณที่ได้จากนิวรอนหนึ่งไปยังอีกนิวรอนหนึ่ง สัญญาณที่ออกจากนิวรอนหนึ่งสามารถกระจายไปยังอีกหลายๆนิวรอนได้ โครงสร้างของโครงข่ายประสาทเทียมสามารถแสดงได้ดังรูปที่ 1.1



รูปที่ 1.1 โครงสร้างของโครงข่ายประสาทเทียม

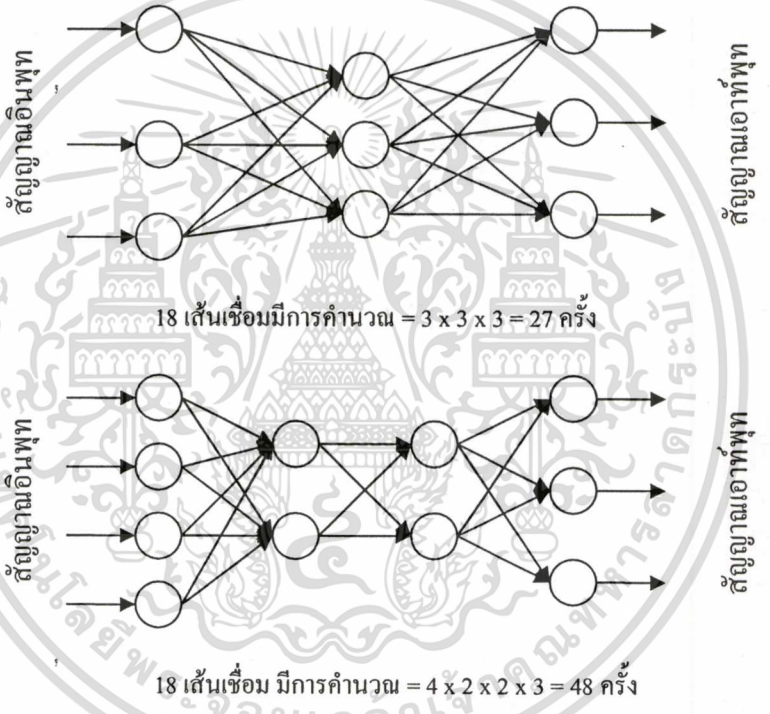
รูปที่ 1.2 แสดงการทำงานที่เกิดขึ้นภายในนิวรอนหนึ่งหน่วย โดยข้อมูลอินพุทจะถูกคูณด้วยค่าน้ำหนักที่เชื่อมอยู่กับอินพุทนั้น ก่อนที่จะถูกรวมเข้ากับข้อมูลที่ได้จากอินพุทตัวอื่น จากนั้นค่าผลรวมที่ได้จะผ่านเข้าไปในแอ็คติเวชั่นฟังก์ชั่นเพื่อกำหนดให้มีค่าอยู่ภายในช่วงที่กำหนด เช่น 0 ถึง +1, -1 ถึง +1 เป็นต้น ในระหว่างการฝึกโครงข่าย ค่าแอ็คติเวชั่นของเอาท์พุทนิวรอนจะถูกเปรียบเทียบกับค่าที่ต้องการ (Desirable value) เพื่อหาค่าผิดพลาดซึ่งจะถูกนำไปปรับค่าน้ำหนักภายในโครงข่ายประสาทเทียมต่อไป



รูปที่ 1.2 แสดงการทำงานที่เกิดขึ้นภายในนิวรอนหนึ่งหน่วย

Haykin S. [17] ได้กล่าวไว้ในตำราโครงข่ายประสาทเทียมในปี ค.ศ. 1994 ว่าโครงข่ายประสาทเทียมที่ถูกสร้างขึ้นเพื่อแก้ปัญหาส่วนใหญ่ประกอบไปด้วย 3 ชั้นได้แก่ ชั้นอินพุท (Input layer) ชั้นซ่อน (Hidden layer) และ ชั้นเอาท์พุท (Output layer) จำนวนของชั้นซ่อนที่เป็นที่นิยมใช้

กันมากที่สุดคือหนึ่งชั้นเนื่องจากมีความซับซ้อนของโครงข่ายไม่มากเท่ากับโครงข่ายที่มีหลายชั้นซ่อน เหตุผลที่โครงข่ายประสาทเทียมต้องการชั้นซ่อน คือ นิวรอนที่อยู่ในชั้นซ่อนทำหน้าที่ตรวจจับคุณลักษณะ (Feature) ของข้อมูลเพื่อใช้ในการทำนายผล ซึ่งคุณลักษณะเหล่านี้สามารถแทนได้ด้วยค่าน้ำหนักที่เชื่อมระหว่างชั้นอินพุทกับชั้นเอาท์พุท โดยทั่วไปแล้ว โครงข่ายประสาทเทียมที่มีชั้นซ่อนเพียงหนึ่งชั้น สามารถแทนฟังก์ชั่นต่อเนื่องใดๆได้ อย่างไรก็ตาม หากต้องการให้โครงข่ายประสาทเทียมเรียนรู้ฟังก์ชั่นที่ไม่ต่อเนื่อง เราอาจจะต้องใช้จำนวนชั้นซ่อนถึงสองชั้น อาจมีบางปัญหาที่มีความซับซ้อนมากๆซึ่งจำเป็นจะต้องใช้จำนวนชั้นซ่อนมากกว่าสองชั้น แต่จำนวนที่เหมาะสำหรับปัญหาโดยทั่วไปที่สุดคือหนึ่งชั้นซ่อนเท่านั้น นอกจากนี้ การมีจำนวนชั้นซ่อนมากจนเกินไปจะทำให้การทำงานของโครงข่ายประสาทเทียมซับซ้อนขึ้น ดังรูปที่ 1.3



รูปที่ 1.3 เปรียบเทียบการคำนวณของโครงข่ายประสาทเทียมแบบหนึ่งและสองชั้นซ่อน

โครงข่ายประสาทเทียมแบบป้อนไปข้างหน้า (Feed-forward) หมายถึง โครงข่ายประสาทเทียมที่นำพาสัญญาณจากนิวรอนของชั้นอินพุทไปยังนิวรอนของชั้นเอาท์พุทโดยอาจจะผ่านชั้นซ่อนหรือไม่ก็ได้ นอกจากนี้แล้ว สัญญาณที่ออกมาจากชั้นเอาท์พุทจะไม่นำกลับมาป้อนให้กับชั้นอินพุทอีกครั้งหนึ่ง

ถึงแม้ว่าโครงข่ายประสาทเทียมจะจำลองการทำงานที่คล้ายคลึงกับการทำงานของสมองที่มีความซับซ้อน การทำงานของโครงข่ายประสาทเทียมสามารถอธิบายด้วยขั้นตอนคร่าวๆ (รายละเอียดของการทำงานถูกอธิบายในหัวข้อต่อไป) ดังต่อไปนี้ พร้อมกับผังงานในรูปที่ 1.4

ก. ออกแบบโครงสร้างของโครงข่ายประสาทเทียม โดยที่โครงข่ายที่ดีจะต้องมีจำนวนของหน่วยซ่อนที่พอเหมาะ หากโครงข่ายมีจำนวนหน่วยซ่อนน้อยจนเกินไป จะทำให้โครงข่ายไม่สามารถเรียนรู้อะไรจากข้อมูลฝึกได้เลย นอกจากนี้ยังทำให้โอกาสที่จะเกิดปัญหาโลคอลมินิมา (Local minima) มีมากขึ้นด้วย ในทางตรงข้าม หากหน่วยซ่อนมีจำนวนมากจนเกินไป อาจทำให้โครงข่ายต้องใช้เวลานานในการฝึก อีกทั้งยังทำให้โครงข่ายสูญเสียความสามารถในการทำนายข้อมูลไปอีกด้วย ด้วยเหตุผลที่โครงข่ายพยายามจำรูปแบบข้อมูลทั้งหมด แทนที่จะเป็นการเรียนรู้

ข. กำหนดค่าน้ำหนักแบบสุ่มให้กับเส้นเชื่อมที่เชื่อมต่อหน่วยต่างๆภายในโครงข่าย การสุ่มค่าน้ำหนักที่ดีจะทำให้โครงข่ายสามารถหลีกเลี่ยงจากปัญหาโลคอลมินิมาได้ ค่าน้ำหนักมักจะขึ้นอยู่กับจำนวนของเส้นเชื่อมที่ต่ออยู่กับหน่วยภายในโครงข่าย อย่างไรก็ดี นักวิจัยโดยส่วนใหญ่เลือกที่จะสุ่มค่าน้ำหนักของโครงข่ายภายในช่วงที่เล็กๆ เช่น -0.1 ถึง +0.1 หรือ -1.0 ถึง +1.0 เป็นต้น

ค. ฝึกโครงข่าย โดยป้อนข้อมูลอินพุทที่ใช้ฝึกโครงข่ายเข้ามาในระบบ จากนั้นหาผลรวมของอินพุทที่คูณกับน้ำหนัก และคำนวณค่าแอ็คติเวชั่นของนิวรอนในโครงข่าย

ง. ทำขบวนการในขั้นตอน ค) สำหรับชั้นซ่อนต่อๆไป จนกระทั่งสามารถคำนวณหาค่าแอ็คติเวชั่นของเอาท์พุทนิวรอน

จ. เปรียบเทียบค่าแอ็คติเวชั่นที่คำนวณได้กับค่าที่ต้องการจากตัวอย่างข้อมูลเพื่อหาค่าผิดพลาดของเอาท์พุท

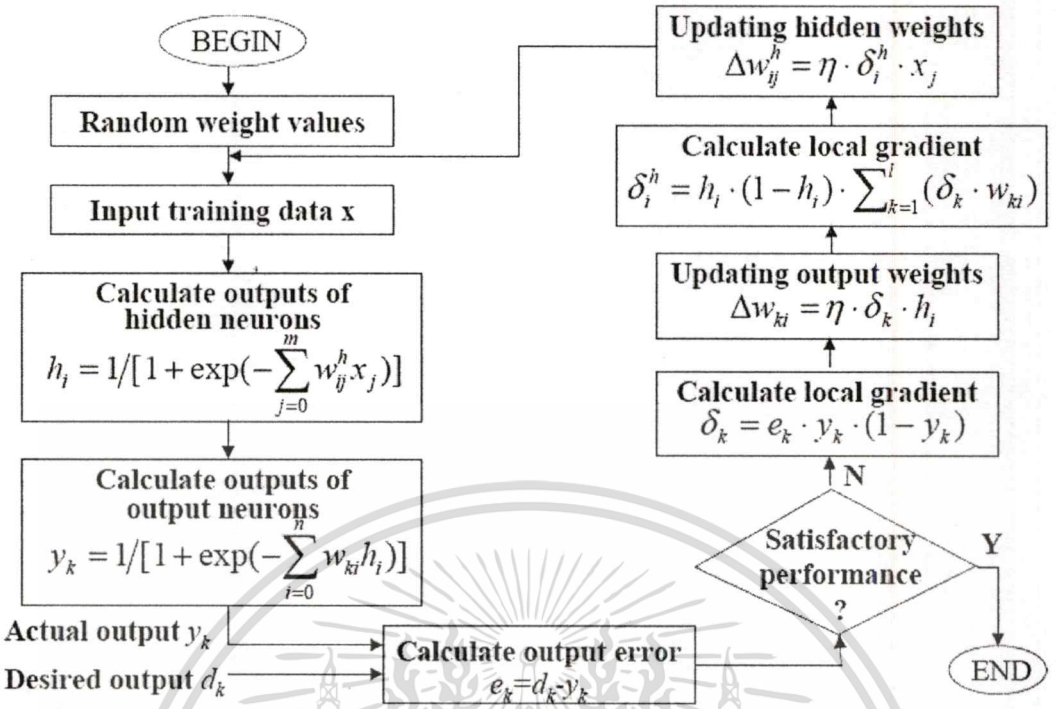
ฉ. ตรวจสอบเงื่อนไขในการหยุดฝึกโครงข่าย เช่น หยุดเมื่อค่าผิดพลาดจากการฝึกมีค่าต่ำกว่าค่าที่กำหนด หากเงื่อนไขในการหยุดเป็นจริง ให้หยุดการฝึกโครงข่ายโดยข้ามไปยังขั้นตอน ญ)

ช. คำนวณค่าโลคอลกราเดี้ยน (Local gradient) แล้วนำมาปรับค่าน้ำหนักที่เชื่อมต่อกับเอาท์พุทนิวรอน

ซ. คำนวณค่าโลคอลกราเดี้ยนแล้วนำมาปรับค่าน้ำหนักที่เชื่อมต่อกับนิวรอนในชั้นซ่อน

ฌ. ย้อนไปยังขั้นตอน ค)

ญ. ใช้โครงข่ายประสาทเทียมในการช่วยแก้ปัญหา



รูปที่ 1.4 แสดงผังงานการทำงานของโครงข่ายประสาทเทียม

ปัจจัยที่ทำให้โครงข่ายประสาทเทียมมีความสามารถในการคำนวณสูงคือโครงสร้างแบบขนานที่มีขนาดใหญ่และความสามารถในการเรียนรู้ ทำให้โครงข่ายประสาทเทียมสามารถทำนายข้อมูลที่ไม่เคยเห็นมาก่อนได้อย่างแม่นยำ อย่างไรก็ตาม ในทางปฏิบัติแล้วโครงข่ายประสาทเทียมยังห่างไกลกับการทำงานที่แท้จริงของสมองของมนุษย์ เนื่องจากโครงข่ายประสาทเทียมยังมีข้อจำกัดในการแก้ปัญหาที่มีความซับซ้อนมากๆ ซึ่งวิธีที่นิยมใช้ในการแก้ปัญหาที่ซับซ้อนนี้คือการแบ่งปัญหาที่ซับซ้อนออกเป็นปัญหาย่อยๆ แล้ว แล้วใช้โครงข่ายประสาทเทียมเข้าไปแก้ปัญหา

การนำโครงข่ายประสาทเทียมไปใช้งานให้ประโยชน์ดังนี้

ก. ความไม่เชิงเส้น (Nonlinearity) โครงข่ายประสาทเทียมสามารถทำงานได้ทั้งเชิงเส้นและไม่เชิงเส้น เนื่องจากนิวรอนภายในโครงข่ายใช้งานแอ็คติเวชั่นฟังก์ชั่นที่แตกต่างกันออกไปซึ่งมีทั้งแบบเชิงเส้นและไม่เชิงเส้น ความสามารถในการทำงานแบบไม่เชิงเส้นนั้นจำเป็นสำหรับปัญหาที่มีอินพุทที่ไม่เป็นเชิงเส้น เช่น สัญญาณเสียง เป็นต้น

ข. สามารถใช้งานได้ดีกับปัญหาที่มีการจับคู่ระหว่างอินพุทและเอาท์พุท (Input-Output Mapping) หรือเป็นที่รู้จักกันดีในวิธีการเรียนรู้แบบมีผู้แนะนำ (Supervised learning)

ค. การปรับตัว (Adaptivity) โครงข่ายประสาทเทียมมีความสามารถในการปรับน้ำหนักเพื่อตอบสนองกับการเปลี่ยนแปลงกับสิ่งแวดล้อมที่อยู่รอบตัว ดังนั้น โครงข่าย

ประสาทเทียมที่ถูกฝึกภายใต้สิ่งแวดล้อมหนึ่งสามารถที่จะปรับตัวเพื่อตอบสนองต่อสิ่งแวดล้อมที่เปลี่ยนแปลงเล็กน้อยได้อย่างรวดเร็ว

ง. การตอบสนองอย่างชัดเจน (Evidential Response) หากเรานำโครงข่ายประสาทเทียมมาใช้ในการจำแนกประเภทของข้อมูล โครงข่ายสามารถตอบสนองต่อข้อมูลได้อย่างชัดเจน โดยไม่เกิดความไม่แน่ใจขึ้น (นั่นคือ จำแนกประเภทโดยไม่อาศัยความน่าจะเป็น)

จ. ทนทานต่อความผิดพลาด (Fault Tolerance) ข้อมูลไม่ได้ขึ้นอยู่กับนิวรอนใดนิวรอนหนึ่ง ซึ่งสามารถกระจายความเสี่ยงในกรณีที่นิวรอนหนึ่งทำงานผิดพลาด เนื่องจากโครงข่ายประสาทเทียมเกิดจากนิวรอนหลายๆตัวมาเชื่อมต่อกัน ดังนั้นความผิดพลาดที่เกิดขึ้นจึงกระจายและถูกซึมซับโดยนิวรอนตัวอื่นๆ

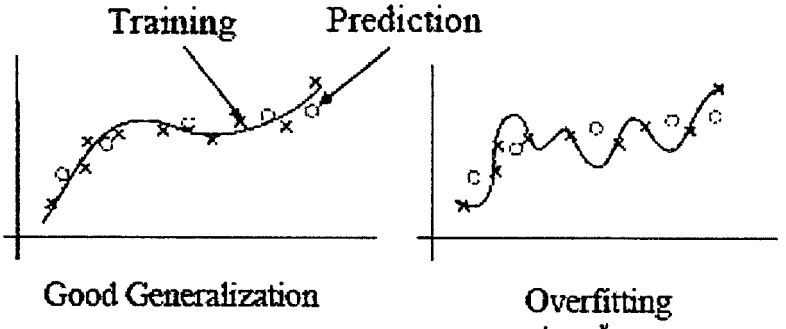
ฉ. ไม่จำเป็นจะต้องมีความรู้เบื้องต้น (Prior knowledge) ก่อนที่จะใช้งานโครงข่ายประสาทเทียม ทำให้สามารถใช้งานได้กับปัญหาหลากหลายประเภท

อย่างไรก็ตาม ปัญหาที่อาจพบได้ในการใช้งานโครงข่ายประสาทเทียมมีดังนี้คือ

ก. ปัญหาโลคอลมินิมา (Local minima) ปัญหานี้เกิดขึ้นเมื่อมีค่าผิดพลาดที่ได้มาจากโครงข่ายประสาทเทียมไม่ใช่ค่าผิดพลาดที่ต่ำที่สุดภายใต้ระนาบเดียวกัน สำหรับชุดข้อมูลหนึ่งๆ ความน่าจะเป็นที่จะเกิดปัญหาโลคอลมินิมาจะลดลงเมื่อจำนวนของน้ำหนักภายในโครงข่ายเพิ่มขึ้น ดังนั้นการเพิ่มจำนวนหน่วยซ่อนภายในโครงข่ายจึงเป็นการหลีกเลี่ยงและลดปัญหาของโลคอลมินิมาได้ทางหนึ่ง

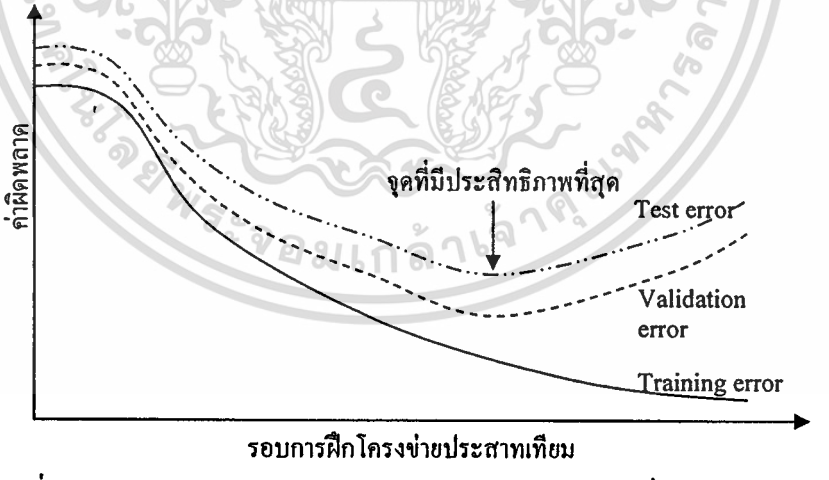
ข. สิ่งรบกวนภายในข้อมูล (Noise) ข้อมูลอื่นบางอย่างอาจไม่ได้เกี่ยวข้องกับการทำนายผลของโครงข่ายประสาทเทียม แต่โครงข่ายอาจจะจำรูปแบบของข้อมูลรบกวนจนทำให้ความสามารถในการทำนายลดน้อยลง (Overfitting ดังแสดงในรูปที่ 1.5) เมื่อทดสอบกับชุดข้อมูลที่ไม่เคยเห็นมาก่อน สิ่งรบกวนภายในข้อมูลอาจเกิดได้ทั้งมีข้อมูลแปลกปลอมอยู่ในนั้น หรืออาจเป็นความไม่สมบูรณ์และความขาดหายของข้อมูลที่ใช้ในการฝึก

ค. เนื่องจากโครงข่ายประสาทเทียมเป็นการทำงานของอัลกอริธึมทางสถิติเพื่อปรับปรุงค่าน้ำหนักภายในโครงข่าย ดังนั้นผลลัพธ์ของโครงข่ายที่ได้จึงไม่อาจจะอธิบายได้ด้วยเหตุผลใดๆ ซึ่งก็ส่งผลให้โครงข่ายประสาทเทียมเป็นกล่องดำที่สามารถใช้แก้ปัญหาที่ไม่สามารถอธิบายด้วยอัลกอริธึมใดๆได้



รูปที่ 1.5 แสดงปัญหาความสามารถในการทำนายลดน้อยลง ที่เกิดขึ้นกับโครงข่ายประสาทเทียม

การออกแบบโครงข่ายประสาทเทียมให้มีประสิทธิภาพในการทำนายข้อมูลที่ไม่เคยเห็นมาก่อนเป็นความท้าทายที่นักวิจัยพยายามค้นคว้าหาคำตอบ โดยทั่วไป การฝึกโครงข่ายจะให้ค่าผิดพลาดออกมาดังแสดงในรูปที่ 1.6 ซึ่งหากเราหยุดฝึกโครงข่ายก่อนที่จะถึงจุดที่มีประสิทธิภาพที่สุด (Optimal point) โครงข่ายจะทำนายข้อมูลได้ไม่ดี แม้กระทั่งกับการทำนายข้อมูลฝึกที่เคยเห็นมาแล้วก็ตาม สถานการณ์นี้ถูกเรียกว่าการเรียนรู้ไม่เพียงพอ (Underfitting) ในทางตรงกันข้าม หากเราหยุดฝึกโครงข่ายหลังจากที่ผ่านจุดที่มีประสิทธิภาพที่ดีที่สุดมาแล้ว โครงข่ายจะมีความสามารถในการทำนายข้อมูลที่ไม่เคยเห็นมาก่อนได้ไม่ดีนัก เนื่องจากโครงข่ายจำทุกๆรูปแบบของข้อมูลที่ใช้ในการฝึกโดยปราศจากการเรียนรู้ (Overfitting)

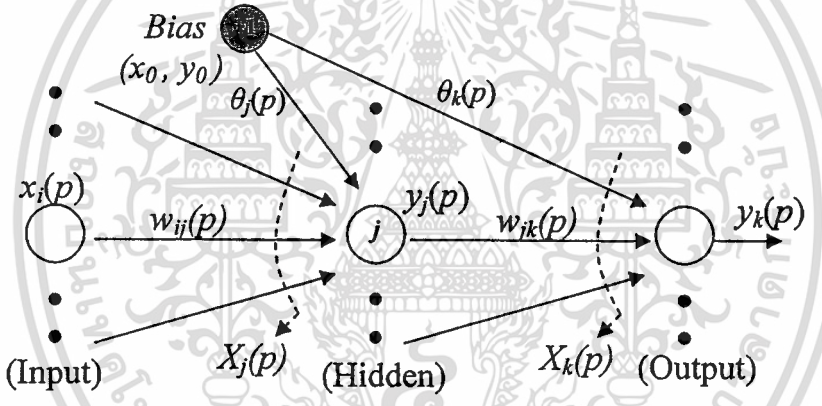


รูปที่ 1.6 แสดงความสัมพันธ์ระหว่างความผิดพลาดกับเวลาที่ใช้ในการฝึก

1.4.2 อัลกอริธึมแบ็คพรอพาเกชั่น

อัลกอริธึมแบ็คพรอพาเกชั่น (Back-propagation algorithm) ถูกนำเสนอโดย Bryson และ Ho ในปี 1969 และเป็นอัลกอริธึมในการเรียนรู้ที่นิยมใช้กันมากที่สุด สำหรับอัลกอริธึมแบ็คพรอพาเกชั่นที่ถูกนำมาใช้งานในการเรียนรู้ของโครงข่ายประสาทเทียมมีหลักการคร่าวๆคือ เมื่อป้อนข้อมูลอินพุทให้แก่ชั้นอินพุทของโครงข่ายแล้ว โครงข่ายจะแพร่กระจายอินพุทที่ได้รับเข้ามาไปยังชั้นต่อไป จนกระทั่งข้อมูลเอาท์พุทถูกสร้างขึ้นโดยชั้นเอาท์พุท ถ้าข้อมูลเอาท์พุทแตกต่างจากข้อมูลเอาท์พุทที่ต้องการ (Desirable output หรือ Target output) ค่าผิดพลาดจะถูกคำนวณและถูกส่งต่อย้อนกลับไปยังชั้นซ่อนและชั้นอินพุทของโครงข่าย จากนั้นค่าน้ำหนักจะถูกเปลี่ยนแปลงตามค่าผิดพลาดที่กระจายมาถึง

วิธีการทำงานโดยละเอียดของอัลกอริธึมแบ็คพรอพาเกชั่นสามารถอธิบายพร้อมกับสัญลักษณ์ต่างๆดังแสดงในรูปที่ 1.7 ดังต่อไปนี้



รูปที่ 1.7 สัญลักษณ์ต่างๆที่ใช้ในโครงข่ายประสาทเทียม

ก. กำหนดค่าเริ่มต้นของน้ำหนักและเทรชโฮล (Threshold) ของโครงข่าย โดยเลือกสุ่มค่าขึ้นมา (โดยปรกติ จะอยู่ภายในช่วง -1.0 ถึง +1.0)

ข. ป้อนข้อมูลชุดฝึกซึ่งประกอบด้วยอินพุท $x_1(p), x_2(p), ..., x_n(p)$ และค่าเอาท์พุทที่ต้องการ $y_1(p), y_2(p), ..., y_n(p)$ โดยที่ p คือจำนวนชุดข้อมูลฝึก จากนั้นคำนวณค่าเอาท์พุทของนิวรอน j ภายในชั้นซ่อนโดยใช้สูตร

$$y_j(p) = \text{ActivationFunction}\left(\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j \right) \qquad (1.1)$$

โดยที่ n คือจำนวนของอินพุทนิวรอนของนิวรอน j ภายในชั้นซ่อน, θ_j คือค่าเทรชโฮลประจำนิวรอน j

จากนั้นกระจายสัญญาณเอาท์พุทของนิวรอนที่ได้ไปยังชั้นเอาท์พุทเพื่อคำนวณค่าเอาท์พุทของแต่ละหน่วยเอาท์พุท k ด้วยสูตรที่คล้ายคลึงกัน

$$y_k(p) = \text{ActivationFunction}\left(\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right) \quad (1.2)$$

โดยที่ m คือจำนวนของอินพุทของนิวรอน k ในชั้นเอาท์พุท

ค. ปรับค่าน้ำหนักภายในโครงข่ายโดยการคำนวณค่ากราเดี้ยนผิดพลาด (Error gradient) สำหรับนิวรอนที่อยู่ในชั้นเอาท์พุท ดังนี้

$$\delta_k(p) = y_k(p) \,.\, [1-y_k(p)] \,.\, e_k(p) \quad (1.3)$$

$$e_k(p) = y_{d,k}(p) - y_k(p) \quad (1.4)$$

โดยที่ $y_{d,k}(p)$ คือ ค่าเอาท์พุทที่ต้องการของหน่วยเอาท์พุท k สำหรับชุดข้อมูล p จากนั้นคำนวณค่าผลต่างของผิดพลาด

$$\Delta w_{jk}(p) = \alpha \,.\, y_j(p) \,.\, \delta_k(p) \quad (1.5)$$

เมื่อ α คืออัตราในการเรียนรู้ (Learning rate) ซึ่งปรกติจะมีค่าน้อยๆ เช่น 0.1, 0.2 จากนั้นทำการปรับค่าน้ำหนักที่เชื่อมกับเอาท์พุทนิวรอน

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p) \quad (1.6)$$

สำหรับนิวรอนที่อยู่ภายในชั้นซ่อน ให้คำนวณค่ากราเดี้ยนผิดพลาด (Error gradient) ดังนี้

$$\delta_j(p) = y_j(p) \,.\, [1-y_j(p)] \,.\, \sum_{k=1}^{l} \delta_k(p) \cdot w_{jk}(p) \quad (1.7)$$

คำนวณค่าผลต่างของค่าผิดพลาดและทำการปรับค่าน้ำหนักที่เชื่อมกับนิวรอนซ่อน

$$\Delta w_{ij}(p) = \alpha \,.\, x_j(p) \,.\, \delta_j(p) \tag{1.8}$$

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \tag{1.9}$$

ง. ทำงานในรอบการทำงานต่อไป จากนั้นวนกลับไปที่ขั้นตอน ข) และทำซ้ำการทำงานจนกระทั่งเงื่อนไขในการหยุดเป็นจริง (โดยปรกติ มักจะให้ค่าผิดพลาดที่ชั้นเอาท์พุทมีค่าน้อยกว่าค่าๆหนึ่ง เช่น 0.001 เป็นต้น)

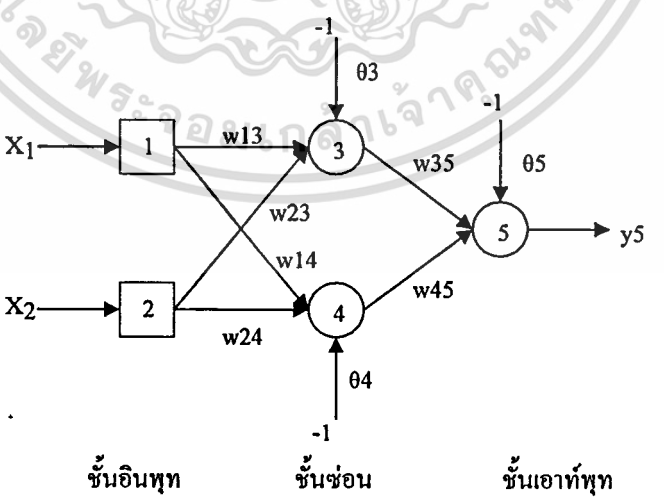
จากตัวอย่างโครงข่ายประสาทเทียมแบบสามชั้น เพื่อใช้แก้ปัญหาเอ็กซ์คลูซีฟ-ออร์ (Exclusive-OR) ดังแสดงในตารางที่ 1.1 และรูปที่ 1.8 กำหนดให้ตัวแปรต่างๆมีค่าเริ่มต้นดังต่อไปนี้

$w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 1.0$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ และ $\theta_5 = 0.3$

นอกจากนี้ สมมุติว่าเราใช้แอ็คติเวชั่นฟังก์ชั่นเป็นแบบซิกมอยด์ (Sigmoid) และใช้ผลรวมค่าผิดพลาดยกกำลังสอง (Sum-squared error) เป็นฟังก์ชั่นคำนวณค่าผิดพลาด

ตารางที่ 1.1 แสดงข้อมูลสำหรับปัญหาเอ็กซ์คลูซีฟ-ออร์

X_1	X_2	Output (y_5)
0	0	0
0	1	1
1	0	1
1	1	0



รูปที่ 1.8 โครงข่ายประสาทเทียมแบบสามชั้น เพื่อใช้แก้ปัญหาเอ็กซ์คลูซีฟ-ออร์

เริ่มต้นจากการป้อนข้อมูลชุดฝึกซึ่งมีค่า x_1 และ x_2 เป็น 1 และมีค่าเอาท์พุท $y_{d,5}$ เป็น 0 จากนั้นทำการคำนวณค่าเอาท์พุทของนิวรอน 3, 4 และ 5 ตามลำดับ

$$y_3 = \text{sigmoid}(x_1.w_{13} + x_2.w_{23} - \theta_3)$$
$$= 1/[1+e^{-(1x0.5 + 1x0.4 - 1x0.8)}] = 0.5250$$

$$y_4 = \text{sigmoid}(x_1.w_{14} + x_2.w_{24} - \theta_4)$$
$$= 1/[1+e^{-(1x0.9 + 1x1.0 - 1x0.1)}] = 0.8808$$

$$y_5 = \text{sigmoid}(y_3.w_{35} + y_4.w_{45} - \theta_5)$$
$$= 1/[1+e^{-(-0.5250x1.2 + 0.8808x1.1 - 1x0.3)}]$$
$$= 0.5097$$

ดังนั้นค่าผิดพลาดที่ได้คือ

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

ขั้นตอนต่อไปเป็นการปรับน้ำหนักและค่าเทรชโฮลด์ตามค่าผิดพลาดที่กระจายกลับไปในโครงข่าย เราคำนวณค่ากราเดี้ยนผิดพลาดสำหรับนิวรอน 5 ที่ชั้นเอาท์พุทดังนี้

$$\delta_5 = y_5(1-y_5)e$$
$$= 0.5097x(1-0.5097)x(-0.5097)$$
$$= -0.1274$$

จากนั้นเราคำนวณผลต่างของน้ำหนักโดยสมมุติให้อัตราการเรียนรู้ (Learning rate) มีค่า 0.1

$$\Delta w_{35} = \alpha . y_3 . \delta_5$$
$$= 0.1 \text{ x } 0.5250 \text{ x } (-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha . y_4 . \delta_5$$
$$= 0.1 \text{ x } 0.8808 \text{ x } (-0.1274) = -0.0112$$

$$\Delta \theta_5 = \alpha . (-1) . \delta_5$$
$$= 0.1 \text{ x } (-1) \text{ x } (-0.1274) = -0.0127$$

ต่อไปเราคำนวณค่ากราเดี้ยนผิดพลาด สำหรับนิวรอน 3 และ 4

$$\delta_3 = y_3(1-y_3) . \delta_5 . w_{35}$$

= 0.5250x(1-0.5250)x(-0.1274)x(-1.2)

= 0.0381

$\delta_4 = y_4(1-y_4) \cdot \delta_5 \cdot w_{45}$

= 0.8808x(1-0.8808)x(-0.1274)x(1.1)

= -0.0147

คำนวณค่าผลต่างของน้ำหนัก

$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3$

= 0.1 x 1 x (0.0381) = 0.0038

$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3$

= 0.1 x 1 x (0.0381) = 0.0038

$\Delta \theta_3 = \alpha \cdot (-1) \cdot \delta_3$

= 0.1 x (-1) x (0.0381) = -0.0038

$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4$

= 0.1 x 1 x (-0.0147) = -0.0015

$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4$

= 0.1 x 1 x (-0.0147) = -0.0015

$\Delta \theta_4 = \alpha \cdot (-1) \cdot \delta_4$

= 0.1 x (-1) x (-0.0147) = 0.0015

สุดท้าย เราปรับค่าน้ำหนักโดยบวกด้วยค่าผลต่างที่คำนวณไว้แล้วก่อนหน้านี้

$w_{13} = w_{13} + \Delta w_{13}$

= 0.5 + 0.0038 = 0.5038

$w_{14} = w_{14} + \Delta w_{14}$

= 0.9 - 0.0015 = 0.8985

$w_{23} = w_{23} + \Delta w_{23}$

= 0.4 + 0.0038 = 0.4038

$w_{24} = w_{24} + \Delta w_{24}$

= 1.0 - 0.0015 = 0.9985

$w_{35} = w_{35} + \Delta w_{35}$

$= -1.2 - 0.0067 = -1.2067$

$w_{45} = w_{45} + \Delta w_{45}$

$= 1.1 - 0.0112 = 1.0888$

$\theta_3 = \theta_3 + \Delta\theta_3$

$= 0.8 - 0.0038 = 0.7962$

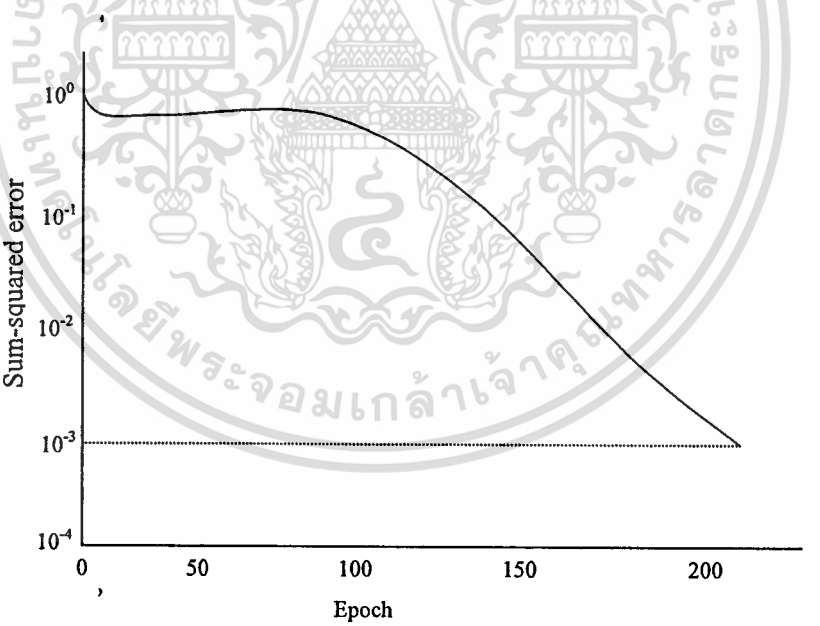
$\theta_4 = \theta_4 + \Delta\theta_4$

$= -0.1 - 0.0015 = -0.0985$

$\theta_5 = \theta_5 + \Delta\theta_5$

$= 0.3 + 0.0127 = 0.3127$

ขบวนการทำงานวนซ้ำไปเรื่อยๆ จนกระทั่งผลรวมค่าผิดพลาดยกกำลังสอง (Sum-squared-error) มีค่าน้อยกว่า 0.001 รูปที่ 1.9 แสดงกราฟของการเรียนรู้โดยเป็นความสัมพันธ์ระหว่างผลรวมค่าผิดพลาดยกกำลังสอง และจำนวนรอบที่ใช้ในการฝึกโครงข่ายประสาทเทียมสำหรับปัญหาเอ็กซ์คลูซีฟ-ออร์ (Exclusive-OR) โครงข่ายใช้จำนวนรอบ (Epoch) ประมาณ 224 หรือ จำนวนรอบของการวนซ้ำ 896 รอบในการเรียนรู้ข้อมูลอินพุท



รูปที่ 1.9 กราฟของการเรียนรู้สำหรับปัญหา Exclusive-OR

ค่าน้ำหนักสุดท้ายที่เราได้จากโครงข่ายที่ถูกฝึกแล้วสามารถแสดงได้ดังนี้

$w_{13} = 4.7621$, $w_{14} = 6.3917$,

$w_{23} = 4.7618$, $w_{24} = 6.3917$,

$w_{35} = -10.3788$, $w_{45} = 9.7691$,

$\theta_3 = 7.3061$, $\theta_4 = 2.8441$, $\theta_5 = 4.5589$

สำหรับผลลัพธ์หลังจากที่ป้อนข้อมูลอินพุทเข้าไปในโครงข่ายสุดท้าย สามารถแสดงได้ในตารางที่ 1.2

ตารางที่ 1.2 แสดงผลลัพธ์สุดท้ายของโครงข่ายประสาทเทียมสำหรับปัญหาเอ็กซ์คลูซีฟ-ออร์ (Exclusive-OR)

Inputs		Desired Output	Actual Output	Error	Squared error
x_1	x_2	y_d	y_5	e	e^2
1	1	0	0.0155	-0.0155	0.000240
0	1	1	0.9849	0.0151	0.000226
1	0	1	0.9849	0.0151	0.000227
0	0	0	0.0175	-0.0175	0.000306
				Sum of squared error	0.001000

1.4.3 ฟังก์ชั่นคำนวณค่าผิดพลาด

ฟังก์ชั่นคำนวณค่าผิดพลาด (Error function) มีประโยชน์ในการระบุประสิทธิภาพของโครงข่าย โดยวัตถุประสงค์ของอัลกอริธึมแบ็คพรอพาเกชั่นก็คือทำให้ค่าผิดพลาดนี้มีค่าน้อยที่สุด ฟังก์ชั่นคำนวณค่าผิดพลาดที่นิยมใช้กันมากที่สุดในการฝึกโครงข่ายประสาทเทียมคือ ผลรวมค่าผิดพลาดยกกำลังสอง (Sum-squared error, SSE), ค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง (Mean-squared error, MSE) และ รากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง (Root-mean-squared error, RMSE)

กำหนดให้ $e_k(n)$ คือค่าผิดพลาดที่เกิดกับนิวรอน k ที่ชั้นเอาท์พุทสำหรับชุดข้อมูลฝึกที่ n

ก. ผลรวมค่าผิดพลาดยกกำลังสอง (SSE)

เนื่องจากค่าเฉลี่ยของผลต่างของความผิดพลาดสำหรับนิวรอน k ถูกกำหนดไว้เท่ากับ $\frac{1}{2}e_k^2(n)$ ดังนั้น ผลรวมค่าผิดพลาดยกกำลังสองที่ได้จากการรวมค่าเฉลี่ยของผลต่างของความผิดพลาดของทุกๆนิวรอนที่ชั้นเอาท์พุทมีค่าเท่ากับ

$$SSE\ (n) = \frac{1}{2}\sum_{k\in K} e_k^2(n)$$ เมื่อ K คือนิวรอนทั้งหมดที่ชั้นเอาท์พุท (1.10)

ข. ค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง (MSE)

ค่าเฉลี่ยค่าผิดพลาดยกกำลังสองเกิดจากการหาค่าเฉลี่ยของผลรวมค่าเฉลี่ยของผลต่างของความผิดพลาดทุกๆนิวรอนที่ชั้นเอาท์พุท ซึ่งก็คือค่าเฉลี่ยของ *SSE* นั่นเอง

$$MSE = \frac{1}{N}\sum_{n=1}^{N} SSE(n)$$ เมื่อ N คือจำนวนข้อมูลชุดฝึก (1.11)

โดยหลักการผลรวมค่าผิดพลาดยกกำลังสองจะให้ค่าผิดพลาดของนิวรอนเอาท์พุททั้งหมดโดยไม่ได้คำนึงถึงจำนวนของชุดข้อมูลที่ใช้ฝึกโครงข่ายว่ามีจำนวนมากน้อยเพียงใด ค่าผิดพลาดที่ได้อาจไม่ค่อยแม่นยำนัก ซึ่งหากเทียบกับค่าเฉลี่ยค่าผิดพลาดยกกำลังสองจะเป็นการคิดค่าผิดพลาดของนิวรอนเอาท์พุทแบบเฉลี่ย ซึ่งย่อมมีความเสถียรและน่าเชื่อถือกว่าผลรวมค่าผิดพลาดยกกำลังสอง แต่เนื่องจากการคำนวณที่ซับซ้อนกว่าของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง นักวิจัยจึงนิยมทดลองใช้ค่าเฉลี่ยค่าผิดพลาดยกกำลังสองดูก่อนว่าให้ผลที่พอใจหรือไม่

ค. รากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง (RMSE)

รากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง คือการหารากที่สองที่เป็นบวกของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง เพื่อเป็นการแยกความแตกต่างของความผิดพลาดได้ด้วยการใช้ค่าตัวเลขที่มากกว่าค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง โดยปรกติค่าเฉลี่ยค่าผิดพลาดยกกำลังสองจะมีค่าน้อย ซึ่งอาจทำให้ไม่เกิดความแตกต่างกันของตัวเลขที่ถูกปัดเศษ ดังนั้นการหารากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสองจะทำให้ค่าความผิดพลาดมีค่ามากขึ้นจนสามารถสังเกตุเห็นความแตกต่างของตัวเลขได้อย่างชัดเจน อย่างไรก็ตาม รากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสองและค่าเฉลี่ยค่าผิดพลาดยกกำลังสองมีความเท่าเทียมกันจนสามารถใช้งานแทนกันได้

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N} SSE(n)}$$ เมื่อ N คือจำนวนข้อมูลชุดฝึก (1.12)

กำหนดให้โครงข่ายประสาทเทียมมีสามหน่วยเอาท์พุทและมีจำนวนของข้อมูลที่ใช้ฝึกโครงข่ายจำนวน 50 ข้อมูล สมมุติว่า ณ รอบของการทำงานของแบ็คพรอพาเกชั่นหนึ่ง มีค่าเอาท์พุทจริง (Actual value) และค่าเอาท์พุทที่ต้องการ (Target value หรือ Desirable value) ดังตารางที่ 1.3

ตารางที่ 1.3 แสดงค่าเอาท์พุทจริงและค่าเอาท์พุทที่ต้องการสำหรับหน่วยเอาท์พุททั้ง 3 ของโครงข่ายประสาทเทียม

Output unit	Actual value	Target value
Output1	1	0.8
Output2	0	0.1
Output3	0	0.2

เราสามารถคำนวณค่าผิดพลาดแบบต่างๆของหน่วยเอาท์พุททั้ง 3 ได้ดังนี้

ก. ผลรวมค่าผิดพลาดยกกำลังสอง(SSE)

$$SSE = [(1-0.8)^2 + (0-0.1)^2 + (0-0.2)^2]/2$$

$$= 0.045$$

ข. ค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง(MSE)

$$MSE = SSE/50 = 0.045/50$$

$$= 0.0009$$

ค. รากที่สองของค่าเฉลี่ยค่าผิดพลาดยกกำลังสอง(RMSE)

$$RMSE = \sqrt{MSE}$$

$$= 0.03$$

1.4.4 แอ็คติเวชั่นฟังก์ชั่น

แอ็คติเวชั่นฟังก์ชั่น (Activation function) คือ ฟังก์ชั่นที่ใช้หาค่าเอาท์พุทของนิวรอนภายในโครงข่ายประสาทเทียมมีมากมายหลายชนิดแต่ที่นิยมใช้งานกันมีเพียงสองหรือสามชนิดเท่านั้น (กราฟสำหรับบางฟังก์ชั่นถูกแสดงในรูปที่ 1.10) สาเหตุที่ต้องมีแอ็คติเวชั่นฟังก์ชั่น คือการทำให้นิวรอนภายในโครงข่ายประสาทเทียมเป็นแบบไม่เชิงเส้น (Nonlinear) เพื่อที่จะทำให้โครงข่ายรองรับการทำงานของฟังก์ชั่นไม่เชิงเส้นได้ ข้อมูลบางชนิดมีลักษณะเป็นแบบไม่เชิงเส้น เช่น เสียง ดังนั้นการที่จะประยุกต์ใช้โครงข่ายประสาทเทียมในการแก้ปัญหาจำเป็นที่จะต้องทำให้โครงข่ายมีคุณสมบัติเป็นแบบไม่เชิงเส้น

ก. ฟังก์ชั่นสเต็ป (Step function)

Y = 1, if X>=0

Y = 0, if X<0

ข. ฟังก์ชั่นเครื่องหมาย (Sign function)

Y = +1, if X>=0

Y = -1, if X<0

ค. ฟังก์ชั่นซิกมอยด์ (Sigmoid function)

$$Y = \frac{1}{1+e^{-X}}$$

โดเมนของฟังก์ชั่นชนิดนี้มีค่าอยู่ในช่วง [0, +1]

ง. ฟังก์ชั่นเชิงเส้น (Linear function)

$$Y = X$$

จ. ฟังก์ชั่นไฮเปอร์โบลิกแทนเจ้นท์ (Hyperbolic tangent function)

$$Y = \frac{e^{X} - e^{-X}}{e^{X} + e^{-X}}$$

$$= \tanh(x/2)$$

โดเมนของฟังก์ชั่นชนิดนี้มีค่าอยู่ในช่วง [-1, +1]

ฉ. ฟังก์ชั่นเอ็กซ์โปเนนท์เชียลลบ (Negative exponential function)

$$Y = e^{-X}$$

โดเมนของฟังก์ชั่นชนิดนี้มีค่าอยู่ในช่วง [0, +INF]

ช. ฟังก์ชั่นซอฟต์แม็กซ์ (Softmax function)

$$Y = \frac{e^{Xi}}{\sum_{i} e^{Xi}}$$

โดเมนของฟังก์ชั่นชนิดนี้มีค่าอยู่ในช่วง [0, +1]

ความแตกต่างและความเหมาะสมสำหรับแอ็คติเวชั่นฟังก์ชั่นชนิดต่างๆสามารถอธิบายได้ดังนี้

ก. ทั้งฟังก์ชั่นสเต็ปและฟังก์ชั่นเครื่องหมายถูกเรียกว่าฟังก์ชั่นฮาร์ดลิมิต (Hard limit function) โดยทั่วไปจะถูกใช้ในงานการแยกประเภทและการจดจำรูปแบบต่างๆ

ข. ฟังก์ชั่นสเต็ปจะให้ค่าเอาท์พุทที่มีค่าบวก ในขณะที่ฟังก์ชั่นเครื่องหมายสามารถให้ค่าได้ทั้งลบและบวก

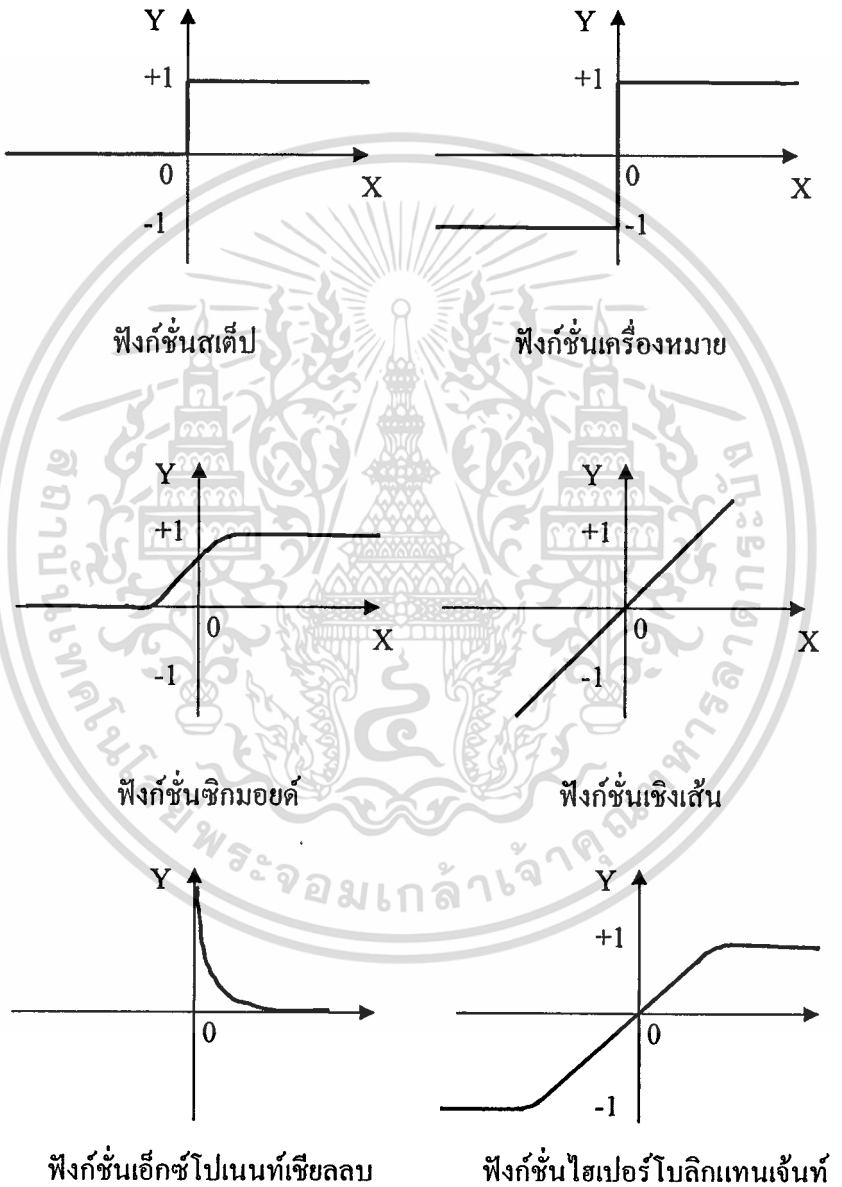
ค. ฟังก์ชั่นซิกมอยด์แปลงอินพุทซึ่งสามารถมีค่าได้ทั้งบวกและลบให้อยู่ในจำนวนจริงระหว่าง 0 และ 1 ฟังก์ชั่นชนิดนี้ถูกเรียกว่าเป็นฟังก์ชั่นมาตรฐานที่ใช้งานในโครงข่ายประสาทเทียมที่ใช้อัลกอริธึมแบบแบ็คพรอพาเกชั่น

ง. ฟังก์ชั่นเชิงเส้นที่ให้ค่าเอาท์พุทเท่ากับค่าอินพุท ฟังก์ชั่นชนิดนี้มักใช้งานเกี่ยวกับการประมาณค่าแบบเส้นตรง (Linear approximation)

จ. ฟังก์ชั่นไฮเปอร์โบลิกแทนเจ้นท์มักถูกใช้งานในการเร่งความเร็วในการฝึกโครงข่ายประสาทเทียม โดยมากจะใช้ในหน่วยที่เป็นหน่วยซ่อน สามารถใช้งานได้ดีกว่าฟังก์ชั่นซิกมอยด์ เนื่องจากความสมมาตรของฟังก์ชั่น

ฉ. ฟังก์ชั่นเอ็กซ์โปเนนท์เชียลลบเหมาะสำหรับหน่วยซ่อนที่เป็นประเภทเรเดียล-เบสิส (Radial-basis) ซึ่งเหมาะสำหรับงานการวิเคราะห์ตัวเลข (Numerical analysis)

ช. ฟังก์ชั่นซอฟต์แม็กซ์เหมาะสำหรับปัญหาที่ต้องการจำแนกประเภทของข้อมูลออกมาเป็นความน่าจะเป็น ทั้งนี้ ฟังก์ชั่นซอฟต์แม็กซ์สามารถใช้ในปัญหาการแยกประเภทข้อมูลโดยไม่ต้องอาศัยวิธีการแยกประเภทอื่นๆเข้ามาช่วย เช่น วินเนอร์-เทค-ออล เป็นต้น

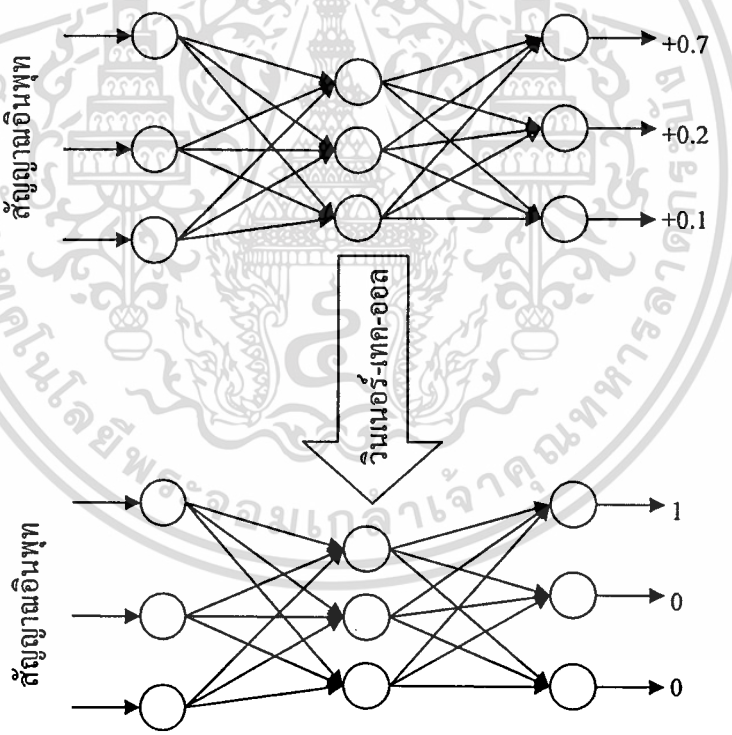


รูปที่ 1.10 แสดงกราฟของ Activation functions

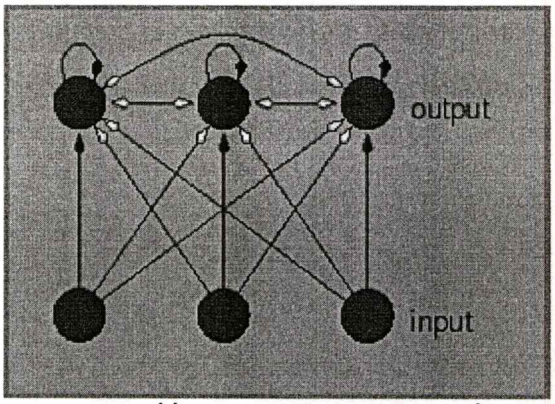
1.4.5 นิวรอนแบบวินเนอร์-เทค-ออล

วินเนอร์-เทค-ออล (Winner-take-all) เป็นวิธีการจำแนกประเภทสำหรับการเรียนรู้ที่มีการแข่งขันกันระหว่างเอาท์พุทนิวรอน (Competitive learning) ซึ่งจะมีเพียงเอาท์พุทนิวรอนเดียวเท่านั้นที่มีค่าเป็นหนึ่ง ส่วนที่เหลือจะมีค่าเป็นศูนย์ทั้งหมด หลักการของวินเนอร์-เทค-ออล คือ ตรวจสอบทุกๆเอาท์พุทนิวรอน และเปรียบเทียบดูว่านิวรอนใดมีค่าแอ็คติเวชั่นสูงที่สุด นิวรอนนั้นจะได้สิทธิ์มีค่าเป็นหนึ่งเพียงนิวรอนเดียวเท่านั้น ตามตัวอย่างดังรูปที่ 1.11

ในรูปที่ 1.12 แต่ละหน่วยเอาท์พุทจะตอบสนองการทำงานของหนึ่งหน่วยอินพุท โดยแต่ละหน่วยอินพุทจะพยายามกระตุ้นให้หน่วยเอาท์พุทของตัวเองให้ทำงานและจะขัดขวางหน่วยเอาท์พุทอื่นๆที่ไม่ใช่ของตนเองไม่ให้ทำงาน ดังนั้นหน่วยอินพุทที่แข็งแรงที่สุดจะทำให้หน่วยเอาท์พุทที่จับคู่มีค่าแอ็คติเวชั่นสูงที่สุดและทำให้หน่วยเอาท์พุทอื่นๆมีค่าแอ็คติเวชั่นลดต่ำลง ในบางโครงข่ายอาจมีการเพิ่มลิ้งค์ที่ทำหน้าที่วนกลับมายังหน่วยเอาท์พุทตัวเอง (Self-loop) เพื่อเป็นการเพิ่มความเสถียรของโครงข่ายประสาทเทียม ดังแสดงในรูปที่ 1.12



รูปที่ 1.11 แสดงการทำงานของนิวรอนแบบวินเนอร์-เทค-ออล



รูปที่ 1.12 แสดงการใช้ลูปที่วิ่งเข้าหาตัวเอง (Self-loop) เพื่อเป็นการเพิ่มความเสถียรของโครงข่ายประสาทเทียม

สำหรับการทำงานแบบขนาน อาจเกิดกรณีที่มีนิวรอน 2 หน่วยที่มีค่าแอ็คติเวชั่นสูงสุดเท่ากัน ผลลัพธ์ของโครงข่ายอาจจะไม่ถูกกำหนดและเกิดการสลับไปสลับมาระหว่างเอาท์พุทของทั้งสองนิวรอน อย่างไรก็ตาม โครงข่ายประสาทเทียมมักถูกใช้งานในระบบคอมพิวเตอร์อนุกรม ซึ่งจะทำให้ปัญหาดังกล่าวหมดไป

แม้ว่าวิธีวินเนอร์-เทค-ออลจะเป็นวิธีที่เรียบง่ายและไม่ซับซ้อน แต่วิธีนี้ก็มีข้อเสีย นั่นคือถ้าค่าน้ำหนักและแอ็คติเวชั่นฟังก์ชั่นของหน่วยเอาท์พุทถูกเลือกอย่างไม่เหมาะสม ผลลัพธ์ที่ได้อาจไม่สามารถบอกประเภทของการแยกประเภทได้ ซึ่งในทางทฤษฎีแล้วเราต้องการหน่วยเอาท์พุทเพียงหน่วยเดียวที่มีค่าแอ็คติเวชั่นที่เป็นบวกสูงที่สุด สำหรับข้อมูลอินพุทที่มีค่าเป็นค่าต่อเนื่อง การใช้วินเนอร์-เทค-ออลอาจก่อให้เกิดการสูญเสียความสามารถในการทำนายข้อมูลที่ยังไม่เคยพบมาก่อน ซึ่งเราสามารถแก้ไขได้โดยการเพิ่มลูปที่วิ่งเข้าหาตัวเอง (Self-loop) เข้าไปที่หน่วยเอาท์พุท ซึ่งจะทำให้เกิดความเสถียรที่หน่วยเอาท์พุท

1.5 ขอบเขตการวิจัย

ขอบเขตของงานวิจัยในวิทยานิพนธ์นี้สามารถอธิบายเป็นข้อๆได้ดังต่อไปนี้

ก. สร้างอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียม

ข. จำนวนหน่วยซ่อนที่ลดลงจะต้องมีความสัมพันธ์ที่สอดคล้องกับความผิดพลาดของการทำนายข้อมูล นั่นคืออัลกอริธึมที่นำเสนอจะต้องกำจัดหน่วยซ่อนให้ได้จำนวนมากๆ โดยโครงข่ายประสาทเทียมยังสามารถทำนายข้อมูลได้อย่างถูกต้องและยอมรับได้

ค. จำนวนชั้นของหน่วยซ่อนภายในโครงข่ายประสาทเทียมมีจำนวนหนึ่งชั้นเท่านั้น

ง. การจับคู่ของข้อมูลอินพุทและหน่วยอินพุทมีลักษณะแบบหนึ่งต่อหนึ่ง

จ. สำหรับปัญหาการแยกประเภท (Classification problem) หน่วยเอาท์พุทแทนการแยกประเภทแบบ 1-of-n นั่นคือ มีหน่วยเอาท์พุท n หน่วยสำหรับข้อมูล n ประเภท และถ้าหน่วยเอาท์พุทใดมีค่าเป็นหนึ่ง หน่วยเอาท์พุทที่เหลือจะต้องมีค่าเป็นศูนย์

ฉ. โครงข่ายประสาทเทียมที่สามารถใช้ได้กับอัลกอริธึมที่นำเสนอจะต้องเป็นโครงข่ายประสาทเทียมที่มีการป้อนข้อมูลไปข้างหน้า (Feed-forward neural network)

ช. การกำจัดหน่วยซ่อนของอัลกอริธึมที่นำเสนอมีลักษณะแบบกำจัดทีละหนึ่งหน่วยซ่อนไปเรื่อยๆ เพื่อป้องกันไม่ให้โครงข่ายประสาทเทียมได้รับผลกระทบมากจนเกินไป

ซ. อัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนจะทำงานได้ถูกต้องก็ต่อเมื่อมีจำนวนข้อมูลแวลิเดชั่น (Validation data set) ที่มากพอ (ประมาณร้อยละยี่สิบห้าของข้อมูลทั้งหมด) และมีการกระจายของข้อมูลอย่างสม่ำเสมอ

1.6 ขั้นตอนของการศึกษา

ขั้นตอนการศึกษาของงานวิจัยในวิทยานิพนธ์นี้สามารถอธิบายได้ดังต่อไปนี้

ก. ศึกษาความรู้พื้นฐานที่เกี่ยวข้องกับงานวิจัย เช่น ทฤษฎีโครงข่ายประสาทเทียม เป็นต้น

ข. ศึกษางานวิจัยที่เกี่ยวข้องที่มีผู้นำเสนอมาแล้ว

ค. ออกแบบแนวความคิดพื้นฐานที่จะนำมาใช้แก้ปัญหาพร้อมทั้งหาเหตุผลเพื่อสนับสนุนแนวความคิดพื้นฐานนั้น

ง. ออกแบบอัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนตามแนวคิดพื้นฐาน

จ. สร้างโปรแกรมที่ทำงานได้ตามอัลกอริธึมที่ได้ออกแบบไว้

ฉ. ทำการทดลองโดยใช้สมมุติฐานของสิ่งแวดล้อมที่เหมือนกับงานวิจัยอื่นที่เกี่ยวข้อง

ช. เปรียบเทียบผลการทดลองกับงานวิจัยอื่น

ซ. วิเคราะห์ผลการทดลองเพื่ออธิบายเหตุผลของปรากฏการณ์ที่เกิดขึ้น

ฌ. สรุปผลการทดลอง

ญ. ส่งบทความวิจัยเพื่อตีพิมพ์ในวารสารระดับนานาชาติและจัดทำรูปเล่มวิทยานิพนธ์

บทที่ 2
งานวิจัยที่เกี่ยวข้อง

2.1 ทบทวนงานวิจัยที่เกี่ยวข้อง

อัลกอริธึมสำหรับการลดขนาดโครงสร้างของโครงข่ายประสาทเทียมมีอยู่สองประเภทใหญ่ๆ คือ คอนสรัคทีฟอัลกอริธึม (Constructive algorithm) [1] ซึ่งเริ่มต้นจากโครงข่ายที่มีขนาดเล็กๆ จากนั้นค่อยๆเพิ่มน้ำหนักหรือหน่วยซ่อนเข้าไปเรื่อยๆจนกระทั่งโครงข่ายเริ่มสูญเสียความสามารถในการทำนายข้อมูล อีกหนึ่งประเภทคือพรูนนิ่งอัลกอริธึม (Pruning algorithm) [2] ซึ่งจะเริ่มต้นจากโครงข่ายที่มีขนาดใหญ่ จากนั้นจึงค่อยๆกำจัดหน่วยซ่อนและน้ำหนักที่ไม่จำเป็นออกไป

มีหลายวิธีที่จะกำจัดหน่วยซ่อนที่ไม่จำเป็นออกไปจากโครงข่าย เช่น Mozer และ Smolensky [3] ได้นำเสนอวิธี Skeletonization ซึ่งจะทำการค้นหาหน่วยซ่อนที่มีความสำคัญน้อยที่สุดโดยดูจากผลกระทบที่มีต่อค่าผิดพลาดจากการฝึกโครงข่าย จากนั้นจึงกำจัดหน่วยซ่อนที่มีผลกระทบต่อค่าผิดพลาดที่น้อยที่สุดออกไป อย่างไรก็ตาม วิธีการนี้อาจมีความผิดพลาดในการกำจัดหน่วยซ่อนเนื่องจากเป็นการวัดค่าความไม่เกี่ยวข้องของหน่วยซ่อนจากข้อมูลที่ใช้ฝึกโครงข่าย ซึ่งโครงข่ายได้มีโอกาสเห็นชุดข้อมูลนี้แล้วในช่วงของการฝึก ในงานวิจัยนี้เราจะแสดงให้เห็นว่าการคำนวณค่าความไม่เกี่ยวข้องของหน่วยซ่อนจากข้อมูลที่ใช้ในการฝึกอาจไม่ได้บ่งบอกถึงความไม่เกี่ยวข้องที่แท้จริงของหน่วยซ่อน นอกจากนี้ Sietsma และ Dow [4-5] ได้นำเสนอวิธีวิเคราะห์โครงข่ายที่ได้รับการฝึกและฮิวริสติกส์ในการระบุว่าหน่วยซ่อนใดที่ไม่ได้เกี่ยวข้องกับโครงข่าย อย่างไรก็ตาม วิธีนี้ไม่ได้ครอบคลุมการทำงานแบบไม่เชิงเส้นภายในโครงข่าย วิธีต่อไปคือ Optimal brain damage (OBD) [6] และ Optimal brain surgeon (OBS) [7] ซึ่ง OBSถูกพัฒนาต่อจาก OBD ทั้งสองวิธีนี้จะทำการคำนวณค่าความสำคัญของแต่ละหน่วยซ่อนและน้ำหนักภายในโครงข่าย ค่าดังกล่าวนี้สามารถบอกถึงระดับของการเปลี่ยนแปลงของค่าผิดพลาดที่เกิดจากการฝึกเมื่อหน่วยซ่อนนั้นถูกกำจัดออกไป อย่างไรก็ดี สมมุติฐานของ OBD ซึ่งบอกว่าเฮสเซี่ยนแมทริกซ์ (Hessian matrix) เป็นลักษณะของแนวทะแยงอาจไม่ถูกต้องนักเนื่องจากเฮสเซี่ยนแมทริกซ์สำหรับหลายๆปัญหาไม่ได้มีลักษณะเป็นแนวทะแยงแต่อย่างใด ทำให้ OBD อาจจะกำจัดหน่วยซ่อนหรือน้ำหนักผิดพลาด ถึงแม้ว่า OBS ได้แก้ไขปัญหานี้ของ OBD แต่ OBS ก็ไม่เหมาะสำหรับปัญหาที่ซับซ้อนมากๆซึ่งจะต้องใช้โครงข่ายที่มีขนาดใหญ่

Murase และคณะวิจัย [8] ได้ทำการวัดค่า *Goodness factor* สำหรับหน่วยซ่อนทุกหน่วยที่อยู่ในโครงข่ายที่ถูกฝึกแล้ว ค่า *Goodness factor* นี้ คือค่าเฉลี่ยของแอ็คติเวชั่นที่มีทิศทางออกจาก

หน่วยซ่อนนั้นไปยังชั้นต่อไป หน่วยซ่อนที่มีค่า *Goodness factor* ที่ต่ำที่สุดจะถูกกำจัดออกไปจากโครงข่าย Shahjahan และคณะวิจัย [9] ได้พัฒนาต่อจากวิธีใน [8] โดยคิดค้นวิธีการกำจัดโหนดแบบไดนามิค (Dynamic node decaying: DNDM) และ Hagiwara [10] ได้นำเสนอวิธี Consuming energy และ Weights power ซึ่งใช้สำหรับกำจัดทั้งหน่วยซ่อนและน้ำหนัก อย่างไรก็ตาม ตามข้อมูลที่ปรากฏอยู่ในงานวิจัย [12] วิธีที่นำเสนอใน [8 – 10] อาจเกิดการผิดพลาดจากหน่วยซ่อนซึ่งมีค่าเอาท์พุทที่มีค่าเป็นศูนย์บ่อยครั้งกว่าหนึ่งซึ่งหน่วยซ่อนนั้นอาจถูกเลือกเป็นหน่วยที่ไม่สำคัญทั้งๆที่หน่วยซ่อนนั้นอาจมีความสำคัญกับโครงข่าย นอกจากนี้ Hagiwara [11] ยังได้เสนอ *Badness factor* เพื่อใช้ระบุหน่วยซ่อนที่แย่ที่สุดในระหว่างการฝึกโครงข่าย หน่วยซ่อนที่มีค่า *Badness factor* มากที่สุดจะถูกพิจารณาว่าเป็นหน่วยซ่อนที่แย่ที่สุดและถูกกำจัดออกไป Engelbrecht [12] นำเสนอฮิวริสติกแบบใช้สถิติโดยหาความสัมพันธ์ระหว่างหน่วยซ่อนและค่าแปรผัน (Variance) ของค่าตัวชี้วัดความสำคัญ (Sensitivity) เมื่อหน่วยซ่อนถูกกำจัดออกไป หน่วยซ่อนที่มีค่าแปรผันของค่าตัวชี้วัดความสำคัญใกล้เคียงศูนย์จะถูกกำจัดออกไป วิธีใน [11-12] อาจจะประสบกับปัญหาเช่นเดียวกับวิธีใน [3] เนื่องจากการวิเคราะห์ค่าตัวชี้วัดความสำคัญใช้ข้อมูลที่มาจากค่าผิดพลาดที่เกิดจากการฝึกโครงข่ายซึ่งอาจขาดความแม่นยำในการเลือกหน่วยซ่อนที่ไม่สำคัญ นอกจากนี้วิธีใน [12] ยังไม่ได้นำเสนอวิธีการหยุดอัลกอริธึมที่ใช้หลักการทางสถิติ แต่กลับใช้กฎพื้นฐานง่ายๆว่าให้หยุดการทำงานถ้าโครงข่ายเริ่มสูญเสียความสามารถในการทำนายข้อมูล เมื่อไม่นานมานี้ Lauret และคณะวิจัย [13] ได้นำเสนออัลกอริธึมในการกำจัดหน่วยภายโครงข่ายสำหรับโครงข่ายประสาทเทียมที่มีชั้นซ่อนเพียงหนึ่งชั้น โดยคำนวณค่าความไม่สำคัญของหน่วยซ่อนจากการวิเคราะห์ฟุริเย (Fourier) ของค่าแปรผันของหน่วยเอาท์พุท อย่างไรก็ตาม งานวิจัยใน [13] สนใจเพียงแค่การลดจำนวนหน่วยซ่อนและเวลาที่ใช้ในการกำจัดหน่วยซ่อนโดยไม่ได้เน้นเรื่องความสามารถในการทำนายข้อมูลของโครงข่าย

2.2 นิยามของคำที่เกี่ยวข้องกับงานวิจัย

ก่อนที่จะนำเสนออัลกอริธึมสำหรับการลดจำนวนหน่วยซ่อน เรานิยามคำต่างที่เกี่ยวข้อง เช่น ค่าความสูญเสียเจนเนอรัลไลเซชั่น (Generalization loss) ตัวชี้โอเวอร์พรุนนิ่ง (Overpruning Indicator) และ ค่าผลต่างของแอ็คติเวชั่น (Activation difference) ดังนี้

2.2.1 ค่าความสูญเสียเจนเนอรัลไลเซชั่น (Generalization Loss: *GL*)

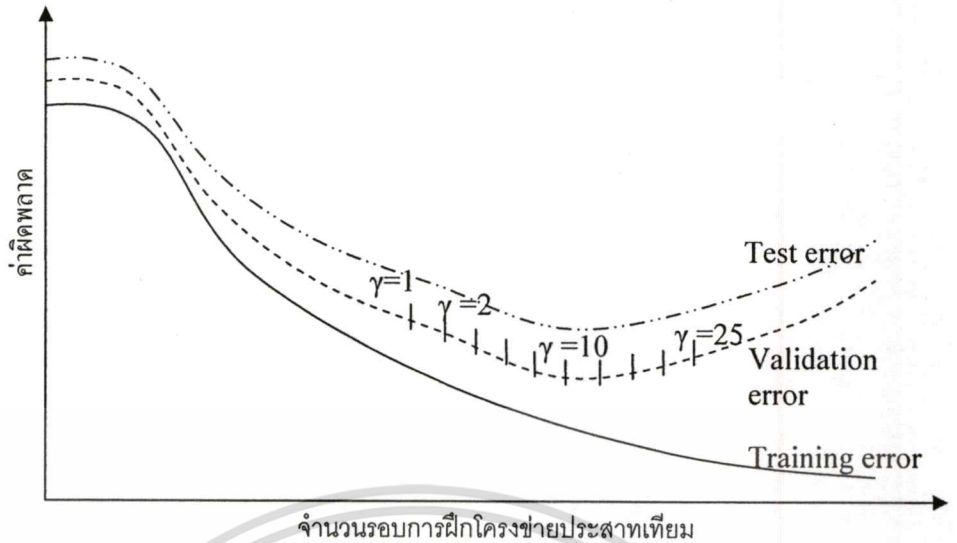
L. Prechelt ได้นำเสนอแนวคิดของค่าความสูญเสียเจนเนอรัลไลเซชั่นในงานวิจัย [14] โดยแนวคิดคือการหยุดขบวนการฝึกโครงข่ายประสาทเทียมก่อนกำหนดเพื่อหลีกเลี่ยงปัญหาการฝึกโครงข่ายมากจนเกินไป ซึ่งจะทำให้ความสามารถในการทำนายของโครงข่ายแย่ลงได้ ค่าความสูญเสียเจนเนอรัลไลเซชั่นที่รอบการฝึกที่ t ถูกนิยามดังต่อไปนี้

$$E_{opt}(t) = min_{\,t' \le t} E_{va}(t') \tag{2.1}$$

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right) \tag{2.2}$$

โดยที่ $E_{va}(t')$ คือค่าผิดพลาดที่เกิดจากชุดข้อมูลแวลิเดชั่นที่รอบที่ t' โดยที่ t' มีค่าเริ่มต้นจากเวลาศูนย์ ขณะที่ $E_{opt}(t)$ คือค่าผิดพลาดต่ำสุดของชุดข้อมูลแวลิเดชั่นภายในช่วงเวลา t รอบก่อนหน้านี้ โดยทั่วไปแล้ว ค่า GL(t) เป็นสัดส่วนระหว่างค่าผิดพลาดปัจจุบันกับค่าผิดพลาดที่ต่ำที่สุดระหว่างช่วงเวลา t รอบ เมื่อค่าความสูญเสียเจนเนอรัลไลเซชั่นมีค่าสูงแสดงว่าโครงข่ายเริ่มสูญเสียความสามารถในการทำนายข้อมูล ซึ่งเป็นจุดที่ควรจะหยุดการฝึกโครงข่าย Prechelt แนะนำว่าการฝึกโครงข่ายควรจะหยุดการฝึกฝนเมื่อค่าความสูญเสียเจนเนอรัลไลเซชั่นมีค่าเกินค่าๆหนึ่งที่กำหนดโดยผู้ใช้งาน ซึ่งอาจไม่เหมือนกันในแต่ละปัญหา โดยกำหนดให้ช่วงเวลาที่ใช้ฝึก(Training strip) โครงข่ายมีความยาวเท่ากับ γ รอบ (Epochs) แนวคิดเรื่องค่าความสูญเสียเจนเนอรัลไลเซชั่นยังสนับสนุนข้อเท็จจริงเรื่องการฝึกโครงข่ายมากจนเกินไป (Overtraining) ซึ่งนำเสนอโดย Chauvin [15] รูปแบบของการฝึกโครงข่ายมากจนเกินความจำเป็นนี้เป็นปัญหาที่เกิดขึ้นบ่อยกับการฝึกโครงข่ายประสาทเทียมโดยทั่วไป

รูปที่ 2.1 แสดงสถานการณ์เมื่อโครงข่ายถูกฝึกมากจนเกินไป เราทำการวัดค่าความสูญเสียเจนเนอรัลไลเซชั่นทุกๆรอบภายในช่วงของการฝึก γ รอบ ถ้าความยาวของช่วงการฝึกนั้นสั้นจนเกินไป การทำนายการเกิดความสูญเสียเจนเนอรัลไลเซชั่นอาจเกิดข้อผิดพลาดอันเนื่องมาจากการตอบสนองต่อการเปลี่ยนแปลงของค่าผิดพลาดมีความไวมากจนเกินไป ในทางตรงกันข้าม ถ้าเราฝึกโครงข่ายนานจนเกินไป เราอาจสูญเสียเวลาที่ใช้ในการตรวจจับค่าความสูญเสียเจนเนอรัลไลเซชั่น สมมุติว่าเราต้องการคำนวณค่าความสูญเสียเจนเนอรัลไลเซชั่นที่ตำแหน่ง γ =25 ดังรูปที่ 2.1 โดยให้ค่าผิดพลาดของชุดข้อมูลแวลิเดชั่นที่ γ =10 (จุดที่มีค่าผิดพลาดต่ำสุด) และ γ =25 (จุดปัจจุบัน) มีค่าเป็น 0.20 และ 0.25 ตามลำดับ ค่าความสูญเสียเจนเนอรัลไลเซชั่นที่ตำแหน่ง γ =25 คือ 100*(0.25/0.20 – 1) = 25 นอกจากนี้ ในส่วนของการทดลอง เราได้นำเสนอวิธีการได้มาซึ่งค่า γ ที่เหมาะสมในหัวข้อการทดลองของงานวิจัยนี้



รูปที่ 2.1 สถานการณ์เมื่อค่าผิดพลาดจากทั้งชุดข้อมูลแวลิเดชั่นและชุดข้อมูลทดสอบมีค่าลดลงจนถึงค่าต่ำสุดและเริ่มที่จะมีค่าเพิ่มขึ้น ขณะที่ค่าผิดพลาดที่เกิดจากชุดข้อมูลฝึกยังมีค่าลดลงเรื่อยๆ

2.2.2 ตัวชี้โอเวอร์พรุนนิ่ง (Overpruning Indicator: *OI*)

โอเวอร์พรุนนิ่ง (Overpruning) เป็นสถานการณ์เมื่อเรากำจัดหน่วยซ่อนภายในโครงข่ายประสาทเทียมมากจนเกินไป ทำให้โครงข่ายมีจำนวนน้ำหนัก (Weights) ไม่เพียงพอที่จะเรียนรู้ชุดข้อมูล เราใช้ตัวชี้โอเวอร์พรุนนิ่งในการเตือนเมื่อโครงข่ายมีจำนวนหน่วยซ่อนไม่เพียงพอต่อการเรียนรู้ ตัวชี้โอเวอร์พรุนนิ่งถูกนิยามดังนี้

$$OI = \frac{(ValidErrRate_{AfterPruning} - ValidErrRate_{BeforePruning})}{ValidErrRate_{BeforePruning}} \times 10 \qquad (2.3)$$

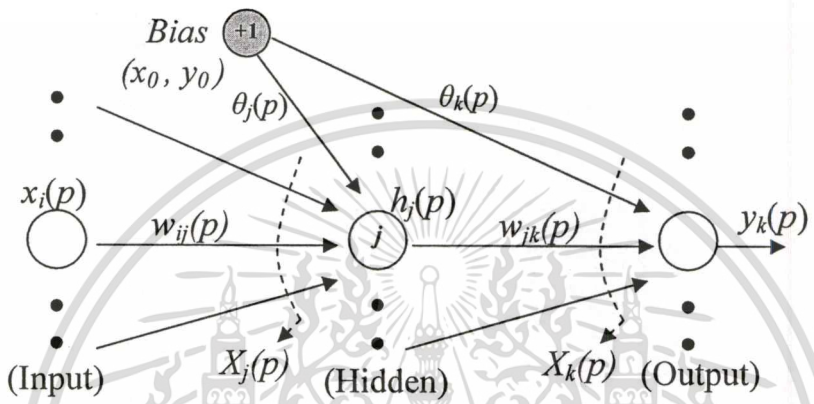
ตัวชี้โอเวอร์พรุนนิ่งถูกคำนวณจากอัตราส่วนของผลต่างระหว่างอัตราความผิดพลาดแวลิเดชั่นก่อน(*ValidErrRateBeforePruning*) และหลังการกำจัดหน่วยซ่อน(*ValidErrRateAfterPruning*) ต่ออัตราผิดพลาดแวลิเดชั่นก่อนกำจัดหน่วยซ่อน สิ่งสำคัญสำหรับตัวชี้โอเวอร์พรุนนิ่งคือค่าเทรชโฮลด์ (Threshold) ซึ่งจะเป็นตัวฟ้องว่าโครงข่ายเกิดโอเวอร์พรุนนิ่งหรือยัง การหาค่าที่เหมาะสมของเทรชโฮลด์ถูกอธิบายไว้ในหัวข้อ 3.3 ของบทที่ 3

2.2.3 ผลต่างของแอ็คติเวชั่น (Activation Difference: *AD*)

ค่าผลต่างของแอ็คติเวชั่น (Activation difference) ถูกใช้ในการวัดความไม่เกี่ยวข้องของหน่วยซ่อนภายในโครงข่ายประสาทเทียม ค่าผลต่างของแอ็คติเวชั่นนี้เป็นผลต่างระหว่างค่าแอ็คติ

เวชั่นของทุกเอาท์พุทยูนิตระหว่างโครงข่ายที่ยังไม่ได้กำจัดหน่วยซ่อนและโครงข่ายที่ถูกกำจัดหน่วยซ่อนใดๆออกไปแล้ว ทั้งนี้กระบวนการดังกล่าวกระทำกับชุดข้อมูลแวลิเดชั่น

เพื่อเป็นการลดปัญหาการผ่านชุดข้อมูลแวลิเดชั่นสองครั้ง การคำนวณผลต่างของค่าแอ็คติเวชั่นจึงถูกฝังอยู่ในโมดูล ValidationError ซึ่งเป็นส่วนของอัลกอริธึมที่คำนวณค่าผิดพลาดที่เกิดจากชุดข้อมูลแวลิเดชั่นเพื่อที่จะระบุจุดที่ควรจะหยุดฝึกโครงข่ายก่อนเวลา (Early stopping) รูปที่ 2.2 แสดงสัญลักษณ์ที่ใช้ภายในโครงข่าย



รูปที่ 2.2 สัญลักษณ์ภายในโครงข่าย

กำหนดให้ $X_j(p)$ คืออินพุทรวมของหน่วยซ่อน j สำหรับแพทเทิร์น (Pattern) p, $h_j(p)$ คือเอาท์พุทของหน่วยซ่อน j, $X_k(p)$ คืออินพุทรวมของหน่วยเอาท์พุทสำหรับแพทเทิร์น p, $y_k(p)$ คือค่าแอ็คติเวชั่นของหน่วย k และ φ คือแอ็คติเวชั่นฟังก์ชั่น ดังสมการดังต่อไปนี้

$$X_j(p) = \sum_i w_{ij}(p) \cdot x_i(p) + \theta_j(p) \tag{2.4}$$

$$h_j(p) = \varphi(X_j(p)) \tag{2.5}$$

$$X_k(p) = \sum_j w_{jk}(p) \cdot h_j(p) + \theta_k(p) \tag{2.6}$$

$$y_k(p) = \varphi(X_k(p)) \tag{2.7}$$

หลังจากที่ $X_k(p)$ ถูกคำนวณแล้ว เรานำอินพุทของหน่วยซ่อน j ไปลบออกจาก $X_k(p)$ เพื่อที่จะแทนการขาดหายไปของหน่วยซ่อน j เราเรียก $X_k(p)$ หลังจากขั้นตอนนี้ว่า $X_{k-j}(p)$ และค่าแอ็คติเวชั่นของหน่วยเอาท์พุท k เมื่อหน่วยซ่อน j ถูกกำจัดออกว่า $y_{k-j}(p)$

$$X_{k-j}(p) = X_k(p) - w_{jk}(p) \cdot h_j(p) \tag{2.8}$$

$$y_{k-j}(p) = \varphi(X_{k-j}(p)) \tag{2.9}$$

เราสามารถทราบระดับของความไม่เกี่ยวข้องของหน่วยซ่อน j โดยการเปรียบเทียบกันระหว่างค่าสมบูรณ์ (Absolute) ของผลต่างระหว่างค่าที่ต้องการ (Desirable value) ของหน่วยเอาท์พุท k หรือ $O_k(p)$ กับค่าแอ็คติเวชั่นของหน่วยเอาท์พุท k เมื่อหน่วยซ่อน j ถูกกำจัด หรือ $y_{k-j}(p)$ นิยามของค่าผลต่างของแอ็คติเวชั่นสามารถอธิบายได้ดังสมการดังต่อไปนี้

$$AD_{k-j}(p) = \left|O_k(p) - y_{k-j}(p)\right| \tag{2.10}$$

ในกรณีที่โครงข่ายมีหน่วยเอาท์พุทมากกว่าหนึ่งหน่วย เราจะต้องคำนวณหาผลรวมของค่าผลต่างของแอ็คติเวชั่น $AD_j(p)$ ของทุกๆหน่วยเอาท์พุทด้วย

$$AD_j(p) = \sum_k AD_{k-j}(p) \tag{2.11}$$

จากนั้นเราทำการหาผลรวมของค่าผลต่างของแอ็คติเวชั่นสำหรับชุดข้อมูลแวลิเดชั่นทั้งหมด ซึ่งเราจะเรียกว่า AD_j และรายงานค่าความไม่เกี่ยวข้องของหน่วยซ่อนต่ออัลกอริธึมเพื่อกำจัดหน่วยซ่อนนั้นออกไป โดยหน่วยซ่อนที่มีค่าผลต่างของแอ็คติเวชั่น (AD_j) ต่ำที่สุดจะถูกกำจัดออกไปก่อน

$$AD_j = \sum_p AD_j(p) \tag{2.12}$$

ถ้าค่าผลต่างของแอ็คติเวชั่นของหน่วยซ่อน j (AD_j) มีค่ามาก หน่วยซ่อน j จะมีความสำคัญต่อโครงข่าย อย่างไรก็ตาม อาจเกิดกรณีที่ ค่าผิดพลาดของโครงข่ายมีค่ามากเมื่อหน่วยซ่อน j ถูกกำจัดออกไป นั่นคือ $\left|O_k(p) - y_k(p)\right|$ มีค่ามาก ในกรณีนี้หน่วยซ่อน j ยังคงต้องถูกกำจัดทิ้งถ้าหน่วย j มีค่าผลต่างของแอ็คติเวชั่น (AD_j) น้อยที่สุดเมื่อเทียบกับหน่วยซ่อนอื่นๆ เหตุผลก็คือว่าเราตัดสินใจที่จะกำจัดหน่วยซ่อน ณ จุดซึ่งโครงข่ายเริ่มสูญเสียความสามารถในการทำนายข้อมูล ดังนั้นโครงข่ายจึงไม่ควรถูกฝึกต่อไปโดยไร้ซึ่งการเปลี่ยนแปลงภายในโครงข่ายเนื่องจากความสามารถในการทำนายข้อมูลจะยิ่งแย่ไปกว่าเดิม ในกรณีที่ดีที่สุดโครงข่ายที่ถูกกำจัดหน่วยซ่อนจะปรับตัวเองให้ทำนายข้อมูลได้ถูกต้องมากขึ้นตามหลักการที่เสนอไว้ใน [2] ในกรณีที่แย่ที่สุดถ้าหน่วยซ่อน j มีความสำคัญต่อโครงข่าย ค่าผิดพลาดแวลิเดชั่น (Validation error) จะเพิ่มขึ้น

อย่างเห็นได้ชัดเจนและก็จะเกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปซึ่งก็คือสถานการณ์ที่จำนวนของหน่วยซ่อนมีไม่เพียงพอต่อการฝึกโครงข่าย

สำหรับสถานการณ์ที่บางหน่วยซ่อนไม่มีความสำคัญกับโครงข่ายแต่กลับมีค่า AD_j มาก หน่วยซ่อนดังกล่าวอาจจะรอดพ้นจากการถูกกำจัดในช่วงแรกของการพรุนนิ่ง อย่างไรก็ตาม ค่า AD_j ของหน่วยซ่อนนี้จะมีค่าน้อยลงเรื่อยๆจนกระทั่งมีค่าน้อยที่สุดเมื่อเทียบกับหน่วยซ่อนอื่นๆ ซึ่งก็จะทำให้หน่วยซ่อนนี้ถูกกำจัดออกไปจากโครงข่ายในที่สุด นอกจากนี้ ค่า AD_j ของหน่วยซ่อนที่มีความสำคัญมักจะมีค่ามากกว่าค่า AD_j ของหน่วยซ่อนที่ไม่สำคัญ เนื่องจากอัลกอริธึมแบ็คพรอพาเกชั่นจะควบคุมน้ำหนักที่เชื่อมต่อกับหน่วยซ่อนที่ไม่สำคัญจนกระทั่งน้ำหนักเหล่านี้มีค่าน้อย ทำให้หน่วยซ่อนที่ไม่สำคัญนี้ไม่มีส่วนร่วมกับการทำงานของโครงข่าย ดังนั้นหน่วยซ่อนที่ไม่สำคัญที่สุดจะให้ค่าผลต่างระหว่างค่าเอาท์พุทที่ต้องการกับค่าแอ็คติเวชั่นเมื่อหน่วยซ่อนนั้นถูกกำจัดออกไปน้อยที่สุด

อาจมีคำโต้แย้งว่าการคำนวณ AD_j นั้นใช้เวลานาน อย่างไรก็ตามเราคำนวณ AD_j ภายในโมดูล ValidationError โดยไม่ให้มีผลกระทบต่อความซับซ้อนของโมดูลดังกล่าว เพื่อเป็นการพิสูจน์ เราแสดงการวิเคราะห์ความซับซ้อนของเวลากับโปรแกรมเทียมสำหรับโมดูล ValidationError ที่ถูกดัดแปลงในรูปที่ 2.3 สำหรับโปรแกรมที่เพิ่มเข้าไปเพื่อคำนวณผลรวมของค่าผลต่างของแอ็คติเวชั่นสำหรับแต่ละหน่วยซ่อน (AD_j) แสดงอยู่ในบรรทัดที่ 8-14 ในบรรทัดที่ 11 เราหลีกเลี่ยงการคำนวณ $X_{k-j}(p)$ อีกครั้งโดยการลบ $X_k(p)$ ซึ่งถูกคำนวณไว้แล้วในบรรทัดที่ 6 ด้วยค่าอินพุทที่มาจากหน่วยซ่อน j

```
Module ValidationError()
(Precondition: ADj must be initialized to zero)
1 For every valid. pattern p
2         For every hidden unit j
3              Xj (p)  =  Σ_i w_ij(p) · x_i(p) + θ_j(p)        /* PJI */
4              hj (p)  =  φ(Xj (p))
5         For every output unit k
6              Xk (p)  =  Σ_j w_jk(p) · h_j(p) + θ_k(p)        /* PKJ */
7              yk (p)  =  φ(Xk (p))
8         For every hidden unit j
9              ADj (p)  =  0
10             For every output unit k
11                Xk-j (p)  =  Xk (p)  - w_jk(p) · h_j(p)
12                yk-j (p)  =  φ(Xk-j (p))
13                ADj (p) =  ADj(p) + |Ok(p) - yk_j(p)|        /* PJK */
14             ADj  =  ADj  +  ADj (p)
/* Proceed to the recognition error rate calculation */
End.
```

รูปที่ 2.3 การดัดแปลงโมดูล ValidationError

กำหนดให้ I คือจำนวนของหน่วยอินพุท, J คือจำนวนของหน่วยซ่อน, K คือจำนวนของหน่วยเอาท์พุท, และ P คือจำนวนของแพทเทิร์นแวลิเดชั่น (Validation patterns) ความซับซ้อนของโปรแกรมทั้งสองเวอร์ชั่นเป็นดังนี้

O(Original module) = O(PJI + PKJ)

O(Modified module) = O(PJI+PKJ+PJK)

= O(PJI+2PKJ)

= O(PJI + PKJ)

เนื่องจากความซับซ้อนของทั้งสองเวอร์ชั่นเหมือนกัน ดังนั้นโปรแกรมที่เพิ่มเข้าไปในบรรทัดที่ 8-14 จึงไม่มีผลต่อความซับซ้อนของโมดูลเดิม เมื่อค่าความสูญเสียเจนเนอรัลไลเซชั่นถูกคำนวณอัลกอริธึมจะสั่งให้โมดูล ValidationError ทำงาน ซึ่งก็จะทำงานโปรแกรมที่ใช้คำนวณค่าผลต่างของแอ็คติเวชั่นไปพร้อมๆกัน ดังนั้นจึงเป็นการคำนวณค่าสองค่าโดยใช้ชุดข้อมูลแวลิเดชั่นเพียงแค่ครั้งเดียว เมื่อเทียบกับการคำนวณ AD_j แบบปรกติโดยไม่ได้ใช้เทคนิคที่นำเสนอ โครงข่ายประสาทเทียมจะต้องใช้เวลาในการคำนวณเท่ากับ O(PJ(JI + KJ))

นอกจากวิธีการคำนวณ AD_j นี้จะใช้ได้กับโครงข่ายประสาทเทียมที่มีชั้นซ่อนเพียงชั้นเดียวแล้ว วิธีการนี้ยังสามารถนำไปใช้กับโครงข่ายที่มีชั้นซ่อนหลายๆชั้นได้อีกด้วย ซึ่งวิธีการคำนวณก็จะมีความคล้ายคลึงกัน แต่ต่างกันตรงที่จะต้องมีการจองหน่วยความจำเพิ่มเติมสำหรับบันทึกค่า X_k สำหรับทุกๆหน่วยเอาท์พุท k และ X_j สำหรับทุกๆหน่วยซ่อน j ค่าเหล่านี้ช่วยให้เราสามารถหลีกเลี่ยงการคำนวณ X_{k-j} ทั้งหมดใหม่ตั้งแต่ต้น ส่วนการกำจัดน้ำหนักภายในโครงข่ายประสาทเทียม เรายังสามารถใช้วิธีการเดิมในการคำนวณ AD_j แต่ต้องมีเงื่อนไขสองข้อเพิ่มเติม คือ ข้อหนึ่ง ถ้า $W_{ij}(p)$ ถูกกำจัด เราจะต้องคำนวณ $h_j(p)$ และ $y_k(p)$ ที่เชื่อมกับ $h_j(p)$ ใหม่ ข้อสอง ถ้า $w_{jk}(p)$ ถูกกำจัด เราจะต้องคำนวณ $y_k(p)$ ที่เชื่อมกับ $w_{jk}(p)$ ใหม่อีกครั้ง

2.3 ความแตกต่างของงานวิจัยกับหลักการที่มีอยู่

เมื่อเปรียบเทียบกับหลักการที่มีอยู่เดิม อัลกอริธึมสำหรับลดจำนวนหน่วยซ่อนที่นำเสนอในวิทยานิพนธ์นี้แตกต่างจากหลักการที่มีอยู่แล้วดังประเด็นต่อไปนี้

ก) อัลกอริธึมที่นำเสนอมีความเรียบง่ายในการทำงาน เนื่องจากแนวคิดของวิธีนี้ตั้งอยู่บนพื้นฐานของสามัญสำนึก คือ กำจัดหน่วยซ่อนเมื่อโครงข่ายประสาทเทียมเริ่มสูญเสียความสามารถในการทำนายข้อมูล และหากหน่วยซ่อนถูกกำจัดมากจนเกินไป อัลกอริธึมก็จะคืนโครงสร้างเดิมของโครงข่ายกลับมาให้

ข) อัลกอริธึมที่นำเสนอหลีกเลี่ยงที่จะคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนด้วยวิธีการที่ซับซ้อนดังเช่นวิธีการที่นำเสนอใน [6, 7] ซึ่งประมาณค่าตัวชี้ความสำคัญจากสมการ (2.13) และ (2.14) สมการทั้งสองเป็นการคำนวณหาเดริเวชั่นอันดับสองของอ็อบเจ็คทีฟฟังก์ชั่น E โดยที่ h_{ii} และ δu_i คือเฮสเซียนแมทริกซ์และค่าความผิดพลาดจากการกำจัดพารามิเตอร์ i ตามลำดับ

$$\delta E = \frac{1}{2}\sum_i h_{ii}\delta u_i^2 \qquad (2.13)$$

$$h_{ii} = \frac{\partial^2 E}{\partial u_i \partial u_i} \qquad (2.14)$$

ในทางตรงข้าม เราเลือกคำนวณค่าชี้ความสำคัญของหน่วยซ่อนจาก AD_j โดยตรง ซึ่งเป็นการหลีกเลี่ยงความซับซ้อนของการคำนวณเดริเวชั่นอันดับสองของอ็อบเจ็คทีฟฟังก์ชั่นข้างต้น

ค) อัลกอริธึมที่นำเสนอยังคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนจากชุดข้อมูลแวลิเดชั่นซึ่งสามารถแทนชุดข้อมูลที่แท้จริงได้ดีกว่าชุดข้อมูลที่ใช้ฝึก ดังนั้นค่าตัวชี้วัดความสำคัญของหน่วยซ่อนที่คำนวณจากชุดข้อมูลแวลิเดชั่นจึงมีความแม่นยำกว่าค่าที่คำนวณมาจากชุดข้อมูลฝึก หากหลักการที่มีอยู่จะเปลี่ยนมาใช้ชุดข้อมูลแวลิเดชั่นเพื่อคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนก็ไม่ใช่เป็นเรื่องง่ายที่จะทำได้ เนื่องจากชุดข้อมูลแวลิเดชั่นจะถูกนำไปรวมกับชุดข้อมูลฝึกเพื่อนำไปฝึกโครงข่ายและคำนวณค่าความสำคัญของหน่วยซ่อน (การคำนวณค่าความสำคัญของหน่วยซ่อนของวิธีอื่นๆ เป็นการคำนวณสะสมระหว่างการฝึกโครงข่าย) ส่งผลให้การหาหน่วยซ่อนที่ไม่สำคัญมีความแม่นยำขึ้นจากเดิม อย่างไรก็ตาม โครงข่ายประสาทเทียมที่ถูกฝึกด้วยชุดข้อมูลทั้งสองจะพยายามจดจำข้อมูลทั้งหมด ทำให้เกิดปัญหาการฝึกโครงข่ายมากจนเกินไป (Overfitting) นอกจากนี้ เรายังไม่สามารถใช้ชุดข้อมูลแวลิเดชั่นมาหยุดการฝึกโครงข่ายก่อนที่จะเกิดปัญหาการฝึกโครงข่ายมากจนเกินไปได้ เนื่องจากโครงข่ายได้เห็นชุดข้อมูลแวลิเดชั่นในช่วงการฝึกมาแล้ว ส่งผลให้การหาหน่วยซ่อนที่ไม่สำคัญมีประสิทธิภาพไม่เต็มที่ ในบทการวิเคราะห์ผลการทดลองของวิทยานิพนธ์นี้ได้มีการทดลองเพื่อสนับสนุนคำกล่าวนี้

ง) โดยทั่วไป ชุดข้อมูลแวลิเดชั่นจะถูกใช้ในการตรวจสอบความถูกต้องของโครงข่ายประสาทเทียมก่อนที่จะนำไปทดสอบด้วยชุดข้อมูลทดสอบเป็นขั้นตอนสุดท้าย โดยชุดข้อมูลแวลิเดชั่นสามารถสลับกับชุดข้อมูลฝึกได้ ซึ่งวิธีการสลับชุดข้อมูลทั้งสองนี้ถูกเรียกว่าการทำแวลิเดชั่นแบบข้าม (Cross validation) อย่างไรก็ตาม ในงานวิจัยที่นำเสนอในวิทยานิพนธ์นี้ไม่ได้ใช้กระบวนการทำแวลิเดชั่นแบบข้ามแต่อย่างใด หากแต่เป็นการฝึกโครงข่ายประสาทเทียมด้วยข้อมูลชุดฝึกเพียงอย่างเดียว หลังจากนั้นจะทำการคำนวณค่าผลต่างของแอ็คติเวชั่นด้วยชุดข้อมูลแวลิเดชั่นซึ่งแยกไว้ต่างหาก

จากชุดข้อมูลฝึก หลังจากที่อัลกอริธึมกำจัดหน่วยซ่อนเสร็จเรียบร้อยแล้ว โครงข่ายประสาทเทียมที่ได้จะถูกทดสอบด้วยชุดข้อมูลทดสอบเป็นขั้นตอนสุดท้าย

หลังจากที่ได้ทบทวนงานวิจัยอื่นๆที่เกี่ยวข้องและศึกษานิยามของคำที่เกี่ยวข้องกับงานวิจัยที่ในนำเสนอ ในบทต่อไปจะเป็นการนำเสนออัลกอริธึมสำหรับการลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียมซึ่งมีความเกี่ยวข้องกับคำต่างๆที่นิยามไว้ในบทนี้โดยเฉพาะ ค่าความสูญเสียเจนเนอรัลไลเซชั่น, ตัวชี้โอเวอร์พรุนนิ่งและค่าผลต่างของแอ็คติเวชั่น

บทที่ 3

อัลกอริธึมการลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียม

3.1 อัลกอริธึมสปาร์ตันซิมพลิซิตี

สำหรับอัลกอริธึมการลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียมที่นำเสนอในวิทยานิพนธ์นี้ถูกเรียกว่าอัลกอริธึมสปาร์ตัน[1]ซิมพลิซิตี (Spartan simplicity algorithm) การทำงานของอัลกอริธึมนี้ถูกอธิบายในรูปที่ 3.1 พร้อมกับผังงานในรูปที่ 3.2

ขั้นตอน 1: ฝึกโครงข่ายเริ่มต้นเป็นจำนวน γ รอบ

ขั้นตอน 2: ถ้าค่าความสูญเสียเจนเนอรัลไลเซชั่นไม่เกินค่า α ให้ย้อนกลับไปยังขั้นตอนที่ 1 มิฉะนั้น หยุดฝึกโครงข่ายและทำต่อไปในขั้นตอนที่ 3

ขั้นตอน 3: คืนสถานะที่ดีที่สุด (สถานะที่มีค่าอัตราผิดพลาดแวลิเดชั่นต่ำที่สุด) ภายใน γ รอบก่อนหน้านี้ ให้เป็นสถานะปัจจุบันของโครงข่าย

ขั้นตอน 4: เก็บรักษาน้ำหนักปัจจุบันและหน่วยซ่อน ก่อนที่จะทำการกำจัดหน่วยซ่อน

ขั้นตอน 5: กำจัดหน่วยซ่อน j ที่มีค่าผลต่างของแอ็คติเวชั่น (AD_j) น้อยที่สุด

ขั้นตอน 6: ฝึกโครงข่ายปัจจุบันต่ออีก γ รอบ

ขั้นตอน 7: ถ้าค่าความสูญเสียเจนเนอรัลไลเซชั่นมีค่าไม่เกินค่า α และตัวชี้โอเวอร์พรุนนิ่งมีค่าน้อยกว่าค่าเทรชโฮลด์ β (เมื่อเทียบกับอัตราผิดพลาดแวลิเดชั่นในขั้นตอนที่ 3) ให้ฝึกโครงข่ายต่ออีก γ รอบโดยย้อนกลับไปยังขั้นตอนที่ 6 มิฉะนั้นให้หยุดฝึกโครงข่ายและทำงานต่อไปในขั้นตอนที่ 8

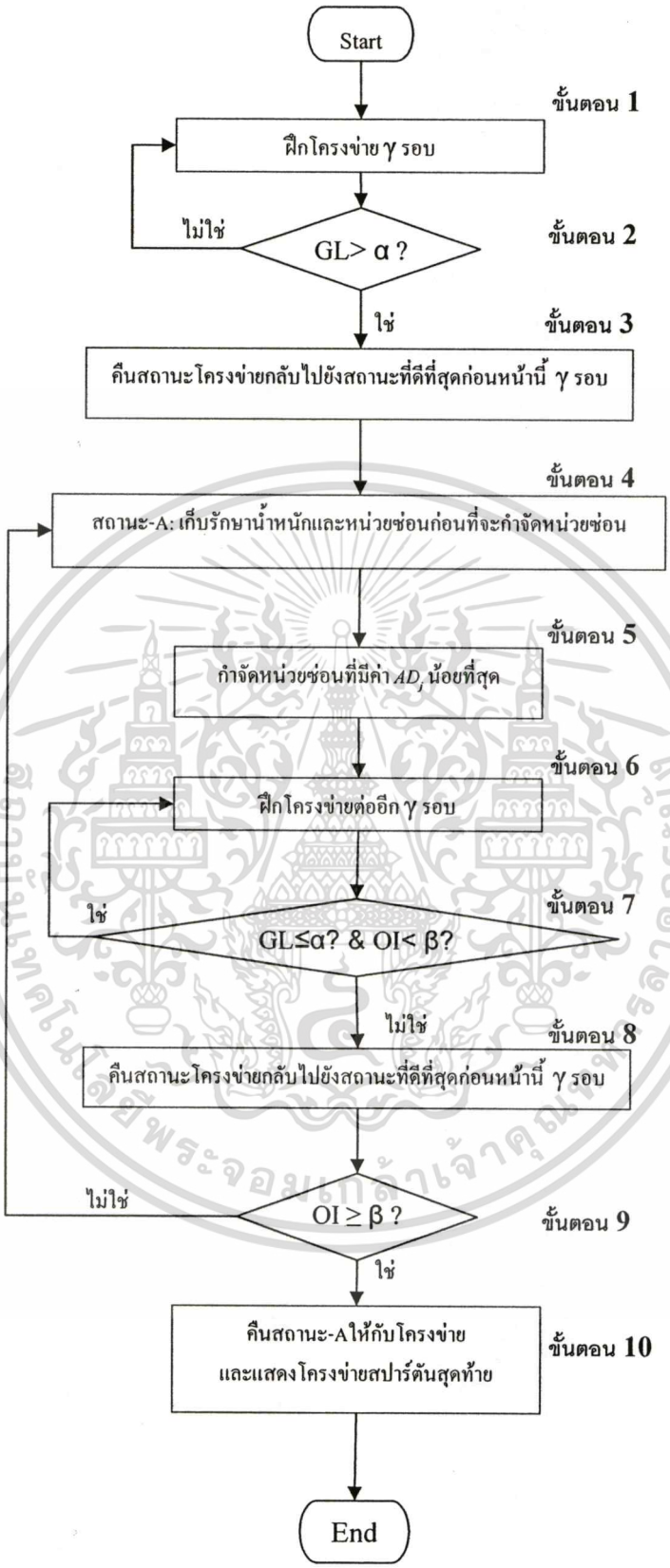
ขั้นตอน 8: คืนสถานะที่ดีที่สุด (สถานะที่มีค่าอัตราผิดพลาดแวลิเดชั่นต่ำที่สุด) ภายใน γ รอบก่อนหน้านี้เพื่อนำมาเป็นสถานะปัจจุบัน

ขั้นตอน 9: ถ้าตัวชี้โอเวอร์พรุนนิ่งไม่ได้เพิ่มขึ้นเกิน β (เมื่อเทียบกับค่าอัตราผิดพลาดแวลิเดชั่นในขั้นตอนที่ 3), ให้ทำการกำจัดหน่วยซ่อนถัดไป โดยกลับไปยังขั้นตอนที่ 4 มิฉะนั้นให้ทำต่อไปยังขั้นตอนที่ 10

ขั้นตอน 10: คืนสถานะที่เก็บรักษาไว้ในขั้นตอนที่ 4 ให้กับโครงข่าย และแสดงผลโครงข่ายสปาร์ตันที่ได้

รูปที่ 3.1 อัลกอริธึมสปาร์ตันซิมพลิซิตี

[1] สปาร์ตันเป็นชนเผ่าพื้นเมืองของเมืองสปาร์ต้า (กรีซสมัยโบราณ) มีชื่อเสียงในด้านความเรียบง่าย มีวินัย และเป็นนักรบที่กล้าหาญ



รูปที่ 3.2 ผังงานสำหรับอัลกอริธึมสปาร์ตันซิมพลิซิตี

อัลกอริธึมเริ่มต้นด้วยการฝึกโครงข่ายเริ่มต้นครั้งละ γ รอบ จนกระทั่งโครงข่ายเริ่มที่จะสูญเสียความสามารถในการทำนายข้อมูล โดยสังเกตจากค่าความสูญเสียเจนเนอรัลไลเซชั่นมีค่าสูงเกินค่าเทรชโฮลด์ α จากนั้นอัลกอริธึมจะทำการย้อนกลับไปยังสถานะของโครงข่ายที่ดีที่สุดภายใน γ รอบที่ผ่านมา โดยการเลือกย้อนกลับไปยังสถานะโครงข่ายที่ดีที่สุดจะทำให้โครงข่ายสามารถรักษาประสิทธิภาพได้ดีที่สุดตลอดเวลา

ก่อนที่ขบวนการกำจัดหน่วยซ่อนจะเริ่มขึ้น อัลกอริธึมจะเก็บรักษาค่าน้ำหนักและหน่วยซ่อนเอาไว้เพื่อรองรับการนำโครงข่ายกลับมายังสถานะเดิมในอนาคตอันเนื่องมาจากเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไป เหตุการณ์นี้เป็นสถานการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปซึ่งจะสังเกตได้จากตัวชี้โอเวอร์พรุนนิ่ง (OI) มีค่าสูงขึ้นเกินเทรชโฮลด์ β อัลกอริธึมจะเริ่มกำจัดหน่วยซ่อนโดยกำจัดหน่วยซ่อนที่มีค่าผลต่างของแอ็คติเวชั่น (AD_j) น้อยที่สุดและทำการฝึกโครงข่ายต่ออีกทีละ γ รอบ ถ้าโครงข่ายไม่สูญเสียประสิทธิภาพในการทำนายข้อมูลและไม่เกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปขึ้น อัลกอริธึมจะทำการฝึกโครงข่ายต่อไปอีก γ รอบ มิฉะนั้นโครงข่ายก็จะคืนสถานะกลับไปยังสถานะที่ดีที่สุดก่อนหน้านี้ หากเกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไป (สังเกตที่ค่า OI มีค่ามากกว่าหรือเท่ากับเทรชโฮลด์ β) อัลกอริธึมจะคืนหน่วยซ่อนและน้ำหนักที่ได้เก็บรักษาไว้ให้กับโครงข่าย และแสดงผลโครงข่ายสปาร์ตันสุดท้าย แต่ถ้าเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปไม่เกิดขึ้น อัลกอริธึมก็จะกำจัดหน่วยซ่อนต่อไป สำหรับวิธีการค้นหาค่าพารามิเตอร์ γ, α และ β ที่เหมาะสมถูกอธิบายในส่วนของการทดลอง

สำหรับปัญหาการจำแนกประเภท (Classification problem) เราควรจะให้ความสำคัญกับจำนวนของข้อมูลที่ทำนายผิดพลาดเมื่อหน่วยซ่อนถูกกำจัดออกไปเป็นอันดับแรก กำหนดให้ $NumErrPat_j$ เป็นจำนวนของข้อมูลที่จำแนกประเภทผิด (Validation error patterns) เมื่อหน่วยซ่อน j หายไปจากโครงข่าย สมการ (3.1) อธิบาย AD_j ที่ถูกปรับปรุงใหม่สำหรับปัญหาการแยกประเภทข้อมูล เราหารพจน์ที่สองด้วยผลคูณของจำนวนชุดข้อมูลและจำนวนหน่วยเอาท์พุท เนื่องจากจำนวนของข้อมูลที่จำแนกผิดประเภท (พจน์ที่หนึ่ง) มีความสำคัญมากกว่าค่าผลต่างของแอ็คติเวชั่น ถ้ามีหน่วยซ่อนมากกว่าหนึ่งหน่วยที่เสมอกันด้วยค่าจำนวนข้อมูลที่จำแนกผิดประเภท เราจะใช้ค่าผลต่างของแอ็คติเวชั่นเพื่อตัดสินว่าหน่วยซ่อนใดมีความสำคัญน้อยกว่ากัน

$$AD_j = NumErrPat_j + \frac{\sum_p AD_j(p)}{P \times K} \tag{3.1}$$

โดย P และ K หมายถึง จำนวนของชุดข้อมูล และจำนวนหน่วยเอาท์พุท ตามลำดับ อนึ่งโปรแกรมสำหรับคำนวณจำนวนข้อมูลที่จำแนกผิดประเภทสามารถแทรกเข้าไปในรอบการทำงาน

เดียวกับการคำนวณหาค่าผลต่างของแอ็คติเวชั่นซึ่งมิได้กระทบต่อความซับซ้อนของโมดูล ValidationError แต่อย่างใด

3.2 ชุดข้อมูลที่ใช้ในการทดลอง

สำหรับปัญหาที่เราได้นำชุดข้อมูลมาทำการทดสอบมีสองประเภทดังนี้

3.2.1 ปัญหาสังเคราะห์

เนื่องจากปัญหาสังเคราะห์ที่ใช้ทดสอบอัลกอริธึมในการลดจำนวนหน่วยซ่อนมีจำนวนมากและหลากหลาย ดังนั้นเราจึงเลือกชุดข้อมูลที่ใช้ในงานที่ถูกอ้างถึงบ่อยที่สุด เราพยายามหลีกเลี่ยงปัญหาที่มีค่าสุ่มเข้ามาเกี่ยวข้อง ทั้งนี้เพื่อความเป็นธรรมในการเปรียบเทียบกับงานอื่น ปัญหาฟังก์ชั่นพาราโบลิก (Parabolic function), ปัญหาฟังก์ชั่นไซน์ (Sine function) และปัญหาการจำแนกประเภทสังเคราะห์ (Artificial classification) นำมาจากงานใน [12]

ก) ปัญหาฟังก์ชั่นพาราโบลิก (*F1*):

$$f_1(z) = z^2 \tag{3.2}$$

$z \sim U(-1, 1)$ โดยที่ U หมายถึงการกระจายแบบเสมอกัน (Uniform Distribution) สำหรับค่าเอาท์พุททั้งหมดจะอยู่ในช่วง [0, 1] และไม่มีสิ่งรบกวนอยู่ในชุดข้อมูลนี้ ฟังก์ชั่นนี้สามารถแสดงได้ดังรูปที่ 3.3 เราใช้ค่า mean-squared error (MSE) แทนที่อัตราการจำแนกประเภทผิดพลาด (Error rate) จากชุดข้อมูลแวลิเดชั่นเพื่อคำนวณค่าความสูญเสียเจนเนอรัลไลเซชั่น เราไม่สามารถใช้ค่าอัตราการจำแนกประเภทผิดพลาดเนื่องจากปัญหานี้ไม่ใช่ปัญหาแยกประเภท

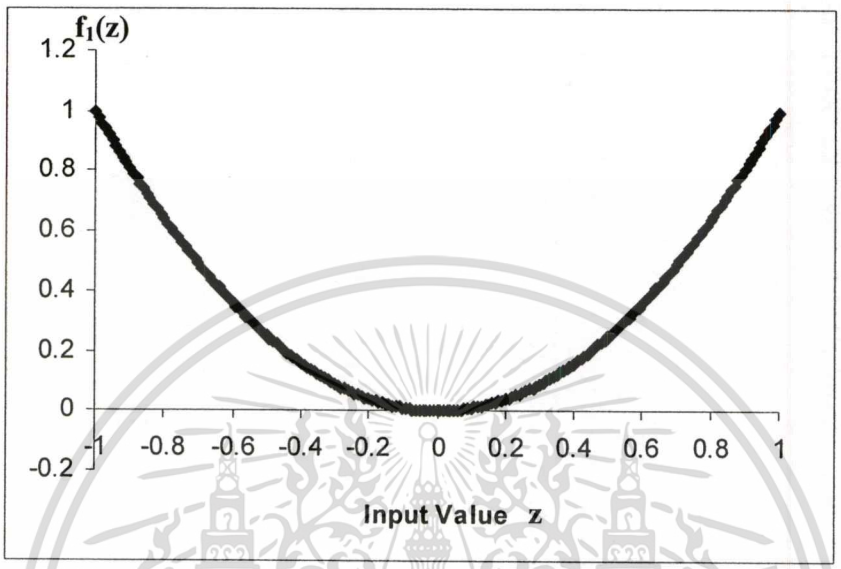
ข) ปัญหาฟังก์ชั่นไซน์ (*F2*)

$$f_2(z) = sin(2\pi z)\, e^{-x} + \zeta \tag{3.3}$$

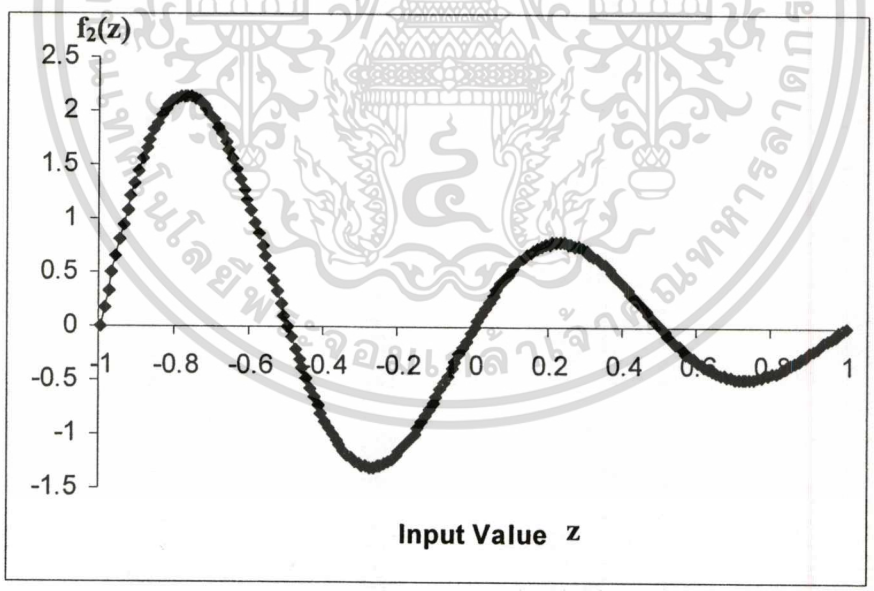
$z \sim U(-1, 1)$ และ $\zeta \sim N(0, 0.1)$ โดยที่ N หมายถึงการกระจายแบบปกติ (Normal Distribution) ค่าเอาท์พุทถูกปรับให้อยู่ในช่วง [0, 1] ฟังก์ชั่นนี้แสดงอยู่ในรูปที่ 3.4 ค่า MSE ยังคงถูกใช้ในการคำนวณค่าความสูญเสียเจนเนอรัลไลเซชั่นด้วยเหตุผลเดียวกันกับ *F1*

ค) ปัญหาการจำแนกประเภทสังเคราะห์ (AC-แสดงในรูปที่ 3.5):

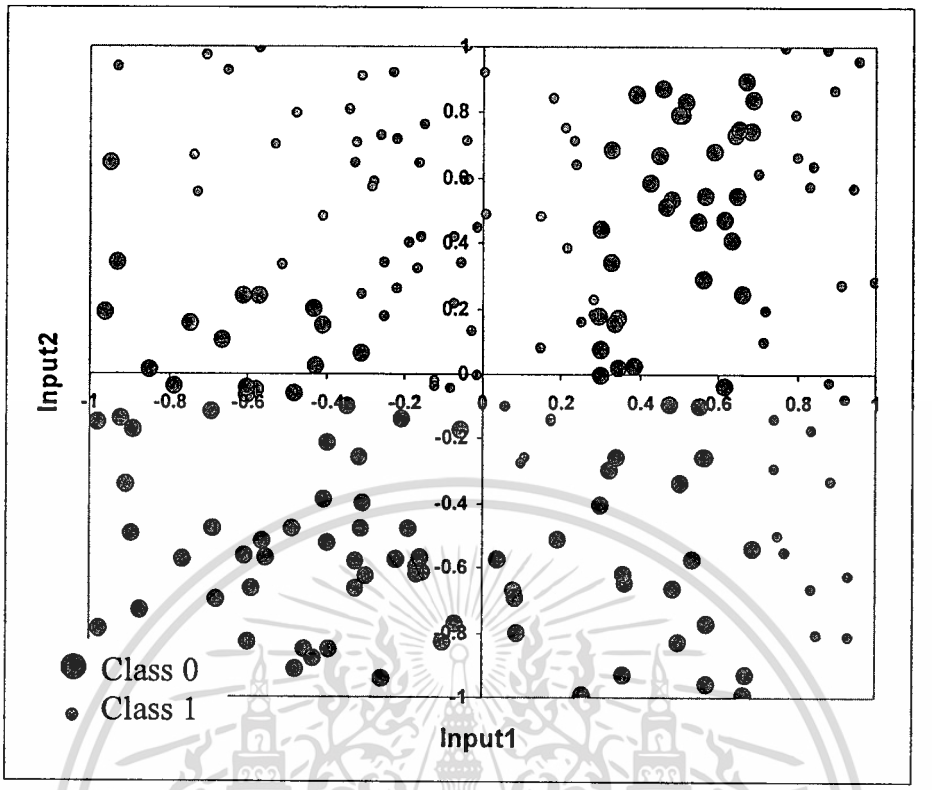
$$Class(z_1, z_2) = \begin{cases} 1 \text{ ; if}(z_1 \geq 0.7) \text{ or } ((z_1 \leq 0.3) \text{ and } (z_2 \geq -0.2\text{-}z_1)) \\ 0 \text{ ; otherwise} \end{cases} \quad (3.4)$$



รูปที่ 3.3 ฟังก์ชั่นพาราโบลิก F1



รูปที่ 3.4 ฟังก์ชั่นไซน์ F2



รูปที่ 3.5 ปัญหาการจำแนกประเภทสังเคราะห์ (AC)

3.2.2 ปัญหามาตรฐาน

เพื่อเป็นการประเมินความสามารถของอัลกอริธึมสปาร์ตันซิมพลิซิตีในการสร้างโครงข่ายสปาร์ตันที่สามารถทำนายข้อมูลที่ยังไม่เคยพบมาก่อนได้ดี เรานำอัลกอริธึมไปประยุกต์ใช้กับปัญหามาตรฐาน นั่นคือ Iris, Breast cancer, Diabetes, Heart disease, Wine, Credit card และ Isolet ข้อมูลนี้ได้รับมาจากแหล่งรวบรวมข้อมูลสำหรับเปรียบเทียบสมรรถนะของ University of California Irvine (UCI benchmark repository)

ก) ปัญหา Iris: ชุดข้อมูลมี 3 คลาส โดยแต่ละคลาสมีข้อมูล 50 ชุดซึ่งจำแนกตามชนิดของพืช ปัญหานี้มีแอททริบิว (Attribute) ซึ่งมีชนิดเป็นข้อมูลต่อเนื่องทั้งหมด 4 แอททริบิว

ข) ปัญหา Breast Cancer: ข้อมูลชุดนี้ได้รับการบริจาคโดย University of Wisconsin Hospitals, Madison ซึ่งมีข้อมูลทั้งสิ้น 699 ชุด โดยแต่ละชุดประกอบไปด้วยแอททริบิวแบบต่อเนื่องทั้งหมด 9 ตัว

ค) ปัญหา Diabetes: ปัญหานี้มี 2 คลาสโดยมีชุดข้อมูลทั้งสิ้น 768 ชุด ข้อมูลเหล่านี้ถูกเก็บมาจากหญิงชาว Pima Indian ซึ่งอาศัยอยู่ใกล้กับเมือง Phoenix, AZ ข้อมูลแต่ละชุดประกอบด้วยแอททริบิวที่เป็นจำนวนจริงทั้งหมด 8 แอททริบิว ปัญหานี้ถือได้ว่าเป็นปัญหาการจำแนกประเภทที่ยากที่สุดปัญหาหนึ่งในการแยกประเภทของข้อมูล เนื่องจากข้อมูลมีจำนวนไม่มากนัก อีกทั้งยังมีข้อมูลรบกวนปะปนอยู่ด้วย

ง) ปัญหา Heart Disease: ข้อมูลชุดนี้ได้รับจาก V.A. Medical Center, Long Beach และ Cleveland Clinic Foundation โดยเป้าหมายของข้อมูลคือการทำนายการปรากฏและไม่ปรากฏของโรคหัวใจ ซึ่งวัดจากแอททริบิวทั้งสิ้น 13 แอททริบิว ชุดข้อมูลมีจำนวนทั้งสิ้น 303 ชุด

จ) ปัญหา Wine: ปัญหานี้เป็นการวิเคราะห์องค์ประกอบทางเคมีของไวน์ที่ผลิตในประเทศอิตาลี ปัญหานี้มีจำนวนแอททริบิว 13 แอททริบิว และมีจำนวนชุดข้อมูลทั้งหมด 178 ชุด

ฉ) ปัญหา Credit Card Assessment: ปัญหานี้ถือเป็นหนึ่งในปัญหาที่ยาก ซึ่งเป็นปัญหาเกี่ยวกับการประเมิน (ผ่าน/ไม่ผ่าน) ผู้สมัครบัตรเครดิตของออสเตรเลีย โดยชุดข้อมูลมีจำนวนแอททริบิว 14 แอททริบิว และมีจำนวนชุดข้อมูล 690 ชุด

ช) ปัญหา Isolet: ปัญหานี้เป็นตัวแทนของปัญหาที่มีขนาดใหญ่ เนื่องจากประกอบไปด้วยข้อมูลเสียงของคนจำนวน 150 คน ซึ่งเปล่งเสียงของตัวอักษรในภาษาอังกฤษ โดยแต่ละคนจะเปล่งเสียงแต่ละตัวอักษรซ้ำกันสองครั้ง จำนวนของแอททริบิวสำหรับปัญหานี้มีจำนวน 617 แอททริบิว และมีจำนวนตัวอย่างทั้งสิ้น 7797 ตัวอย่าง ถือได้ว่าปัญหานี้เหมาะสำหรับการทดสอบการทำงานของอัลกอริธึมว่าสามารถทำงานได้กับปัญหาที่มีขนาดใหญ่และมีความซับซ้อนที่มากขึ้นได้ดีเพียงใด

ปัญหาเหล่านี้มีความท้าทายในเรื่องของความซับซ้อนของข้อมูลจริง นอกจากนี้ แอททริบิวที่เป็นอินพุทส่วนใหญ่ถูกนอร์มัลไลซ์ (Normalize) จนกระทั่งค่าเฉลี่ยของแอททริบิวเหล่านี้มีค่าใกล้เคียงศูนย์ ทั้งนี้เพื่อเป็นการเร่งความเร็วขบวนการเรียนรู้ของแบ็คพรอพาเกชั่น [16] ซึ่งไม่ได้ส่งผลกระทบต่อผลการทดลองแต่อย่างใด สำหรับปัญหาที่แอททริบิวถูกนอร์มัลไลซ์ให้มีค่าอยู่ระหว่าง 0.00 และ 1.00 โดยฟังก์ชั่นเส้นตรง คือ ปัญหา Diabetes, Heart disease, Wine และ Credit card ชุดข้อมูลส่วนใหญ่ยังถูกสลับที่แบบสุ่มก่อนที่จะถูกแบ่งออกเป็นสามส่วน (ชุดฝึก, ชุดแวลิ

เดชั่น, ชุดทดสอบ) ยกเว้นปัญหา Wisconsin breast cancer, Diabetes และ Heart disease เนื่องจากชุดข้อมูลของปัญหาเหล่านี้ถูกสลับที่อยู่แล้ว

3.3 วิธีการเลือกค่าพารามิเตอร์ให้กับอัลกอริธึม

ตารางที่ 3.1 แสดงการแบ่งชุดข้อมูลเป็นสามชุดคือ ชุดฝึก (Training set), ชุดแวลิเดชั่น (Validation set) และ ชุดทดสอบ (Test set) โดยยึดตามสัดส่วนที่ได้แสดงไว้ในงานวิจัย [9, 12, 18] นอกจากนี้ตารางนี้ยังแสดงค่าพารามิเตอร์ที่ใช้ในอัลกอริธึม ตัวอย่างเช่น ค่าเทรชโฮลด์ α และจำนวนเริ่มต้นของหน่วยซ่อน ค่าพารามิเตอร์เหล่านี้เกิดจากการสังเกตภายใต้การทำงานในรอบแรกของอัลกอริธึมสปาร์ตันซิมพลิซิตีเพื่อจะได้มาซึ่งค่าที่เหมาะสมที่สุด ตัวอย่างวิธีการได้มาซึ่งค่าพารามิเตอร์ทั้งสองสำหรับปัญหา Wisconsin Breast cancer และปัญหา Credit card ถูกแสดงไว้ในตารางที่ 3.2 และ 3.3 ตามลำดับ

ตารางที่ 3.1 การแบ่งข้อมูลสำหรับแต่ละปัญหา

Benchmark	#Attributes	#Classes	#Train patterns	#Valid. patterns	#Test patterns	α	Initial # of hidden units
F1	1	approximation	80	40	40	n/a	5
F2	1	approximation	100	50	50	n/a	10
AC	2	2	100	50	50	n/a	10
Iris	4	3	75	37	38	10	6-11
Breast Cancer	9	2	349	175	175	6	6-8
Diabetes	8	2	384	192	192	2	6-8
Heart	13	2	151	76	76	1	6-8
Wine	13	3	89	44	45	10	6-8
Credit card	14	2	346	172	172	2	12-14
Isolet	617	26	3899	1949	1949	0.05	78

ตารางที่ 3.2 การเลือกค่าที่เหมาะสมของ α และจำนวนเริ่มต้นของหน่วยซ่อนสำหรับปัญหา Wisconsin Breast Cancer

α	Initial # of hidden units	Final # of hidden units	Test Err Rate
0.0	12	3	0.0500
	10	2	0.0294
	8	1	0.0212
	6	1	0.0181
	4	2	0.0644
2.0	12	2	0.0512
	10	2	0.0344
	8	1	0.0232
	6	1	0.0198
	4	2	0.0617
6.0	12	2	0.0459
	10	1	0.0287
	8	1	0.0172
	6	1	0.0172
	4	2	0.0574
12.0	12	4	0.0689
	10	1	0.0632
	8	1	0.0229
	6	1	0.0402
	4	2	0.0574

ตารางที่ 3.3 การเลือกค่าที่เหมาะสมของ α และจำนวนเริ่มต้นของหน่วยซ่อนสำหรับปัญหา Credit card

α	Initial # of hidden units	Final # of hidden units	Test Err Rate
0.0	14	1	0.1010
	12	1	0.0980
	10	2	0.1151
	8	2	0.1298
	6	2	0.1211
2.0	14	1	0.0980
	12	1	0.0980
	10	1	0.1045
	8	1	0.1241
	6	2	0.1080
9.0	14	1	0.0980
	12	1	0.0980
	10	2	0.1111
	8	3	0.1241
	6	2	0.1111
12.0	14	1	0.0980
	12	1	0.0980
	10	2	0.1111
	8	2	0.1437
	6	3	0.1111

เราพิจารณาจำนวนของหน่วยซ่อนสุดท้ายและค่าอัตราการผิดพลาดของการจำแนกชุดข้อมูลทดสอบ โดยเราปรับให้ค่า α และจำนวนหน่วยซ่อนเริ่มต้นให้เปลี่ยนแปลง เราจะเห็นได้ว่าช่วงของ α สำหรับปัญหา Wisconsin breast cancer ไม่มีผลต่ออัลกอริธึมมากนักเพราะว่าทั้งค่าผิดพลาดทดสอบ (Test error) และจำนวนหน่วยซ่อนสุดท้ายจะมีค่าใกล้เคียงกันสำหรับทุกๆค่าของ α อย่างไรก็ตามที่ α =12 ประสิทธิภาพของการจำแนกประเภทตกลงจากเดิม ดังนั้นช่วงที่เหมาะสมสำหรับ α ควรมีค่าเท่ากับ 6.0 เนื่องจากให้ค่าผิดพลาดทดสอบและจำนวนหน่วยซ่อนสุดท้ายดีที่สุด สำหรับจำนวนหน่วยซ่อนเริ่มต้นนั้น เรามีวิธีการพิจารณาคล้ายคลึงกับ α ช่วงที่เหมาะสมของจำนวนหน่วยซ่อนเริ่มต้นควรจะทำให้โครงข่ายมีจำนวนหน่วยซ่อนสุดท้ายที่น้อยที่สุดและมีค่าผิดพลาดจากการทดสอบที่น้อยที่สุด ดังนั้นจำนวนหน่วยซ่อนเริ่มต้นสำหรับปัญหา Wisconsin breast cancer จึงควรมีค่าอยู่ระหว่าง 6-8 โดยวิธีการพิจารณาที่คล้ายคลึงกันนี้ เราสามารถเลือกค่าของพารามิเตอร์สำหรับปัญหาอื่นๆได้อย่างเหมาะสม สำหรับปัญหา F1, F2 และ AC เราให้จำนวนหน่วยซ่อนเริ่มต้นมีค่าเท่ากันกับงานวิจัยใน [12] นอกจากนี้ ชุดข้อมูลสำหรับปัญหาเหล่านี้เป็นข้อมูลสังเคราะห์ที่ไม่มีข้อมูลรบกวน (Noise) จึงทำให้โครงข่ายไม่เกิดความสูญเสียเจนเนอรัลไลเซชั่นในขณะฝึก ดังนั้นเราจึงหยุดฝึกโครงข่ายเมื่อค่าผิดพลาดแวลิเดชั่นมีค่าน้อยกว่า 0.001

พารามิเตอร์อีกหนึ่งตัวที่จำเป็นต้องใช้ในอัลกอริธึมคือค่าเทรชโฮลด์สำหรับตัวชี้โอเวอร์พรุนนิ่ง (β) เราทดลองฝึกโครงข่ายประสาทเทียมในแต่ละปัญหาแล้วค่อยๆกำจัดหน่วยซ่อนที่ไม่สำคัญออก จากนั้นวัดค่าเทรชโฮลด์ β เมื่อหน่วยซ่อนถูกกำจัดมากจนเกินไปโดยใช้การคำนวณค่าตัวชี้โอเวอร์พรุนนิ่งจากสมการที่ (2.3) ตารางที่ 3.4 แสดงแนวทางที่ใช้ในการหาค่าที่เหมาะสมของ β สำหรับปัญหาต่างๆ อนึ่ง ค่าเทรชโฮลด์ β สำหรับปัญหาใดๆไม่ควรมีค่าน้อยเกินไปเพราะอัลกอริธึมจะไวต่อการตรวจจับการเกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปซึ่งจะทำให้ได้โครงข่ายที่ยังมีหน่วยซ่อนที่ไม่จำเป็นหลงเหลืออยู่ ในทางตรงกันข้าม ถ้าค่าเทรชโฮลด์ β มีค่ามากจนเกินไป อัลกอริธึมจะทำงานต่อเนื่องจนไม่สามารถหยุดทำงานได้เนื่องจากโครงข่ายมีจำนวนหน่วยซ่อนไม่เพียงพอต่อการฝึก ทำให้เราไม่สามารถทำให้ค่าผิดพลาดของโครงข่ายมีค่าน้อยจนเป็นที่ยอมรับได้ ดังนั้น เราแนะนำให้มีการทำการทดลองในตารางที่ 3.4 ก่อนเลือกค่า β ที่เหมาะสม

ตารางที่ 3.4 การหาค่าเทรชโฮลด์ที่เหมาะสมของ OI (β)

	OI's threshold (β)	Std.Dev.
F1	61.54	0.25
F2	48.0	0.18
AC	51.72	0.09
Iris	165.0	1.21
Cancer	94.44	0.51
Diabetes	30.0	0.10
Heart	28.13	0.09
Wine	170	1.78
Credit Card	38.75	0.34
Isolet	1300.0	56.20

เพื่อให้เกิดความเป็นระบบในการหาค่าพารามิเตอร์ α และ β แนวทางในการหาค่าพารามิเตอร์ที่เหมาะสมสามารถแสดงได้ดังนี้

<u>การหาค่าเทรชโฮลด์ของการสูญเสียเจนเนอรัลไลเซชั่น (α)</u>

ก) กำหนดให้ $\alpha = 0.0$ และทดลองทำงานอัลกอริธึมสปาร์ตันซิมพลิซิตี

ข) บันทึกค่าจำนวนหน่วยซ่อนสุดท้ายและค่าอัตราผิดพลาดทดสอบ (Test error rate)

ค) ค่อยๆเพิ่ม α ด้วยค่าคงที่[2]ที่น้อยๆ เช่น +0.1, +0.5, +1.0 เป็นต้น

ง) ทดลองทำงานอัลกอริธึมสปาร์ตันซิมพลิซิตีอีกครั้งหนึ่ง

จ) บันทึกค่าจำนวนหน่วยซ่อนสุดท้ายและค่าอัตราผิดพลาดทดสอบ (Test error rate)

ฉ) ถ้าค่าที่บันทึกได้มีค่าน้อยกว่าค่าที่บันทึกในรอบก่อนหน้านี้ ให้ทำงานซ้ำในขั้นตอน ค) – จ)

<u>การหาค่าเทรชโฮลด์ของการเกิดโอเวอร์พรุนนิ่ง (β)</u>

ก) ฝึกโครงข่ายตั้งต้นด้วยจำนวนหน่วยซ่อนเริ่มต้นที่เหมาะสม นั่นคือ จำนวนหน่วยซ่อนที่ทำให้โครงข่ายสามารถเข้าสู่คอนเวอร์เจนซ์ (Convergence)

ข) บันทึกค่าอัตราผิดพลาดแวลิเดชั่นก่อนเกิดโอเวอร์พรุนนิ่ง

ค) กำจัดหน่วยซ่อนที่มีความสำคัญน้อยที่สุด (มีค่า AD_j น้อยที่สุด) ออกไปจากโครงข่าย

ง) ฝึกโครงข่ายที่ถูกกำจัดหน่วยซ่อนออกไปแล้วอีกครั้งหนึ่งพร้อมทั้งตรวจสอบดูค่าอัตราผิดพลาดแวลิเดชั่น

[2] ถ้าค่าที่เพิ่มขึ้นนี้ไม่ได้ส่งผลให้ค่าอัตราผิดพลาดทดสอบมีค่าลดลงอย่างเห็นได้ชัดเจน เราควรจะเลือกค่าที่ใหญ่ขึ้นเพื่อลดเวลาในการทดลองทำงานของอัลกอริธึม

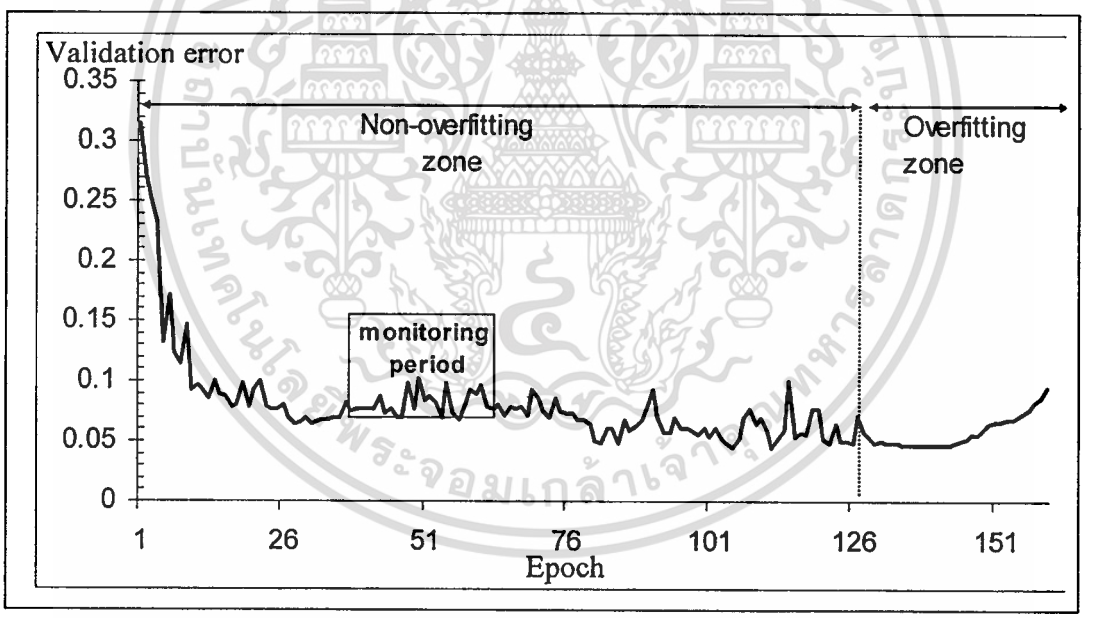
จ) ถ้าค่าอัตราผิดพลาดแวลิเดชั่นมีค่าเพิ่มขึ้นอย่างชัดเจนและไม่มีแนวโน้มที่จะลดลงเลยไม่ว่าเราจะฝึกโครงข่ายนานเท่าใด ให้หยุดฝึกโครงข่าย และข้ามไปยังขั้นตอนที่ ช)

ฉ) ทำงานซ้ำในขั้นตอน ข) – จ)

ช) บันทึกค่าอัตราผิดพลาดแวลิเดชั่นหลังจากที่เกิดโอเวอร์พรุนนิ่ง

ซ) คำนวณค่าเทรชโฮลด์ β โดยใช้สมการที่ (2.3)

ดังที่ได้กล่าวไว้ในอัลกอริธึมว่า เราตรวจจับการเกิดค่าความสูญเสียเจนเนอรัลไลเซชั่นภายใน γ รอบเพื่อที่จะหาจุดที่ควรจะหยุดการฝึกโครงข่าย ต่อไปนี้เป็นการสาธิตวิธีการได้มาซึ่งจำนวนรอบ (γ) ที่เหมาะสม ขั้นตอนแรกเราทำการฝึกโครงข่ายประสาทเทียมมาตรฐานที่มี 3 ชั้น (ชั้นอินพุท, ชั้นซ่อน และชั้นเอาท์พุท) โดยใช้อัลกอริธึมแบ็คพรอพาเกชั่น และทำการตรวจสอบดูการเปลี่ยนแปลงของค่าผิดพลาดแวลิเดชั่น รูปที่ 3.6 แสดงการสุ่มวางหน้าต่างตรวจสอบ (Monitoring window) ลงบนกราฟของค่าผิดพลาดแวลิเดชั่น



รูปที่ 3.6 หน้าต่างตรวจสอบถูกสุ่มวางบนกราฟของค่าผิดพลาดแวลิเดชั่น

ตารางที่ 3.5 การทำนายค่าความสูญเสียเจนเนอรัลไลเซชั่นผิดพลาดของช่วงการตรวจสอบที่มีขนาดต่างๆ

Monitoring period (epochs)	The number of wrong predictions						
	Iris	Cancer	Diabetes	Heart	Wine	Credit	Isolet
5	4	6	7	7	4	5	15
15	2	4	5	6	3	4	9
20	1	2	2	2	3	2	4
25	**0**	**0**	**1**	**1**	**0**	**1**	**1**
30	0	0	1	1	0	1	1
35	0	0	1	1	0	1	1

มีสองปัจจัยที่สำคัญที่เราพิจารณาในการเลือกช่วงที่เหมาะสมสำหรับการตรวจสอบค่าความสูญเสียเจนเนอรัลไลเซชั่น ปัจจัยเหล่านี้คือเวลาที่ถูกใช้ไปในการตรวจสอบและจำนวนครั้งที่ทำนายผิดพลาดว่าโครงข่ายเกิดความสูญเสียเจนเนอรัลไลเซชั่น ช่วงเวลาที่มีขนาดเล็กจะใช้เวลาน้อยในการทำนายแต่อาจทำให้โครงข่ายทำนายข้อมูลผิดเป็นจำนวนมาก ในทางตรงกันข้ามช่วงเวลาที่มีขนาดใหญ่จะใช้เวลานานในการทำนาย แต่จะสามารถทำนายได้ถูกต้องและแม่นยำกว่า เราทำการวัดจำนวนครั้งของการทำนายผิดพลาดสำหรับช่วงที่มีขนาดต่างๆกันหกช่วง โดยวางหน้าต่างตรวจสอบลงบนส่วนที่ไม่เกิดปัญหาการฝึกมากจนเกินไป (Non-overfitting zone) และส่วนที่เกิดปัญหาการฝึกมากจนเกินไป (Overfitting zone) การทำนายผิดพลาดในช่วงที่ไม่เกิดปัญหาการฝึกมากจนเกินไปจะเกิดขึ้นเมื่อเรามีช่วงในการตรวจสอบสั้นเกินไปและช่วงนั้นไม่สามารถคลุมค่าผิดพลาดที่ขึ้นลงเพียงเล็กน้อยไว้ได้ ส่วนการทำนายผิดในช่วงที่เกิดปัญหาการฝึกมากจนเกินไปจะเกิดขึ้นเมื่อค่าความแตกต่างของค่าผิดพลาดระหว่างซ้ายมือและขวามือของหน้าต่างตรวจสอบมีค่าน้อยมากและไม่สามารถผ่านค่าเทรชโฮล์ด α ได้ ดังนั้น หน้าต่างตรวจสอบที่มีขนาดเล็กจึงมักจะมีจำนวนครั้งของการทำนายผิดพลาดมากกว่าหน้าต่างที่มีขนาดใหญ่ๆ เราทำการทดลองโดยการสุ่มวางหน้าต่างที่มีขนาดต่างกันหกค่าลงบนกราฟของค่าผิดพลาด 100 ครั้งคละกันในทุกๆชุดข้อมูล จากนั้นจึงบันทึกผลที่ได้ลงในตารางที่ 3.5 สังเกตว่าจำนวนการทำนายผิดระหว่างช่วง 25-35 รอบนั้นไม่ได้มีความแตกต่างกันมากนัก ดังนั้นช่วงของการตรวจสอบค่าความสูญเสียเจนเนอรัลไลเซชั่นที่เหมาะสมจึงมีค่าเท่ากับ 25 รอบ เนื่องจากจะให้ผลของการทำนายค่าความสูญเสียเจนเนอรัลไลเซชั่นที่ถูกต้องและใช้เวลาน้อยที่สุด นอกจากนี้ ค่าที่ได้นี้สอดคล้องกับแนวคิดจากงานวิจัยอื่น [9] ที่มีอยู่แล้วอีกด้วย

บทที่ 4

การทดลองและการเปรียบเทียบผลการทดลอง

4.1 การทดลอง

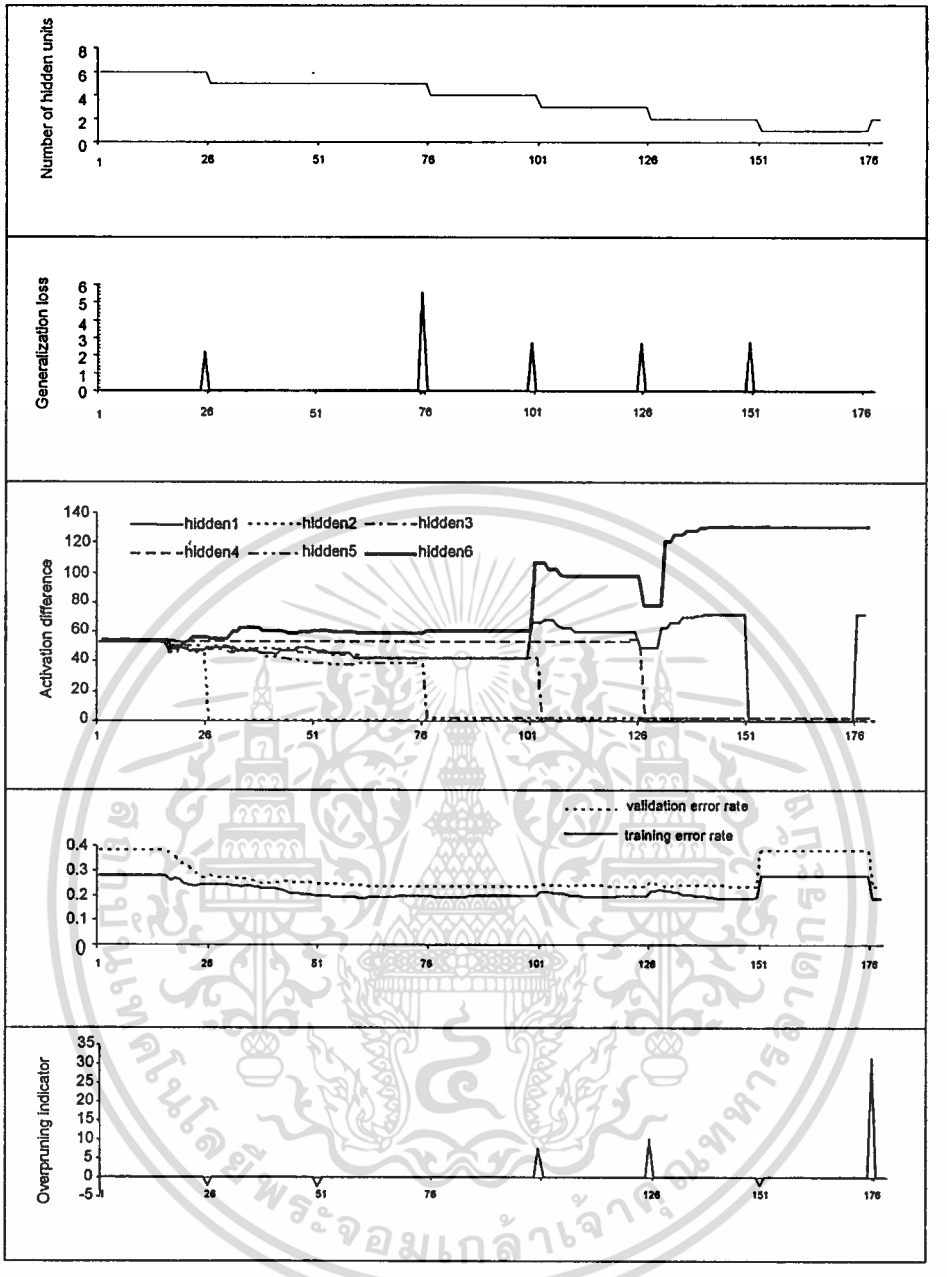
ผลการทดลองเกิดจากการเฉลี่ยผลที่วัดจากการทดลองทั้งหมด 30 ครั้งโดยใช้ค่าน้ำหนักเริ่มต้นและจำนวนหน่วยซ่อนเริ่มต้นที่แตกต่างกัน การทดลองทั้งหมดถูกกระจายอยู่บนค่าอัตราการเรียนรู้ (Learning rate) ในช่วง 0.1-0.5 โครงข่ายที่ได้และอัตราการทำนายข้อมูลผิดสำหรับแต่ละปัญหาถูกแสดงในตารางที่ 4.1 ค่าผิดพลาดที่แสดงในตารางหมายถึงค่าผลรวมของค่าผิดพลาดกำลังสอง (Sum-squared error) แต่สำหรับปัญหา F1 และ F2 เราใช้ค่าเฉลี่ยของค่าผิดพลาดกำลังสอง (Mean-squared error) สำหรับอัตราผิดพลาด (Error rate) หมายถึงอัตราของข้อมูลที่โครงข่ายสปาร์ตันทำนายผิด ในปัญหาที่ไม่ใช่ปัญหาจำแนกประเภท (F1 และ F2) เราไม่แสดงค่าอัตราผิดพลาดเนื่องจากไม่สามารถคำนวณได้

ตารางที่ 4.1 ผลการทดลองที่ได้จากสปาร์ตันซิมพลิซิตีในแต่ละปัญหา

Problem		Final # of hidden units	Training set		Validation set		Test set	
			error	err rate	error	err rate	error	err rate
F1	Mean	2	0.0211	-	0.0255	-	0.0318	-
	SD	0	0.0106	-	0.0176	-	0.0163	-
F2	Mean	5	0.0230	-	0.0249	-	0.0285	-
	SD	0	0.0018	-	0.0007	-	0.0019	-
AC	Mean	3	0.0254	0.0266	0.0272	0.0286	0.0281	0.0300
	SD	0	0.0240	0.0251	0.0639	0.0642	0.0098	0.0095
Iris	Mean	2.25	0.0110	0.0126	0.0138	0.0145	0.0142	0.0146
	SD	1.09	0.0067	0.0098	0.0027	0.0056	0.0020	0.0021
Cancer	Mean	2.43	0.0113	0.0100	0.0160	0.0183	0.0180	0.0200
	SD	0.23	0.0096	0.0048	0.0053	0.0043	0.0077	0.0069
Diabetes	Mean	2.16	0.1823	0.1898	0.1832	0.1935	0.1957	0.2144
	SD	0.98	0.0045	0.0051	0.0230	0.0039	0.1516	0.0061
Heart	Mean	2.27	0.0932	0.1148	0.0979	0.1554	0.1081	0.1751
	SD	0.21	0.0133	0.0068	0.0051	0.0107	0.0019	0.0054
Wine	Mean	4.53	0.0016	0.0017	0.0044	0.0050	0.0057	0.0068
	SD	0.96	0.0061	0.0062	0.0055	0.0021	0.0030	0.0039
Credit	Mean	3.25	0.1026	0.1113	0.1061	0.1115	0.1095	0.1119
	SD	1.24	0.0030	0.0041	0.0932	0.0126	0.0310	0.0200
Isolet	Mean	20.28	0.0115	0.0101	0.0490	0.0507	0.0500	0.0510
	SD	0.74	0.0058	0.0087	0.0070	0.0115	0.0054	0.0059

รูปที่ 4.1 แสดงการทำงานของอัลกอริธึมสปาร์ตันซิมพลิซิตีของปัญหา Diabetes ซึ่งผลการทดลองนี้เป็นกราฟแสดงจำนวนของหน่วยซ่อน, ค่าความสูญเสียเจนเนอรัลไลเซชั่น, ค่าผลต่างของแอคติเวชั่นของแต่ละหน่วยซ่อน, อัตราความผิดพลาดของชุดข้อมูลฝึกและชุดข้อมูลแวลิเดชั่น, และการเปลี่ยนแปลงของตัวชี้โอเวอร์พรุนนิ่ง อัลกอริธึมใช้เวลา 25 รอบ ในการฝึกโครงข่ายเริ่มต้นซึ่งมีจำนวนหน่วยซ่อน 6 หน่วย จนกระทั่งโครงข่ายเริ่มสูญเสียเจนเนอรัลไลเซชั่นในรอบที่ 25 (สังเกตจากค่าความสูญเสียเจนเนอรัลไลเซชั่นสูงขึ้นถึง 2.20 ซึ่งเกินค่าเทรชโฮลด์ α ที่มีค่า 2.0 จากตารางที่ 3.1) จากนั้นในรอบที่ 26 อัลกอริธึมเลือกที่จะกำจัดหน่วยซ่อนที่ 2 เนื่องจากมันมีค่าผลต่างของแอคติเวชั่นน้อยที่สุด หลังจากนั้นโครงข่ายจะถูกฝึกต่อไปจนกระทั่งเริ่มสูญเสียเจนเนอรัลไลเซชั่นอีกครั้งในรอบที่ 76, 101, 126 และ 151 ซึ่งอัลกอริธึมตัดสินใจกำจัดหน่วยซ่อนที่ 5, 3, 4 และ 1 ตามลำดับ น่าสนใจที่หน่วยซ่อนที่ 1 มีค่าผลต่างของแอคติเวชั่นที่น้อยกว่าหน่วยซ่อนที่ 4 แต่มันสามารถอยู่ในโครงข่ายได้นานกว่าหน่วยซ่อนที่ 4 ดังนั้น การเปลี่ยนแปลงค่าผลต่างของแอคติเวชั่นชี้ให้เห็นถึงการแข่งขันกันของหน่วยซ่อนตลอดช่วงเวลาของการกำจัดหน่วยซ่อน แม้ว่าหน่วยซ่อนนั้นจะมีค่าผลต่างของหน่วยซ่อนที่น้อยกว่า แต่มันอาจจะมีชีวิตอยู่ได้นานกว่าหน่วยซ่อนอื่นๆ

ที่รอบ 176 เกิดสถานการณ์โอเวอร์พรุนนิ่งขึ้นเนื่องจากค่าตัวชี้โอเวอร์พรุนนิ่ง (Overpruning Indicator หรือ OI) มีค่าสูงขึ้นจนถึงค่าเทรชโฮลด์ β ซึ่งมีค่า 30 (ดังแสดงในตารางที่ 3.4) ซึ่งแสดงให้เห็นว่าโครงข่ายประสาทเทียมมีจำนวนหน่วยซ่อนไม่เพียงพอต่อการเรียนรู้ ดังนั้นอัลกอริธึมจึงคืนหน่วยซ่อนที่ 1 ซึ่งถูกกำจัดทิ้งไปพร้อมทั้งน้ำหนักที่เชื่อมต่อกับหน่วยซ่อนกลับมาและให้โครงข่ายสปาร์ตันซึ่งมีจำนวนหน่วยซ่อนเพียง 2 หน่วย



รูปที่ 4.1 ตัวอย่างกราฟแสดงผลของอัลกอริธึมสปาร์ตันซิมพลิซิตีสำหรับปัญหา Diabetes

4.2 การเปรียบเทียบผลการทดลองกับวิธีอื่น

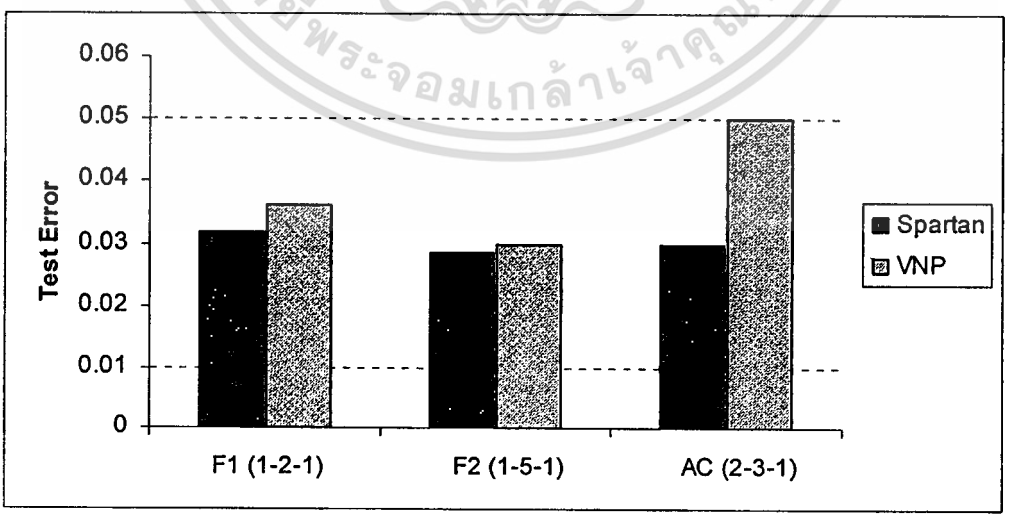
ในหัวข้อนี้ เราเปรียบเทียบผลของอัลกอริธึมสปาร์ตันซิมพลิซิตีกับอัลกอริธึมในการกำจัดหน่วยซ่อนที่เป็นที่รู้จัก นั่นคือ Skeletonization [3], Optimal brain damage (OBD) [6], Optimal brain surgeon (OBS) [7], Dynamic node decaying method (DNDM) [9], Magnitude based pruning (MBP) [10], Variance nullity pruning (VNP) [12] และ EPNet [18] สำหรับวิธี OBD, OBS, Skeletonization และ MBP ถูกสร้างไว้แล้วอยู่ในโปรแกรมจำลองการทำงาน Stuttgart NN simulator (SNNS) ดังนั้น เราจึงทดลองวิธีเหล่านี้บนโปรแกรม SNNS ส่วนวิธีอื่นๆ เรานำผล

การทดลองที่ระบุไว้ในงานทดลองดั้งเดิมมาเปรียบเทียบกับผลการทดลองของเราโดยกำหนดให้สภาพแวดล้อมและเงื่อนไขในการทดลองมีลักษณะเหมือนกัน

สำหรับปัญหา ฟังก์ชั่น F1, ฟังก์ชั่น F2 และ AC (Artificial classification) ถูกเปรียบเทียบกับอัลกอริธึม VNP ผลการเปรียบเทียบในตารางที่ 4.2 และรูปที่ 4.2 สามารถบ่งบอกได้ชัดเจนว่าทั้งสองอัลกอริธึมมีความสามารถใกล้เคียงกันในแง่ของค่าความผิดพลาดแบบค่าเฉลี่ยยกกำลังสองของการทดสอบ (Testing mean-squared error), อัตราความถูกต้องของการทดสอบ (Testing accuracy) และจำนวนของหน่วยซ่อน โดยผลที่ได้จากอัลกอริธึมสปาร์ตันซิมพลิซิตีจะดีกว่าผลของ VNP เล็กน้อย เราไม่ได้เปรียบเทียบอัลกอริธึมที่นำเสนอกับ VNP ในทุกๆชุดข้อมูลที่มีอยู่ในงานวิจัย [12] เนื่องจากอัลกอริธึม VNP กำจัดทั้งหน่วยซ่อนและหน่วยอินพุทออกจากโครงข่าย ซึ่งต่างจากอัลกอริธึมของเราที่กำจัดหน่วยซ่อนเพียงอย่างเดียว อีกจุดหนึ่งที่ทำให้ VNP แตกต่างจากวิธีของเราก็คือ VNP เริ่มต้นค่าน้ำหนักของโครงข่ายใหม่ทุกครั้งที่หน่วยซ่อนถูกกำจัดออกไปจากโครงข่าย โครงข่ายจะต้องถูกฝึกใหม่ตั้งแต่แรก ซึ่งอาจจะนำไปสู่ปัญหาโลคอลมินิมา (Local minima) ถ้าการกำหนดค่าเริ่มต้นของน้ำหนักถูกกระทำในช่วงที่ไม่เหมาะสม [17]

ตารางที่ 4.2 การเปรียบเทียบระหว่าง VNP และสปาร์ตันซิมพลิซิตี

	Final architecture		Testing MSE		Testing accuracy	
	Spartan	VNP	*Spartan*	VNP	*Spartan*	VNP
F1	1-2-1	1-2-1	0.0318	0.0362	n/a	n/a
F2	1-5-1	1-5-1	0.0285	0.0299	n/a	n/a
AC	2-3-1	2-3-1	n/a	n/a	97%	95%



รูปที่ 4.2 กราฟแสดงการเปรียบเทียบผลการทดลองสำหรับปัญหา F1, F2 และ AC

เนื่องจากวิธีการที่เรานำเสนอนี้ เริ่มกำจัดหน่วยซ่อนที่จุดที่โครงข่ายเกิดการสูญเสียความสามารถในการทำนายข้อมูลและทำการฝึกโครงข่ายต่อไปโดยใช้ค่าน้ำหนักเดิม ดังนั้นจึงอาจมีข้อโต้แย้งว่าโครงข่ายอาจจะประสบปัญหาโลคอลมินิมาได้ ในความเป็นจริงแล้ว ถ้าหน่วยซ่อนถูกกำจัดออกไปจากโครงข่าย (ซึ่งน้ำหนักต่างๆที่เชื่อมกับหน่วยซ่อนนั้นก็จะถูกกำจัดไปพร้อมๆกันด้วย) โครงข่ายจะถูกกระทำโดยเปรียบเสมือนกับโดนเขย่ารอบๆพื้นผิวค่าผิดพลาด (Error surface) ทำให้โครงข่ายมีโอกาสที่จะหลุดออกจากโลคอลมินิมาได้ นอกจากนี้ การใช้น้ำหนักเดิมเพื่อฝึกโครงข่ายที่ถูกกำจัดหน่วยซ่อนยังเป็นการกระจายน้ำหนักที่ถูกกำจัดออกไปยังน้ำหนักอื่นๆที่ยังคงอยู่ภายในโครงข่าย ทำให้ช่วยประหยัดเวลาในการฝึกโครงข่าย

ตารางที่ 4.3 แสดงการเปรียบเทียบสปาร์ตันซิมพลิซิตีกับงานวิจัยอื่นๆ สำหรับปัญหา Iris อัลกอริธึมสปาร์ตันซิมพลิซิตีมีผลลัพธ์ที่ดีกว่าทั้งในแง่ของจำนวนของหน่วยซ่อนและค่าอัตราผิดพลาดของการทดสอบ แม้ว่า DNDM จะมีค่าเฉลี่ยของผลการทดลองที่เกิดจากการสลับชุดข้อมูลระหว่างข้อมูลฝึก, ข้อมูลแวลิเดชั่น และข้อมูลทดสอบในระหว่างการฝึก ผลลัพธ์ของ DNDM ก็ไม่ได้ดีเทียบเท่ากับผลของอัลกอริธึมสปาร์ตันซิมพลิซิตี

ตารางที่ 4.3 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและอัลกอริธึมอื่นๆสำหรับปัญหา Iris

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization	DNDM
# hidden units	2.25	4	4	4	3.5	2.67
Test error rate	0.0146	0.02	0.02	0.02	0.018	0.0159

ตารางที่ 4.4 แสดงตัวอย่างของโครงข่ายสปาร์ตันที่ได้โดยอัลกอริธึมสปาร์ตันซิมพลิซิตีสำหรับปัญหา Iris โครงข่ายที่ได้นี้เป็นโครงข่ายที่มีขนาดเล็กโดยมีจำนวนหน่วยซ่อนเพียงแค่สองหน่วยและให้ค่าอัตราผิดพลาดทดสอบที่ต่ำ โดยถ้าชุดข้อมูลทดสอบจำนวน 38 ชุดถูกส่งเข้าไปทำนายในโครงข่ายสปาร์ตันนี้ เราจะได้ค่าอัตราผิดพลาดทดสอบเท่ากับ 0.0146

ตารางที่ 4.4 แสดงน้ำหนักภายในโครงข่ายสปาร์ตันสำหรับปัญหา Iris

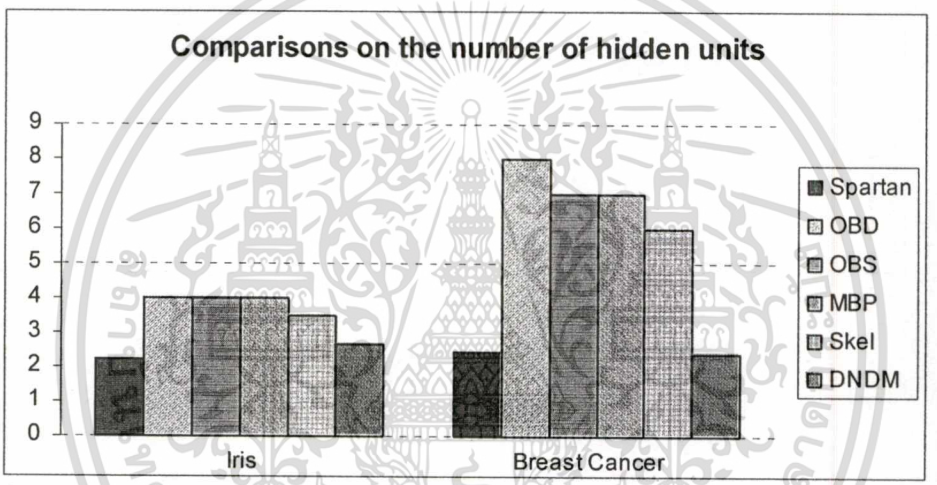
Inputs	I_1	I_2	I_3	I_4	Bias	Output1	Output2	Output3
Hidden1	2.3809	2.3005	-3.6304	-2.8563	1.4623	-4.3191	2.3457	2.2992
Hidden2	-0.3917	-1.6751	2.5088	1.2970	-0.3560	1.8007	2.4811	-4.2425
Bias	-	-	-	-	-	0.3071	-3.8836	-0.0124

ตารางที่ 4.5 แสดงการเปรียบเทียบผลการทดลองกับงานอื่นๆสำหรับปัญหา Breast cancer จะเห็นได้ว่าค่าเฉลี่ยของจำนวนหน่วยซ่อนของโครงข่ายสปาร์ตันมีค่าต่ำกว่าของวิธีอื่นๆ สำหรับจำนวนของหน่วยซ่อนและอัตราผิดพลาดของการทดสอบนั้น โครงข่ายสปาร์ตันจะให้ค่าที่

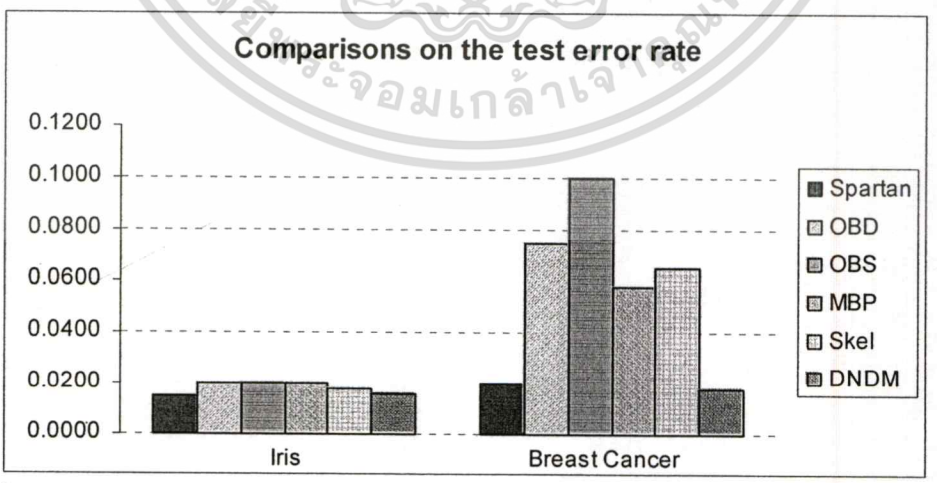
ใกล้เคียงกับค่าเปรียบเทียบที่ได้จากวิธี DNDM เพื่อเป็นการง่ายต่อการสังเกตผลการเปรียบเทียบสำหรับปัญหา Iris และ Breast Cancer กราฟแสดงการเปรียบเทียบสำหรับทั้งสองปัญหาจึงถูกแสดงในรูปที่ 4.3 และ 4.4

ตารางที่ 4.5 การเปรียบเทียบผลลัพธ์ระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหา Breast cancer

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization	DNDM
# hidden units	2.43	8	7	7	6	2.38
Test error rate	0.020	0.075	0.100	0.058	0.065	0.018



รูปที่ 4.3 กราฟแสดงการเปรียบเทียบจำนวนหน่วยซ่อนสำหรับปัญหา Iris และ Breast Cancer



รูปที่ 4.4 กราฟแสดงการเปรียบเทียบอัตราความผิดพลาดสำหรับปัญหา Iris และ Breast Cancer

หนึ่งในปัญหาที่ยากที่สุดในงานวิจัยเกี่ยวกับการเรียนรู้ของเครื่องจักร คือปัญหา Diabetes เนื่องจากมีจำนวนข้อมูลน้อยและมีข้อมูลรบกวนค่อนข้างมาก ผลจากการเปรียบเทียบแสดงให้เห็นว่าสปาร์ตันซิมพลิซิตีมีประสิทธิภาพที่ดีกว่าวิธีอื่นๆ โดยให้จำนวนหน่วยซ่อนที่น้อยที่สุด นอกจากนี้ยังให้ค่าอัตราผิดพลาดทดสอบที่ใกล้เคียงกับวิธี MBP ดังแสดงในตารางที่ 4.6

ตารางที่ 4.6 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหา Diabetes

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization	DNDM	EPNet
# hidden units	2.16	3.8	3.7	4.5	3.5	2.73	3.4
Test error rate	0.2144	0.1962	0.1988	0.1910	0.210	0.2234	0.224

ตารางที่ 4.7 แสดงผลการเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆสำหรับปัญหา Heart disease สำหรับปัญหานี้เป็นปัญหาเดียวที่สปาร์ตันซิมพลิซิตีไม่ได้ให้ผลที่ดีที่สุดในแง่ของจำนวนของหน่วยซ่อนและค่าเฉลี่ยอัตราผิดพลาดของการทดสอบ อย่างไรก็ตาม โครงข่ายสปาร์ตันของเราให้ผลการทดลองที่ใกล้เคียงกับผลของวิธี DNDM ซึ่งมีค่าจำนวนหน่วยซ่อนและค่าอัตราผิดพลาดทดสอบที่ดีกว่าวิธีอื่นๆ สำหรับกราฟแสดงการเปรียบเทียบในปัญหา Diabetes และ Heart disease ถูกแสดงในรูปที่ 4.5 และ 4.6

ตารางที่ 4.7 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหา Heart disease

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization	DNDM	EPNet
# hidden units	2.27	4.5	4.5	5.2	3.2	1.83	4.1
Test error rate	0.1751	0.1578	0.1578	0.1478	0.1651	0.186	0.168

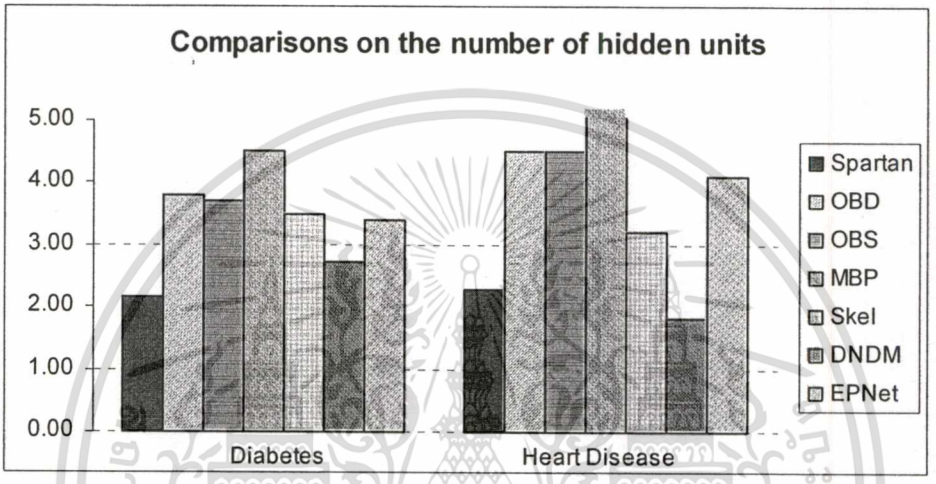
อันเนื่องมาจากประสิทธิภาพที่ใกล้เคียงระหว่าง อัลกอริธึมสปาร์ตันซิมพลิซิตีกับ Dynamic node decaying method (DNDM) [9] เราสามารถจำแนกประเด็นที่ DNDM แตกต่างจากอัลกอริธึมสปาร์ตันซิมพลิซิตี ได้ดังต่อไปนี้

ก. DNDM กำจัดหน่วยซ่อนโดยคำนวณ Goodness factor เป็นค่าความไม่เกี่ยวข้องของหน่วยซ่อน ซึ่งต่างจากอัลกอริธึมสปาร์ตันซิมพลิซิตีที่คำนวณค่าผลต่างของแอ็คติเวชั่น (AD_j) เพื่อใช้วัดความสำคัญของหน่วยซ่อนแต่ละหน่วย j

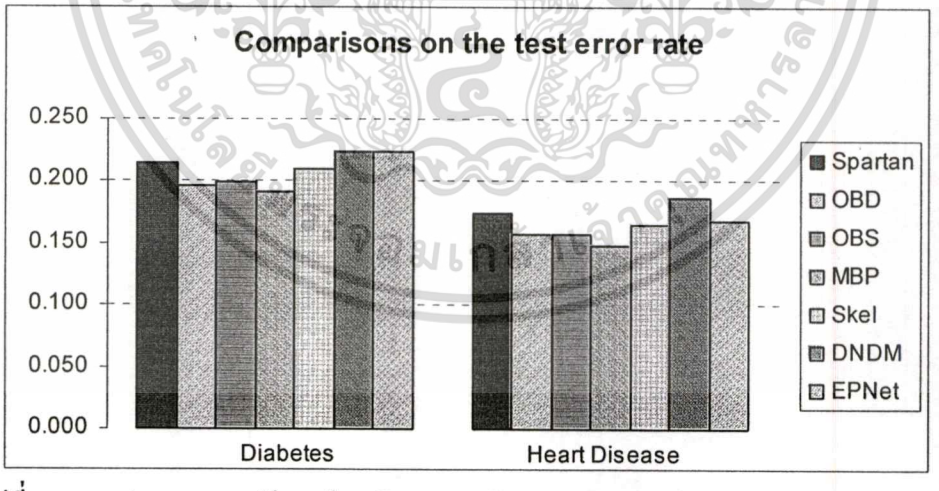
ข. DNDM คำนวณ Goodness factor จากข้อมูลชุดฝึก แต่อัลกอริธึมสปาร์ตันซิมพลิซิตีคำนวณค่าผลต่างของแอ็คติเวชั่นจากชุดมูลแวลิเดชั่น

ค. DNDM ปรับปรุงความสามารถในการทำนายข้อมูลของโครงข่ายประสาทเทียมโดยวิธีการลดทอนโหนด (Node decay) แต่อัลกอริธึมสปาร์ตันซิมพลิซิตีปรับปรุงความสามารถในการทำนายข้อมูลจากการทำการรู้จำบนชุดข้อมูลแวลิเดชั่น

ง. DNDM ไม่มีการตรวจสอบการกำจัดหน่วยซ่อนมากจนเกินไป แต่อัลกอริธึมสปาร์ตันซิมพลิซิตีตรวจสอบตัวชี้โอเวอร์พรุนนิ่งว่าโครงข่ายประสาทเทียมมีจำนวนหน่วยซ่อนที่เพียงพอต่อเรียนรู้ชุดข้อมูลหรือไม่



รูปที่ 4.5 กราฟแสดงการเปรียบเทียบจำนวนหน่วยซ่อนสำหรับปัญหา Diabetes และ Heart disease



รูปที่ 4.6 กราฟแสดงการเปรียบเทียบอัตราความผิดพลาดสำหรับปัญหา Diabetes และ Heart disease

การเปรียบเทียบผลลัพธ์สำหรับปัญหา Wine ถูกแสดงในตารางที่ 4.8 โดยอาศัยข้อมูลเปรียบเทียบที่แสดงไว้ในงาน [18] น่าสนใจว่าโครงข่ายสปาร์ตันมีค่าอัตราผิดพลาดของการ

ทดสอบต่ำกว่าวิธีอื่นๆมาก นอกจากนั้น โครงข่ายสปาร์ตันยังคงมีโครงสร้างที่เล็กที่สุดเมื่อเทียบกับวิธีอื่น เราไม่ได้เปรียบเทียบผลกับวิธี VNP เนื่องจาก VNP ลดจำนวนทั้งหน่วยอินพุทและหน่วยซ่อน

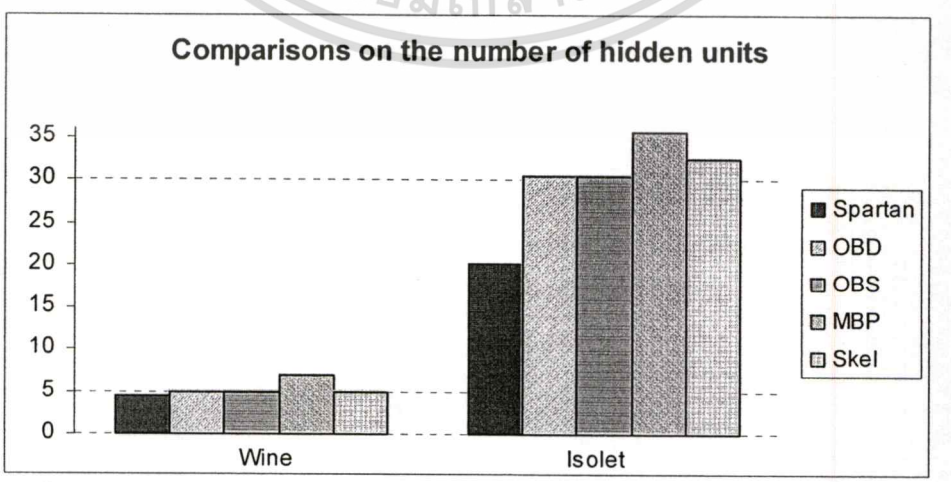
ตารางที่ 4.8 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหา Wine

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization
# hidden units	4.53	5	5	7	5
Test error rate	0.0068	0.028	0.028	0.014	0.029

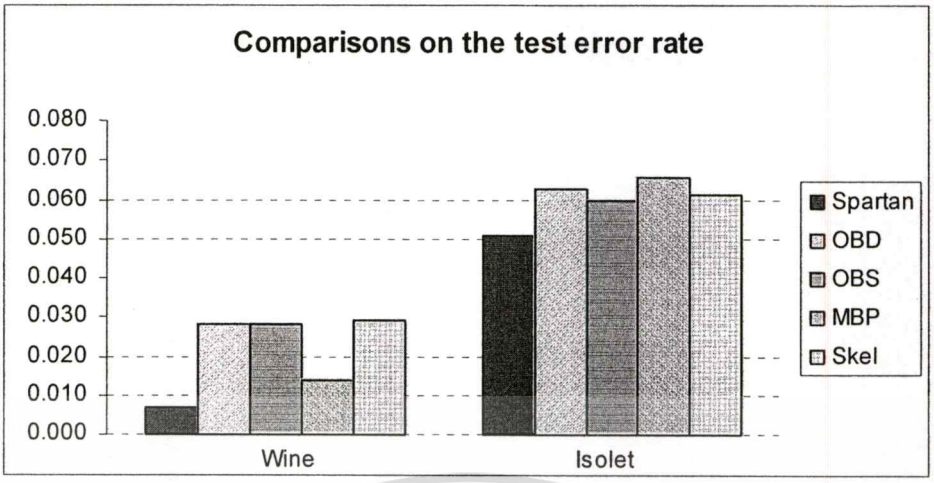
ตารางที่ 4.9 แสดงการเปรียบเทียบผลของวิธีที่นำเสนอกับวิธีอื่นๆสำหรับปัญหา Isolet ชุดข้อมูลนี้เป็นเสียงของคนทั้งหมด 150 คนซึ่งพูดชื่อของตัวอักษรแต่ละตัวโดยพูดตัวอักษรละสองครั้ง ปัญหา Isolet ถือได้ว่าเป็นปัญหาที่ดีในการทดสอบความสามารถของอัลกอริธึมในการทำงานกับปัญหาที่มีขนาดใหญ่เนื่องจากปัญหานี้มีจำนวนของแอททริบิวและจำนวนคลาสของเอาท์พุทที่มาก สปาร์ตันซิมพลิซิตีสามารถลดจำนวนหน่วยซ่อนจาก 78 ลงไปถึง 20.28 ในขณะที่ให้ค่าอัตราผิดพลาดทดสอบที่ใกล้เคียงกับผลการทดลองกับวิธีอื่นๆ สำหรับกราฟแสดงการเปรียบเทียบในปัญหา Wine และ Isolet ถูกแสดงในรูปที่ 4.7 และ 4.8

ตารางที่ 4.9 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหา Isolet

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization
# hidden units	20.28	30.55	30.40	35.5	32.5
Test error rate	0.0510	0.0627	0.0597	0.0655	0.0610



รูปที่ 4.7 กราฟแสดงการเปรียบเทียบจำนวนหน่วยซ่อนสำหรับปัญหา Wine และ Isolet

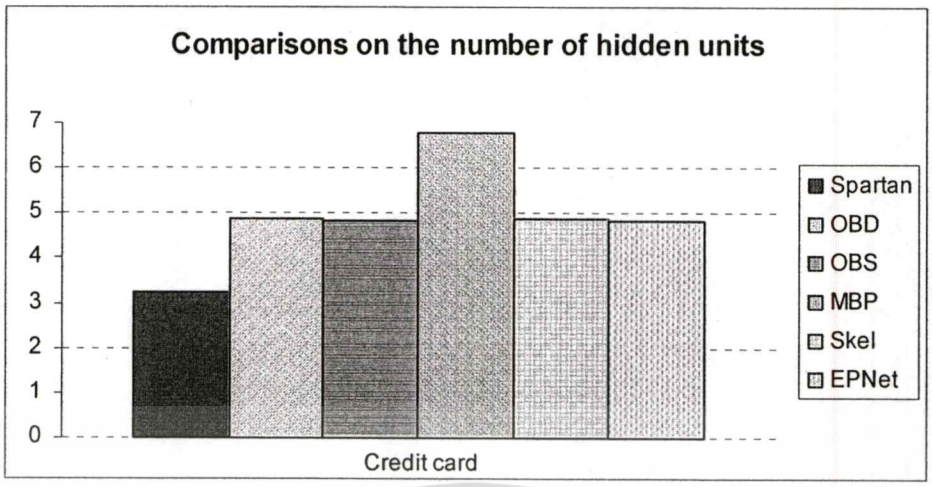


รูปที่ 4.8 กราฟแสดงการเปรียบเทียบอัตราความผิดพลาดสำหรับปัญหา Wine และ Isolet

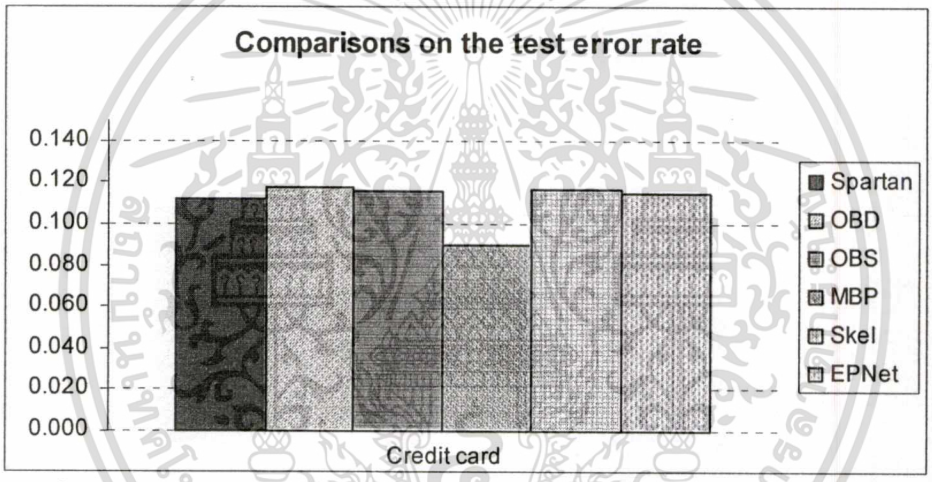
ปัญหาที่ยากอีกปัญหาคือ ปัญหา Credit card assessment ปัญหานี้มีจำนวนของแอททริบิวที่มีจำนวนมาก นั่นคือ 14 แอททริบิว นอกจากนี้ แอททริบิวบางตัวเท่านั้นที่เกี่ยวข้องกับการจำแนกประเภท การเปรียบเทียบผลการทดลองแสดงไว้ในตารางที่ 4.10 จะเห็นได้ว่าโครงข่ายสปาร์ตันมีค่าที่ต่ำที่สุดทั้งจำนวนหน่วยซ่อนและอัตราผิดพลาดทดสอบ วิธีอื่นๆ (ยกเว้น EPNet) ได้ผลลัพธ์มาจากการทำแวลิเดชั่นแบบสิบทบ (Ten-fold cross validation) แต่ยังคงมีอัตราความผิดพลาดที่สูงกว่าโครงข่ายสปาร์ตัน แม้ว่า EPNet ดูเหมือนว่าจะมีประสิทธิภาพใกล้เคียงกับสปาร์ตันซิมพลิซิตีแต่ค่าเฉลี่ยของจำนวนหน่วยซ่อนยังคงสูงกว่าค่าที่ได้จากโครงข่ายสปาร์ตัน (3.25) สำหรับกราฟแสดงการเปรียบเทียบในปัญหา Credit card ถูกแสดงในรูปที่ 4.9 และ 4.10

ตารางที่ 4.10 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีอื่นๆกับปัญหาCredit card

Algorithm	*Spartan*	OBD	OBS	MBP	Skeletonization	EPNet
# hidden units	3.25	4.9	4.85	6.8	4.9	4.83
Test error rate	0.1119	0.118	0.116	0.09	0.117	0.115



รูปที่ 4.9 กราฟแสดงการเปรียบเทียบจำนวนหน่วยซ่อนสำหรับปัญหา Credit card



รูปที่ 4.10 กราฟแสดงการเปรียบเทียบอัตราความผิดพลาดสำหรับปัญหา Credit card

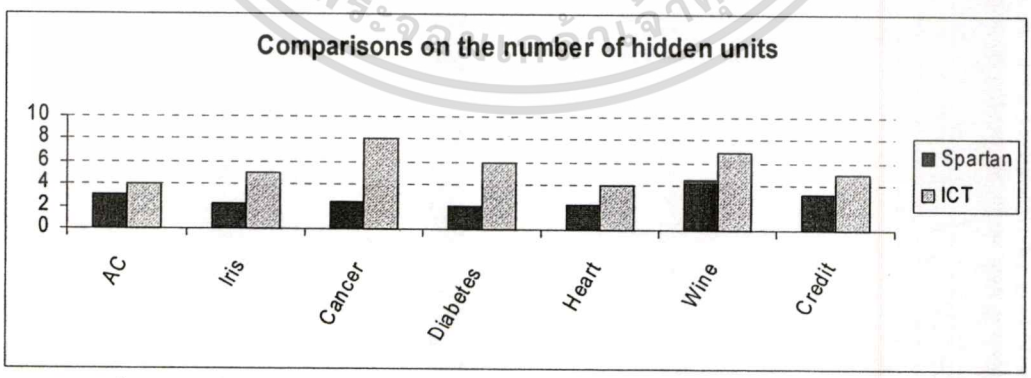
เพื่อเป็นการแสดงให้เห็นถึงความเป็นไปได้ของประสิทธิภาพของสปาร์ตันซิมพลิซิตีเราได้ทำการเปรียบเทียบผลลัพธ์กับวิธีมาตรฐาน (Intuitive construction technique: ICT) ชุดข้อมูลและเงื่อนไขการทดลองถูกควบคุมให้เป็นเช่นเดียวกับการทดลองก่อนหน้านี้ สำหรับวิธี ICT นี้ เราไม่ได้กำจัดหน่วยซ่อน แต่เราเริ่มต้นด้วยโครงข่ายที่มีโครงสร้างเล็กๆ (นั่นคือโครงข่ายที่ไม่มีหน่วยซ่อนอยู่เลย) จากนั้นทำการฝึกโครงข่ายโดยใช้อัลกอริธึมแบ็คพรอพาเกชั่น เราใช้การแทนแบบ 1-of-n และหลักการของวินเนอร์-เทค-ออล (Winner-take-all) ในการจำแนกประเภทข้อมูล นอกจากนี้เรายังใช้หน้าต่างเลื่อน (Sliding window) บนกราฟค่าผิดพลาดแวลิเดชั่นเพื่อหาจุดที่จะหยุดฝึกโครงข่ายตั้งแต่เนิ่นๆ ถ้าเราไม่สามารถเห็นค่าใหม่ที่น้อยที่สุดของค่าผิดพลาดแวลิเดชั่นภายใน 25 รอบ เราจะหยุดการฝึกพร้อมกับคืนน้ำหนักกลับไปยังจุดที่มีค่าผิดพลาดแวลิเดชั่นต่ำที่สุด จากนั้นเราจะเพิ่มหน่วยซ่อนเข้าไปหนึ่งหน่วยและฝึกโครงข่ายต่อไปในลักษณะเดิม อัลกอริธึม ICT หยุด

ทำงานเมื่อค่าผิดพลาดแวลิเดชั่นใหม่มีค่ามากกว่าค่าผิดพลาดแวลิเดชั่นที่น้อยที่สุดของโครงข่ายก่อนหน้านี้ สำหรับแต่ละโครงข่าย เราทำการทดลองกับค่าน้ำหนักเริ่มต้นและค่าอัตราการเรียนรู้ (Learning rate) ที่หลากหลายเพื่อให้ได้ผลลัพธ์ที่ดีที่สุด จากนั้น เราเลือกโครงข่ายที่ให้ผลลัพธ์ที่ดีที่สุดเมื่อถูกทดสอบด้วยชุดข้อมูลแวลิเดชั่นจากโครงข่ายทั้งหมดสิบโครงข่าย

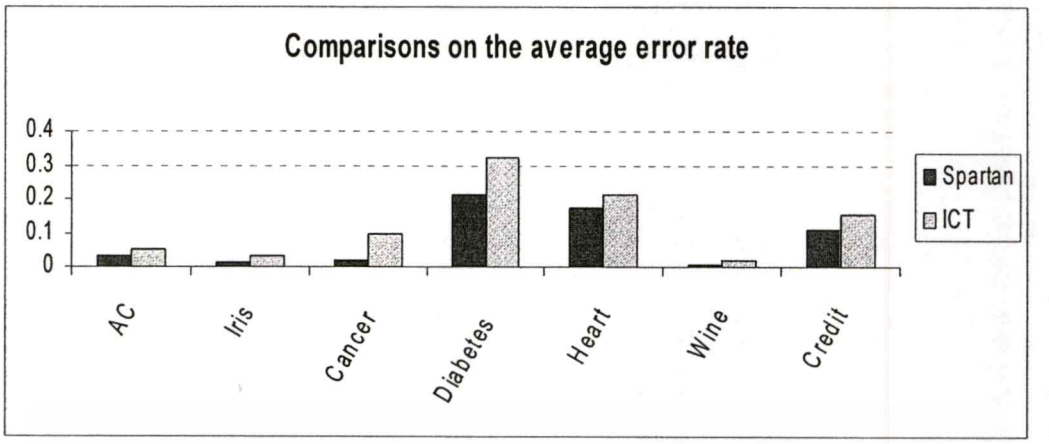
ตารางที่ 4.11 แสดงค่าเฉลี่ยของผลลัพธ์ของสปาร์ตันซิมพลิซิตีกับค่าที่ดีที่สุดที่ได้จากวิธีมาตรฐาน ICT เราจะเห็นได้ว่าวิธีของเรามีผลลัพธ์ที่ดีกว่าวิธีมาตรฐานในทุกๆกรณี ถึงแม้ว่าวิธีของเราอาจจะแตกต่างจากวิธีมาตรฐานในการสร้างโครงข่ายของหลายๆคนเนื่องจากทุกคนมีวิธีที่หลากหลายในการสร้างโครงข่ายประสาทเทียมเพื่อให้ได้โครงข่ายที่มีความเป็นเจนเนอรัลไลเซชั่นและมีโครงสร้างที่เล็ก อย่างน้อยวิธีของเราก็น่าจะแทนวิธีมาตรฐานที่อยู่ในใจของหลายๆคนได้ ไม่มากก็น้อย สำหรับกราฟแสดงการเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีมาตรฐาน ICT ถูกแสดงในรูปที่ 4.11 และ 4.12

ตารางที่ 4.11 การเปรียบเทียบระหว่างสปาร์ตันซิมพลิซิตีและวิธีมาตรฐาน ICT

	AC		Iris		Cancer		Diabetes	
	#HD	Err	#HD	Err	#HD	Err	#HD	Err
Spartan	3	0.030	2.25	0.0146	2.43	0.020	2.16	0.2144
ICT	4	0.050	5	0.030	8	0.100	6	0.3210
	Heart		Wine		Credit			
	#HD	Err	#HD	Err	#HD	Err		
Spartan	2.27	0.1751	4.53	0.0068	3.25	0.1119		
ICT	4	0.2110	7	0.0220	5	0.1520		



รูปที่ 4.11 กราฟแสดงการเปรียบเทียบจำนวนหน่วยซ่อนระหว่าง Spartan และ ICT



รูปที่ 4.12 กราฟแสดงการเปรียบเทียบอัตราความผิดพลาดระหว่าง Spartan และ ICT

บทที่ 5

การวิเคราะห์ผลการทดลอง

5.1 ผลกระทบของการใช้ชุดข้อมูลแวลิเดชั่นคำนวณค่าความสำคัญของหน่วยซ่อน

เราแสดงผลลัพธ์จากการทดลองบางอย่างที่ช่วยสนับสนุนว่าการคำนวณค่าชี้วัดความสำคัญของหน่วยซ่อนจากชุดข้อมูลแวลิเดชั่นจะสามารถบอกความไม่เกี่ยวข้องที่แท้จริงของหน่วยซ่อนได้เสมอ หรืออีกนัยหนึ่ง อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถเลือกหน่วยซ่อนที่มีผลกระทบน้อยที่สุดต่ออัตราผิดพลาดแวลิเดชั่น ตารางที่ 5.1 แสดงอัตราผิดพลาดแวลิเดชั่นหลังจากที่หน่วยซ่อนหนึ่งๆถูกกำจัดออกในสองขั้นตอนแรกของการกำจัดหน่วยซ่อน แต่ละช่องในตารางที่ 5.1 แสดงค่าผลต่างของแอ็คติเวชั่น (ค่าบน) ซึ่งถูกคำนวณมาจากชุดข้อมูลแวลิเดชั่นและค่าอัตราผิดพลาดแวลิเดชั่น(ค่าล่าง) หลังจากที่หน่วยซ่อนนั้นถูกกำจัด เราแสดงเพียงสองขั้นตอนแรกของการกำจัดหน่วยซ่อนเพื่อให้เห็นถึงประเด็นที่เราจะนำเสนอ เราจะเห็นได้จากตารางที่ 5.1 ว่า ณ ขั้นตอนใดๆของการกำจัดหน่วยซ่อน อัลกอริธึมสามารถเลือกกำจัดหน่วยซ่อนที่มีค่าผลต่างของแอ็คติเวชั่นที่น้อยที่สุดออกไปจากโครงข่าย ซึ่งทำให้ค่าอัตราผิดพลาดแวลิเดชั่นหลังจากที่หน่วยซ่อนดังกล่าวถูกกำจัดมีค่าน้อยที่สุดเมื่อเปรียบเทียบกับการกำจัดหน่วยซ่อนอื่นๆ ตัวเลขที่แสดงเป็น**แบบหนา**หมายถึงหน่วยซ่อนที่ถูกเลือกในแต่ละขั้นของการกำจัดหน่วยซ่อน สำหรับค่าอัตราผิดพลาดแวลิเดชั่นก่อนที่หน่วยซ่อนจะถูกกำจัดในแต่ละขั้นตอนถูกแสดงในตารางที่ 5.2

ในทางตรงกันข้าม การใช้ชุดข้อมูลฝึกในการคำนวณค่าผลต่างของแอ็คติเวชั่น (ดังแสดงในตารางที่ 5.3 โดยแต่ละช่องประกอบไปด้วยค่าผลต่างของแอ็คติเวชั่นและอัตราผิดพลาดแวลิเดชั่นหลังจากที่หน่วยซ่อนถูกกำจัดออกไป) มีข้อเสียเปรียบอยู่สองประการ ประการแรก ระหว่างช่วงแรกๆของการกำจัดหน่วยซ่อน โครงข่ายประสาทเทียมที่ถูกฝึกแล้วอาจจะเคยชินกับชุดข้อมูลฝึกจนกระทั่งค่าผลต่างของแอ็คติเวชั่นระหว่างก่อนและหลังการกำจัดหน่วยซ่อนมีค่าเป็นศูนย์ (ดังแสดงเป็น*ตัวเอน*ที่ขั้นตอนแรกของการกำจัดหน่วยซ่อนของปัญหา F2, AC และ Diabetes) สถานการณ์นี้เป็นอันตรายอย่างยิ่งในการกำจัดหน่วยซ่อนเนื่องจากค่าอัตราผิดพลาดแวลิเดชั่นมีความแตกต่างกันแม้ว่าค่าผลต่างของแอ็คติเวชั่นจะมีค่าเท่ากัน สิ่งที่ดีที่สุดที่อัลกอริธึมสามารถทำได้ก็คือการสุ่มเลือกหน่วยซ่อนที่จะกำจัดซึ่งอาจส่งผลให้อัตราผิดพลาดแวลิเดชั่นมีค่าแย่ที่สุด ประการที่สอง ค่าผลต่างของแอ็คติเวชั่นที่ต่ำที่สุดอาจไม่ได้ให้ค่าอัตราผิดพลาดแวลิเดชั่นที่ต่ำที่สุด สถานการณ์นี้แสดงโดยใช้ตัวเลขหนา ตัวอย่างเช่น ณ ขั้นที่สองของการกำจัดหน่วยซ่อนในปัญหา Iris หน่วยซ่อนที่ 4 มีค่าผลต่างของแอ็คติเวชั่น(4.0226) ต่ำที่สุด โดยมีค่าอัตราผิดพลาดแวลิเดชั่นหลังจากที่หน่วยซ่อนดังกล่าวถูกกำจัดเท่ากับ 0.0541 อย่างไรก็ตาม ค่าอัตราผิดพลาดแวลิเดชั่นนี้

ไม่ใช่เป็นค่าที่ต่ำที่สุดเมื่อเทียบกับการกำจัดหน่วยซ่อนอื่นๆ (แสดงด้วย**ตัวหนาและขีดเส้นใต้**) จากผลในตารางที่ 5.1 และ 5.3 เราสามารถสรุปสถานการณ์เมื่อหน่วยซ่อนหนึ่งๆถูกกำจัดออกไปจากโครงข่ายประสาทเทียมได้ดังรูปที่ 5.1

ตารางที่ 5.1 ขั้นของการกำจัดหน่วยซ่อนเมื่อคำนวณ AD_j จากชุดข้อมูลแวลิเดชั่น

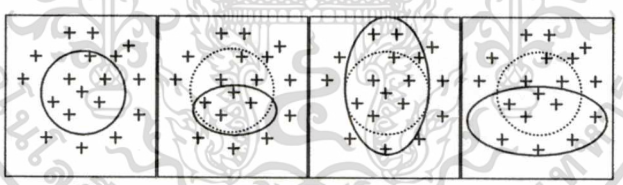
Phase	F1	F2	AC	Iris	Cancer	Diabetes	Heart	Wine	Credit
1	4.8493	14.149	7.2339	11.0287	12.0542	47.1713	15.0441	3.0346	18.1131
	0.0097	0.0776	0.0400	0.2973	0.0686	0.2448	0.1974	0.0682	0.1169
	3.9594	2.8180	23.1190	**1.0287**	18.0812	**47.1489**	20.0484	6.0347	**18.1026**
	0.0063	0.0037	0.1400	**0.0270**	0.1029	**0.2448**	0.2632	0.1364	**0.1169**
	6.2165	12.786	20.4391	2.0239	**10.0672**	50.1674	20.0517	6.0418	18.1101
	0.0302	0.0910	0.0400	0.0541	**0.0571**	0.2604	0.2632	0.1364	0.1169
	4.9793	**2.6838**	**6.7866**	13.0261	11.0599	53.1264	**15.0440**	22.041	19.1076
	0.0165	**0.0035**	**0.0000**	0.3514	0.0629	0.2760	**0.1974**	0.5000	0.1234
	25.330	18.324	43.0466	3.0274	11.0516	47.1502	18.0512	5.0365	39.1076
	0.2422	0.1316	0.4000	0.0811	0.0629	0.2448	0.2368	0.1136	0.2532
	1.4119	2.8654	19.2218	1.0290	12.0583	55.1765	17.0475	**3.0329**	18.1120
	0.0009	0.0037	0.1800	0.0270	0.0686	0.2865	0.2237	**0.0682**	0.1169
2	4.9634	14.836	24.0645	11.0321	13.0549	42.1503	**15.0441**	**4.0331**	18.0747
	0.0101	0.0845	0.1400	0.2973	0.0743	0.2188	**0.1974**	**0.0909**	0.1169
	4.0350	6.4943	19.3053	**2.0265**	16.0789	43.1450	20.0484	5.0326	**18.0725**
	0.0066	0.0123	0.0400	**0.0541**	0.0914	0.2240	0.2632	0.1136	**0.1169**
	6.2180	12.773	**9.7216**	13.0300	10.0713	53.1075	20.0517	5.0397	20.0704
	0.0302	0.0917	**0.0000**	0.3514	0.0571	0.2760	0.2632	0.1136	0.1299
	4.9829	18.045	43.1003	5.0304	**10.0591**	**37.1184**	18.0512	21.040	39.0901
	0.0166	0.1264	0.4000	0.1351	**0.0571**	**0.1927**	0.2237	0.4773	0.2532
	25.313	**3.2020**	19.1007	13.0364	12.0589	57.1636	17.0475	5.0345	19.0747
	0.2419	**0.0041**	0.1600	0.3514	0.0686	0.2969	0.2368	0.1136	0.1234
Final Test Err	0.0333	0.0180	0.02000	0.02632	0.01724	0.21875	0.14473	0.0889	0.09803

ตารางที่ 5.2 อัตราผิดพลาดแวลิเดชั่นก่อนที่จะกำจัดหน่วยซ่อนในแต่ละขั้น

Phase	F1	F2	AC	Iris	Cancer	Diabetes	Heart	Wine	Credit
1	0.0157	0.0035	0.0000	0.0541	0.0320	0.2448	0.1570	0.1136	0.0980
2	0.0031	0.0023	0.0000	0.0541	0.0450	0.1874	0.1650	0.0891	0.1054

ตารางที่ 5.3 ขั้นของการกำจัดหน่วยซ่อนเมื่อคำนวณ AD_j จากชุดข้อมูลฝึก

Phase	F1	F2	AC	Iris	Cancer	Diabetes	Heart	Wine	Credit
1	10.050	*0.0000*	*0.0000*	13.0399	20.0987	*0.0000*	**19.0687**	**9.0677**	49.2557
	0.0433	*0.0898*	*0.0800*	0.0811	0.0686	*0.2448*	**0.1974**	**0.1111**	0.1169
	20.012	*0.0000*	*0.0000*	4.0308	31.1652	*0.0000*	**18.0844**	24.0780	50.2299
	0.0428	*0.0074*	*0.0800*	0.0270	0.1029	*0.2448*	**0.2632**	0.1556	0.1169
	21.170	*0.0000*	*0.0000*	2.0152	**15.1111**	*0.0000*	35.0960	18.0820	50.2488
	0.0437	*0.0841*	*0.1200*	0.0270	**0.0571**	*0.2604*	0.2632	0.1111	0.1169
	19.850	*0.0000*	*0.0000*	13.0338	**13.0997**	*0.0000*	**19.0684**	30.0741	51.2419
	0.0446	*0.0008*	*0.0800*	0.0811	**0.0629**	*0.2760*	**0.1974**	0.4889	0.1234
	19.114	*0.0000*	*0.0000*	3.0225	15.0871	*0.0000*	**21.0857**	23.0819	68.2352
	0.0559	*0.0861*	*0.3800*	0.0541	0.0629	*0.2448*	**0.2368**	0.1333	0.2532
	18.470	*0.0000*	*0.0000*	27.0407	20.1073	*0.0000*	**20.0772**	**13.0660**	51.2527
	0.0481	*0.0065*	*0.2200*	0.3784	0.0686	*0.2865*	**0.2237**	**0.0889**	0.1169
2	**10.713**	18.012	74.3726	16.0401	24.0870	107.317	19.0669	24.0755	**48.1580**
	0.0436	0.0891	0.0800	0.1351	0.0914	0.1927	0.1974	0.1333	**0.1234**
	20.640	**4.1564**	74.7385	**4.0311**	42.1454	111.349	72.1084	18.0792	**49.1872**
	0.0429	**0.0074**	0.0800	**0.0270**	0.1143	0.2604	0.4211	0.1111	**0.1169**
	20.620	22.066	79.8323	13.0337	12.1128	145.303	19.0666	30.0736	**50.1764**
	0.0438	0.0841	0.1000	0.0811	0.0514	0.2760	0.1974	0.4889	**0.1169**
	21.176	10.144	119.262	**4.0226**	16.0819	104.319	21.0870	23.0806	71.1951
	0.0447	0.0865	0.3800	**0.0541**	0.0686	0.1927	0.2368	0.1333	0.2597
	23.396	**10.107**	82.6963	27.0402	26.0904	146.366	19.0770	13.0628	**51.1921**
	0.0579	**0.0063**	0.2200	0.3784	0.0857	0.4792	0.2368	0.0889	**0.1169**
Final Test Err	0.0439	0.0383	0.16000	0.05410	0.02873	0.22396	0.22368	0.09091	0.10457



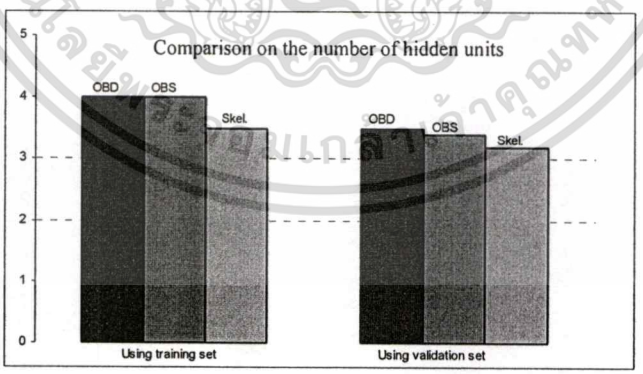
รูปที่ 5.1 ผลที่เกิดขึ้นจากการกำจัดหน่วยซ่อนหนึ่งๆ

จากรูปที่ 5.1 กากบาทในวงกลมแสดงถึงชุดข้อมูลแวลิเดชั่นที่ถูกแยกประเภทได้อย่างถูกต้องในโครงข่ายประสาทเทียมปัจจุบัน เมื่อหน่วยซ่อนหนึ่งๆถูกกำจัดออกไป ค่าอัตราผิดพลาดแวลิเดชั่นที่วัดได้หลังจากที่หน่วยซ่อนถูกกำจัดสามารถจำแนกได้เป็นสามประเภทดังต่อไปนี้ ประเภทแรก (ดังแสดงในรูปที่ 5.1 (ก)) คือสถานการณ์เมื่อค่าผิดพลาดก่อนและหลังการกำจัดหน่วยซ่อนมีค่าเท่ากัน ซึ่งหมายความว่าหน่วยซ่อนที่ถูกกำจัดไม่มีความสำคัญต่อโครงข่ายเลย อัลกอริธึมในการกำจัดหน่วยซ่อนส่วนใหญ่พยายามค้นหาหน่วยซ่อนที่ให้ผลเหมือนกับในรูปที่ 5.1(ก) ประเภทที่สองดังแสดงในรูปที่ 5.1(ข) คือสถานการณ์ที่โครงข่ายมีประสิทธิภาพที่แย่ลงเมื่อเรากำจัดหน่วยซ่อนออกไปจากโครงข่าย ค่าเจนเนอรัลไลเซชั่นจะลดลงขณะที่เวลาที่ใช้ในการฝึก

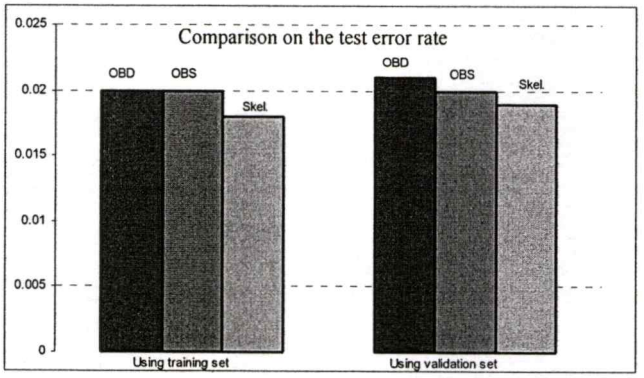
โครงข่ายก็จะเพิ่มขึ้นด้วย รูปที่ 5.1(ค) แสดงผลลัพธ์ในประเภทที่สาม เมื่อค่าเจนเนอรัลไลเซชั่นของโครงข่ายมีค่าดีขึ้นจากเดิม อย่างไรก็ดีเหตุการณ์นี้เกิดขึ้นไม่บ่อยนัก รูปที่ 5.1(ง) แสดงสถานการณ์เมื่อข้อมูลที่เคยทำนายถูกก่อนที่หน่วยซ่อนจะถูกกำจัด กลับทำนายไม่ถูกเมื่อหน่วยซ่อนถูกกำจัดออกไปแล้ว และในทางตรงข้ามเมื่อข้อมูลบางส่วนที่ทำนายผิด แต่กลับทำนายได้ถูกต้องหลังจากการกำจัดหน่วยซ่อน อันเนื่องมาจากความสามารถในการคำนวณค่าผิดพลาดแวลิเดชั่นและค่าผลต่างของแอ็คติเวชั่นล่วงหน้า อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถเลือกกำจัดหน่วยซ่อนที่มีผลกระทบน้อยที่สุดต่อค่าอัตราผิดพลาดแวลิเดชั่นเมื่อเทียบกับหน่วยซ่อนอื่นๆ ดังนั้นอัลกอริธึมของเราเลือกทางเลือกที่ดีที่สุดไม่ว่าจะเกิดเหตุการณ์ใดขึ้นก็ตาม

5.2 การใช้ชุดข้อมูลแวลิเดชั่นคำนวณค่าความสำคัญของหน่วยซ่อนกับวิธีอื่นๆ

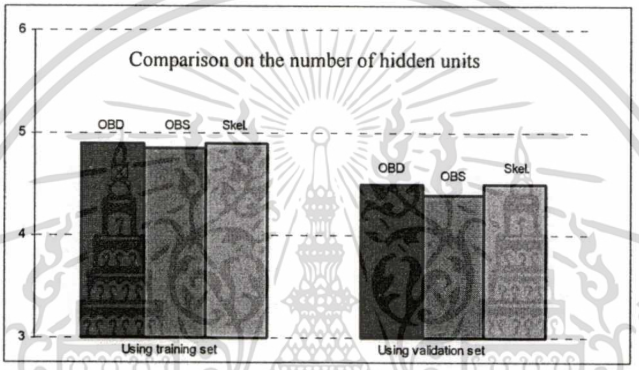
ในหัวข้อนี้ เราจะวิเคราะห์ผลการทดลองในแง่ของการใช้ชุดข้อมูลแวลิเดชั่นสำหรับคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนในวิธีการกำจัดหน่วยซ่อนอื่นๆที่มีอยู่ นั่นคือ OBD, OBS และ Skeletonization เพื่อวิเคราะห์ว่าชุดข้อมูลแวลิเดชั่นมีผลอย่างไรต่อการคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อน เราทำการแก้ไขอัลกอริธึมสำหรับกำจัดหน่วยซ่อนเหล่านั้นซึ่งมีอยู่แล้วในโปรแกรม Stuttgart NN simulator (SNNS) โดยใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อน การทดลองนี้ถูกกระทำภายใต้เงื่อนไขเดียวกันกับที่ได้ระบุไว้ก่อนหน้านี้ การเปรียบเทียบผลการทดลองระหว่างอัลกอริธึมดั้งเดิมกับอัลกอริธึมที่ถูกดัดแปลงถูกแสดงไว้ในรูปที่ 5.2 – 5.7 เราเลือกทำการทดลองกับชุดข้อมูลสามชุดคือ Iris, Credit card และ Isolet เพื่อแทนชุดข้อมูลที่มีขนาดเล็ก กลาง และใหญ่ ตามลำดับ



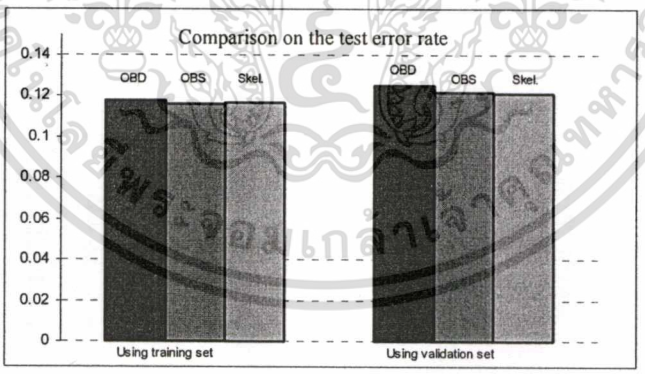
รูปที่ 5.2 จำนวนของหน่วยซ่อนสำหรับปัญหา Iris



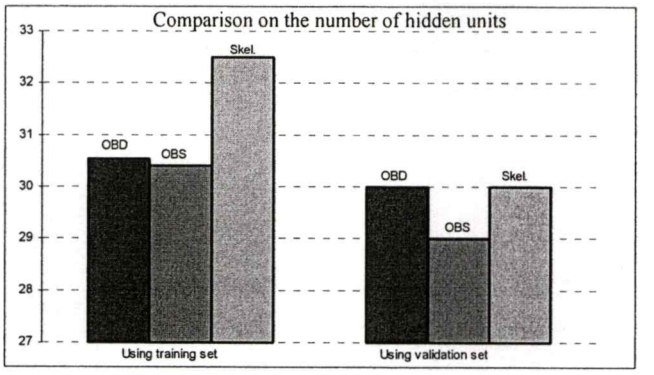
รูปที่ 5.3 อัตราผิดพลาดทดสอบสำหรับปัญหา Iris



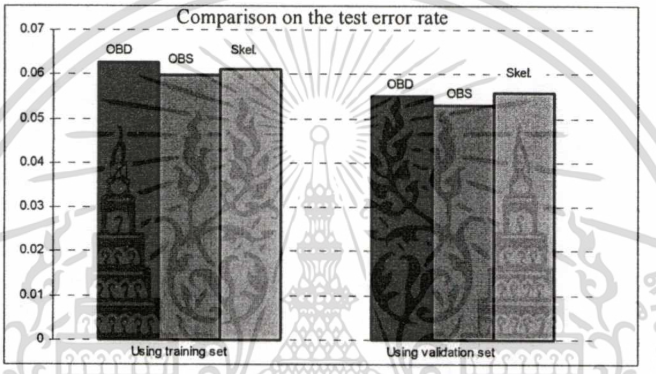
รูปที่ 5.4 จำนวนหน่วยซ่อนสำหรับปัญหา Credit card



รูปที่ 5.5 อัตราผิดพลาดทดสอบสำหรับปัญหา Credit card



รูปที่ 5.6 จำนวนหน่วยซ่อนสำหรับปัญหา Isolet



รูปที่ 5.7 อัตราผิดพลาดทดสอบสำหรับปัญหา Isolet

ดังจะสังเกตเห็นได้ว่า ในทุกๆกรณีของทุกๆชุดข้อมูล การใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณค่าชี้วัดความสำคัญของหน่วยซ่อนสามารถให้จำนวนหน่วยซ่อนสุดท้ายต่ำที่สุดเมื่อเทียบกับการใช้ชุดข้อมูลฝึกในการคำนวณ นอกจากนี้ การใช้ชุดข้อมูลแวลิเดชั่นยังคงความสามารถในการทำนายข้อมูลที่ไม่เคยเห็นมาก่อนได้ดีเทียบเท่าหรือดีกว่าการใช้ชุดข้อมูลฝึกอีกด้วย ผลการทดลองนี้จึงเป็นการยืนยันผลกระทบในเชิงบวกของการใช้ชุดข้อมูลแวลิเดชั่นคำนวณค่าชี้วัดความสำคัญของหน่วยซ่อนได้เป็นอย่างดี อนึ่ง อาจมีคำถามว่าเหตุใดวิธีการกำจัดหน่วยซ่อนที่มีชื่อเสียงหลายวิธีจึงไม่ใช้ชุดข้อมูลแวลิเดชั่นสำหรับคำนวณค่าความสำคัญของหน่วยซ่อน คำถามนี้สามารถอธิบายได้ดังนี้ วิธีการกำจัดหน่วยซ่อนเหล่านี้ไม่ได้ถูกออกแบบมาเพื่อใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณ เนื่องจากค่าชี้วัดความสำคัญเหล่านี้ถูกคำนวณสะสมตลอดช่วงเวลาของการฝึกโครงข่าย ชุดข้อมูลที่ใช้คำนวณค่าชี้วัดความสำคัญจึงต้องเป็นชุดข้อมูลฝึก หากจะใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณชุดข้อมูลแวลิเดชั่นจะต้องถูกนำไปรวมกับชุดข้อมูลฝึกเพื่อนำไปฝึกโครงข่ายและคำนวณค่าความสำคัญของหน่วยซ่อน ส่งผลให้การหาหน่วยซ่อนที่ไม่สำคัญมีความแม่นยำขึ้นจากเดิม อย่างไรก็ตาม โครงข่ายประสาทเทียมที่ถูกฝึกด้วยชุดข้อมูลทั้งสองจะพยายามจดจำข้อมูลทั้งหมด ทำให้เกิดปัญหาการฝึกโครงข่ายมากจนเกินไป (Overfitting) นอกจากนี้ เรายังไม่สามารถใช้ชุด

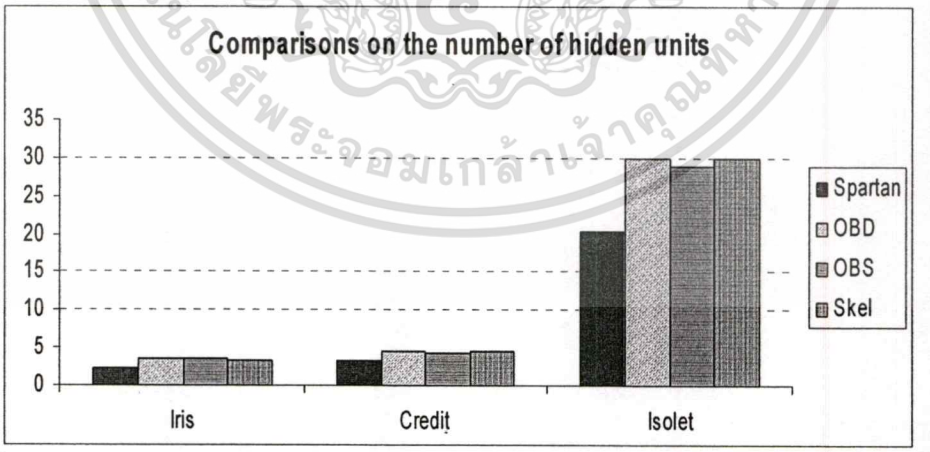
ข้อมูลแวลิเดชั่นมาหยุดการฝึกโครงข่ายก่อนที่จะเกิดปัญหาดังกล่าวได้ เนื่องจากโครงข่ายได้เห็นชุดข้อมูลแวลิเดชั่นในช่วงการฝึกมาแล้ว ส่งผลให้การหาหน่วยซ่อนที่ไม่สำคัญมีประสิทธิภาพไม่เต็มที่

ถึงแม้ว่าชุดข้อมูลแวลิเดชั่นจะถูกใช้ในการคำนวณค่าความสำคัญของหน่วยซ่อนในอัลกอริธึมสำหรับกำจัดหน่วยซ่อนอื่นๆ เป็นที่น่าสนใจว่าอัลกอริธึมสปาร์ตันซิมพลิซิตียังคงมีประสิทธิภาพที่เหนือกว่าอัลกอริธึมอื่นๆ ดังแสดงในตารางที่ 5.4

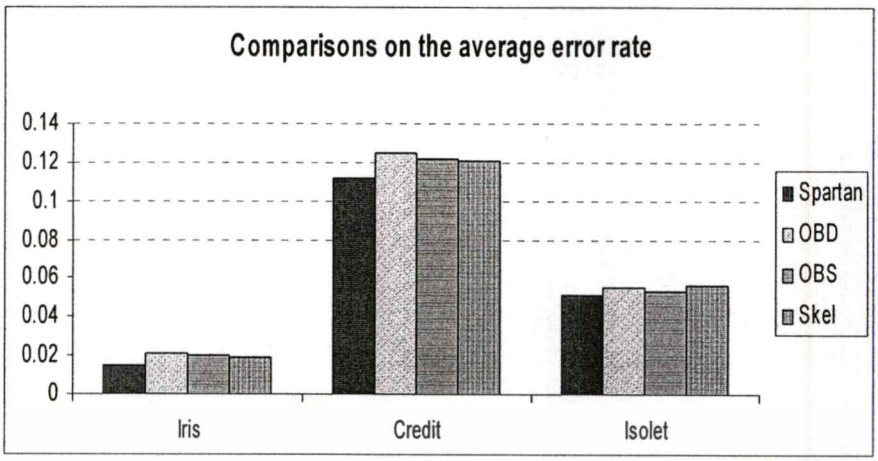
จากตารางที่ 5.4 อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถได้จำนวนหน่วยซ่อนเฉลี่ยสุดท้ายและอัตราผิดพลาดทดสอบเฉลี่ยน้อยที่สุด นอกจากนี้ อัลกอริธึมของเรายังใช้เวลาในการทำงานน้อยที่สุดอีกด้วย ทั้งนี้เป็นเพราะความเรียบง่ายของอัลกอริธึมและการออกแบบวิธีการลดความซับซ้อนของการคำนวณค่าชี้วัดความสำคัญของหน่วยซ่อน ในขณะที่ OBS ใช้เวลาในการทำงานนานที่สุดอันเนื่องมาจากการคำนวณเฮสเซี่ยนแมทริกซ์ที่มีความซับซ้อน สำหรับกราฟแสดงการเปรียบเทียบระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีกับวิธีอื่นๆถูกแสดงในรูปที่ 5.8 และ 5.9

ตารางที่ 5.4 การเปรียบเทียบระหว่างโครงข่ายสปาร์ตันและวิธีอื่นๆเมื่อชุดข้อมูลแวลิเดชั่นถูกใช้ในการคำนวณค่าความสำคัญของหน่วยซ่อน

	Spartan			OBD			OBS			Skeletonization		
	Avg. #HD	Avg. err rate	Avg. CPU time (s.)	Avg. #HD	Avg. err rate	Avg. CPU time (s.)	Avg. #HD	Avg. err rate	Avg. CPU time (s.)	Avg. #HD	Avg. err rate	Avg. CPU time (s.)
Iris	2.25	0.015	935	3.5	0.021	960	3.4	0.020	2000	3.2	0.019	945
Credit	3.25	0.112	1950	4.5	0.125	2005	4.4	0.122	5895	4.5	0.121	1987
Isolet	20.28	0.051	12610	30	0.055	12670	29	0.053	54810	30	0.056	12650

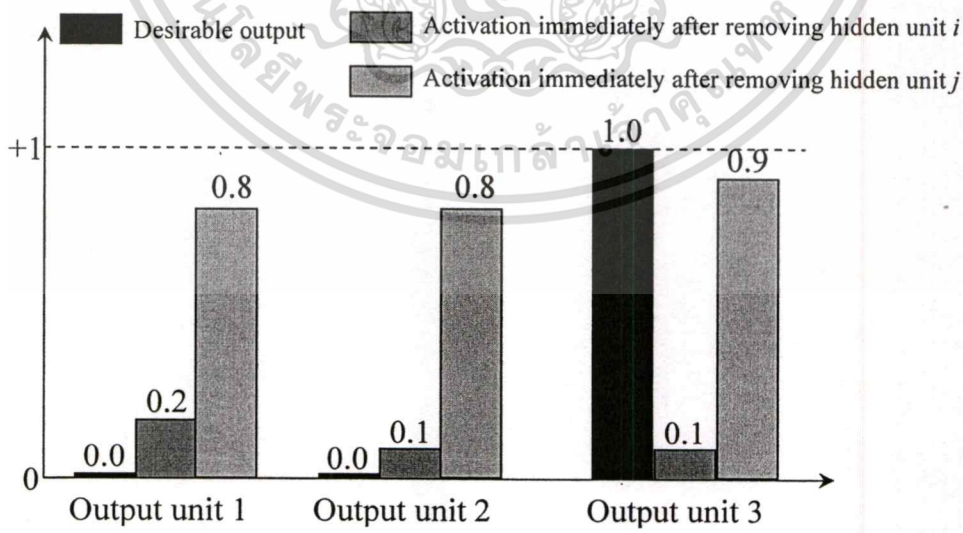


รูปที่ 5.8 กราฟเปรียบเทียบจำนวนหน่วยซ่อนระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีกับวิธีอื่นๆ



รูปที่ 5.9 กราฟเปรียบเทียบอัตราผิดพลาดระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีกับวิธีอื่นๆ

ก่อนที่จะอธิบายว่าเหตุใดอัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถทำงานได้ดีกว่าอัลกอริธึมอื่นๆ ถึงแม้ว่าเราจะใช้ชุดข้อมูลแวลิเดชั่นในการคำนวณค่าความสำคัญของหน่วยซ่อนในวิธีอื่นๆก็ตาม เราจะทบทวนวิธีการคำนวณค่าชี้วัดความสำคัญของหน่วยซ่อนของอัลกอริธึมอื่นๆ ดังนี้ ค่าชี้ความสำคัญของหน่วยซ่อนสำหรับวิธี OBD, OBS และ Skeletonization ถูกคำนวณมาจากการคำนวณเดริเวชั่นลำดับแรกหรือลำดับสอง (First/ Second derivation) ของฟังก์ชั่นผิดพลาด (Error function) หลังจากที่พารามิเตอร์ (น้ำหนัก หรือหน่วยซ่อน) ถูกกำจัดออกไปจากโครงข่าย พารามิเตอร์ที่มีค่าชี้วัดความสำคัญน้อยที่สุดก็จะถูกกำจัดออกไปจากโครงข่าย ดังตัวอย่างที่แสดงในรูปที่ 5.10



รูปที่ 5.10 กราฟแสดงค่าเอาท์พุทที่ต้องการ (Desirable outputs) และค่าแอ็คติเวชั่นหลังจากที่หน่วยซ่อน i และ j ออกทันที

จากรูปที่ 5.10 เป็นตัวอย่างการกำจัดหน่วยซ่อนของโครงข่ายที่มีหน่วยเอาท์พุทสามหน่วย กำหนดให้ ρ_n เป็นค่าชี้วัดความสำคัญเมื่อหน่วยซ่อน n ถูกกำจัดออกไปจากโครงข่าย จากกราฟที่แสดงในรูปที่ 5.10 เป็นที่ชัดเจนว่าหน่วยซ่อน i มีค่าความสำคัญน้อยกว่าหน่วยซ่อน j ($\rho_i < \rho_j$) เนื่องจากค่าผิดพลาดหลังจากที่กำจัดหน่วยซ่อน i มีค่าน้อยกว่าค่าผิดพลาดเมื่อหน่วยซ่อน j ถูกกำจัด ดังนั้น อัลกอริธึมทั้งสามนี้จึงเลือกกำจัดหน่วยซ่อน i ออกจากโครงข่าย อย่างไรก็ตาม การใช้วิธีการจำแนกประเภทแบบวินเนอร์-เทค-ออล (Winner-take-all) อัลกอริธึมเหล่านี้จะทำนายข้อมูลผิด เนื่องจากโครงข่ายที่ถูกกำจัดหน่วยซ่อนสร้างเอาท์พุทเป็น <1, 0, 0> แทนที่จะเป็น <0, 0, 1> ในทางตรงข้าม ถ้าอัลกอริธึมเหล่านั้นเลือกที่จะกำจัดหน่วยซ่อน j โครงข่ายที่ได้จะสามารถทำนายข้อมูลนี้ได้อย่างถูกต้องเนื่องจากโครงข่ายสร้างเอาท์พุทเป็น <0, 0, 1> ดังนั้น ค่าชี้วัดความสำคัญสำหรับอัลกอริธึมอื่นๆที่เป็นที่รู้จักจึงประเมินความสำคัญของหน่วยซ่อนผิดพลาด ปัญหานี้สามารถแก้ไขได้โดยการนำจำนวนของข้อมูลแวลิเดชั่นที่ทำนายผิดพลาดมาพิจารณาเป็นอันดับแรก ดังนั้นอัลกอริธึสปาร์ตันซิมพลิซิตีจึงให้ความสำคัญกับจำนวนข้อมูลแวลิเดชั่นที่ทำนายผิดพลาดดังแสดงในสมการที่ (3.1) นอกจากนี้ เพื่อให้เห็นความถี่ที่อัลกอริธึมสำหรับกำจัดหน่วยซ่อนทำนายความสำคัญของหน่วยซ่อนผิดพลาด เราแสดงจำนวนของการกำจัดหน่วยซ่อนผิดพลาดจากการสุ่มทั้งหมดหนึ่งร้อยครั้งในตารางที่ 5.5 เราจะเห็นได้ว่าอัลกอริธึมสปาร์ตันซิมพลิซิตีไม่เคยกำจัดหน่วยซ่อนผิดเลย ในขณะที่ OBD มีจำนวนครั้งของการกำจัดหน่วยซ่อนผิดพลาดมากที่สุด ถ้าเกิดการกำจัดหน่วยซ่อนผิดพลาด ค่าผิดพลาดเอาท์พุทจะมีขนาดใหญ่และโครงข่ายจะเข้าสู่คอนเวอร์เจนซ์ด้วยความยากลำบาก ดังนั้น จึงทำให้โครงข่ายยังคงเหลือหน่วยซ่อนที่ไม่จำเป็นอยู่ซึ่งหน่วยซ่อนเหล่านี้จะขัดขวางความสามารถในการทำนายข้อมูลของโครงข่ายประสาทเทียม

ตารางที่ 5.5 จำนวนการกำจัดหน่วยซ่อนผิดหน่วยสำหรับทั้งสามปัญหา

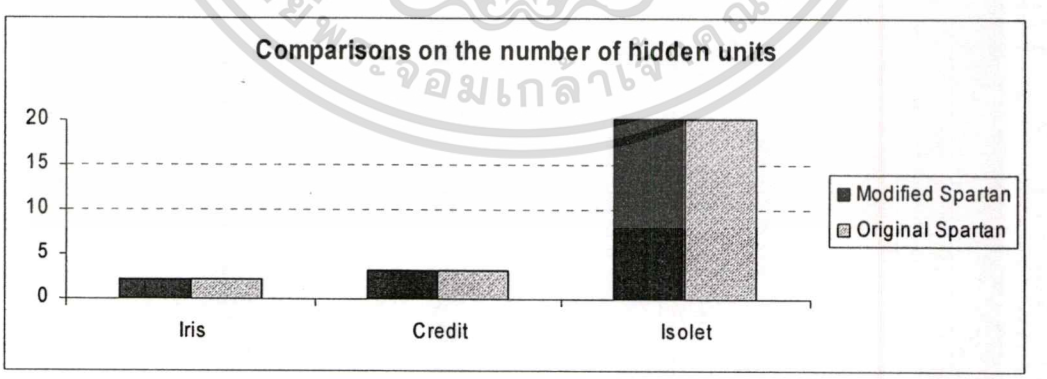
	Spartan	OBD	OBS	Skeletonization
Iris	0	3	2	2
Credit	0	7	3	5
Isolet	0	13	9	14

ประเด็นที่น่าสนใจประการหนึ่งเกี่ยวกับวิธีที่เรานำเสนอคือ มีความเป็นไปได้เพียงใดที่อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถกำจัดหน่วยซ่อนได้ครั้งละมากกว่าหนึ่งหน่วย เราแสดงความเป็นไปได้นี้โดยดัดแปลงอัลกอริธึมดังนี้ สำหรับหน่วยซ่อนซึ่งมีค่า AD_j เกินค่าเทรชโฮลด์ เรากำจัดหน่วยซ่อนเหล่านั้นและฝึกโครงข่ายต่อไป ถ้าจำนวนหน่วยซ่อนถูกกำจัดมากจนเกินไป เราจะคืนหน่วยซ่อนที่ถูกกำจัดที่มีค่า AD_j มากที่สุดกลับมาและทำการฝึกโครงข่ายต่อ ขบวนการคืนและ

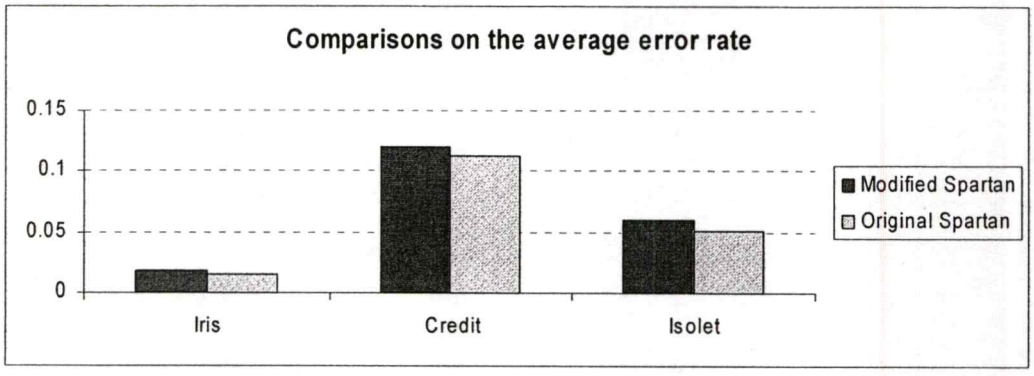
กำจัดหน่วยซ่อนจะดำเนินต่อไปจนกระทั่งตัวชี้โอเวอร์พรุนนิ่งหายไป จากนั้นจึงแสดงโครงข่ายสุดท้ายที่ได้ ตารางที่ 5.6 แสดงการเปรียบเทียบระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีที่ถูกดัดแปลงกับอัลกอริธึมดั้งเดิม (จากตารางที่ 5.4) ผลของทั้งสองเวอร์ชั่นในแง่จำนวนหน่วยซ่อนสุดท้ายเฉลี่ยและอัตราผิดพลาดเฉลี่ยไม่ได้แตกต่างกันมากนัก ในทางตรงกันข้าม เวลาของซีพียูของอัลกอริธึมที่ดัดแปลงถูกลดลงประมาณครึ่งหนึ่งในทุกๆชุดข้อมูลเนื่องจากสามารถกำจัดหน่วยซ่อนได้ครั้งละหลายๆหน่วย อย่างไรก็ตาม เราต้องการวิธีการคำนวณค่าเทรชโฮลด์สำหรับ AD_j ที่เหมาะสม เนื่องจากการทดลองนี้เป็นการใช้ค่าเทรชโฮล์ด ซึ่งถูกเลือกมาจากการทดลองทำงานอัลกอริธึมในรอบแรก สิ่งนี้จึงถือได้ว่าเป็นงานที่สามารถต่อยอดได้ในอนาคต สำหรับกราฟแสดงการเปรียบเทียบระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีที่ถูกดัดแปลงกับอัลกอริธึมดั้งเดิมถูกแสดงในรูปที่ 5.11, 5.12 และ 5.13

ตารางที่ 5.6 การเปรียบเทียบระหว่างโครงข่ายสปาร์ตันที่ถูกดัดแปลงและโครงข่ายสปาร์ตันดั้งเดิม

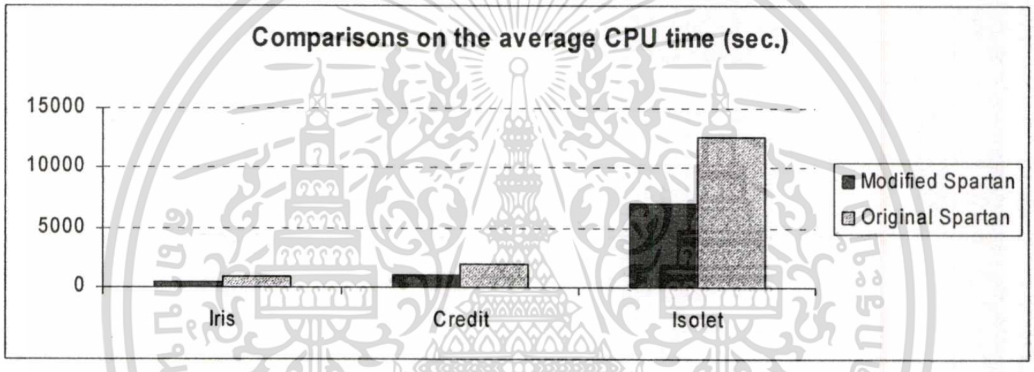
	Modified Spartan			Original Spartan		
	Avg. #HD	Avg. err rate	Avg. CPU time (s.)	Avg. #HD	Avg. err rate	Avg. CPU time (s.)
Iris	2.21	0.018	509	2.25	0.015	935
Credit	3.20	0.120	1015	3.25	0.112	1950
Isolet	20.21	0.060	6983	20.28	0.051	12610



รูปที่ 5.11 กราฟเปรียบเทียบจำนวนหน่วยซ่อนระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีที่ถูกดัดแปลงกับอัลกอริธึมดั้งเดิม



รูปที่ 5.12 กราฟเปรียบเทียบอัตราผิดพลาดระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีที่ถูกดัดแปลงกับอัลกอริธึมดั้งเดิม



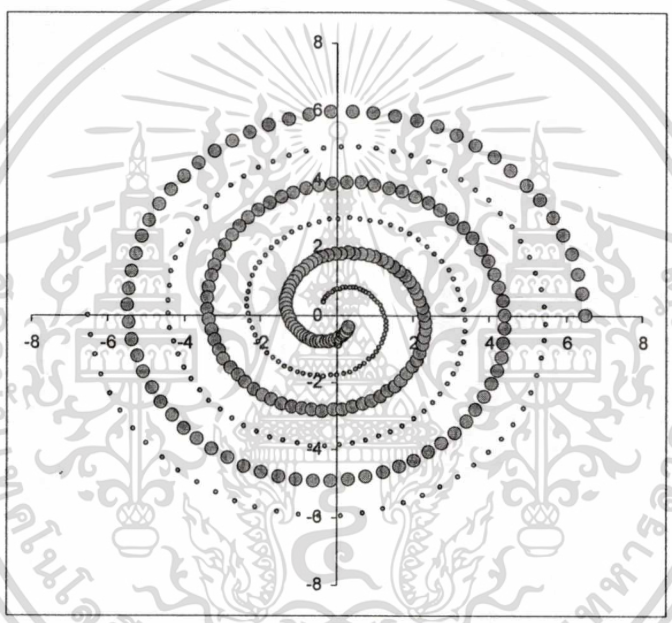
รูปที่ 5.13 กราฟเปรียบเทียบเวลาทำงานระหว่างอัลกอริธึมสปาร์ตันซิมพลิซิตีที่ถูกดัดแปลงกับอัลกอริธึมดั้งเดิม

5.3 การขยายผลการทดลองสำหรับปัญหาที่มีลักษณะไม่ใช่เชิงเส้น (Nonlinear)

เพื่อเป็นการทดสอบความสามารถในการกำจัดหน่วยซ่อนของอัลกอริธึมสปาร์ตันซิมพลิซิตีกับปัญหาที่มีลักษณะไม่เชิงเส้น ซึ่งต้องการจำนวนหน่วยซ่อนที่มากขึ้นในการจำแนกประเภท ปัญหา Two Spirals จึงถูกนำมาใช้ทดสอบ

ปัญหา Two Spirals (ถูกนำเสนอครั้งแรกโดย Alexis Wieland ปัจจุบันถือว่าเป็นชุดข้อมูลที่สำคัญของ Carnegie Mellon Repository) เป็นปัญหาที่มีความยากในการจำแนกประเภท เนื่องจากปัญหานี้เป็นปัญหาที่ไม่เป็นเชิงเส้นอย่างมาก (Highly nonlinear) ปัญหานี้เกิดจากการสังเคราะห์ข้อมูลสองแกนคือ x และ y โดยเอาท์พุทที่ได้มีค่าเป็นหนึ่งถ้าคู่ลำดับ x และ y นั้นอยู่บนเส้นก้นหอย A และเอาท์พุทจะมีค่าเป็นศูนย์ถ้าคู่ลำดับนั้นอยู่บนเส้นก้นหอย B ดังแสดงในรูปที่ 5.14 สำหรับตัวอย่างโปรแกรมเทียมที่สร้างชุดข้อมูลนี้คือ

```
data spirals;
    pi = 22/7;
    do i = 0 to 96;
      angle = i*pi/16.0;
      radius = 6.5*(104-i)/104;
      x = radius*cos(angle);
      y = radius*sin(angle);
      c = 1;
      print(x, y, c);
      x = -x;
      y = -y;
      c = 0;
      print(x, y, c);
    end;
run;
```



รูปที่ 5.14 กราฟจากชุดข้อมูลจากปัญหา Two Spirals

Baum และ Lang [19] แนะนำว่า สำหรับโครงข่ายประสาทเทียมที่มีจำนวนชั้นซ่อนหนึ่งชั้น ควรจะมีจำนวนหน่วยซ่อนสำหรับปัญหา Two Spirals จำนวนเท่ากับ 50 หน่วย ทั้งนี้เพื่อให้ได้อัตราความถูกต้องของการจำแนกประเภทร้อยละ 90 ดังนั้น เราจึงใช้จำนวนหน่วยซ่อนเริ่มต้นที่มีจำนวนมากพอที่จะทำให้โครงข่ายประสาทเทียมมีความสามารถในการทำนายข้อมูลได้ไม่ต่ำกว่าร้อยละ 90 ในการทดลองนี้เราใช้จำนวนหน่วยซ่อนเริ่มต้น 60 หน่วย โดยมีค่าพารามิเตอร์สำหรับอัลกอริธึมสปาร์ตันชิมพลิซิตี คือ $\beta = 420$ สิ่งหนึ่งที่ควรอธิบายในที่นี้ คือ ข้อมูลสำหรับปัญหานี้จะเป็นข้อมูลเพียงชุดเดียวและมีจำนวนชุดข้อมูลค่อนข้างน้อย นอกจากนี้ ปัญหานี้มีความไม่เป็นเชิงเส้นอย่างสูง ดังนั้นโครงข่ายประสาทเทียมจึงต้องการชุดข้อมูลทั้งหมด 194 จุดมาใช้ในการฝึกโครงข่าย อนึ่ง ปัญหานี้เป็นปัญหาที่ไม่มีข้อมูลรบกวน (Non-noise problem) ซึ่งจะไม่เกิดปัญหาการฝึกโครงข่ายมากจนเกินไป (Overfitting problem) ดังนั้น ค่าพารามิเตอร์ α จึงไม่ถูกใช้งาน เรา

จะหยุดการฝึกโครงข่ายประสาทเทียมเมื่อค่าอัตราผิดพลาดมีค่าน้อยกว่าค่าที่ยอมรับได้ ในที่นี้ เราเลือกค่า 0.11 ผลการทดลองที่ได้จากการค่าเฉลี่ยจากการทำงานทั้งสิ้น 30 ครั้งถูกแสดงในตารางที่ 5.7

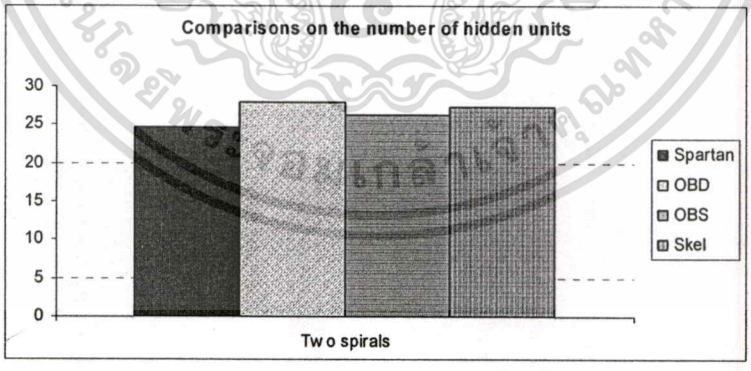
ตารางที่ 5.7 ผลการทดลองที่ได้จากสปาร์ตันซิมพลิซิตีในปัญหา Two Spirals

Problem		Final # of hidden units	err rate
Two Spirals	Mean	24.59	0.1031
	SD	1.89	0.0750

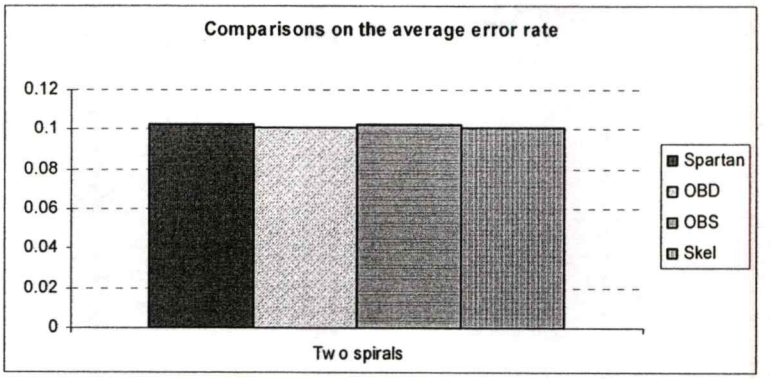
กราฟจากการทำงานของอัลกอริธึมสปาร์ตันซิมพลิซิตีสำหรับปัญหา Two Spirals สามารถแสดงได้ในรูปที่ 5.17 นอกจากนี้ การเปรียบเทียบผลการทดลองกับอัลกอริธึม OBD, OBS และ Skeletonization ยังถูกแสดงในตารางที่ 5.8 (สำหรับกราฟถูกแสดงในรูป 5.15 และ 5.16) ซึ่งทั้งสามอัลกอริธึมนี้ใช้เงื่อนไขในการทดลองเหมือนกับ Spartan

ตารางที่ 5.8 การเปรียบเทียบผลทดลองกับอัลกอริธึมอื่นๆสำหรับปัญหา Two Spirals

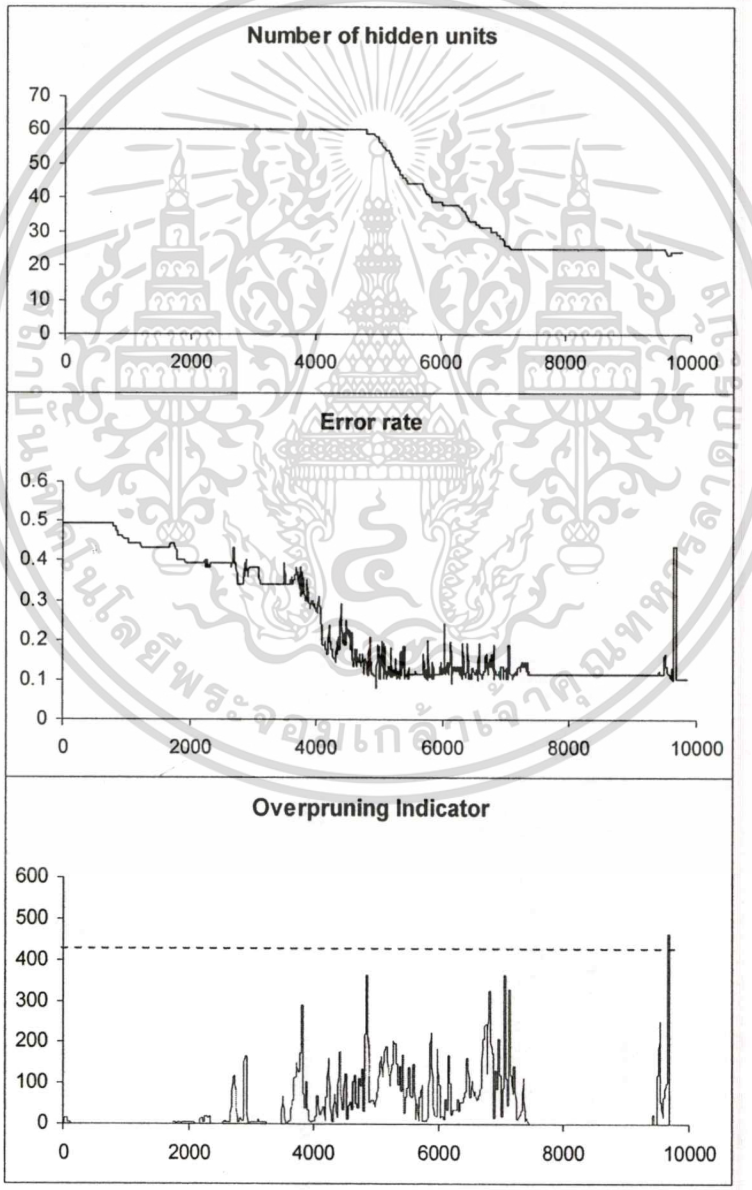
Problem	*Spartan*		OBD		OBS		Skeletonization	
	Avg. #HD	Avg. err rate	Avg. #HD	Avg. err rate	Avg. #HD	Avg. err rate	Avg. #HD	Avg. err rate
Two Spirals	24.59	0.1031	27.85	0.1007	26.24	0.1025	27.32	0.1011



รูปที่ 5.15 กราฟเปรียบเทียบจำนวนหน่วยซ่อนสำหรับปัญหา Two Spirals



รูปที่ 5.16 กราฟเปรียบเทียบอัตราผิดพลาดสำหรับปัญหา Two Spirals



รูปที่ 5.17 กราฟแสดงการทำงานของอัลกอริธึมสปาร์ตันซิมพลิซิตีสำหรับปัญหา Two Spirals

จากผลการทดลองกับปัญหา Two Spirals จะเห็นได้ว่าอัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถลดจำนวนของหน่วยซ่อนจากเดิม 60 หน่วย เหลือเพียง 24 หน่วย โดยยังคงให้อัตราความผิดพลาดในการทำนายข้อมูลใกล้เคียงกับของเดิม นอกจากนี้ จากการเปรียบเทียบผลการทดลองกับอัลกอริธึมอื่นๆ อัลกอริธึมสปาร์ตันซิมพลิซิตียังสามารถกำจัดหน่วยซ่อนได้มากกว่าอัลกอริธึมอื่นๆ และยังให้ค่าอัตราผิดพลาดที่ใกล้เคียงกับวิธีอื่นอีกด้วย ดังนั้น ผลการทดลองนี้จึงสามารถใช้ยืนยันประสิทธิภาพของอัลกอริธึมสปาร์ตันซิมพลิซิตีในการประยุกต์ใช้กับปัญหาที่มีลักษณะไม่เชิงเส้นได้เป็นอย่างดี

5.4 วิเคราะห์ผลเปรียบเทียบกับการหาจำนวนหน่วยซ่อนโดยวิธี Brute force

หัวข้อนี้เป็นการวิเคราะห์จำนวนหน่วยซ่อนของปัญหา Two Spirals เมื่อใช้วิธีการ Brute force ในการหาจำนวนหน่วยซ่อนที่ดีที่สุด โดยขั้นตอนในการทดลองด้วยวิธี Brute force มีดังนี้

ก) กำหนดให้หน่วยซ่อนมีจำนวนที่พอเหมาะที่จะทำให้โครงข่ายประสาทเทียมมีประสิทธิภาพที่ดี (ในที่นี้ เราจะใช้จำนวนหน่วยซ่อนเท่ากับ 60 เพื่อให้สอดคล้องกับการทดลองในหัวข้อย่อยก่อนหน้านี้)

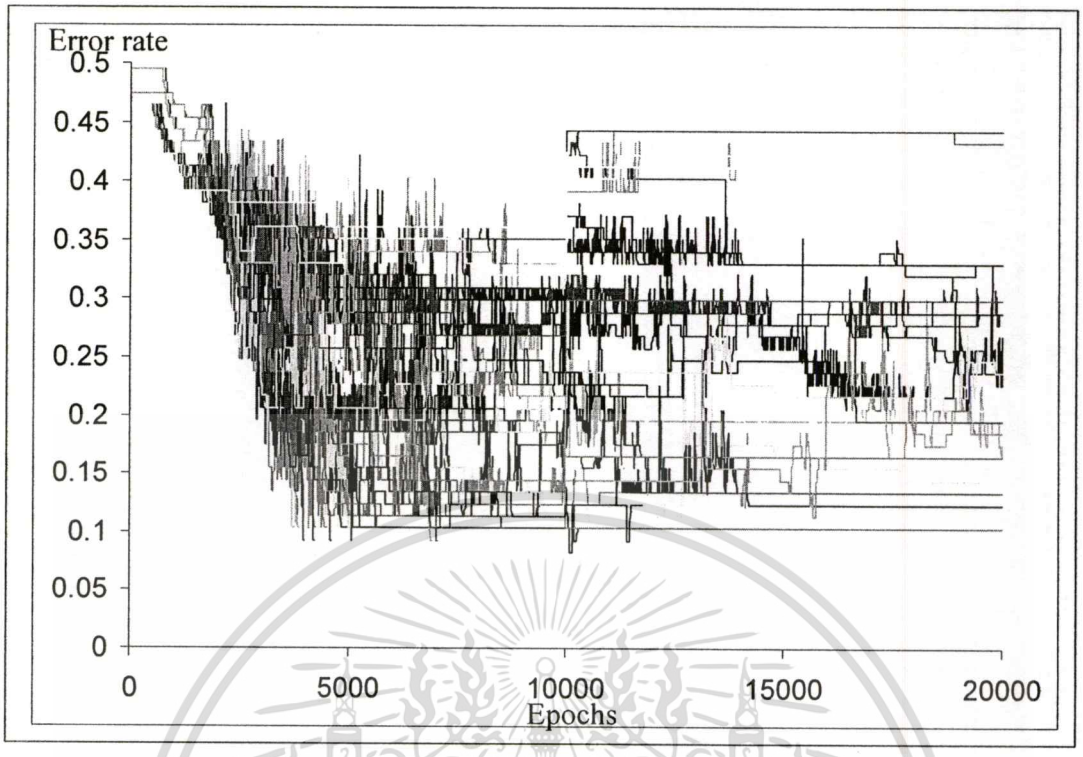
ข) สุ่มค่าน้ำหนักของโครงข่ายใหม่ทั้งหมด

ค) ฝึกโครงข่ายประสาทเทียมด้วยอัลกอริธึมแบ็คพรอพาเกชั่นมาตรฐาน จนกระทั่งโครงข่ายได้ค่าอัตราความผิดพลาดที่ยอมรับได้ (ในที่นี้ กำหนดให้ค่าอัตราผิดพลาดมีค่าเท่ากับ 0.1031 เพื่อให้เท่ากับอัตราค่าผิดพลาดเฉลี่ยของอัลกอริธึมสปาร์ตันซิมพลิซิตี)

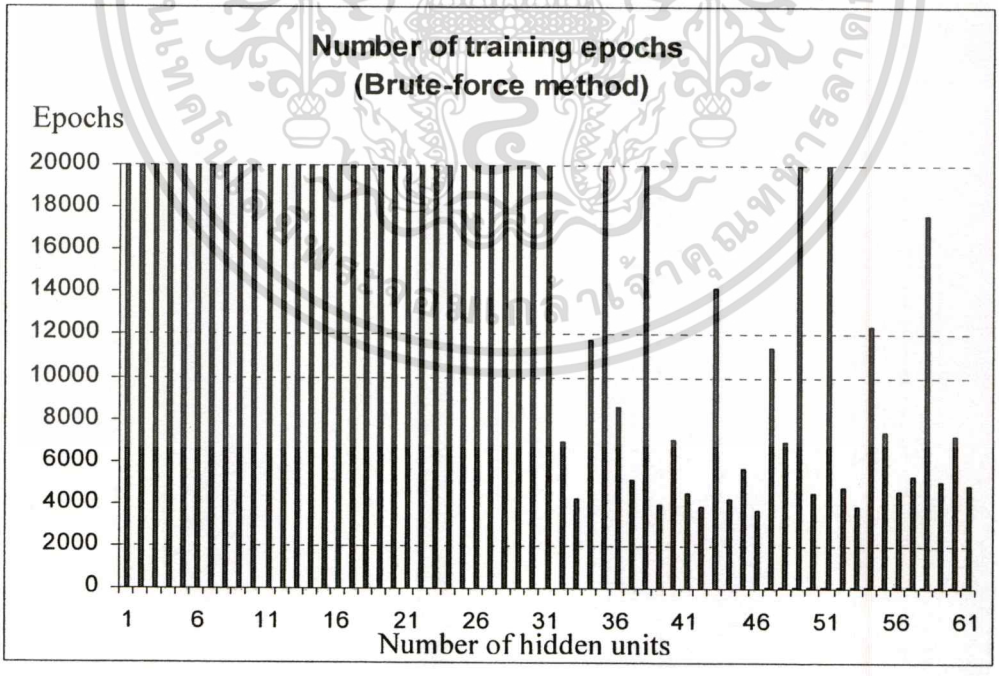
ง) หากโครงข่ายประสาทเทียมไม่สามารถฝึกจนเข้าสู่คอนเวอร์เจ้นซ์ได้ภายใน 20,000 รอบได้ เราจะถือว่าโครงข่ายประสาทเทียมติดอยู่ในปัญหาโลคอลมินิม่า (Local minima) และยกเลิกการฝึกโครงข่ายนั้นไป

จ) ลดจำนวนหน่วยซ่อนลงหนึ่งหน่วยและย้อนกลับไปยังขั้นตอน ข) จนกระทั่งจำนวนหน่วยซ่อนมีค่าเป็น 1

รูปที่ 5.18 แสดงกราฟความสัมพันธ์ระหว่างอัตราผิดพลาด (แนวตั้ง) และจำนวนรอบในการฝึกโครงข่าย (แนวนอน) เราจะสังเกตได้ว่า โครงข่ายประสาทเทียมที่มีจำนวนหน่วยซ่อนที่แตกต่างกันมีความสามารถในการเข้าสู่คอนเวอร์เจ้นซ์ได้ไม่เหมือนกัน รูปที่ 5.19 แสดงรายละเอียดของความสามารถในการเข้าสู่คอนเวอร์เจ้นซ์ของโครงข่ายทั้ง 60 โครงข่าย



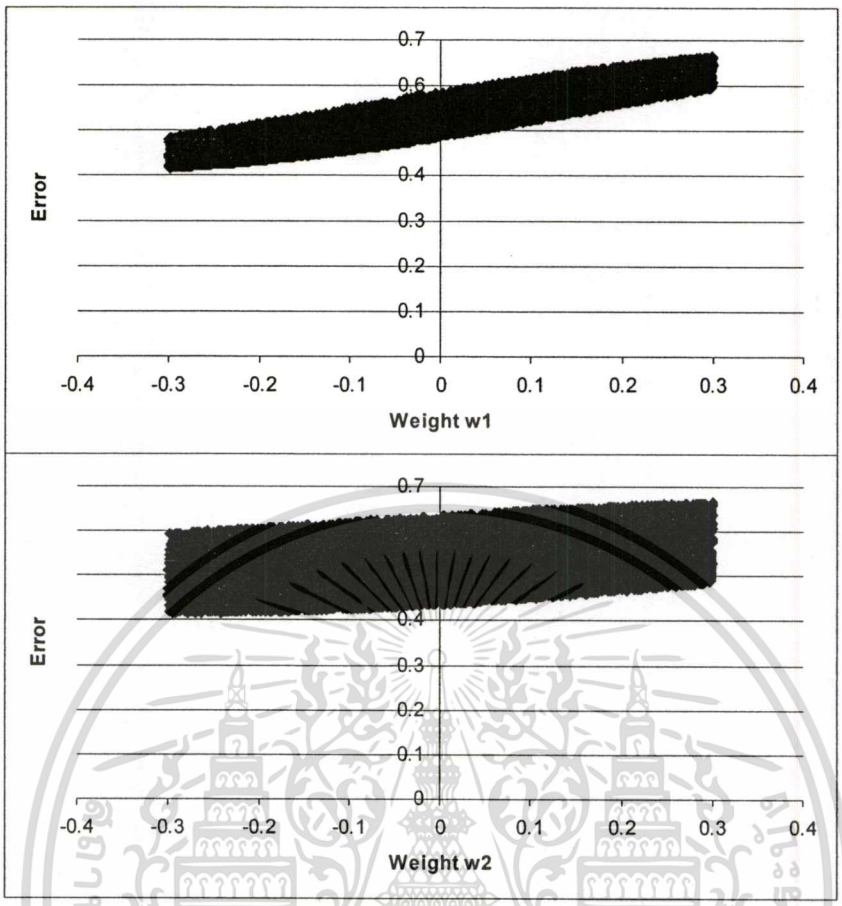
รูปที่ 5.18 ความสามารถในการเข้าสู่คอนเวอร์เจ้นซ์สำหรับปัญหา Two Spirals โดยวิธี Brute Force



รูปที่ 5.19 จำนวนรอบที่ใช้ในการเข้าสู่คอนเวอร์เจ้นซ์ในแต่ละโครงข่ายที่มีจำนวนหน่วยซ่อนต่างกัน

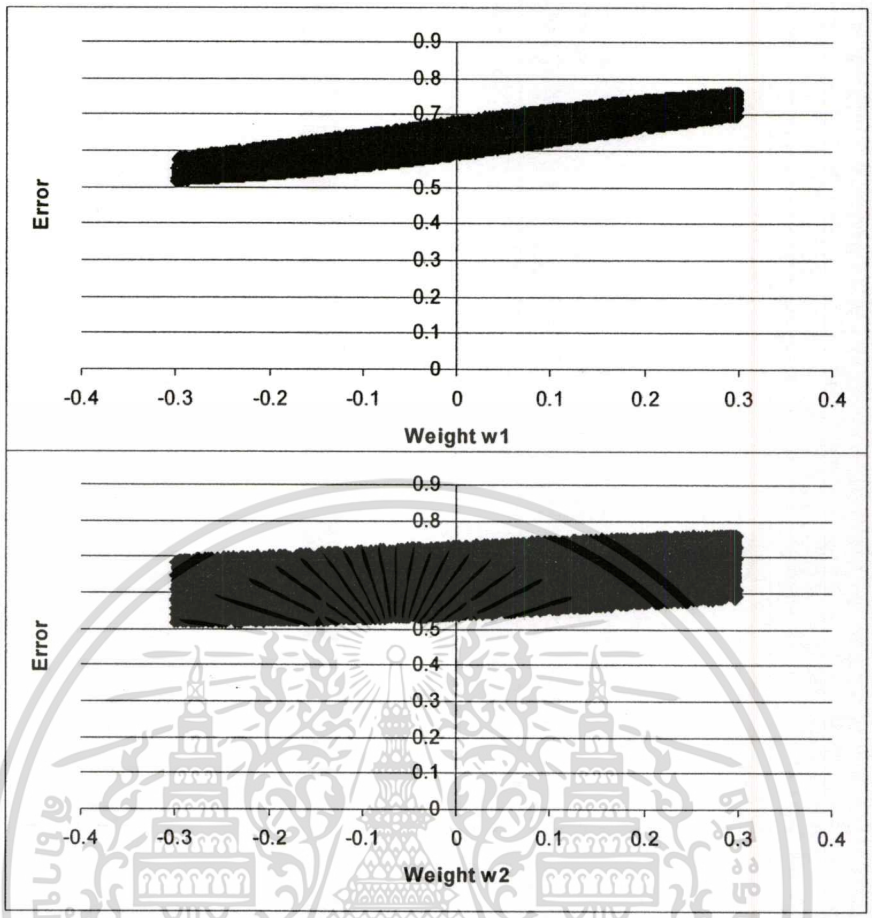
จากรูปที่ 5.19 เราจะเห็นได้ว่าเมื่อเราทดลองลดจำนวนหน่วยซ่อนทีละหน่วยแล้วฝึกโครงข่ายให้เข้าสู่คอนเวอร์เจ้นซ์ จะมีบางโครงข่ายเท่านั้นที่สามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ภายในเวลา 20,000 รอบ นอกจากนี้ จำนวนหน่วยซ่อนที่ทำให้โครงข่ายประสาทเทียมสามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ยังไม่มีคุณสมบัติเชิงเส้น เช่น จำนวนหน่วยซ่อน 51 หน่วยไม่สามารถทำให้โครงข่ายเข้าสู่คอนเวอร์เจ้นซ์ แต่จำนวนหน่วยซ่อน 52 และ 50 กลับสามารถทำให้โครงข่ายเข้าสู่คอนเวอร์เจ้นซ์ได้ เป็นต้น นอกจากนี้ จำนวนหน่วยซ่อนที่น้อยที่สุดที่สามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ด้วยวิธี Brute force คือ 32 เมื่อเปรียบเทียบผลการทดลองกับอัลกอริธึมสปาร์ตันซิมพลิซิตีในหัวข้อย่อยที่แล้ว จำนวนของหน่วยซ่อนสุดท้ายที่เราได้คือ 24 ซึ่งถือว่ามีความแตกต่างกันอย่างชัดเจน อนึ่ง การลดลงของหน่วยซ่อนของอัลกอริธึมสปาร์ตันซิมพลิซิตีจาก 60 จนถึง 24 ยังเป็นการแสดงให้เห็นว่าอัลกอริธึมนี้สามารถหลีกเลี่ยงปัญหาจำนวนหน่วยซ่อนที่ไม่เป็นเชิงเส้นดังเช่นวิธี Brute force ได้ประสบมาแล้ว

ถึงแม้ว่าการมีจำนวนหน่วยซ่อนที่ลดลงจะช่วยให้โครงข่ายประสาทเทียมใช้เวลาในการทำงานที่น้อยลง อย่างไรก็ตาม หากการกำหนดค่าเริ่มต้นให้กับน้ำหนักต่างๆภายในโครงข่ายประสาทเทียมถูกกระทำอย่างไม่เหมาะสม โครงข่ายนั้นอาจไม่สามารถค้นพบค่าผิดพลาดที่ต่ำที่สุดได้ เพื่อเป็นการสนับสนุนแนวคิดนี้ เราได้ทำการทดลองกับปัญหา Two spirals โดยเริ่มต้นจากโครงข่ายประสาทเทียมที่มีจำนวนหน่วยซ่อนเท่ากับ 50 หน่วย และกำหนดค่าเริ่มต้นแบบสุ่มที่มีช่วงอยู่ระหว่าง -0.3 ถึง +0.3 ให้กับน้ำหนักทั้งหมดภายในโครงข่าย แล้วนำชุดข้อมูลของปัญหา Two spirals ส่งผ่านโครงข่ายประสาทเทียมนี้และวัดค่าผิดพลาด หลังจากนั้น เราเลือกน้ำหนัก W1 และ W2 ที่อยู่ภายในโครงข่ายเพื่อสุ่มค่าน้ำหนักใหม่ ส่วนน้ำหนักอื่นๆให้ใช้ค่าน้ำหนักชุดเดิม สำหรับทุกๆค่าที่สุ่มขึ้นมาใหม่ของ (W1, W2) ให้วัดค่าผิดพลาดที่ได้จากการผ่านชุดข้อมูลของปัญหา Two spirals เข้าในโครงข่ายเพื่อนำมาสร้างกราฟดังรูปที่ 5.20 เราจะสังเกตเห็นได้ว่าช่วงของค่าผิดพลาด (Error scope) ที่เกิดขึ้นจากการแปรผันค่าน้ำหนัก W1 และ W2 (โดยที่น้ำหนักอื่นๆยังมีค่าคงที่) มีค่าอยู่ในช่วง 0.4 – 0.7



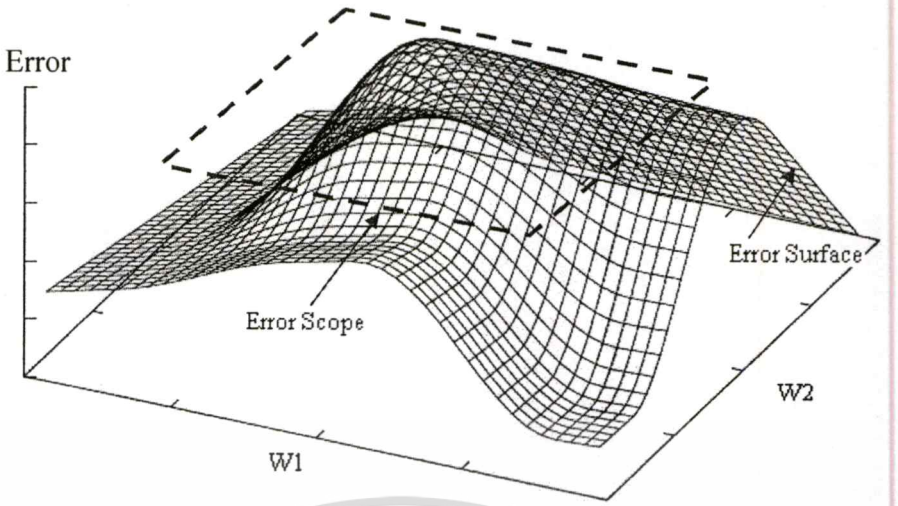
รูปที่ 5.20 ค่าผิดพลาดที่ได้จากการแปรผันค่าน้ำหนัก W1 และ W2 โดยค่าน้ำหนักอื่นๆยังมีค่าคงที่

เมื่อเรากำจัดหน่วยซ่อนภายในโครงข่ายประสาทเทียมให้เหลือหน่วยซ่อนเพียงแค่ 40 หน่วย (เราไม่กำจัดหน่วยซ่อนที่เชื่อมอยู่กับน้ำหนัก W1 และ W2) และกำหนดค่าเริ่มต้นให้กับน้ำหนักที่เหลืออยู่ใหม่ทั้งหมด แล้วนำน้ำหนัก W1 และ W2 มากำหนดค่าแบบสุ่มเช่นดังที่ได้ทำมาแล้วก่อนหน้านี้ กราฟที่ได้มีลักษณะดังรูปที่ 5.21 เราจะเห็นได้ว่าช่วงของค่าผิดพลาด (Error scope) ที่เกิดขึ้นจากการแปรผันค่าน้ำหนัก W1 และ W2 (หลังจากหน่วยซ่อนถูกกำจัดเหลือเพียง 40 หน่วย และน้ำหนักอื่นๆถูกคงค่าไว้ในขณะที่น้ำหนัก W1 และ W2 ถูกแปรผันแบบสุ่ม) มีค่าเพิ่มขึ้นอยู่ในช่วง 0.5 – 0.8



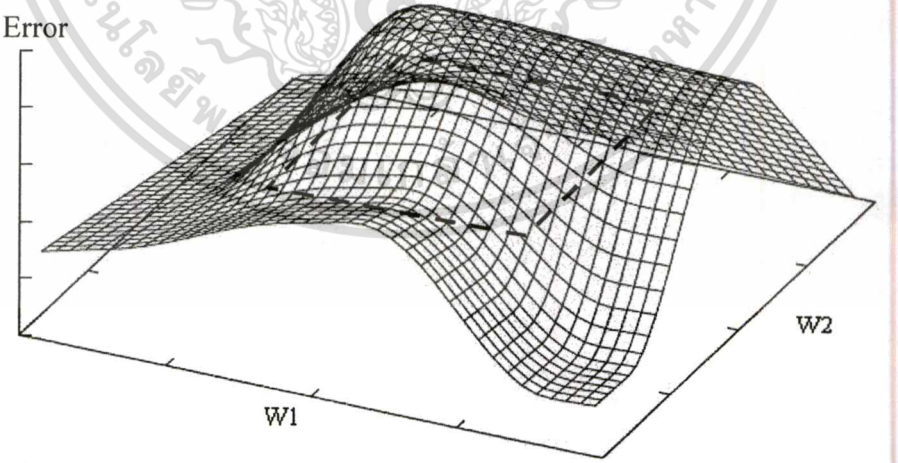
รูปที่ 5.21 ค่าผิดพลาดที่ได้จากการแปรผันค่าน้ำหนัก W1 และ W2 หลังจากเหลือหน่วยซ่อนเพียง 40 หน่วย

ช่วงของค่าผิดพลาด (Error scope) ที่มีค่าเพิ่มขึ้นบ่งบอกให้เราทราบว่าพื้นผิวของค่าผิดพลาด (Error surface) ของโครงข่ายมีค่าเพิ่มขึ้นเมื่อหน่วยซ่อนมีจำนวนลดลง ดังนั้นค่าผิดพลาดที่ได้จากการฝึกโครงข่ายประสาทเทียมที่ถูกกำจัดหน่วยซ่อนและสุ่มค่าเริ่มต้นของน้ำหนักใหม่ทั้งหมดจะไม่สามารถเข้าสู่ค่าผิดพลาดที่ต่ำที่สุดได้เลย (ค่าผิดพลาดต่ำสุดที่ได้จากรูปที่ 5.13 คือ 0.4 แต่ค่าผิดพลาดต่ำสุดที่ได้จากการทดลองในรูปที่ 5.14 คือ 0.5) เพื่ออธิบายปรากฏการณ์ที่เกิดขึ้นนี้ รูปที่ 5.22 แสดงกราฟสามมิติจำลองของพื้นผิวความผิดพลาดก่อนกำจัดหน่วยซ่อน ช่วงของค่าผิดพลาด (Error scope) ของโครงข่ายประสาทเทียมที่มีจำนวนหน่วยซ่อนมากจะสามารถครอบคลุมพื้นผิวค่าผิดพลาด (Error surface) ได้เป็นบริเวณกว้าง ทำให้โอกาสที่จะพบคำตอบที่ดีมีมากขึ้นด้วย

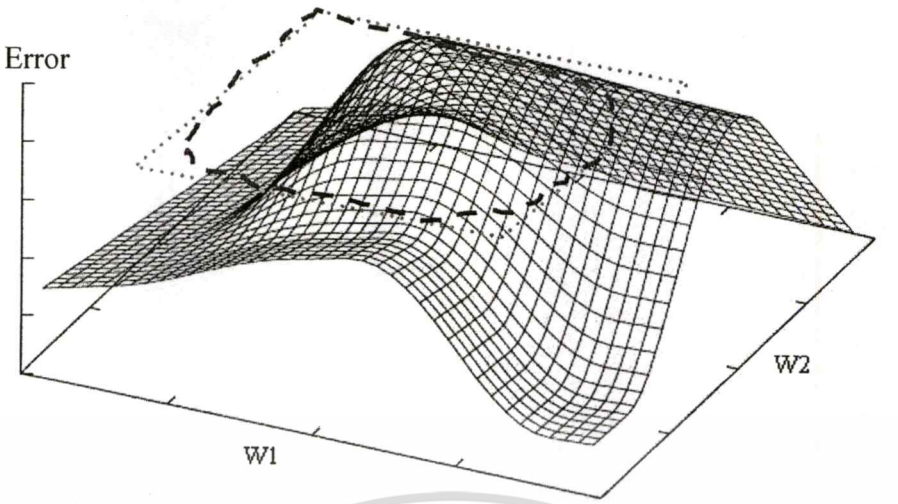


รูปที่ 5.22 ขอบเขตของค่าผิดพลาดที่คลอบคลุมพื้นผิวค่าผิดพลาดก่อนกำจัดหน่วยซ่อน

ในขณะที่รูปที่ 5.23 แสดงกราฟสามมิติจำลองของพื้นผิวความผิดพลาดหลังกำจัดหน่วยซ่อน ช่วงของค่าผิดพลาด (Error scope) ของโครงข่ายประสาทเทียมที่มีจำนวนหน่วยซ่อนที่น้อยลงจะครอบคลุมพื้นผิวค่าผิดพลาด (Error surface) ได้เป็นบริเวณแคบลง ทำให้โอกาสที่จะพบคำตอบที่ดีมีน้อยลงด้วย ในทางตรงข้าม หากเราค่อยๆกำจัดหน่วยซ่อนทีละหน่วยตามลำดับของหน่วยซ่อนที่มีความสำคัญน้อยที่สุดโดยไม่ได้สุ่มค่าเริ่มต้นใหม่ให้กับน้ำหนักภายในโครงข่ายประสาทเทียม ช่วงของค่าผิดพลาดจะค่อยๆแคบลงโดยไม่ทำให้คำตอบที่ดีหายไปจากช่วงของค่าผิดพลาดนี้ ดังแสดงในรูปที่ 5.24



รูปที่ 5.23 ขอบเขตของค่าผิดพลาดที่คลอบคลุมพื้นผิวค่าผิดพลาดหลังกำจัดหน่วยซ่อน

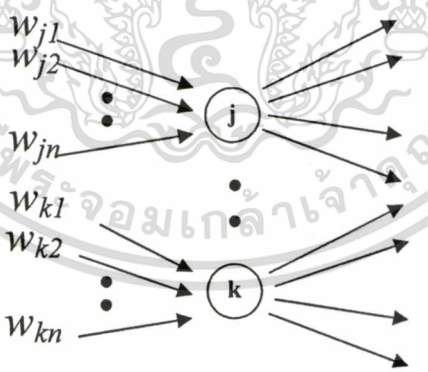


รูปที่ 5.24 ขอบเขตของค่าผิดพลาดที่คลอบคลุมพื้นผิวค่าผิดพลาดหลังกำจัดหน่วยซ่อนด้วยวิธีค่อยๆกำจัดหน่วยซ่อนทีละหน่วยตามลำดับความไม่สำคัญ

5.5 วิเคราะห์ผลเปรียบเทียบกับการกำจัดหน่วยซ่อนที่ซ้ำซ้อนกัน

เพื่อเป็นการวิเคราะห์ผลการทดลองเปรียบเทียบกับการกำจัดหน่วยซ่อนที่ซ้ำซ้อนกันโดยพิจารณาจากหน่วยซ่อนคู่ที่มีค่าน้ำหนักเข้า (Incoming Weight) ที่คล้ายกันที่สุด ซึ่งคำนวณได้ดังนี้

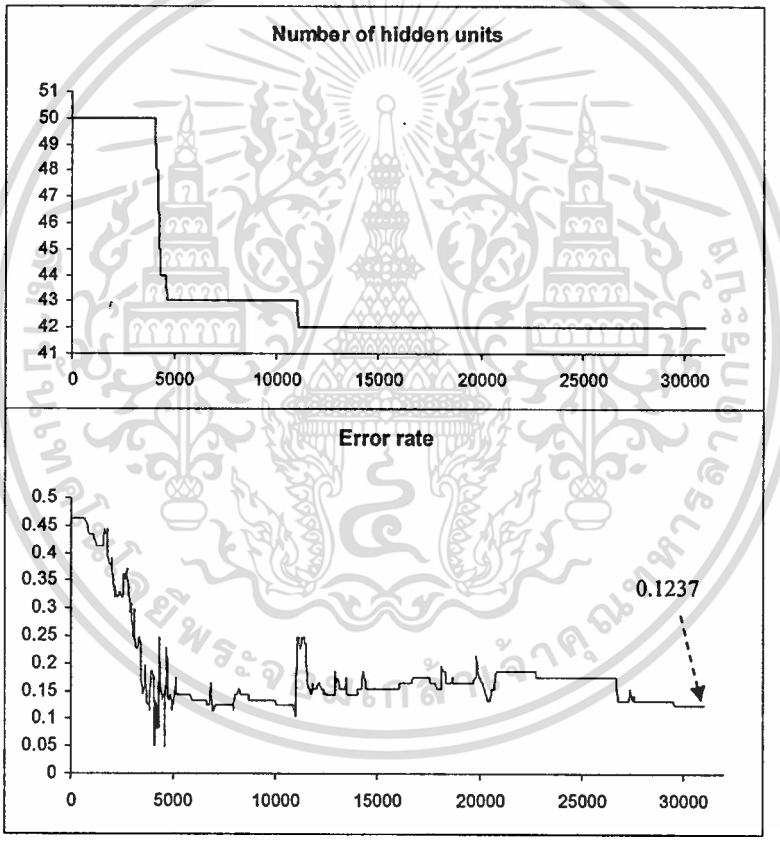
รูปที่ 5.25 แสดงการวิเคราะห์หน่วยซ่อนสองหน่วย j และ k ว่ามีความซ้ำซ้อนกันหรือไม่ เราคำนวณค่าความซ้ำซ้อนของหน่วยซ่อนจากน้ำหนักที่เข้ามายังหน่วยซ่อนดังสมการต่อไปนี้



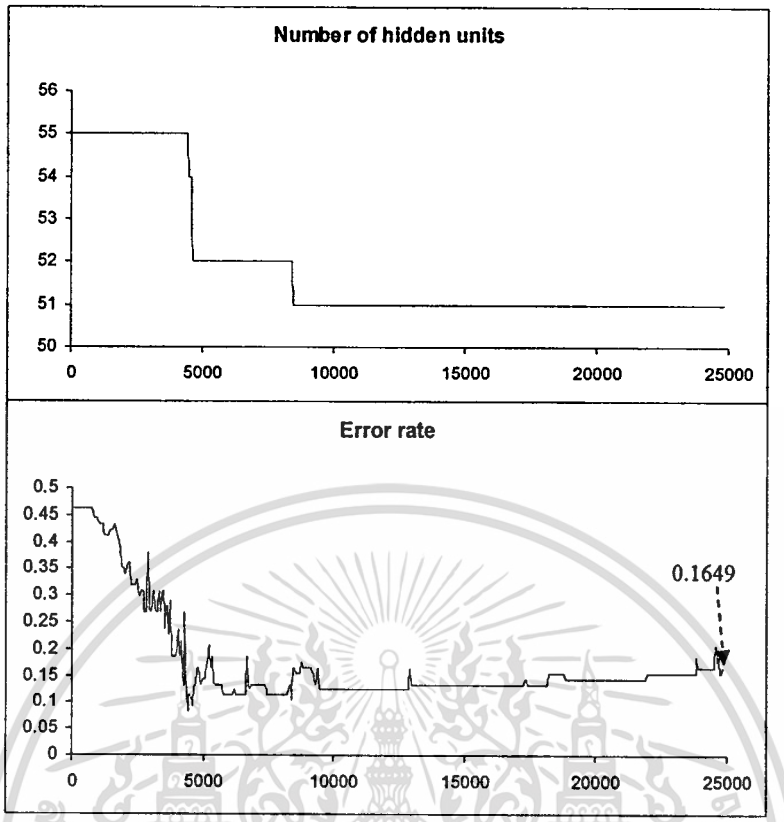
รูปที่ 5.25 การวิเคราะห์หน่วยซ่อนที่ซ้ำซ้อนกัน

$$\text{Hidden unit redundancy (j , k)} = \sum_{i=1}^{Input} w_{ji} \cdot w_{ki} \qquad (5.1)$$

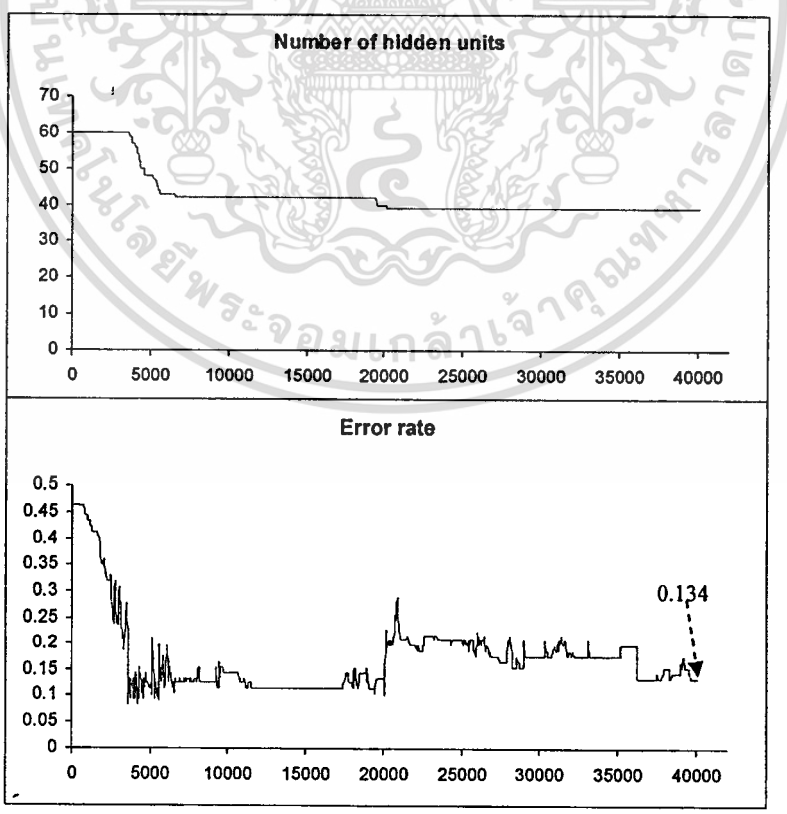
เราทำการจับคู่หน่วยซ่อนเพื่อวัดค่าความซ้ำซ้อน โดยคู่ของหน่วยซ่อนใดที่ให้ค่าความซ้ำซ้อนสูงที่สุดแสดงว่ามีความซ้ำซ้อนกันและหนึ่งในนั้นควรจะถูกกำจัดออกซึ่งเราจะกำจัดหน่วยซ่อนที่มีค่าผลรวมของน้ำหนัก (ทั้งเข้าและออก) ที่น้อยที่สุด เราเริ่มต้นจากการฝึกโครงข่ายประสาทเทียมจนกระทั่งค่าอัตราผิดพลาดมีค่าน้อยกว่า 0.11 (เป็นค่าที่ใกล้เคียงกับการทดลองในหัวข้อย่อยก่อนหน้านี้และถือว่าเป็นค่าที่ยอมรับได้) จากนั้นทำการหาหน่วยซ่อนที่จะถูกกำจัดโดยใช้สมการที่ (5.1) หากโครงข่ายใช้เวลาในการทำงานนานจนเกินไป (เกิน 20,000 รอบนับจากจุดที่กำจัดหน่วยซ่อนล่าสุด) เราจะหยุดการทำงานทันที รูปที่ 5.26 – 5.29 แสดงการกำจัดหน่วยซ่อนของโครงข่ายประสาทเทียมโดยวิธีวิเคราะห์ความซ้ำซ้อนกับโครงข่ายที่มีหน่วยซ่อนเริ่มต้น 50, 55, 60 และ 70 ตามลำดับ



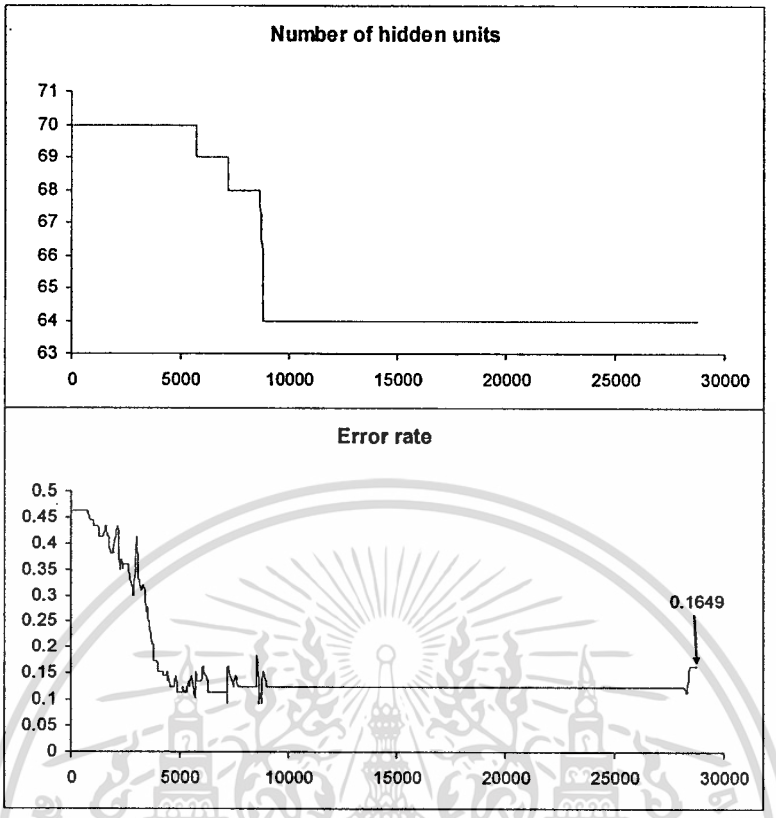
รูปที่ 5.26 ผลการทดลองสำหรับปัญหา Two Spirals โดยมีจำนวนหน่วยซ่อนเริ่มต้น 50 หน่วย



รูปที่ 5.27 ผลการทดลองสำหรับปัญหา Two Spirals โดยมีจำนวนหน่วยซ่อนเริ่มต้น 55 หน่วย



รูปที่ 5.28 ผลการทดลองสำหรับปัญหา Two Spirals โดยมีจำนวนหน่วยซ่อนเริ่มต้น 60 หน่วย



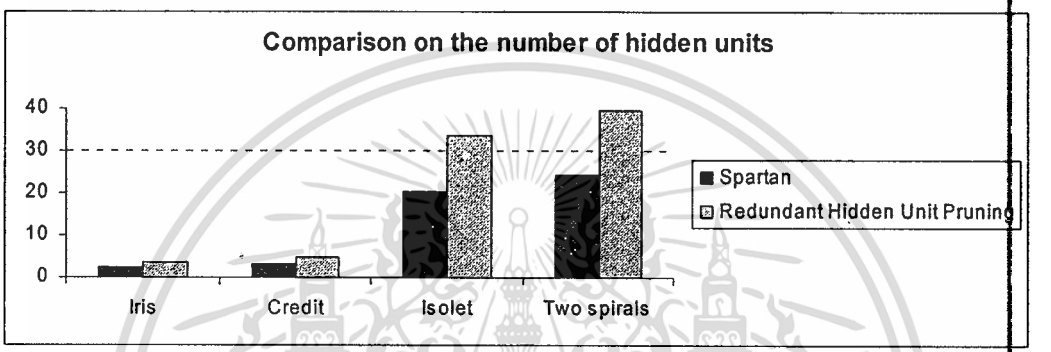
รูปที่ 5.29 ผลการทดลองสำหรับปัญหา Two Spirals โดยมีจำนวนหน่วยซ่อนเริ่มต้น 70 หน่วย

ปัญหาหนึ่งที่มีความเหมือนกันจากผลการทดลองในรูปที่ 5.26 – 5.29 คือ โครงข่ายทุกโครงข่ายประสบกับปัญหา Local minima ซึ่งค่าอัตราผิดพลาดไม่มีการเปลี่ยนแปลงไม่ว่าโครงข่ายจะถูกฝึกนานเพียงใด นอกจากนี้ จำนวนหน่วยซ่อนที่อยู่ภายในโครงข่ายยังสามารถถูกกำจัดได้เพิ่มอีก แต่เนื่องจากค่าอัตราผิดพลาดที่ยังสูงอยู่ จึงทำให้เราไม่สามารถลดจำนวนหน่วยซ่อนได้มากกว่านี้ ดังนั้น ปัญหาที่เกิดขึ้นนี้จึงเป็นสิ่งหนึ่งที่สามารถยืนยันได้ว่าการกำจัดหน่วยซ่อนด้วยวิธีการวิเคราะห์ความซ้ำซ้อนของหน่วยซ่อนไม่เหมาะสมกับปัญหาที่มีความไม่เป็นเชิงเส้นสูง ดังเช่นปัญหา Two Spirals

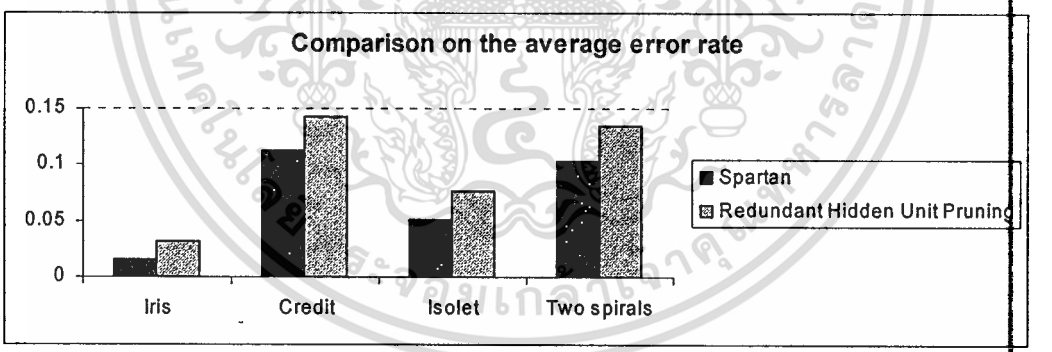
อย่างไรก็ตาม สำหรับปัญหาที่มีระดับความซับซ้อนไม่มากนัก เช่น ปัญหา Iris และ Credit card วิธีการกำจัดหน่วยซ่อนด้วยวิธีการวิเคราะห์ความซ้ำซ้อนของหน่วยซ่อนสามารถทำงานได้ดีพอสมควร แต่เมื่อทดสอบกับปัญหาที่ซับซ้อนขึ้นเช่น Isolet ผลการทดลองที่ได้นั้นไม่ดีนัก เนื่องจากให้ค่าอัตราผิดพลาดที่สูงและจำนวนหน่วยซ่อนที่มาก ดังผลการทดลองที่แสดงในตาราง 5.9 (สำหรับกราฟถูกแสดงในรูปที่ 5.30 และ 5.31) ซึ่งเป็นค่าเฉลี่ยจากการทดลองทั้งสิ้น 30 ครั้ง โดยใช้ค่าเริ่มต้นเดียวกันกับที่ทดลองด้วยอัลกอริธึมสปาร์ตันซิมพลิซิตี

ตารางที่ 5.9 การเปรียบเทียบผลการทดลองการกำจัดหน่วยซ่อนกับปัญหา Iris, ปัญหา Credit card, ปัญหา Isolet และปัญหา Two Spirals

Problem	*Spartan*		Hidden Unit Redundancy Method	
	Avg. #HD	Avg. err rate	Avg. #HD	Avg. err rate
Iris	2.25	0.015	3.54	0.0310
Credit	3.25	0.112	4.69	0.1429
Isolet	20.28	0.051	33.58	0.0759
Two spirals	24.59	0.1031	39.75	0.1344



รูปที่ 5.30 กราฟเปรียบเทียบจำนวนหน่วยซ่อนเมื่อเทียบกับวิธี Redundant Hidden Unit Pruning

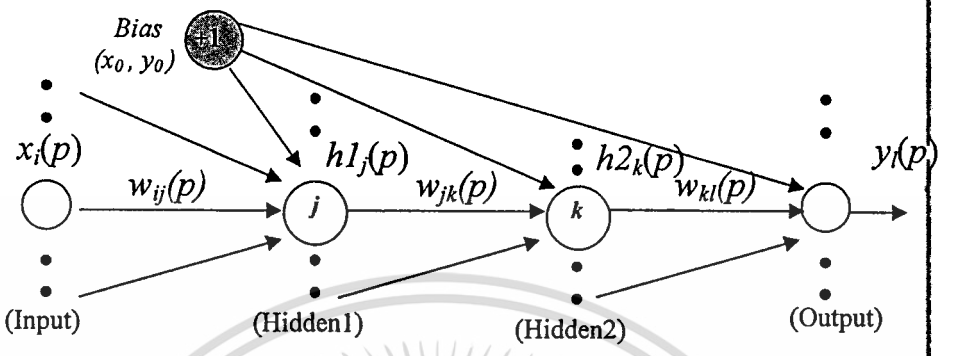


รูปที่ 5.31 กราฟเปรียบเทียบอัตราผิดพลาดเมื่อเทียบกับวิธี Redundant Hidden Unit Pruning

5.6 การขยายผลอัลกอริธึมสปาร์ตันซิมพลิซิตีในการกำจัดหน่วยซ่อนในโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้น

เพื่อเป็นการขยายผลอัลกอริธึมสปาร์ตันซิมพลิซิตีให้มีความสามารถเพิ่มขึ้น แนวทางในการกำจัดหน่วยซ่อนของโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้นจึงถูกนำเสนอในหัวข้อย่อยนี้ ในที่นี้ เรานำเสนอเพียงการกำจัดหน่วยซ่อนของโครงข่ายประสาทเทียมที่มีหน่วยซ่อนเพียงสอง

ชั้นเท่านั้น เนื่องจากสามารถเป็นตัวแทนของโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้นได้ อีกทั้งยังง่ายต่อการเข้าใจ รูปที่ 5.32 แสดงโครงข่ายที่มีหน่วยซ่อนสองชั้นซึ่งจะถูกใช้ในการขยายผลอัลกอริธึม



รูปที่ 5.32 สัญลักษณ์ภายในโครงข่ายที่มีหน่วยซ่อนสองชั้น

การที่จะทำให้อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถทำงานได้กับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้นได้ เราจะต้องดัดแปลงวิธีการคำนวณค่าผลต่างของแอ็คติเวชั่น (AD_j) เพื่อหลีกเลี่ยงการคำนวณซ้ำซ้อนซึ่งอาจเกิดขึ้นได้

กำหนดให้ $X_j(p)$ คืออินพุทรวมของหน่วยซ่อน j สำหรับแพทเทิร์น (Pattern) p, $h1_j(p)$ คือเอาท์พุทของหน่วยซ่อน j ในชั้นที่ 1, $h2_k(p)$ คือเอาท์พุทของหน่วยซ่อน k ในชั้นที่ 2, $X_k(p)$ คืออินพุทรวมของหน่วยซ่อนในชั้นที่สองสำหรับแพทเทิร์น p, $X_l(p)$ คืออินพุทรวมของหน่วยเอาท์พุทสำหรับแพทเทิร์น p, $y_l(p)$ คือค่าแอ็คติเวชั่นของหน่วย l ในชั้นเอาท์พุท และ φ คือ แอ็คติเวชั่นฟังก์ชั่น ดังสมการดังต่อไปนี้

$$X_j(p) = \sum_i w_{ij}(p) \cdot x_i(p) + \theta_j(p) \tag{5.2}$$

$$h1_j(p) = \varphi \ (X_j(p)) \tag{5.3}$$

$$X_k(p) = \sum_j w_{jk}(p) \cdot h1_j(p) + \theta_k(p) \tag{5.4}$$

$$h2_k(p) = \varphi \ (X_k(p)) \tag{5.5}$$

$$X_l(p) = \sum_l w_{kl}(p) \cdot h2_k(p) + \theta_l(p) \tag{5.6}$$

$$y_l(p) = \varphi \ (X_l(p)) \tag{5.7}$$

หากเรากำจัดหน่วยซ่อน m ภายในชั้น Hidden1 ทั้ง $X_k(p)$ และ $X_l(p)$ ที่เกี่ยวข้องกับหน่วยซ่อนนั้นจะต้องถูกคำนวณใหม่ แต่ถ้าเรากำจัดหน่วยซ่อน m ภายในชั้น Hidden2 เฉพาะ $X_l(p)$ ที่

เกี่ยวข้องกับหน่วยซ่อนนั้นเท่านั้นที่จะถูกคำนวณใหม่ เราเรียก $X_k(p)$ และ $X_l(p)$ หลังจากขั้นตอนการกำจัดหน่วยซ่อน m นี้ว่า $X_{k-m}(p)$ และ $X_{l-m}(p)$ ตามลำดับ นอกจากนี้ เรายังเรียกค่าแอ็คติเวชั่นของหน่วยเอาท์พุท l เมื่อหน่วยซ่อน m ถูกกำจัดออกว่า $y_{l-m}(p)$

//กรณีหน่วยซ่อนในชั้น Hidden1 ถูกกำจัด

$$X_{k-m}(p) = X_k(p) - w_{mk}(p) \cdot h1_m(p) \tag{5.8}$$

$$h2_{k-m}(p) = \varphi \ (X_{k-m}(p)) \tag{5.9}$$

$$X_{l-m}(p) = \sum_l w_{kl}(p) \cdot h2_{k-m}(p) + \theta_l(p) \tag{5.10}$$

$$y_{l-m}(p) = \varphi \ (X_{l-m}(p)) \tag{5.11}$$

//กรณีหน่วยซ่อนในชั้น Hidden2 ถูกกำจัด

$$X_{l-m}(p) = X_l(p) - w_{ml}(p) \cdot h2_m(p) \tag{5.12}$$

$$y_{l-m}(p) = \varphi \ (X_{l-m}(p)) \tag{5.13}$$

โดยการเปรียบเทียบกันระหว่างค่าสมบูรณ์ (Absolute) ของผลต่างระหว่างค่าที่ต้องการ (Desirable value) ของหน่วยเอาท์พุท l หรือ $O_l(p)$ กับค่าแอ็คติเวชั่นของหน่วยเอาท์พุท l เมื่อหน่วยซ่อน m ถูกกำจัด หรือ $y_{l-m}(p)$ เราสามารถทราบระดับของความไม่เกี่ยวข้องของหน่วยซ่อน m นิยามของค่าผลต่างของแอ็คติเวชั่นสามารถอธิบายได้ดังสมการดังต่อไปนี้

$$AD_{l-m}(p) = \left| O_l(p) - y_{l-m}(p) \right| \tag{5.14}$$

ในกรณีที่โครงข่ายมีหน่วยเอาท์พุทมากกว่าหนึ่งหน่วย เราจะต้องคำนวณหาผลรวมของค่าผลต่างของแอ็คติเวชั่น $AD_m(p)$ ของทุกๆหน่วยเอาท์พุทด้วย

$$AD_m(p) = \sum_l AD_{l-m}(p) \tag{5.15}$$

จากนั้นเราทำการหาผลรวมของค่าผลต่างของแอ็คติเวชั่นสำหรับชุดข้อมูลแวลิเดชั่นทั้งหมด ซึ่งเราจะเรียกว่า AD_m และรายงานค่าความไม่เกี่ยวข้องของหน่วยซ่อนต่ออัลกอริธึมเพื่อกำจัดหน่วยซ่อนนั้นออกไป โดยหน่วยซ่อนที่มีค่าผลต่างของแอ็คติเวชั่น (AD_m) ต่ำที่สุดจะถูกกำจัดออกไปก่อน

$$AD_m = \sum_p AD_m(p) \tag{5.16}$$

สิ่งหนึ่งที่เป็นปัญหาในการคำนวณค่าผลต่างของแอ็คติเวชั่นในโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้นคือ หากหน่วยซ่อนภายในชั้น Hidden1 ถูกกำจัด เราจะไม่สามารถประหยัดการคำนวณด้วยการใช้ค่า X_i (p) ที่คำนวณไว้แล้วก่อนหน้านี้มาคำนวณได้ ดังนั้นในสมการที่ (5.10) เราจึงต้องทำการคำนวณไปข้างหน้า (Forward computation) ใหม่อีกครั้ง ผลจากปัญหานี้ทำให้การคำนวณค่า AD_m ใช้เวลาค่อนข้างนาน

เพื่อเป็นการเปรียบเทียบความซับซ้อนของการคำนวณค่า AD_m ในโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้น การวิเคราะห์ความซับซ้อนของการคำนวณจึงถูกแสดงด้วยโมดูล ValidationError ที่ถูกดัดแปลงดังรูปที่ 5.33

Module ValidationError()
(Precondition: AD_j must be initialized to zero)
1 For every valid. pattern **p**
2 For every hidden unit **j** in layer **hidden1**
3 $X_j(p) = \sum_i^I w_{ij}(p) \cdot x_i(p) + \theta_j(p)$ /* PJI */
4 $h1_j(p) = \varphi(X_j(p))$
2 For every hidden unit **k** in layer **hidden2**
3 $X_k(p) = \sum_j^J w_{j_k}(p) \cdot h1_j(p) + \theta_k(p)$ /* PKJ */
4 $h2_k(p) = \varphi(X_k(p))$
5 For every output unit **l**
6 $X_l(p) = \sum_k^K w_{kl}(p) \cdot h2_k(p) + \theta_l(p)$ /* PLK */
7 $y_l(p) = \varphi(X_l(p))$

// Remove hidden unit in Hidden1 layer
8 For every hidden unit **m** in **layer hidden1**
9 $AD_{mH1}(p) = 0$
10 For every hidden unit **k** in **layer hidden2**
11 $X_{k-m}(p) = X_k(p) - w_{mk}(p) \cdot h1_m(p)$
12 $h2_{k-m}(p) = \varphi(X_{k-m}(p))$
13 For every output unit **l**
14 $X_{l-m}(p) = \sum_l w_{kl}(p) \cdot h2_{k-m}(p) + \theta_l(p)$
15 $y_{l-m}(p) = \varphi(X_{l-m}(p))$
16 $AD_{mH1}(p) = AD_m(p) + \left| O_l(p) - y_{l-m}(p) \right|$ /* PJKL */
17 $AD_{mH1} = AD_{mH1} + AD_{mH1}(p)$

// Remove hidden unit in Hidden2 layer
18 For every hidden unit **m** in **layer hidden2**
19 $AD_{mH2}(p) = 0$
20 For every output unit **l**
21 $X_{l-m}(p) = X_l(p) - w_{ml}(p) \cdot h2_m(p)$
22 $y_{l-m}(p) = \varphi(X_{l-m}(p))$
23 $AD_{mH2}(p) = AD_{mH2}(p) + \left| O_l(p) - y_{l-m}(p) \right|$ /* PKL */
24 $AD_{mH2} = AD_{mH2} + AD_{mH2}(p)$

/* Proceed to the recognition error rate calculation */
End.

รูปที่ 5.33 การดัดแปลงโมดูล ValidationError

$$O(\text{Original module}) = O(PJI + PKJ + PLK)$$

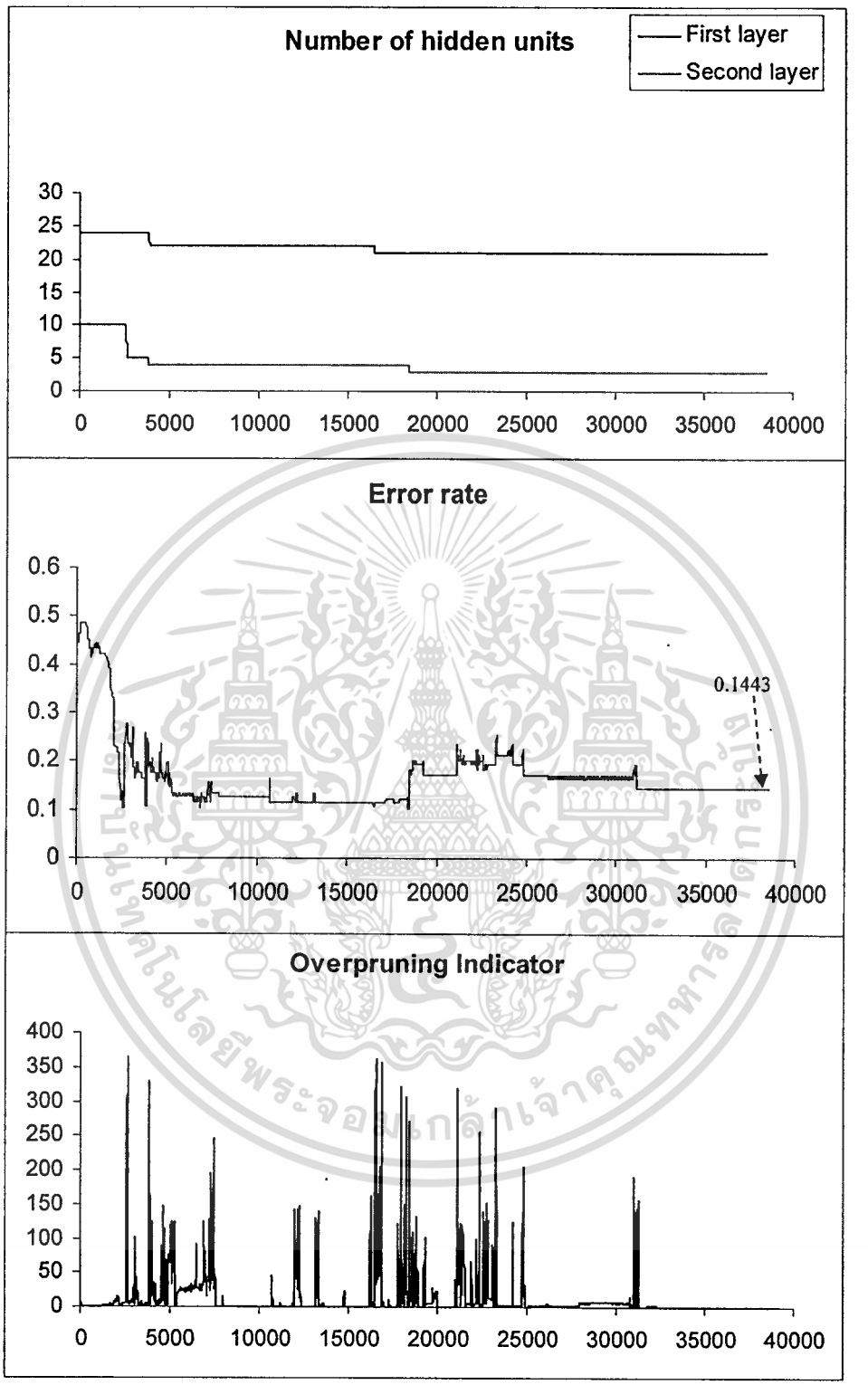
$$O(\text{Modified module}) = O(PJI + PKJ + PLK + PJKL + PKL)$$
$$= O(PJI + PKJ + 2PKL + PJKL) = O(PJKL)$$

จะเห็นได้ว่าความซับซ้อนของเวลาในการทำงานถูกเปลี่ยนแปลงเพิ่มขึ้นเป็น O(PJKL) ซึ่งเป็นข้อจำกัดอย่างหนึ่งของอัลกอริธึมสปาร์ตันซิมพลิซิตี ดังนั้นจึงไม่เหมาะกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้น อย่างไรก็ตาม หากปราศจากเทคนิคในการคำนวณในชั้นแรกของหน่วยซ่อน เราจะต้องใช้เวลาในการคำนวณค่าผลต่างของแอ็คติเวชั่นเท่ากับ O(PIJKL) ซึ่งมากกว่าเวลาในการคำนวณที่ได้จากเทคนิคที่นำเสนอ

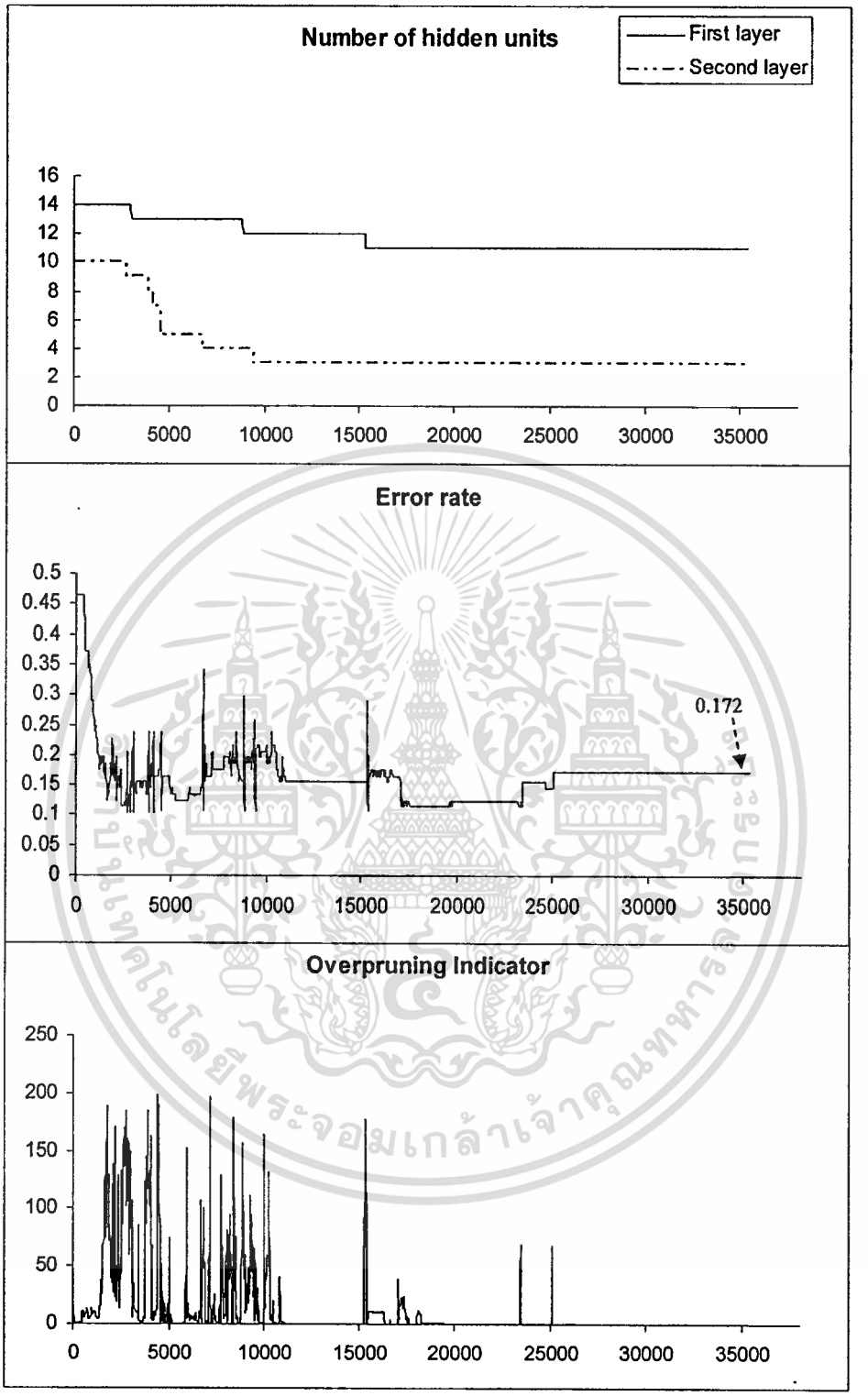
ผลการทดลองกับปัญหาที่มีระดับความไม่เป็นเชิงเส้นสูง นั่นคือ ปัญหา Two Spirals ถูกแสดงในรูปที่ 5.34 สำหรับปัญหาอื่นๆที่ได้กล่าวถึงในปริญญานิพนธ์นี้ล้วนแล้วแต่สามารถใช้โครงข่ายประสาทเทียมที่มีหน่วยซ่อนชั้นเดียวในการแก้ปัญหาได้ ดังนั้นจึงไม่มีประโยชน์ที่จะทดลองปัญหาเหล่านี้กับโครงข่ายประสาทเทียมที่ต้องการหน่วยซ่อนสองชั้น

จากผลการทดลองในรูปที่ 5.34 เราทำการทดลองกับโครงข่ายประสาทเทียมที่มีจำนวนหน่วยซ่อนในชั้นแรกและชั้นที่สองเท่ากับ 24 และ 10 (นั่นคือโครงข่ายเริ่มต้นที่เราใช้ในการทดลองคือ 2-24-10-2) ตามลำดับ เหตุผลที่เราเลือกจำนวนหน่วยซ่อนเป็นจำนวนดังกล่าวเพราะว่าจากผลการทดลองปัญหา Two Spirals กับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหนึ่งชั้น เราได้จำนวนหน่วยซ่อนที่น้อยที่สุดที่ได้จากอัลกอริธึมสปาร์ตันซิมพลิซิตีเท่ากับ 24 ดังนั้น เราจึงทดลองเพิ่มหน่วยซ่อนในชั้นที่สองอีก 10 หน่วย เพื่อทดสอบว่าอัลกอริธึมสามารถลดจำนวนหน่วยซ่อนในชั้นที่สองลงได้มากน้อยเพียงใด จากผลการทดลองที่เราได้ โครงข่ายประสาทเทียมสุดท้ายที่เราได้คือ 2-21-3-2 เราจะเห็นได้ว่าจำนวนหน่วยซ่อนในชั้นแรกลดลงจาก 24 เหลือ 21 และจำนวนหน่วยซ่อนในชั้นที่สองลดลงจาก 10 เหลือเพียง 3 หน่วย ผลการทดลองนี้แสดงให้เห็นว่าอัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถลดจำนวนหน่วยซ่อนของโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นได้ โดยให้จำนวนหน่วยซ่อนสุดท้ายที่ใกล้เคียงกับจำนวนหน่วยซ่อนที่น้อยที่สุดที่ได้จากการทดลองกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหนึ่งชั้น

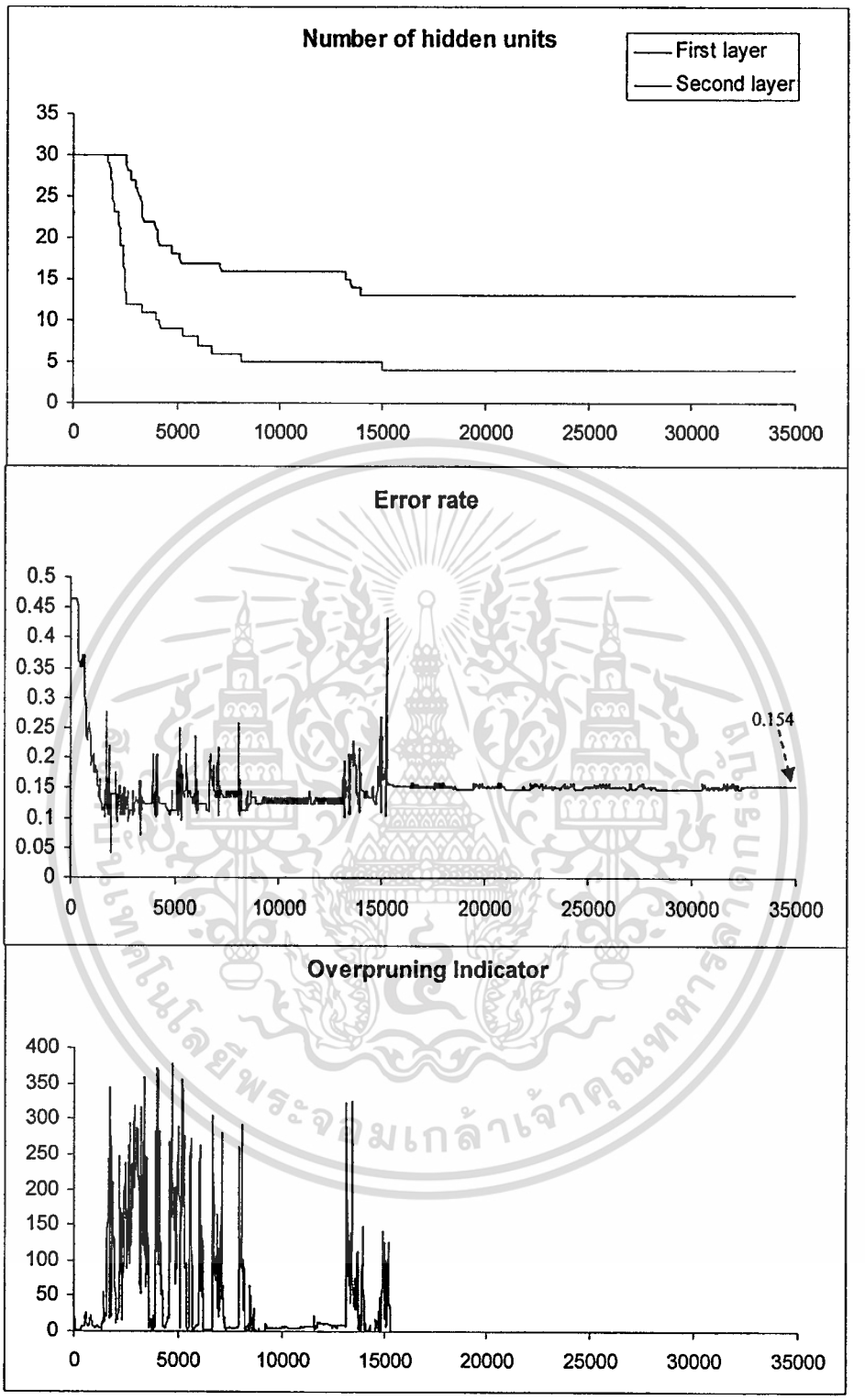
สำหรับผลการทดลองกับโครงข่ายประสาทเทียมเริ่มต้นแบบ 2-14-10-2 และ 2-30-30-2 ถูกแสดงในรูปที่ 5.35 และ 5.36 ตามลำดับ



รูปที่ 5.34 ผลการทดลองกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นสำหรับปัญหา Two Spirals โดยใช้โครงข่ายเริ่มต้น 2-24-10-2 และได้โครงข่ายสุดท้าย 2-21-3-2

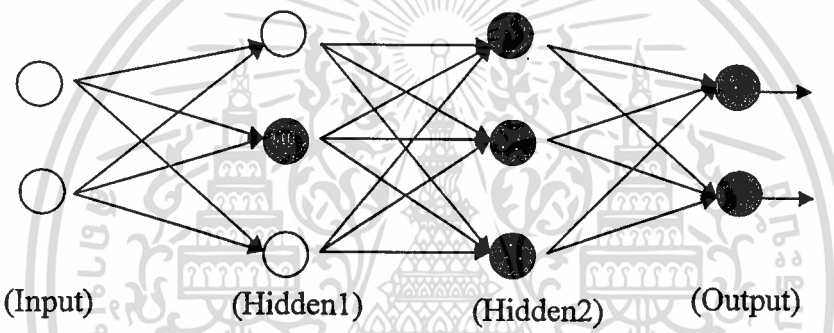


รูปที่ 5.35 ผลการทดลองกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นสำหรับปัญหา Two Spirals โดยใช้โครงข่ายเริ่มต้น 2-14-10-2 และได้โครงข่ายสุดท้าย 2-11-3-2

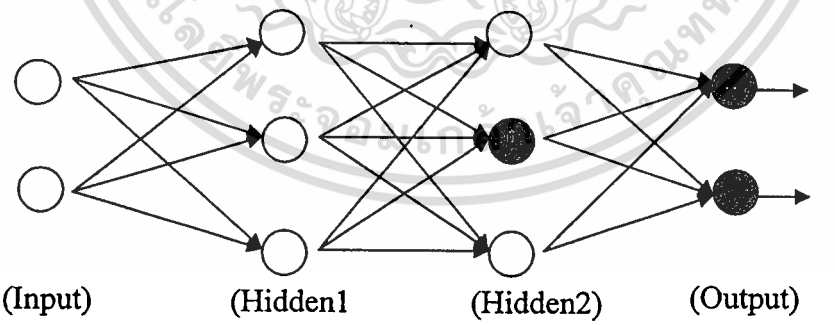


รูปที่ 5.36 ผลการทดลองกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นสำหรับปัญหา Two Spirals โดยใช้โครงข่ายเริ่มต้น 2-30-30-2 และได้โครงข่ายสุดท้าย 2-13-4-2

สิ่งหนึ่งที่สังเกตได้จากการทดลองกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นคือ ค่าอัตราความผิดพลาดที่ได้จากโครงข่ายประสาทเทียมสุดท้ายจะมีค่าสูงกว่าค่าอัตราความผิดพลาดที่ได้จากโครงข่ายประสาทเทียมที่มีหน่วยซ่อนชั้นเดียว ทั้งนี้เป็นผลมาจากการกำจัดหน่วยซ่อนในชั้นแรกจะส่งผลกระทบกับค่าแอ็คติเวชั่นของหน่วยเอาท์พุทมากกว่าการกำจัดหน่วยซ่อนในชั้นที่สอง ซึ่งจากการทดลองกับโครงข่ายทั้งสามแบบ (2-24-10-2, 2-14-10-2 และ 2-30-30-2) ล้วนเกิดการกำจัดหน่วยซ่อนที่ชั้นแรกแทบทั้งสิ้น ดังนั้นจึงส่งผลให้อัตราความผิดพลาดของหน่วยเอาท์พุทมีค่าสูงขึ้น รูปที่ 5.37 แสดงหน่วยที่ได้รับผลกระทบจากการกำจัดหน่วยซ่อน m ในชั้นที่หนึ่ง ซึ่งมีจำนวนมากกว่าหน่วยที่ได้รับผลกระทบจากการกำจัดหน่วยซ่อน n ในชั้นที่สอง ดังแสดงในรูปที่ 5.38 ดังนั้น อัลกอริธึมสปาร์ตันซิมพลิซิตีจึงไม่เหมาะกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้น



รูปที่ 5.37 หน่วยซ่อนที่ได้รับผลกระทบจากการกำจัดหน่วยซ่อนในชั้นที่หนึ่ง



รูปที่ 5.38 หน่วยซ่อนที่ได้รับผลกระทบจากการกำจัดหน่วยซ่อนในชั้นที่สอง

นอกจากนี้ อีกสิ่งหนึ่งที่เราสามารถสังเกตได้จากการทดลองนี้คือ อัลกอริธึมสปาร์ตันซิมพลิซิตีมักจะกำจัดหน่วยซ่อนในชั้นหน่วยซ่อนที่สองมากกว่าชั้นแรก เป็นผลให้โครงข่ายประสาทเทียมสุดท้ายมีจำนวนหน่วยซ่อนในชั้นแรกมากกว่าจำนวนหน่วยซ่อนในชั้นที่สองเสมอ อนึ่ง เราไม่ได้ทดลองกำจัดหน่วยซ่อนในโครงข่ายประสาทเทียมที่มีหน่วยซ่อนสองชั้นกับอัลกอริธึม

สำหรับกำจัดหน่วยซ่อนแบบอื่นๆ เนื่องจากอัลกอริธึมเหล่านั้นถูกออกแบบมาให้รองรับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนเพียงชั้นเดียวเท่านั้น

บทที่ 6

สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้นำเสนอวิธีในการกำจัดหน่วยซ่อนที่ไม่จำเป็นออกไปจากโครงข่ายประสาทเทียมที่มีชั้นซ่อนจำนวนหนึ่งชั้นและมีการป้อนข้อมูลไปข้างหน้าโดยคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนจากชุดข้อมูลแวลิเดชั่น วิธีการที่นำเสนอเป็นวิธีที่เรียบง่ายแต่กลับให้ผลลัพธ์ที่ดีซึ่งทำให้โครงข่ายสามารถทำนายข้อมูลที่ยังไม่เคยเห็นมาก่อนได้อย่างแม่นยำ อัลกอริธึมเริ่มต้นด้วยการฝึกโครงข่ายเริ่มต้นจนกระทั่งโครงข่ายเริ่มที่จะสูญเสียความสามารถในการทำนายข้อมูล จากนั้นอัลกอริธึมจะทำการย้อนกลับไปยังสถานะของโครงข่ายที่ดีที่สุดภายในรอบการฝึกโครงข่ายที่ผ่านมาเพื่อทำให้โครงข่ายรักษาประสิทธิภาพในการทำนายข้อมูลได้ดีที่สุดตลอดเวลา ก่อนที่ขบวนการกำจัดหน่วยซ่อนจะเริ่มขึ้น อัลกอริธึมจะเก็บรักษาค่าน้ำหนักและหน่วยซ่อนเอาไว้เพื่อรองรับการนำโครงข่ายกลับมายังสถานะเดิมในอนาคต อัลกอริธึมจะเริ่มกำจัดหน่วยซ่อนโดยกำจัดหน่วยซ่อนที่มีค่าผลต่างของแอ็คติเวชั่น (Activation difference) น้อยที่สุด ค่าผลต่างของแอ็คติเวชั่นนี้เป็นผลต่างระหว่างค่าแอ็คติเวชั่นของทุกหน่วยเอาท์พุทระหว่างโครงข่ายที่ยังไม่ได้กำจัดหน่วยซ่อนและโครงข่ายที่ถูกกำจัดหน่วยซ่อนออกไปแล้ว (ทั้งนี้กระบวนการดังกล่าวกระทำกับชุดข้อมูลแวลิเดชั่น) ถ้าโครงข่ายไม่สูญเสียประสิทธิภาพในการทำนายข้อมูลและไม่เกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไปขึ้น อัลกอริธึมจะทำการฝึกโครงข่ายต่อไปอีก มิฉะนั้นโครงข่ายก็จะคืนสถานะกลับไปยังสถานะที่ดีที่สุดก่อนหน้านี้และกำจัดหน่วยซ่อนต่อไป หากเกิดเหตุการณ์ที่จำนวนของหน่วยซ่อนถูกกำจัดมากจนเกินไป อัลกอริธึมจะคืนหน่วยซ่อนและน้ำหนักให้กับโครงข่าย และแสดงผลโครงข่ายสปาร์ตันสุดท้าย

วิทยานิพนธ์นี้ได้แสดงให้เห็นถึงประสิทธิภาพของอัลกอริธึมที่นำเสนอโดยใช้ชุดข้อมูลสังเคราะห์จากสามชุดปัญหาและชุดข้อมูลจริงจากเจ็ดชุดปัญหา ความถูกต้อง (Correctness) ของผลการทดลองถูกยืนยันโดยปัญหาซึ่งเราทราบจำนวนที่น้อยที่สุดของหน่วยซ่อน นั่นคือ F1, F2 และ AC นอกจากนี้ ความถูกต้องของอัลกอริธึมที่นำเสนอยังสามารถยืนยันได้จากผลการทดลองในปัญหามาตรฐานซึ่งมีประสิทธิภาพที่ดีกว่าหรือเทียบเท่ากับอัลกอริธึมอื่นๆ ความสามารถในการทนทาน (Robustness) ของอัลกอริธึมถูกทดสอบโดยชุดข้อมูลที่มีสัญญาณรบกวนมาก เช่น ปัญหา Diabetes, Heart disease และ Credit card เป็นต้น ความสามารถในการทำงานกับปัญหาที่มีขนาดใหญ่ (Scalability) ของอัลกอริธึมถูกยืนยันโดยการทดลองกับปัญหา Isolet ซึ่งให้ผลเป็นที่น่าพอใจ

การวิเคราะห์ผลการทดลองในแง่ของการใช้ชุดข้อมูลแวลิเดชั่นสำหรับคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อนในวิธีการกำจัดหน่วยซ่อนอื่นๆที่มีอยู่แล้ว (นั่นคือ OBD, OBS และ Skeletonization) ได้ถูกนำเสนอในวิทยานิพนธ์เพื่อวิเคราะห์ว่าชุดข้อมูลแวลิเดชั่นมีผลอย่างไรต่อการคำนวณค่าตัวชี้วัดความสำคัญของหน่วยซ่อน ผลปรากฏว่าอัลกอริธึมที่ใช้ทดสอบมีประสิทธิภาพที่ดีขึ้นจากเดิม แต่ยังมีประสิทธิภาพด้อยกว่าผลการทดลองที่ได้จากอัลกอริธึมสปาร์ตันซิมพลิซิตี ทั้งนี้เนื่องจากอัลกอริธึมสปาร์ตันซิมพลิซิตีให้ความสำคัญกับจำนวนของข้อมูลที่ทำนายผิดพลาดเมื่อหน่วยซ่อนถูกกำจัดออกไปเป็นอันดับแรก ถ้ามีหน่วยซ่อนมากกว่าหนึ่งหน่วยที่เสมอกันด้วยค่าจำนวนข้อมูลที่จำแนกผิดประเภท อัลกอริธึมจะใช้ค่าผลต่างของแอ็คติเวชั่นเพื่อตัดสินว่าหน่วยซ่อนใดมีความสำคัญน้อยกว่ากัน

นอกจากนี้ ยังได้มีการขยายผลการทดลองสำหรับปัญหาที่มีระดับของความไม่เป็นเชิงเส้นสูง นั่นคือ ปัญหา Two Spirals ซึ่งถือว่าเป็นปัญหาที่มีความยากในการจำแนกประเภท ผลปรากฏว่าอัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถลดจำนวนของหน่วยซ่อนจากเดิม 60 หน่วย เหลือเพียง 24 หน่วย โดยยังคงให้อัตราความผิดพลาดในการทำนายข้อมูลใกล้เคียงกับโครงข่ายก่อนที่จะถูกกำจัดหน่วยซ่อน นอกจากนี้ จากการเปรียบเทียบผลการทดลองกับอัลกอริธึมอื่นๆ อัลกอริธึมสปาร์ตันซิมพลิซิตียังสามารถกำจัดหน่วยซ่อนได้มากกว่าอัลกอริธึมอื่นๆและยังให้ค่าอัตราผิดพลาดที่ใกล้เคียงกับวิธีอื่นอีกด้วย ดังนั้น ผลการทดลองนี้จึงสามารถใช้ยืนยันประสิทธิภาพของอัลกอริธึมสปาร์ตันซิมพลิซิตีในการประยุกต์ใช้กับปัญหาที่มีลักษณะไม่เชิงเส้นได้

จากการวิเคราะห์การฝึกโครงข่ายประสาทเทียมด้วยวิธี Brute force ในวิทยานิพนธ์นี้ โครงข่ายประสาทเทียมอาจจะสามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ด้วยจำนวนหน่วยซ่อนจำนวนหนึ่ง แต่อาจไม่สามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ด้วยจำนวนหน่วยซ่อนที่ใกล้เคียงกัน ปัญหานี้ได้ถูกทดสอบกับอัลกอริธึมสปาร์ตันซิมพลิซิตี ผลปรากฏว่าอัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถหลีกเลี่ยงปัญหานี้ได้ เนื่องจากหากเราค่อยๆกำจัดหน่วยซ่อนทีละหน่วยตามลำดับของหน่วยซ่อนที่มีความสำคัญน้อยที่สุดโดยไม่ได้สุ่มค่าเริ่มต้นใหม่ให้กับน้ำหนักภายในโครงข่ายประสาทเทียม ช่วงของค่าผิดพลาด (Error Scope) ที่วางอยู่บนพื้นผิวค่าผิดพลาด (Error Surface) จะค่อยๆแคบลงโดยไม่ทำให้คำตอบที่ดีหายไปจากช่วงของค่าผิดพลาดนี้ จึงทำให้โครงข่ายประสาทเทียมสามารถเข้าสู่คอนเวอร์เจ้นซ์ได้ถึงแม้ว่าหน่วยซ่อนจะถูกกำจัดออกไปทีละหน่วยก็ตาม

ปัจจัยห้าประการที่ทำให้อัลกอริธึมสปาร์ตันซิมพลิซิตีประสบความสำเร็จคือ ประการแรก อัลกอริธึมของเราสามารถคำนวณค่าผลต่างของแอ็คติเวชั่นเอาท์พุทระหว่างก่อนและหลังการกำจัดหน่วยซ่อน โดยค่าผลต่างของแอ็คติเวชั่นนี้ถูกคำนวณจากชุดข้อมูลแวลิเดชั่นซึ่งโครงข่ายไม่ได้ใช้ในการฝึกฝน ประการที่สอง การคำนวณค่าผลต่างของแอ็คติเวชั่นเอาท์พุทสามารถกระทำได้ภายในโปรแกรมส่วนที่คำนวณหาค่าผิดพลาดแวลิเดชั่นซึ่งทำให้ประหยัดการคำนวณด้วยการส่งชุดข้อมูลแวลิเดชั่นเพียงครั้งเดียวและการคำนวณนี้ไม่มีผลกระทบต่อความซับซ้อนของ

อัลกอริธึมแบ็คพรอพาเกชั่น ประการที่สาม สำหรับปัญหาการจำแนกประเภท อัลกอริธึมคำนวณค่าตัวชี้วัดความสำคัญโดยพิจารณาจำนวนชุดข้อมูลแวลิเดชั่นที่จำแนกผิดพลาดเมื่อแต่ละหน่วยซ่อนถูกกำจัดเป็นหลัก ซึ่งจะช่วยแก้ปัญหาที่ค่าตัวชี้วัดความสำคัญของหน่วยซ่อนให้ค่าผิดพลาดถึงแม้ว่าชุดข้อมูลแวลิเดชั่นจะถูกใช้ในการคำนวณก็ตาม ประการที่สี่ การวัดความสูญเสียความสามารถในการทำนายข้อมูล (Generalization loss) ช่วยให้อัลกอริธึมสปาร์ตันซิมพลิซิตีสามารถหลีกเลี่ยงปัญหาการฝึกมากจนเกินไป (Overfitting) โดยยอมให้มีการหยุดการฝึกโครงข่ายก่อนถึงเวลาอันควรในระหว่างที่ทำการฝึก ประการที่ห้า อัลกอริธึมสปาร์ตันซิมพลิซิตีไม่ได้กำหนดค่าเริ่มต้นใหม่ของน้ำหนักภายในโครงข่ายหลังจากที่หน่วยซ่อนถูกกำจัดออกไป การใช้ค่าน้ำหนักเดิมจะช่วยป้องกันไม่ให้เกิดเหตุการณ์ที่โครงข่ายที่มีขนาดเล็กลงตอบสนองในทางลบกับการกำหนดค่าเริ่มต้นใหม่ทั้งหมด

6.2 ข้อเสนอแนะ

จุดที่สามารถปรับปรุงให้ดีขึ้นสำหรับอัลกอริธึมสปาร์ตันซิมพลิซิตีมีอยู่สองประเด็นคือ ประเด็นแรก เราสามารถตรวจจับสถานการณ์ที่มีการกำจัดหน่วยซ่อนมากจนเกินไปได้รวดเร็วขึ้นเพื่อหลีกเลี่ยงการคืนสถานะให้โครงข่ายซึ่งจะใช้เวลานาน โดยอาจใช้วิธีทางสถิติในการคำนวณว่าหน่วยซ่อนที่มีค่าตัวชี้วัดความสำคัญที่น้อยที่สุดไม่ได้มีความสำคัญที่แตกต่างกันอย่างชัดเจนกับหน่วยซ่อนอื่นๆ ซึ่งหากเหตุการณ์นี้เกิดขึ้นอัลกอริธึมก็สามารถหยุดการทำงานได้โดยไม่ต้องรอให้เกิดสถานการณ์ที่มีหน่วยซ่อนไม่เพียงพอ ประเด็นที่สอง เราสามารถออกแบบวิธีในการคำนวณค่าเทรชโฮลด์สำหรับ AD_j ที่เหมาะสมเพื่อใช้ในการกำจัดหน่วยซ่อนทีละหลายๆหน่วยซึ่งจะเป็นผลให้เวลาในการทำงานลดลงไปด้วย

ข้อจำกัดของอัลกอริธึมสปาร์ตันซิมพลิซิตีคือสมมุติฐานที่ว่าเราจะต้องมีจำนวนชุดข้อมูลแวลิเดชั่นที่มากพอ (ประมาณร้อยละ 25 ของชุดข้อมูลทั้งหมด) มิฉะนั้น อัลกอริธึมนี้จะให้ความสำคัญกับชุดข้อมูแวลิเดชั่นซึ่งมีจำนวนน้อย ทำให้ไม่สามารถทำนายชุดข้อมูลที่ยังไม่เคยเห็นมาก่อนได้อย่างถูกต้อง นอกจากนี้ อัลกอริธึมที่นำเสนอยังไม่เหมาะที่จะนำไปประยุกต์ใช้กับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้น เนื่องจากจะทำให้อัตราค่าผิดพลาดมีค่าสูงขึ้นและทำให้หน่วยซ่อนในชั้นท้ายสุดเท่านั้นที่มักจะถูกกำจัดออกจากโครงข่าย โดยหน่วยซ่อนภายในชั้นต้นๆแทบจะไม่ถูกกำจัดออกไปจากโครงข่ายเลย และการประยุกต์ใช้อัลกอริธึมสปาร์ตันซิมพลิซิตีกับโครงข่ายประสาทเทียมที่มีหน่วยซ่อนหลายชั้นยังทำให้โครงข่ายสุดท้ายที่ได้มีความสามารถในการทำนายข้อมูลไม่ดีนักเมื่อเทียบกับโครงข่ายดั้งเดิมก่อนที่จะถูกกำจัดหน่วยซ่อน

เอกสารอ้างอิง

[1]Kwok T. Y. and Yeung D. Y. "Constructive algorithms for structure learning in feedforward neural networks for regression problems." **IEEE Trans. Neural Net,** vol.8, no.3, 1997, pp.630-645.

[2] Reed R. "Pruning algorithms -A survey." **IEEE Trans. Neural Net,** vol.4, no.5, September 1993, pp.740-747.

[3] Mozer M.C. and Smolensky P. "Skeletonization: A technique for trimming the fat from a network via relevance assessment." **Advances in Neural Information Processing Systems,** vol.1, Morgan Kaufman, San Mateo, CA, 1989, pp. 107-115.

[4] Sietsma J. and Dow R.J.F. "Neural net pruning - why and how." **Proc. IEEE Int. Joint Conf. Neural Networks,** vol. I (San Diego), 1988, pp.325-333.

[5] Sietsma J. and Dow R.J.F. "Creating artificial neural networks that generalize." **Neural Networks,** vol.4, no.1, 1991, pp. 67-69.

[6] LeCun Y., Denker J.S., and Solla S.A. "Optimal brain damage." **Advances in Neural Information Processing Systems,** vol.2, Morgan Kaufman, San Mateo, CA, 1990, pp.598-605.

[7] Hassibi B. and Stork D.G. "Second order derivatives for network pruning: Optimal brain surgeon." **Advances in Neural Information Processing Systems,** vol. 5, Morgan Kaufman, San Mateo, CA, 1993, pp.162-171.

[8] Murase K., Matsunaga Y., and Nakade Y. "A Back-Propagation Algorithm which Automatically Determines the Number of Association Units." **Proc. IEEE Int. Joint Conf. Neural Networks,** vol. 1, 1991, pp. 783-788.

[9] Shahjahan Md., Murase K. "A Dynamic Node Decaying Method for Pruning Artificial Neural Networks." **IEICE Trans. Inf. & System,** vol. E86-D, no. 4, April 2003, pp. 736-751.

[10] Hagiwara M. "Removal of Hidden Units and Weights for Back Propagation Networks." **Proc. IEEE Int. Joint Conf. Neural Networks,** vol. 1, 1993, pp. 351-354.

[11] Hagiwara M. "Novel BackPropagation Algorithm for Reduction of Hidden Units and Acceleration of Convergence using Artificial Selection." **Proc. IEEE Int. Joint Conf. Neural Networks,** vol. I, 1990, pp. 625-630.

[12] Engelbrecht A. P. "A new pruning heuristic based on variance analysis of sensitivity information." **IEEE Trans. Neural Network,** vol. 12, no. 6, Nov. 2001, pp. 1386-1399.

[13] Lauret P., Fock E., and Mara T. A. "A node pruning algorithm based on a Fourier amplitude sensitivity test method." **IEEE Trans. Neural Network,** vol.17, no.2, 2006, pp.273-293.

[14] Prechelt L. "Proben1- A set of neural network benchmark problems and benchmarking rules." **Tech. Report,** Karlsruhe, Germany, Sept. 1994.

[15] Chauvin Y. "Generalization performance of overtrained backpropagation networks." **Neural Networks, Proc. EUROSIP Workshop, L. B. Almeida and C. J. Wellekens, Eds.,** February. 1990, pp. 46-55.

[16] LeCun Y. "Efficient Learning and Second-order Methods, A Tutorial." **NIPS,** Denver, 1993.

[17] Haykin S. "Neural Networks: A Comprehensive Foundation." **Macmillan College Publishing Company,** New York, 1994.

[18] Yao X. and Liu Y. "A New Evolutionary System for Evolving Artificial Neural Networks." **IEEE Trans. Neural Network,** vol.8, no.3, May 1997, pp.694-713.

[19] Baum E.B. and Lang K. J. "Constructing hidden units using examples and queries" **Lippmann P., Moody J. E., and Touretzky D.S. (Eds.): Advances in neural information processing systems.,** vol. 3, Morgan Kaufmann Pub., 1991.

ภาคผนวก

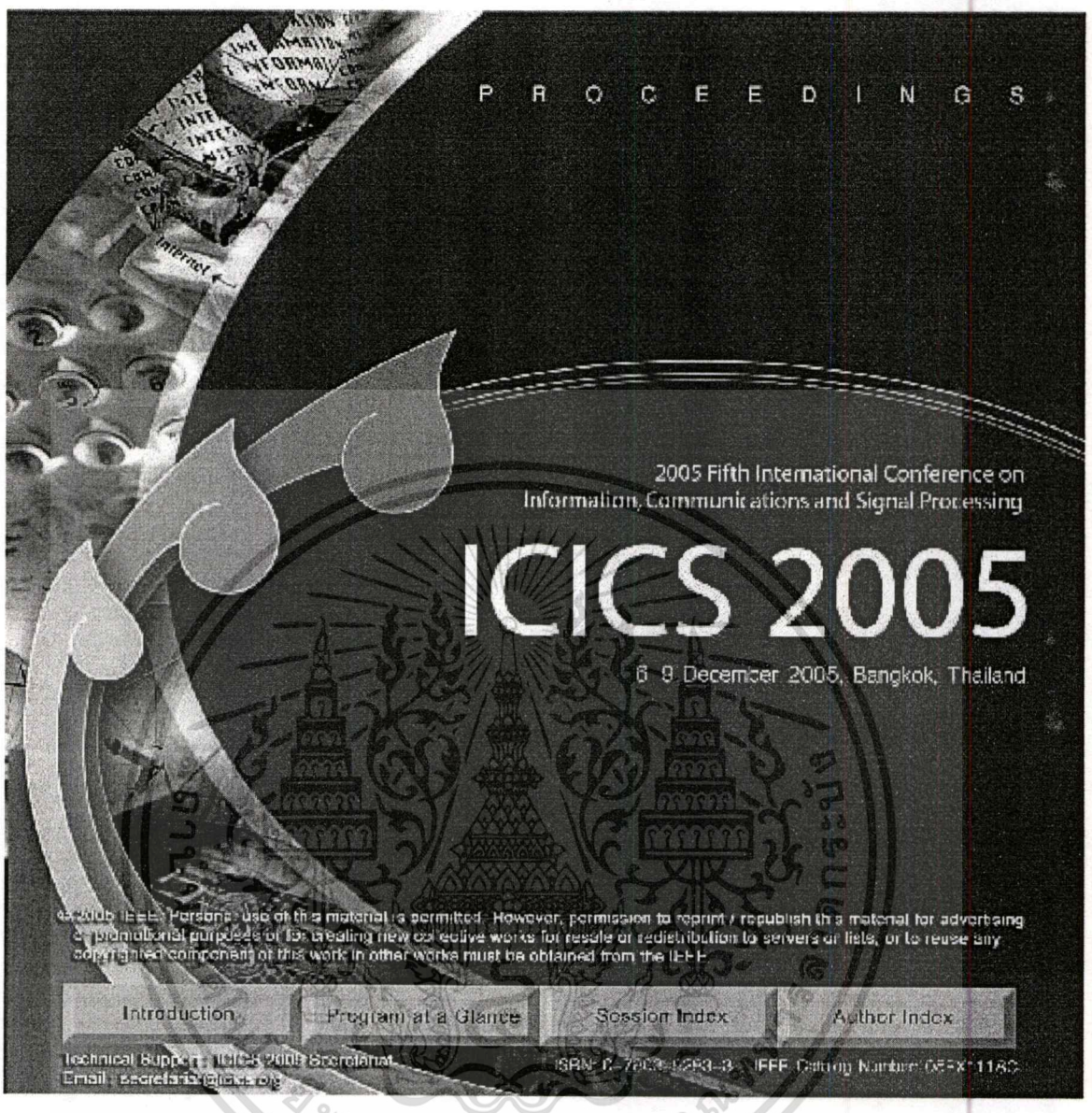
ภาคผนวก ก.

ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

1. Kietikul Jearanaitanakij. and Ouen Pinngern "Determining the Orders of Feature and Hidden Unit Prunings of Artificial Neural Networks." **Proceedings of the Fifth International Conference on Information, Communications and Signal Processing (ICICS). Bangkok, Thailand,** December 6-9, 2005.

2. Kietikul Jearanaitanakij. and Ouen Pinngern "Determine the Irrelevance of Hidden Unit from the Validation Set." **Proceedings of the International Symposium on Communications And Information Technologies (ISCIT), Bangkok, Thailand,** October 18-20, 2006.

3. Kietikul Jearanaitanakij. and Ouen Pinngern "Spartan Simplicity: A Pruning Algorithm for Neural Nets." **Journal of Circuits, Systems, and Computers (JCSC), World Scientific Publishing Company,** Vol. 17, No. 4, August 2008.



W3C.3

Determining the Orders of Feature and Hidden Unit Prunings of Artificial Neural Networks

Kietikul JEARANAITANAKIJ
Department of Computer Engineering
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
kjkietik@kmitl.ac.th

Ouen PINNGERN
Department of Computer Engineering
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
kpouen@kmitl.ac.th

***Abstract*— There is a great deal of research undertaken for pruning away features and hidden units in order to reduce the size of Artificial Neural Networks (ANNs). However, none of these methods mentions about the relationship between the pruned unit and the number of epochs needed for retraining when the unit is pruned away from the network. In this paper, we present two heuristics for determining the pruning orders, which lead to the near smallest number of retraining epochs. The heuristics are based on the employment of the modified information gain calculated from all features in training data. Then, we test our proposed heuristics on an exclusive-or data set. The experimental results show the success of using information gain as a criterion for determining the pruning orders.**

Keywords- Artificial Neural Networks, pruning order, feature pruning, hidden unit pruning, information gain

I. INTRODUCTION

An artificial neural network can be defined as a model of reasoning based on the human brain. Among models of ANNs, Backpropagation (BP), developed by Rumelhart et al. [1] in 1986, is the most widely used method for training feed-forward neural networks. However, the typical BP method trains ANNs without any reduction in size, which is sometimes too bulky. In order to produce a compact network, there are two issues that we need to minimize: the number of features of a data set and the number of hidden units. In addition, removing features and hidden units in wrong order may extend the retraining time. Therefore, we focus on the solution to minimize the retraining time.

There are a large number of investigations undertaken to reduce the size of ANNs. Belue and Bauer [2] reported several saliency measures in order to select the feature to be removed from the network. Setiono and Liu [3] proposed the network accuracy, by adding a penalty term to the error function of the network, on the training data set as a criterion for determining feature removal. Mozer and Smolensky [4] described a *Skeletonization* method estimating which unit is the least important according to the smallest effect on the training error and deleting it during training. Sietsma and Dow [5], [6] suggested an interactive method in which they inspect a trained network and identify a hidden unit that has a constant activation over all training patterns. Then, the hidden unit which does not influence the output is pruned away. Murase et al. [7] measured the *Goodness Factors* of the hidden units in the trained network. The unit which has the lowest value of the *Goodness Factor* is removed from the hidden layer. Hagiwara [8] presented the *Consuming Energy* and the *Weights Power* methods for removal of both hidden units and weights, respectively.

Among these methods to reduce the size of ANNs, none mentions about the order of the unit pruning that can lead to the near minimum retraining time. In this paper, we propose two heuristics for determining the pruning orders of features and hidden units in ANNs. These heuristics employ a power of information gain as a criterion for pruning orders. We perform experiments on some variations of the exclusive-or problem. The outputs show that the proposed heuristics prune away unnecessary features and hidden units in ANNs, resulting in a remarkable retraining time.

The rest of this paper is organized into the following orders. In Section 2, we explain a modified version of information gain in order to classify the continuous data set. In Section 3, we employ information gain to prune away features and hidden units. Next, in Section 4, we describe the experimental study, data sets used, and experimental results. Finally, in Section 5, we summarize our findings and suggest possible directions for future investigations.

II. MODIFICATION OF INFORMATION GAIN

We begin by defining a modification of information gain in order to classify a continuous data set. Entropy, a measure commonly used in the information theory, characterizes the (im)purity of an arbitrary collection of examples. Given a collection S, containing examples with each of the C outcomes, the entropy of S is

$$Entropy\ (S) = \sum_{I \in C} [-p(I)\log_2 p(I)], \qquad (1)$$

where $p(I)$ is the proportion of S belonging to class I. Note that S is not a feature but an entire sample set. Entropy is 0 if all members of S belong to the same class. The scale of the entropy is 0 (purity) to 1 (impurity). The next measure is an information gain. This was first defined by Shannon and

Weaver [9] to measure the expected reduction in entropy. For a particular feature A, *Gain(S, A)* means the information gain of the sample set S on the feature A and is defined by the following equation:

$$Gain(S,A) = Entropy(S) - \sum_{v \in A}[(|S_v|/|S|).Entropy(S_v)], \quad (2)$$

where Σ is the summation on all possible values (v) of the feature A; Sv is the subset of S for which feature A has value v; $|Sv|$ is the number of elements in Sv, and; $|S|$ is the number of elements in S.

We consider the feature which has a high value of information gain being important to the data set classification. A typical information gain categorizes the data set whose feature values are discrete and bound into a particular range. In contrast, if possible values (v) of feature A are uncountable, for example, feature A has real values in the range of (-1, 1), there will be a huge number of computations taking place in the summation of (2). Therefore, we need to revise information gain in order to serve a continuous feature. We modify information gain by quantizing the range of a continuous feature into k intervals, where k is the minimal constant depending on the range of a particular feature. Thus, the following condition must be added.

$$|v| = k. \quad (3)$$

In order to discretize feature A into k intervals, we use *MD* (*Maximal Discernibility*) heuristic [10] to find a minimal set of intervals. A group of examples with continuous features will fall into an appropriate space. A large number of possible values of the feature A then become countable and hence *Gain(S, A)* is easy to compute.

III. PRUNING HEURISTICS

This section describes two heuristics used as ordering criterions for the feature and the hidden unit prunings in ANNs. The goal of these heuristics is to determine the pruning orders leading to the near smallest number of epochs for retraining the network. The heuristics are based on the employment of the modified information gain as presented in the previous section.

A. Feature Pruning Heuristic

Removing unimportant input features in the training set leads to a compact network which costs fewer computations. Our feature pruning heuristic reverses the concept of the ID3 algorithm (Quinlan, 1986, [11]), which selects the next testing feature in descending order of information gains associated with features. In contrast, our heuristic prunes away features in the training set in ascending order of their information gains. Before the feature pruning begins, we need to train the original network and calculate the information gains of all the features in the training set. After the initial network has been trained, the feature which has the smallest information gain is selected as the unit to be removed. This is simply performed by logically setting weights between the selected feature and hidden units into zero. Then, the network is retrained. Setting the weights connected to the selected feature into zero is the same as removing it. We do not physically prune away the selected feature from the data set because we can use the original data set for training the network without any changes. If the network successfully converges, the same process of removing the feature will be repeated. There is only one feature removed at a time until the network cannot converge. Then, the process restores the last pruned unit and network configurations. There is no need trying any other feature removals because the network will not converge. In addition, we discover that the number of epochs needed for retraining depends on the value of the information gain of the removed feature. When the feature which has the lowest value of information gain is removed, the network spends the smallest number of epochs for retraining. Thus, the experimental results in Section 4 show that removing unimportant features in this fashion leads to the smallest number of retraining epochs.

B. Hidden Unit Pruning Heuristic

From the previous subsection, an information gain can tell us how a feature is important to the network. Similarly, we find that the importance of the hidden unit is also congruent with that of the information gain. The hidden unit pruning heuristic proposed here is based on the propagated information gains from feature units. Before going further, let us define some notations used in this section such as information gain of feature unit i ($Gain_i$), incoming information gain of a hidden unit ($Gain_{In}$), outgoing information gain of a hidden unit ($Gain_{Out}$), the weight from the i-th unit of the (n-1)-th layer to the j-th unit of the n-th layer ($w_{ij}^{n-1,n}$), and, similarly, the weight from the j-th unit of the n-th layer to the k-th unit of the (n+1)-th layer ($w_{jk}^{n,n+1}$). All notations are shown in Fig. 1.

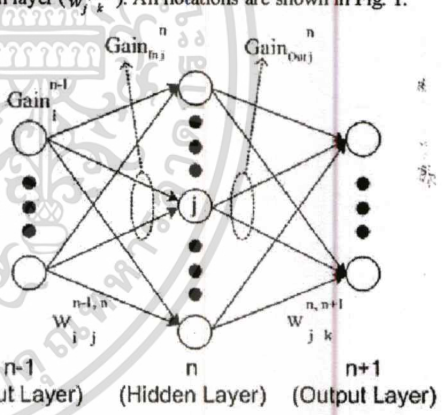


Figure 1. Network and notations

The amount of information received at a hidden unit is the summation, on training patterns, of the total squared production between weights, which connect from feature units to a hidden unit in a hidden layer, and information gains of all feature units. Then the result is averaged over the number of training patterns and the number of feature units. We define the incoming information gains of the j-th hidden unit in n-th layer ($Gain_{In_j^n}$) as the following:

$$Gain_{In\ j}^{\ n} = \frac{1}{P \times I} \times \sum_{P} \sum_{i} (w_i^{n-1,n}{}_j \times Gain_i^{\ n-1})^2, \quad (4)$$

where P and I are the number of training patterns and the number of feature units in the (n-1)-th layer, respectively. This $Gain_{In\ j}^{\ n}$ is, in turn, used for calculating the outgoing information gain of the j-th hidden unit. The degree of importance of a particular hidden unit can be determined by the outgoing information gain of the hidden unit ($Gain_{Out\ j}^{\ n}$). The outgoing information gain of a particular hidden unit is the summation, on training patterns, of the total squared production between weights, which connect from the hidden unit to output units, and the incoming information gain of that hidden unit. Then the result is averaged over the number of training patterns and the number of output units. The outgoing information gain of the j-th hidden unit in the n-th layer ($Gain_{Out\ j}^{\ n}$) is given by:

$$Gain_{Out\ j}^{\ n} = \frac{1}{P \times O} \times \sum_{P} \sum_{k} (w_j^{n,\ n+1}{}_k \times Gain_{In\ j}^{\ n})^2, \quad (5)$$

where O is the number of output units in the (n+1)-th layer. Note that the number of training patterns, P, in both (4) and (5) is the number of training patterns that the network has seen so far.

The hidden unit which has the lowest outgoing information gain should be firstly removed from the trained network because it does not affect the convergence time for retraining the network that much. Similar to the feature pruning heuristic, there is only one hidden unit removed at a time until the network cannot converge. Then, the last pruned unit and network configurations are restored. In Section 4, our experimental results illustrate the comparison on the number of retraining epochs in all circumstances of the removals and, hence, verify our stated heuristic.

IV. Experimental Study

Here we perform experiments under a well-known standard benchmark that is the exclusive-or (XOR) problem.

A. Experimental Results

The following subsections show the outputs of our proposed heuristics: the feature pruning and the hidden unit pruning heuristics.

1) Feature pruning experiment

In order to verify that the number of epochs needed for retraining depends on the value of information gain of the removed feature, we establish an experiment on the XOR problem to have two essential input features and three unnecessary features. Our feature pruning heuristic can prune away three unnecessary features within a small period of retraining time. Table 1 shows the relationship between the information gain of a feature and the number of epochs needed for retraining when that feature is pruned away.

In Table 1, removing feature number 3, which has the lowest information gain, leads to the smallest number of retraining epochs. For the first two features, they are so essential to the output classification that the network will not get a convergence if they are removed. As a result, the order of feature removals can be represented as: 3, 4, and 5. Note that, after feature number 3 is removed, the number of epochs needed for retraining the other features in Table 1 may not be the same. However, the removal order does not alter. The number of input features and the sum-squared error are depicted in Fig. 2 and 3, respectively.

TABLE I RELATIONSHIP BETWEEN INFORMATION GAIN AND CONVERGENCE

Feature number	Information Gain	Number of epochs needed for retraining
1	0.5	n/a
2	0.5	n/a
3	0.065	720
4	0.125	850
5	0.191	1090

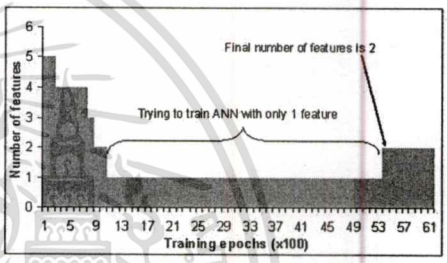


Figure 2. The number of input features during retraining period

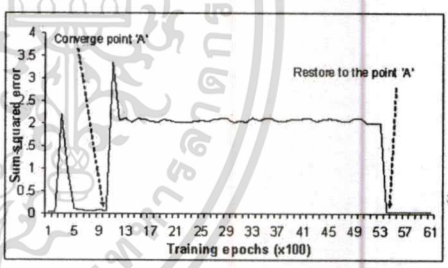


Figure 3. Sum of squared errors of feature pruning

2) Hidden unit pruning experiment

In this experiment, we focus on the hidden unit pruning which considers an outgoing information gain as a pruning criterion. Similar to the previous investigation, we use the XOR problem. The number of features is set to two, while the initial number of hidden units is set to ten, which is a fairly large number so that the network gets convergence easily. Table 2 shows the relationship between the outgoing information gain of a hidden unit and the number of epochs needed for retraining when that hidden unit is pruned away.

In Table 2, the outgoing information gain is a distinctly good criterion for prioritizing the unimportant hidden units to

be pruned away from the trained network. The hidden unit which has the lowest outgoing information gain should be firstly removed because it takes the smallest retraining time comparing to others. Note that the values in Table 2 are used only once for the first time of hidden unit pruning. These values are changed when we begin to prune away the next unit. In contrast to Table 1, the order of values in Table 2 can be altered after each hidden unit pruning. In addition, the results in Fig. 4 and 5 indicate that the number of hidden units begins to reach a theoretical number, which is two, at approximately 125 learning epochs.

TABLE II. RELATIONSHIP BETWEEN OUTGOING INFORMATION GAIN AND THE NUMBER OF RETRAINING EPOCHS

Hidden unit number	Outgoing Information Gain	Number of epochs needed for retraining
1	0.0000002	20
2	0.0000266	32
3	0.0000772	33
4	0.3543068	47
5	0.3764098	99
6	0.7774660	530
7	1.0322675	910
8	3.6764264	3315
9	3.9485948	3423
10	5.0070827	4903

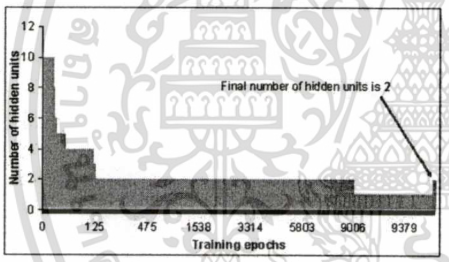


Figure 4. The number of hidden units during retraining period

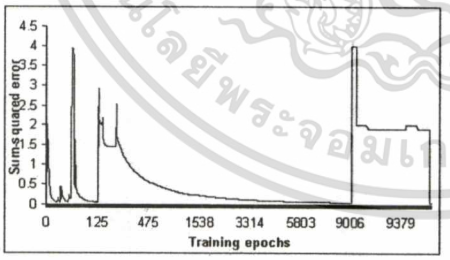


Figure 5. Sum of squared errors of hidden unit pruning

V. CONCLUSIONS

We have proposed two heuristics for determining the pruning orders of features and hidden units in ANNs. Both heuristics are based on the utilization of the information gain as a criterion. In the input layer, information gain indicates the degree of importance of a feature. The feature which has the smallest information gain is not important to the output classification and it should be ignored. As a result, we have the smallest number of epochs needed for retraining. In the hidden layer, the hidden unit which has the smallest outgoing information gain tends to propagate rather small amount of information to the output units in the next layer. Consequently, removing that unit from the network gives little effect on the retraining time. The experimental results on the XOR problem indicate that the pruning orders of features and hidden units as suggested by our heuristics lead ANNs to the near minimum retraining time. Besides the three-layer ANNs, the multi-layer neural networks can also use our heuristics to efficiently decrease their pruning time. However, our pruning heuristics are not completely immune. One apparent problem found is that the network may encounter the local minimum problem during the ending period of both feature and hidden unit pruning processes, resulting in a slow convergence. One of the possible solutions is to apply the convergence acceleration technique to significantly speed up the convergence in that situation.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in Parallel Distributed Processing: Exploration in the Microstructure of Cognition: vol.1: Foundations, eds. D.E. Rumelhart and J.L. McClelland, pp.318-362, The MIT Press, Cambridge, Massachusetts, 1986.

[2] L.M. Belue and K.W. Bauer, Jr., "Determining input features for multilayer perceptron," Neurocomputing, vol. 7, no. 2, pp. 111-122, 1995.

[3] R. Setiono and H. Liu, "Neural-Network Feature Selector," IEEE Trans. on Neural Networks, vol. 8, no.3, pp.654-662, 1997.

[4] M.C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, vol.1, pp. 107-115, Morgan Kaufman, San Mateo, CA, 1989.

[5] J. Sietsma and R.J.F. Dow, "Creating artificial neural networks that generalize," Neural Networks, vol.4, no.1, pp. 67-69, 1991

[6] J. Sietsma and R.J.F. Dow, "Neural net pruning – why and how," in Proc. IEEE Int. Conf. Neural Networks, vol. I (San Diego), pp.325-333, 1988.

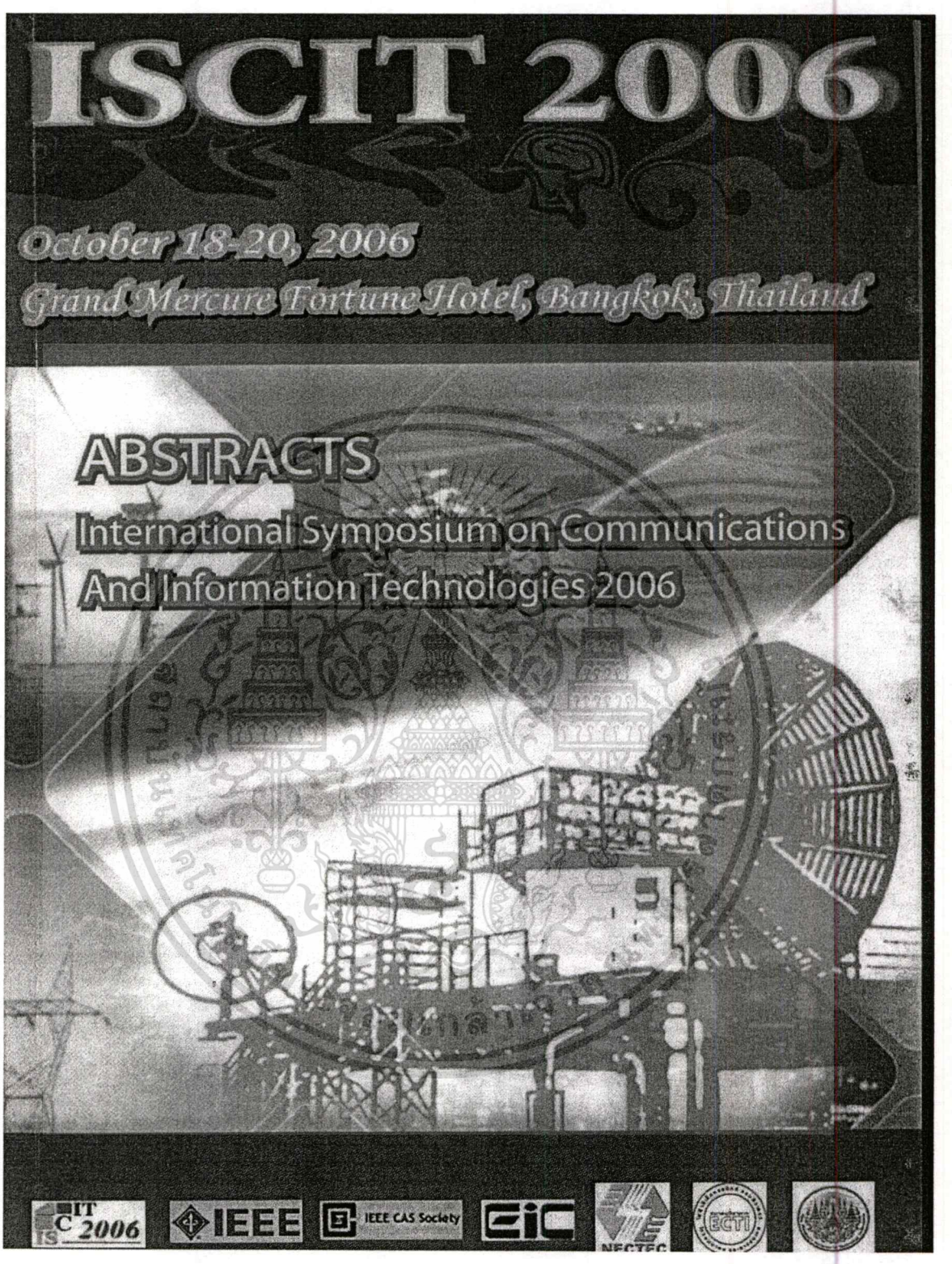
[7] K. Murase, Y. Matsunaga, and Y. Nakade, "A Back-Propagation Algorithm which Automatically Determines the Number of Association Units," Proc. IEEE Int. Conf. Neural Networks, vol. 1, pp. 783-788, 1991.

[8] M. Hagiwara, "Removal of Hidden Units and Weights for Back Propagation Networks," Proc. IJCNN'93, vol. 1, pp. 351-354, 1993.

[9] Shannon, C. E. and Weaver, W., The Mathematical Theory of Communication, University of Illinois Press, Urbana, Illinois, 1949.

[10] Nguyen H. S., Skowron A., "Quantization of real values attributes, Rough set and Boolean Reasoning Approaches," Proc. of the Second Joint Annual Conference on Information Sciences, Wrightsville Beach, NC, pp. 34-37, 1995.

[11] Quinlan, J. R., "Induction of decision trees," Machine Learning, vol. 1, issue 1, pp. 81–106., 1986.



Determine the Irrelevance of Hidden Unit from the Validation Set

Kietikul Jearanaitanakij and Ouen Pinngern
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
E-mail: kjkietik@kmitl.ac.th

***Abstract*— This paper proposes a method to determine the irrelevance of the hidden unit in the artificial neural network. Unlike other approaches, we calculate the sensitivity of the hidden unit from the validation set, instead of the training set. The advantage of using the validation set to calculate the sensitivity is that we never overestimate the relevance of hidden unit. In other words, we always remove the unit that has the least effect on the validation set error. As a result, the pruned neural network has the highest generalization when compared with other choices of removals. Our sensitivity is based on the activation difference of the output unit. This activation difference is the gap between the activation of output units when a particular hidden unit is present and when it is removed. We have applied our technique to two standard benchmark problems. The experimental results show that the proposed technique can correctly determine the least irrelevant hidden unit.**

I. INTRODUCTION

Artificial neural network is a statistical model whose weights are estimated from a training set. If the network is trained with a sufficient training patterns and a sufficient number of units, it can learn any function [1]. However, having too many hidden units than necessary can produce a neural network that has a poor generalization. In order to prune unnecessary hidden unit away from the network, we need to know the actual irrelevance of the unit. There are a lot of sensitivity methods that are used for determining the importance of hidden unit. Mozer and Smolensky [2] described a *Skeletonization* method estimating which unit is the least important according to the smallest effect on the training error and deleting it during training. This may remove the wrong unit since the error is based on the training patterns. We demonstrate in this paper that calculating the sensitivity from the training set may not reflect the actual irrelevance of the hidden unit. Optimal brain damage (*OBD*) [3] and its variant, optimal brain surgeon (*OBS*) [4], compute a saliency for each parameter, which can be either a unit or a weight, indicating the influence that small perturbations to the parameter have on approximation training error. Parameters with low saliency are removed. For computational simplicity, *OBD* assumes that the Hessian matrix is diagonal; in fact, however, Hessian for many problems are strongly non-diagonal, and this lead *OBD* to eliminate the wrong weights [4]. On the other hand, *OBS* is impractical for complex problems which require a large network. Murase et al. [5] measured the *goodness factors* of the hidden units in the trained network. Basically, the *goodness factor* is the average of the activation that fans out to the next layer. The unit which has the lowest value of the *goodness factor* is removed from the hidden layer. According to Engelbrecht [6], the work in [5] can suffer from the flaw when the unit's output is more frequently zero than one, that unit may be chosen as being unimportant, while this is not necessarily the case.

In this paper, we propose a novel method to determine the irrelevance of hidden unit by using the activation difference. This activation difference is the gap between the activation of output units when a particular hidden unit is present and when it is removed. We calculate the sensitivity from the validation set because the validation patterns can represent the unseen data better than the training patterns. We make assumption here that there are sufficient validation patterns and the patterns are randomly distributed over both training and validation data. The experimental results show the consistency of using the activation difference, making it suitable for determining the relevance of hidden unit.

The rest of this paper is organized as follows. In Section 2, we explain the notion of activation difference and some network notations used in this paper. In Section 3, we describe the experimental study, data sets used, and experimental results. In Section 4, we discuss the experimental verifications of using the validation set to calculate the sensitivity. Finally, in Section 5, the conclusions are given.

II. ACTIVATION DIFFERENCE (*AD*)

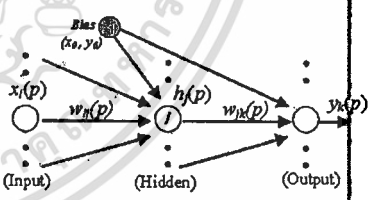


Fig. 1. Network notations.

Fig. 1 represents the network notations. We define the total synaptic input of the hidden unit j on pattern p, $X_j(p)$, the

output of the hidden unit j, $h_j(p)$, the total synaptic input of the output unit k on pattern p, $X_k(p)$, and the activation of the unit k, $y_k(p)$, as follows:

$$X_j(p) = \sum_i w_{ij}(p) \cdot x_i(p) \quad (1)$$

$$h_j(p) = \frac{1}{1 + e^{-X_j(p)}} \quad (2)$$

$$X_k(p) = \sum_j w_{jk}(p) \cdot h_j(p) \quad (3)$$

$$y_k(p) = \frac{1}{1 + e^{-X_k(p)}} \quad (4)$$

Once $X_k(p)$ has been calculated, we take the synaptic input of the hidden unit j out of $X_k(p)$ in order to represent the absence of the unit j. We call $X_k(p)$ after this process as $X_{k-j}(p)$, and the activation of the output unit k when the hidden unit j is removed as $y_{k-j}(p)$.

$$X_{k-j}(p) = X_k(p) - w_{jk}(p) \cdot h_j(p) \quad (5)$$

$$y_{k-j}(p) = \frac{1}{1 + e^{-X_{k-j}(p)}} \quad (6)$$

The activation difference is the gap between the target value of the output unit k, $O_k(p)$, and the activation of the unit k when the hidden unit j is absent, $y_{k-j}(p)$. The formal definition of the activation difference is given as the following:

$$AD_{k-j}(p) = \left| O_k(p) - y_{k-j}(p) \right| \quad (7)$$

In case of the network which has more than one output unit, the summation of activation differences, $AD_j(p)$, of all output units is required.

$$AD_j(p) = \sum_k AD_{k-j}(p) \quad (8)$$

Then we sum the activation difference on all validation patterns, resulting in AD_j. Therefore, the most irrelevant hidden unit is the unit which has the lowest activation difference (AD_j).

$$AD_j = \sum_p AD_j(p) \quad (9)$$

For a classification problem, it is wise to consider the number of misclassified validation patterns, when each hidden unit is missing, as the first priority. Let $NumErrPat_j$ be the number of misclassified validation patterns, when a hidden unit j is missing. Eq. 10 defines the revised AD_j for a classification problem. If hidden units are tied at a particular number of error patterns, we then use the average activation difference (the second term of Eq. 10) to break the tie.

$$AD_j = NumErrPat_j + \frac{\sum_p AD_j(p)}{P \times K} \quad (10)$$

where P and K are the number of validation patterns and the number of output units, respectively. In order to make $NumErrPat_j$ more important than AD_j, the sum of activation differences in Eq. 10 is averaged on P and K. As a result, the first term of Eq. 10 is a positive integer, while the second term is a positive floating point number, which is less than one.

III. Experimental Study

The experiments are performed on two standard real-world benchmarks. The data set is divided into three sets: 50% of the data set for training set, 25% of the data set for validation set, and 25% of the data set for test set. We use the standard error function, i.e., sum-squared error (*SSE*).

$$SSE = \frac{1}{2} \cdot \sum_p^P \sum_k^K (O_k(p) - y_k(p))^2 \quad (11)$$

where K and P are the number of output units and the number of training patterns, respectively. The output units for n classes are encoded into 1-of-n output representation. The output having the highest activation designates the class (winner-takes-all.)

A. *Data sets (originally from UCI repository)*

- *The Iris Data Set*

The data set has 3 classes; setosa, versicolor, and virginia. Each class has 50 instances and each refers to a type of plant. There are 4 continuous attributes for this problem.

- *The Wine Data Set*

These data are the results of a chemical analysis of wines grown in Italy. The analysis determined the quantities of 13 constituents (attributes) found in each of the three types of wines. There are 178 instances in this data set.

B. *Experimental results*

We train the neural network for each problem until the convergence condition is satisfied. Then we try every hidden unit removal and record the validation error rate right after the unit is pruned away. Table I reports validation error rate right after a particular hidden unit is removed in the first two pruning phases. Each cell in Table I shows the activation difference (upper value), which is calculated from the validation set, and the validation error rate (lower value) right

after the hidden unit is removed. Due to space limitations, only the first two pruning phases are reported.

TABLE I
PRUNING PHASES WHEN AD_1 IS CALCULATED FROM THE VALIDATION SET.

Phase	Iris	Wine
1	16.5431	3.6415
	0.4455	0.0818
	4.5411	7.2416
	0.1217	0.1637
	3.0359	7.2502
	0.0812	0.1637
	19.5392	14.7120
	0.5271	0.5204
	1.5431	7.5548
	0.0405	0.1363
	1.5435	**3.6395**
	0.0405	**0.0818**
2	16.5482	17.2012
	0.4460	0.5356
	19.5546	6.0391
	0.5271	0.1363
	19.5450	6.0476
	0.5271	0.1363
	7.5456	**4.8397**
	0.2026	**0.1091**
	3.0398	6.0414
	0.0810	0.1363

For a particular pruning phase, we choose the hidden unit which has the smallest activation difference as the unit to be pruned away. It is clear that the validation error rate, right after the unit is removed, is minimal, comparing with other unit removals. The **bold**-font number represents the unit which we select in each pruning phase. The pruned network is retrained when we get rid of the unit we chose. Note that the validation error rate before the unit is removed in each pruning phase is shown in Table II.

TABLE II
VALIDATION ERROR RATE BEFORE PRUNING IN EACH PHASE.

Phase	Iris	Wine
1	0.0612	0.0963
2	0.0612	0.0963

On the other hand, using the training set to calculate the activation difference, Table III shows the activation difference and validation error rate right after the hidden unit is removed. We can see the disadvantage of using training set to calculate the activation difference. The lowest activation difference may not reflect the lowest validation error rate. The situation can be shown in **bold** number. For example, at the second pruning phase of the iris problem, the fourth hidden unit has the lowest activation difference (6.0339) with the validation error rate of 0.0811. However, this validation error rate is not optimal, comparing to other hidden unit removals (in **bold**-underlined numbers), i.e., AD = 6.0466 with Error rate = 0.0405. This kind of event also happens to the first pruning phase of the wine problem.

TABLE III
PRUNING PHASES WHEN AD_1 IS CALCULATED FROM THE TRAINING SET.

Phase	Iris	Wine
1	19.5595	**10.8812**
	0.1216	**0.1333**
	6.0462	28.8936
	0.0405	0.1867
	3.0228	21.6884
	0.0405	0.1333
	19.5507	**15.6761**
	0.1217	**0.1066**
	4.5337	27.6982
	0.0811	0.1599
	30.1424	36.0889
	0.5676	0.5867
2	24.0601	28.8906
	0.2026	0.1599
	6.0466	21.6950
	0.0405	0.1333
	19.5505	36.0883
	0.1216	0.5866
	6.0339	27.6967
	0.0811	0.1599
	32.0402	**8.6754**
	0.5676	**0.1066**

IV. EXPERIMENTAL VERIFICATIONS

In order to see the impact of using the validation set to calculate the sensitivity on the other method, we perform the same experiment to the *Skeletonization* algorithm [2]. We modify the *Skeletonization* algorithm to computing the sensitivity (ρ) of each parameter by using the validation set, instead of the training set. Table IV shows the comparison results on the wine benchmark. We start both versions by using 6 initial hidden units and trains the networks to the optimal validation error. At the pruning phase, the original *Skeletonization* chooses the unit 1 and the modified version chooses the unit 3. The validation error rate produced by the modified version (0.0451) is better than that of the original version (0.0511.) This partially endorses our notion about using the validation set to calculate the sensitivity of the hidden unit. However, the most irrelevance unit is the hidden unit 2 since the network produces the lowest validation error rate (0.0322), when the unit is removed. This situation can be described in Fig. 2 for a particular validation pattern.

According to the original recipe of *Skeletonization* [2], the sensitivity of the unit i (ρ_i) is calculated from the first derivation of the linear error, which is defined as $\sum_p \sum_k \left| t_{pk} - o_{pk} \right|$. Note that t_{pk} and o_{pk} are the target value and the activation of the output unit k on the pattern p,

TABLE IV
COMPARISON RESULTS BETWEEN TWO VERSIONS OF SKELETONIZATION.

Phase	Original Skel.	Modified Skel.
1	**9.0008**	9.2501
	0.0511	0.0511
	10.6586	**9.3447**
	0.0322	**0.0322**
	9.1676	**8.9622**
	0.0451	**0.0451**
	11.3125	10.5231
	0.0519	0.0519
	10.5874	12.5334
	0.0776	0.0776
	11.5731	11.4569
	0.0519	0.0519
Testing error rate (after further training with 5 hidden units)	0.0285	0.0278

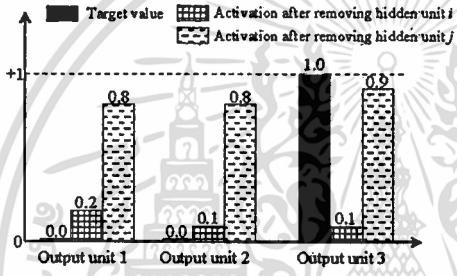


Fig. 2. The situation where the sensitivity overestimates the relevance of hidden unit on the wine problem

respectively. In Fig. 2, it is clear that $\rho_i < \rho_j$. Therefore, we pick the hidden unit i and prune it away. However, by using winner-takes-all classification, this pattern is misclassified because the pruned network generates output <1,0,0>, instead of <0,0,1>. On the other hands, if we choose to prune the hidden unit j, the pruned network correctly classifies this pattern since it generates <0,0,1>. Therefore, for a classification problem, we cannot rely on the error generated from the error function. We should consider the number of patterns which are correctly classified by the resulting network. As a result, we consider the number of misclassified patterns as the first priority (Eq. 10.)

Although we neither investigate all benchmarks nor test our assumption to all pruning algorithms, the experiments on a well-known technique with a real world data set can partially support our notion. It is worth to note here that calculating the sensitivity from the validation set will increase the computational complexity of the original algorithm. Therefore, an intriguing technique should be applied to reduce the computations for a practical usage.

V. CONCLUSIONS

We proposed a new approach of using the output activation difference, which is computed from the validation set, to determine the relevance of individual hidden units. The least relevant hidden unit is the unit that has the smallest activation difference. One of the future developments of the activation difference is to invent the pruning algorithm that employs the ability of the activation difference to determine the irrelevant hidden unit and produces the compact-sized neural network that is generalized.

REFERENCES

[1] Hornik, K., M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol.2, pp. 359-366, 1989.
[2] M.C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in Neural Information Processing Systems, vol.1, pp. 107-115, Morgan Kaufman, San Mateo, CA, 1989.
[3] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," Advances in Neural Information Processing Systems, vol.2, pp.598-605, Morgan Kaufman, San Mateo, CA, 1990.
[4] B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," Advances in Neural Information Processing Systems, vol. 5, pp.162-171, Morgan Kaufman, San Mateo, CA, 1993.
[5] K. Murase, Y. Matsunaga, and Y. Nakade, "A Back-Propagation Algorithm which Automatically Determines the Number of Association Units," Proc. IEEE Int. Conf. Neural Networks, vol. 1, pp. 783-788, 1991.
[6] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," IEEE Trans. Neural Net, vol. 12, no. 6, pp. 1386-1399, Nov. 2001.

World Scientific Publishing Co Pte Ltd

Postal & Correspondence: Farrer Road, P O Box 128, Singapore 912805
Office: 5 Toh Tuck Link, Singapore 596224 Tel: (65) 6466 5775 Fax: (65) 6467 7667
E-mail: wspc@wspc.com.sg
http://www.worldscientific.com/

31 October 2008

To

Dr. Kietikul Jearanaitanakij
Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang,
Chalongkrung Road, Ladkrabang,
Bangkok, 10520
Thailand

RE: ACCEPTED PAPER IN
JOURNAL OF CIRCUITS, SYSTEMS, AND COMPUTERS (JCSC)

This is to confirm that the following paper has been accepted for publication in Journal of Circuits, Systems, and Computers:

"Spartan Simplicity: A Pruning Algorithm for Neural Nets" by K. Jearanaitanakij and O. Pinngern.

Yours sincerely,

Yeow Hwa QUEK (Mr.)
Editor, JCSC
World Scientific Publishing Co Pte Ltd
E-mail: yhquek@wspc.com.sg
Fax: (65) 6467-7667
Tel: (65) 6466-5775

USA Office: World Scientific Publishing Co., Inc., 27 Warren Street, Suite 401-402, Hackensack, NJ 07601, USA. Tel: 1-201-487-9655 Fax: 1-201-487-9656 E-mail: sales@wspc.com
London Office: World Scientific Publishing (UK) Ltd., 57 Shelton Street, Covent Garden, London WC2H 9HE, UK. Tel: 44-020-7836-0888 Fax: 44-020-7836-2303 E-mail: sales@wspc.co.uk
Shanghai Office: Global Consultancy (Shanghai) Pte Ltd., 99, Huangpu Road, Room 2008, Shanghai Quad International Tower, Shanghai 200080, PR China. Tel: 86-21-6325-0932 Fax: 86-21-6325-0935
Hong Kong Office: World Scientific Publishing (HK) Co Ltd., P O Box 72982, Kowloon Central Post Office, HONG KONG. Tel: 852-3-771-8791 Fax: 852-3-771-8153 E-mail: wspc@pacific.net.hk
Taiwan Office: World Scientific Publishing Co Pte Ltd., 4F-5, No. 80, Sec 3, Hsin-Hsing S. Road, Taipei, TAIWAN. Tel: 886-2-2368-7365 Fax: 886-2-2368-0498 E-mail: wspc@ms13.hinet.net
India Office: World Scientific Publishing Co Pte Ltd., No 10, Sastri Med Road, T. Nagar, Chennai 600 017, INDIA. Fax: 91-44-2820-6496 Tel: 91-44-2829-7114 Email: sales-in@wspc.com

YAHOO! MAIL Classic

Re: Final paper and copyright transfer form Tuesday, December 4, 2007 8:19 PM

From: "Yeow-Hwa QUEK" <yhquek@wspc.com.sg>
To: "Kietikul Jearanaitanakij" <kietikul@yahoo.com>

Dear Dr. Jearanaitanakij,

Thank you for your recent email to me with the final files of the accepted paper.

There is no need to send the signed copyright form by regular mail.

This paper is tentatively scheduled for publication in JCSC Vol. 17, No. 4 (August 2008).

Best regards,
Yeow-Hwa Quek.

Kietikul Jearanaitanakij wrote:

> Dear Mr. Yeow-Hwa Quek,
>
> Please see the attachments for the final paper (an editable file and pdf according to MS-Word style) and the signed copyright transfer form. Would you like me to mail the hard copy of the signed copyright transfer form to your address as well? Thank you.
>
> Sincerely,
>
> Kietikul Jearanaitanakij
>
> --- Yeow-Hwa QUEK <yhquek@wspc.com.sg> wrote:
>
> > Date: Mon, 03 Dec 2007 17:59:15 +0800
> > From: Yeow-Hwa QUEK <yhquek@wspc.com.sg>
> > To: Kietikul Jearanaitanakij <kietikul@yahoo.com>
> > Subject: Re: Inquiry about the process after paper acceptance
> >
> > Dear Dr. Jearanaitanakij,
> >
> > Further to the recent email from Prof. Ahmadi,
> > please prepare your final
> > paper according to our styles and templates at
> >
> > *
>
> http://www.worldscinet.com/style_files/jcsc/123-readme_2e.shtml

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Journal of Circuits, Systems, and Computers
Vol. 17, No. 4 (2008) 1–28

World Scientific

SPARTAN SIMPLICITY: A PRUNING ALGORITHM FOR NEURAL NETS

KIETIKUL JEARANAITANAKIJ* and OUEN PINNGERN

Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok, 10520 Thailand
**kjetetik@kmitl.ac.th*

Received 26 April 2007
Revised 2 November 2007
Accepted 29 November 2007

Having more hidden units than necessary can produce a neural network that has a poor generalization. This paper proposes a new algorithm for pruning unnecessary hidden units away from the single-hidden layer feedforward neural networks, resulting in a Spartan[a] network. Our approach is simple and easy to implement, yet produces a very good result. The idea is to train the network until it begins to lose its generalization. Then the algorithm measures the sensitivity and automatically prunes away the most irrelevant unit. We define this sensitivity as the absolute difference between the desirable output and the output of the pruned network. Unlike other pruning methods, our algorithm is distinct in calculating the sensitivity from the validation set, instead of the training set, without increasing the asymptotic time complexity of the back-propagation algorithm. In addition, for a classification problem, we raise a point that the sensitivities of some well-known pruning algorithms may still underestimate the irrelevance of hidden unit even though the validation set is used in measuring the sensitivity. We resolve this problem by considering the number of misclassified patterns as the main concern. The Spartan simplicity algorithm is applied to three artificial and seven standard benchmarks. In most problems, the algorithm can produce a compact-sized network with high generalization ability in comparison with other pruning algorithms.

Keywords: Artificial neural network; pruning; classification; generalization; hidden unit.

1. Introduction

A back-propagation algorithm used in an artificial neural network is a statistical model whose parameters are estimated from a training set. If the network is trained with a sufficient number of training patterns and an appropriate activation function, the network can learn any function.[1] However, the generalization of the network is degraded if there are unnecessary hidden units.

[a]Spartan, a native of Sparta (ancient Greece), was simple and self-disciplined, yet a brave warrior.

There are various methods to prune away unnecessary hidden units. Mozer and Smolensky[3] described a Skeletonization method estimating the least important unit according to the gap between the error when the unit is removed and the error when it is left in place. This may not reflect true significance of hidden unit since the error is calculated from the training set which was seen during the training period. Optimal brain damage (OBD)[3] and its variant, optimal brain surgeon (OBS),[4] compute a saliency for each parameter, which can be either a unit or a weight, indicating the influence that small perturbations to the parameter have on the training error. For computational simplicity, the OBD assumes that the Hessian matrix is diagonal; in fact, however, Hessians for many problems are strongly non-diagonal. This leads the OBD to eliminate wrong weights.[4] On the other hand, the OBS is impractical for complex problems which require a large network. Murase *et al.*[5] measured the goodness factors of the hidden units in the trained network. The goodness factor is the average activation that fans out to the next layer. Shahjahan *et al.*[6] extended the work in Ref. 5 to develop a dynamic node decaying as a method for hidden unit pruning. According to Engelbrecht,[7] the works in Ref. 5 can suffer from the flaw when the unit's output is more frequently 0 than 1. That unit may be chosen as being unimportant, while this is not necessarily the case. Lauret *et al.*[8] recently proposed a node-pruning algorithm for a single-hidden layer neural network. The irrelevance of the hidden unit is determined by analyzing the Fourier decomposition of the output unit's variance. However, their results focus only on reducing the number of units and the pruning time with little concern about the generalization ability. Engelbrecht[7] proposed a new statistical pruning heuristic by expressing parameter relevancy as a function of the variance in sensitivity over all training patterns and using hypothesis tests to remove parameters for which the variance in sensitivity is approximately zero. However, the method in Ref. 7 does not provide the statistic test to stop pruning. The simple rule of thumb "stop pruning when generalization deteriorates too much" is used instead.

In this paper, we propose a sensitivity analysis pruning algorithm for the single-hidden layer feed-forward neural networks using stochastic back-propagation of *error*. Our goal is to prune away as many unnecessary hidden units as possible while maintaining high generalization ability. We calculate the sensitivity from the validation set, because the validation set can represent the unseen data more truly than the training patterns. In addition, for a classification problem, the sensitivities of some well-known pruning algorithms may still underestimate the irrelevance of hidden unit even though the validation set is used in measuring the sensitivity. We resolve this problem by considering the number of misclassified patterns as the main concern. The Spartan simplicity algorithm is tested on various standard benchmarks. The results show that the algorithm produces a compact-sized network with remarkable generalization ability.

The rest of this paper is organized as follows. In Sec. 2, we explain terms that are used throughout the paper. In Sec. 3, we explain in detail the Spartan simplicity algorithm. In Sec. 4, we describe the experimental study, data sets

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

used, experimental results, and comparisons with other pruning methods. In Sec. 5, we discuss the experimental verifications of our notions. Finally, in Sec. 6, the conclusions and possible future development are given.

2. Relevant Terms

Before explaining in detail the Spartan simplicity algorithm in the next section, let us define some related terms used in this algorithm.

2.1. *Generalization loss (GL)*

Prechelt[9] proposed the concept of generalization loss. The idea is to stop early the training in order to avoid overfitting of the network to a particular training set used. Generalization loss at epoch t is defined as follows:

$$E_{opt}(t) = \min_{t' \le t} E_{va}(t'), \tag{1}$$

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right). \tag{2}$$

Here, $E_{va}(t')$ is the validation error at epoch t', whereas $E_{opt}(t)$ is the lowest validation error obtained in epochs up to t. $GL(t)$ is the ratio between the current and minimum validation errors during epoch t. High generalization loss is one candidate reason to stop training. Prechelt suggested that the training should be early stopped when the generalization loss exceeds a certain threshold, which may vary from problem to problem, within the training strip of length γ epochs.

Generalization loss supports the notion of overtraining, which is described by Chauvin.[10] The overtraining is a frequently found situation in training neural networks. Figure 1 illustrates the overtraining situation along with the training strip of length γ epochs. In Sec. 4, we demonstrate how to derive a suitable value of γ.

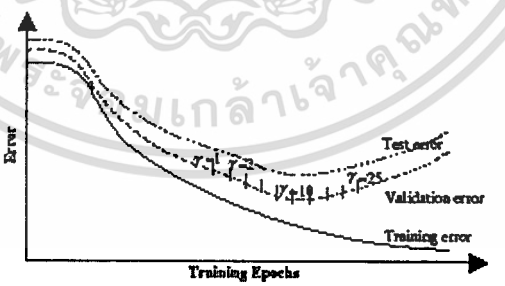


Fig. 1. The overtraining situation.

2.2. *Overpruning indicator (OI)*

The overpruning is the situation when we prune too many hidden units away from the network, resulting in the network without sufficient weights to learn patterns from the data set. We use the overpruning indicator to alert when the network is overpruned. The definition of overpruning indicator is given by

$$OI = \frac{(ValidErrRate_{\text{AfterPruning}} - ValidErrRate_{\text{BeforePruning}})}{ValidErrRate_{\text{BeforePruning}}} \times 10. \quad (3)$$

2.3. *Activation difference (AD)*

The sensitivity of the Spartan simplicity algorithm is the absolute difference between the desirable output and the output of the pruned network. We call this sensitivity as the activation difference, and it has to be calculated over all the patterns in the validation set. Thus, its computation is embedded into the module where the validation error is calculated in order to avoid passing the validation set twice.

Figure 2 represents the network notations. We define the total input of the hidden unit j on the validation pattern p, $X_j(p)$, the output of the hidden unit j, $h_j(p)$, the total input of the output unit k, $X_k(p)$, the activation function, φ, the activation of the unit k, $y_k(p)$, and the weight connects to the bias unit, $\theta(p)$, as follows:

$$X_j(p) = \sum_i w_{ij}(p) \cdot x_i(p) + \theta_j(p) \quad (4)$$

$$h_j(p) = \varphi(X_j(p)), \quad (5)$$

$$X_k(p) = \sum_j w_{jk}(p) \cdot h_j(p) + \theta_k(p), \quad (6)$$

$$y_k(p) = \varphi(X_k(p)). \quad (7)$$

Once $X_k(p)$ has been calculated, we subtract $X_k(p)$ with the input from the hidden unit j in order to represent the absence of the unit j. We call $X_k(p)$ after this process as $X_{k-j}(p)$, and the activation of the output unit k when the hidden

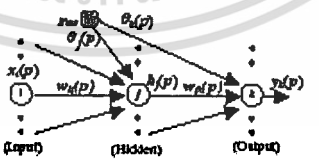


Fig. 2 Network notations.

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

unit j is removed as $y_{k-j}(p)$.

$$X_{k-j}(p) = X_k(p) - w_{jk}(p) \cdot h_j(p)\,, \tag{8}$$

$$y_{k-j}(p) = \varphi(X_{k-j}(p))\,. \tag{9}$$

The activation difference is the absolute difference between the desirable value of the output unit k, $O_k(p)$, and the activation of the unit k when the hidden unit j is removed, $y_{k-j}(p)$. The formal definition of the activation difference is given by

$$AD_{k-j}(p) = |O_k(p) - y_{k-j}(p)|\,. \tag{10}$$

In case of the network which has more than one output unit, the summation of activation differences, $AD_j(p)$, of all output units is required.

$$AD_j(p) = \sum_k AD_{k-j}(p)\,. \tag{11}$$

Then, we sum the activation difference over all the validation patterns, resulting in AD_j, and report the sensitivity to the pruning algorithm. Therefore, the most irrelevant hidden unit is the unit which has the smallest activation difference (AD_j):

$$AD_j = \sum_p AD_j(p)\,. \tag{12}$$

If AD_j is large, the unit j is relevant (by the above criterion.) However, it can also be the case that the smallest AD_j is large. If that is the case, the unit j still has to be removed. The reason is that we decide to prune the hidden unit at the point where the network begins to lose generalization. We should not further train the network, because the generalization will be worse (as shown in Fig. 1.) At best, if the unit j is irrelevant, the pruned network will receive a better generalization after further training by the rule of thumb. If the network has only a limited number of degrees of freedom (which are hidden units here), it will use them to adapt to the largest regularities in the data.[11] At worst, if the unit j is relevant to the network, the validation error will significantly increase and the overpruning will be detected.

It is interesting to consider a situation when an insignificant hidden unit has a large AD_j. This insignificant unit may survive during the early pruning phase. However, its AD_j will be smaller than AD_j of the significant unit and finally be deleted when the pruning process continues. AD_j of a significant unit is always larger than AD_j of an insignificant one since a back-propagation algorithm controls all weights connected to an insignificant hidden unit to be so small that the unit has no participation to the output layer. As a result, the most insignificant unit produces the smallest gap between the desirable value and the activation of the output unit when the insignificant unit is deleted.

We can compute AD_j in the validation error module without disturbing the module's complexity. In order to prove our claim, the analysis of the asymptotic computational complexity along with the pseudo-code of the modified validation error module is shown in Fig. 3. An additional code portion for calculating the sum of activation difference, AD_j, is in lines 8–14. In line 11, we avoid calculating

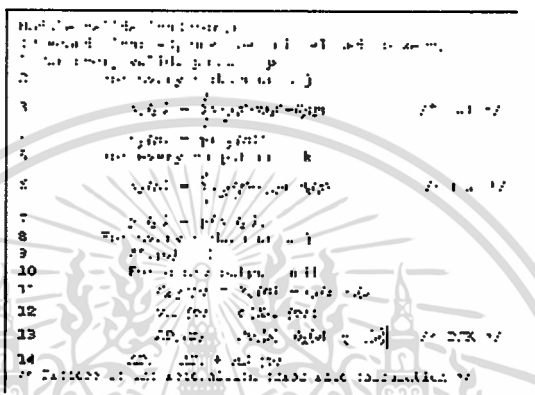


Fig. 3. A modified version of the ValidationError module.

$X_{k-j}(p)$ from the scratch by subtracting $X_k(p)$, which is already calculated in line 6, with the information from unit j. Let I be the number of input units, J be the number of hidden units, K be the number of output units, and P be the number of validation patterns. The asymptotic time complexities of both versions are

$$\begin{aligned} O(\text{Original module}) &= O(PJI + PKJ), \\ O(\text{Modified module}) &= O(PJI + PKJ + PJK) \\ &= O(PJI + 2PKJ) \\ &= O(PJI + PKJ). \end{aligned}$$

Therefore, the embedded code does not increase the asymptotic computational complexity of the original module. In addition, our technique spends fewer steps than a straightforward approach to calculate AD_j. By performing a forward computation when each hidden unit j is removed, a plain neural network needs approximately $O(PJ(JI + KJ))$ in order to compute AD_j.

Besides the single hidden layer network, our approach can be generalized to prune units in the network with multiple hidden layers, and to prune weights. In the network with multiple hidden layers, AD_j can be computed in the same manner as in the single hidden layer network. However, there must be some additional memory space allocated for recording X_k of every output unit k and X_j of every hidden unit j. These values avoid calculating X_{k-j} from the scratch. In weight pruning, we can still use the same process along with the following two conditions to calculate AD_j: first, if $w_{ij}(p)$ is removed, both $h_j(p)$ and every $y_k(p)$ that connect to $h_j(p)$ must be recomputed; second if $w_{jk}(p)$ is deleted, only $y_k(p)$ connected with $w_{jk}(p)$ needs to be recalculated.

3. Spartan Simplicity Algorithm

The algorithm is described in Fig. 4. We train the initial network for a set of γ epochs until the network begins to lose its generalization, i.e., the generalization loss (GL) exceeds the threshold α. Then we move the network back to the optimal state, where the validation error is minimal, within the previous γ epochs. Before the pruning process begins, the algorithm saves current weights and hidden units for future restoration in the overpruning situation. Then, the algorithm prunes away the hidden unit which has the smallest activation difference (AD_j), and trains the network for another set of γ epochs. If neither the network loses its generalization nor the overpruning occurs, the algorithm will continue another set of training. Otherwise, it moves back to the optimal state within the previous set of training. Finally, if the overpruning occurs, i.e., the overpruning indicator (OI) exceeds the threshold β, the algorithm will restore the hidden unit and all its corresponding weights before returning to a Spartan network. Note that choosing a suitable value of γ, α, and β for each problem is explained in Sec. 4.

For a non-noised problem, instead of checking the generalization loss, we stop the training when there is no progress, e.g., the validation error decreases less than 0.001. For a classification problem, it is worth considering the number of misclassified validation patterns, when each hidden unit is removed, as the first priority (the reason is discussed in Sec. 5.) Let $NumErrPat_j$ be the number of misclassified validation patterns, when a hidden unit j is removed. The revised AD_j for a classification problem is defined by Eq. (13). The first term on the right side of Eq. (13) is a positive integer, while the last term is the average AD_j, which

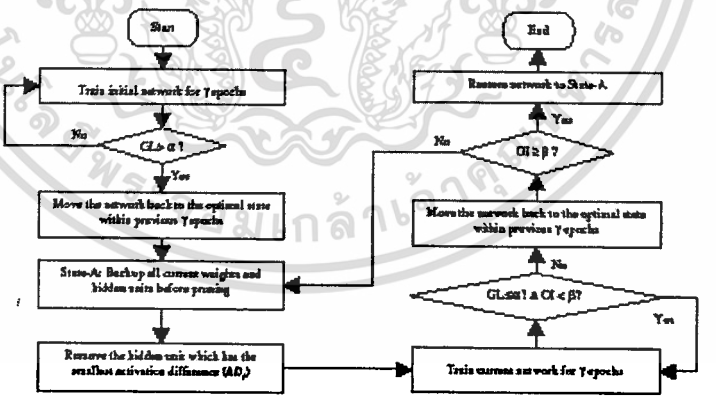


Fig. 4. Spartan simplicity algorithm.

has a value between 0.0 and +1.0.

$$AD_j = NumErrPat_j + \frac{\sum_p AD_j(p)}{P \times K}, \tag{13}$$

where P and K are the number of validation patterns and the number of output units, respectively.

4. Experimental Study

The experiments are performed on both artificial and standard real-world benchmarks. The data set is divided into three sets: 50% of the data set for training set, 25% of the data set for validation set, and 25% of the data set for test set. We use the standard error function, i.e., sum-squared error (SSE), for most problems, except for F1 and F2. In order to conform to the experiments in Ref. 7, we use mean-squared error (MSE) for F1 and F2. The SSE and MSE[12] are defined as follows:

$$SSE = \frac{1}{2} \cdot \sum_p^P \sum_k^K (O_k(p) - y_k(p))^2, \tag{14}$$

$$MSE = \frac{1}{2P} \cdot \sum_p^P \sum_k^K (O_k(p) - y_k(p))^2. \tag{15}$$

We use the standard sigmoid function in all experiments. For a classification problem, the output units for n classes are encoded into 1-of-n output representation. The output having the highest activation designates the class (winner-takes-all.) All experiments are simulated on Pentium 4, 2.66 GHz.

4.1. *Artificial benchmarks*

Three well-known artificial benchmarks are chosen to evaluate the correctness of the proposed algorithm. Parabolic function (F1), sine function (F2), and artificial classification (AC) are originally from Ref. 7. The minimum numbers of hidden units for these benchmarks are known in advance[7] — two units for F1, five units for F2, and three units for AC.

- Parabolic function (F1 — illustrated in Fig. 5): $f(x) = x^2$, (16)
  where $x \sim U(-1, 1)$. All the output values are in the range [0, 1].
- Sine function (F2 — illustrated in Fig. 6): $f(x) = \sin(2\pi x)e^{-x} + \zeta$, (17)
  where $x \in U(-1, 1)$ and $\zeta \sim N(0, 0.1)$. Output values are scaled to the range [0, 1].
- Artificial classification (AC — illustrated in Fig. 7):

$$\text{Class} = \begin{cases} 1; & \text{if } (z_1 \geq 0.7) \text{ or } ((z_1 \leq 0.3) \text{ and } (z_2 \geq -0.2 - z_1)), \\ 0; & \text{otherwise}. \end{cases}$$

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

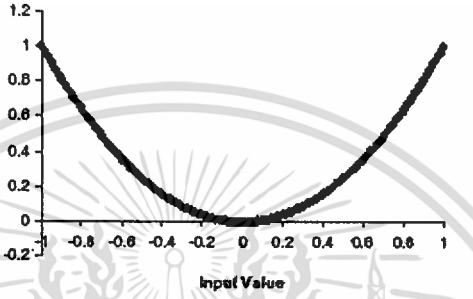


Fig. 5. Function F1.

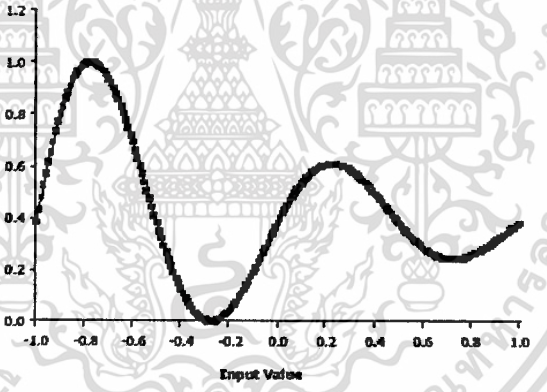


Fig. 6. Function F2.

4.2. *Real-world benchmarks*

All real-world data sets are obtained from the UCI benchmark repository. These problems represent the challenging benchmarks in the field of machine learning because of their relatively high noise level. Data sets and their brief details are given in Table 1. In order to accelerate the back-propagation learning process, input attributes should be pre-processed by a simple normalization so that its mean value, averaged over the entire set, is close to zero.[13] Thus, the input attributes of some benchmarks, e.g., diabetes, heart disease, wine, and credit card, are rescaled to between 0.0 and 1.0 by a linear function. In order to comply with the experimental

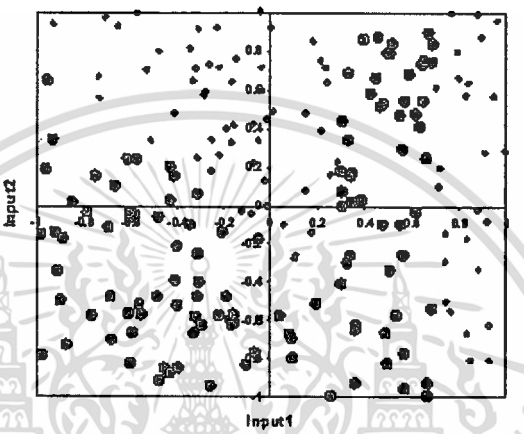


Fig. 7. Artificial classification.

results reported in Refs. 6, 7 and 14, the size of each data set used in our experiment is partitioned as given in Table 1. Most of the data sets are randomised before partitioning, except for Wisconsin breast cancer, diabetes, and heart disease problems, which are already mingled in the original data sets.

4.3. *Parameter choosing*

Table 1 also shows the values of two parameters used in the algorithm, i.e., the *GD*'s threshold (α) and the initial numbers of hidden units. These parameters are investigated under preliminary runs of the Spartan simplicity algorithm in order to

Table 1. Data sets used to test the Spartan simplicity algorithm.

Benchmark	# attributes	# classes	# Train patterns	# Valid. patterns	# Test patterns	α	Initial # of hidden units
P1	1	approximation	80	40	40	n/a	5
P2	1	approximation	100	50	50	n/a	10
AC	2	2	100	50	50	n/a	10
Iris	4	3	75	37	38	10	6–11
Breast Cancer	9	2	349	175	175	6	6–8
Diabetes	8	2	384	192	192	2	6–8
Heart	13	2	151	76	76	1	6–8
Wine	13	3	89	44	45	10	6–8
Credit card	14	2	346	172	172	2	12–14
Isolet	617	26	3899	1949	1949	120	78

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Table 2. Choosing appropriate α for Breast Cancer (a) and Credit card (b).

α	Initial # of hidden units	Final # of hidden units (a)	Test err. rate	α	Initial # of hidden units	Final # of hidden units (b)	Test err. rate
0.0	12	3	0.0500	0.0	14	1	0.1538
	10	2	0.0364		12	1	0.0990
	8	1	0.0313		10	2	0.1111
	6	1	0.0181		8	2	0.1389
	4	2	0.0844		6	2	0.1311
2.0	12	2	0.0513	[illegible]			
	10	2	0.0844				
	8	1	0.0263		10	1	0.1026
	6	1	0.0190		8	1	0.1261
	4	2	0.0617		6	2	0.1200
[illegible]	12	2	0.0459	6.0	14	1	0.0990
	10	1	0.0247		12	1	0.0990
	[illegible]				10	2	0.1111
	[illegible]				8	2	0.1261
	4	2	0.0574		6	2	0.1111
12.0	12	4	0.0539	12.0	14	1	0.0990
	10	1	0.0382		12	1	0.0990
	8	1	0.0239		10	2	0.1111
	6	1	0.0402		8	2	0.1437
	4	2	0.0574		6	2	0.1111

find the most appropriate value. For example, finding both parameters for breast cancer and credit card problems are shown in Tables 2(a) and 2(b), respectively. We consider the final number of hidden units and the test error rate while allowing α and the initial number of hidden units to vary. A suitable value of α for each problem should produce a final network with a small number of hidden units and a low test error rate. Therefore, we select $\alpha = 6$ and $\alpha = 2$ for breast cancer and credit card problems, respectively. Similarly, the most appropriate range of initial number of hidden units should allow the network to have a small number of final hidden units and a low test error rate. As a result, the initial number of hidden units is between 6–8 and 12–14 for breast cancer and credit card problems, respectively. Based on similar considerations, we can select the initial number of hidden units for other benchmarks. For benchmarks F1, F2, and AC, the initial numbers of hidden units are the same as those used by the works in Ref. 7. The network never loses its generalization on these three artificial benchmarks since they are non-noised problems. Thus, we stop training when the validation error decreases less than 0.001.

Another parameter that needs an explanation is the *OI*'s threshold (β). We run the Spartan simplicity algorithm on each benchmark and measure the threshold β when hidden units are overpruned by using Eq. (3) If the network is overpruned, its validation error rate will significantly increase. Even if the network is further trained, there will hardly be any decrease on the validation error rate. We use this

Table 3. Finding the OI's threshold (β).

	F1	F2	AC	Iris	Cancer	Diabetes	Heart	Wine	Credit	Isolet
OI's threshold (β)	61.54	48.0	51.72	165.0	94.44	30.0	28.13	170	38.75	24.0
Std. Dev.	0.25	0.18	0.09	1.21	0.51	0.10	0.00	1.78	0.34	0.20

threshold to alert the overpruning situation instead of waiting for the decreasing on the validation error rate. Some preliminary experiments as in Table 3 should be run before choosing a suitable threshold for each problem. Table 3 gives an average threshold β and its variance for each data set over 10 preliminary runs of the Spartan simplicity algorithm.

As indicated in Secs. 2 and 3, we monitor the generalization loss within γ epochs to stop the training early in order to avoid overtraining. Here, we demonstrate how we derive γ. We train the initial network with a standard back-propagation algorithm and monitor the variation of the validation error. In Fig. 8, the monitoring window is randomly placed on the validation error curve. There are two essential factors that we consider on selecting a suitable monitoring period. These factors are the time needed for detecting the generalization loss and the number of mispredictions. A small-sized window spends short time to predict the generalization loss but may have a large number of wrong predictions. Conversely, a large-sized window spends longer time to predict the generalization loss but has a smaller number of wrong predictions. We measure the number of wrong predictions among six different values of monitoring period in both non-overfitting and overfitting zones. A misprediction in the non-overfitting zone occurs when we have too small a monitoring period, and it cannot suppress the trivial ripples along the error curve. Similarly, a wrong prediction in the overfitting zone occurs when the error difference between the left and right rims of a monitoring window does not reach a certain threshold. The results of 100 random samples are recorded, and the comparison on different sizes of monitoring period is given in Table 4. Notice that the numbers of wrong predictions during the monitoring period 25–35 epochs are not different.

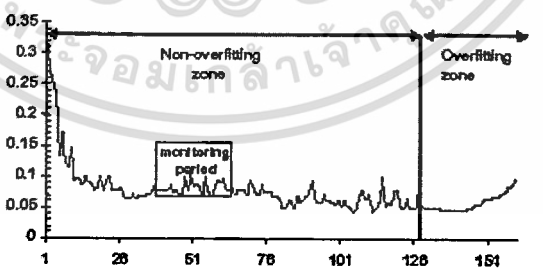


Fig. 8. Sliding windows moving along validation error.

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Table 4. The number of wrong predictions among six different monitoring periods.

	The number of wrong predictions						
Monitoring period (epochs)	Iris	Cancer	Diabetes	Heart	Wine	Credit	Isolet
5	4	6	7	7	4	5	15
15	2	4	5	6	3	4	9
20	1	2	2	2	3	2	4
25	0	0	1	1	0	1	1
30	0	0	1	1	0	1	1
35	0	0	1	1	0	1	1

Therefore, the monitoring period of 25 epochs should give a reasonable trade off between the time needed for determining the generalization loss and the number of wrong predictions.

4.4. *Experimental results*

Each experimental result is an average of 30 runs on different weight initializations and initial number of hidden units. These runs are uniformly distributed over the learning rate range of 0.1–0.5. In order to avoid the effect of a sophisticated network initialization, which may mislead the experimental results, we simply initialize all weights to be within the range of -0.1 and $+0.1$. The produced architecture and classification errors on every problem are shown in Table 5.

For one example, let us consider a sample run of the proposed algorithm on the diabetes problem. Figure 9 illustrates the number of hidden units, the generalization loss, the activation difference of each hidden unit, the error rates of training and validation sets, and the overpruning indicator. The algorithm spends 25 epochs on training the initial network, which has six hidden units, until the network loses its generalization at epoch 25 as the generalization loss rises up to 2.20 and exceeds the GL's threshold ($\alpha = 2$ from Table 1). At epoch 26, the algorithm decides to prune away unit 2 since it has the smallest AD_j. Then, the training continues until the network loses its generalization again at epochs 75, 100, 125, and 150, where the algorithm decides to prune units 5, 3, 4, and 1, respectively. It is interesting that unit 1 has smaller AD_j than unit 4 during the first hundred epochs, but it can live longer than unit 4. Therefore, the variation of AD_j indicates the competition, which occurs throughout the pruning process, among the hidden units. At epoch 175, overpruning occurs since the OI immediately reaches 30, which is the OI's threshold ($\beta = 30$ from Table 3). Therefore, the algorithm restores unit 1 and all its corresponding weights, and then returns a Spartan network with two hidden units.

4.5. *Comparisons*

The implementations of the OBD, the OBS, Skeletonization, and Magnitude-based pruning (MBP)[15] provided in the Stuttgart NN simulator (SNNS) are used in generating results for comparisons. For other methods not provided in the SNNS, we

Table 5. Results produced by the Spartan simplicity algorithm on various problems.

Problem		Final # of hidden units	Training set		Validation set		Test set	
			Error	Err. rate	Error	Err. rate	Error	Err. rate
F1	Mean	2	0.0211	—	0.0255	—	0.0318	—
	SD	0	0.0106	—	0.0176	—	0.0163	—
F2	Mean	5	0.0230	—	0.0248	—	0.0285	—
	SD	0	0.0018	—	0.0007	—	0.0019	—
AC	Mean	3	0.0254	0.0286	0.0272	0.0286	0.0281	0.0300
	SD	0	0.0240	0.0251	0.0639	0.0642	0.0008	0.0005
Iris	Mean	2.25	0.0110	0.0126	0.0138	0.0145	0.0142	0.0146
	SD	1.09	0.0067	0.0098	0.0027	0.0056	0.0020	0.0021
Cancer	Mean	2.43	0.0113	0.0100	0.0160	0.0183	0.0180	0.0200
	SD	0.23	0.0026	0.0048	0.0053	0.0043	0.0077	0.0069
Diabetes	Mean	2.16	0.1823	0.1898	0.1832	0.1935	0.1957	0.2144
	SD	0.98	0.0045	0.0051	0.0230	0.0030	0.1516	0.0061
Heart	Mean	2.27	0.0932	0.1148	0.0979	0.1534	0.1081	0.1751
	SD	0.21	0.0133	0.0098	0.0051	0.0107	0.0019	0.0054
Wine	Mean	4.53	0.0016	0.0017	0.0044	0.0050	0.0057	0.0058
	SD	0.96	0.0061	0.0062	0.0055	0.0021	0.0030	0.0039
Credit	Mean	3.25	0.1026	0.1113	0.1061	0.1115	0.1005	0.1119
	SD	1.24	0.0030	0.0041	0.0932	0.0126	0.0310	0.0200
Iris1st	Mean	20.28	0.0115	0.0101	0.0400	0.0507	0.0500	0.0510
	SD	0.74	0.0058	0.0087	0.0070	0.0115	0.0054	0.0059

take results from the original works by setting our experiment under the same conditions (the dynamic node decaying method (DNDM) from Ref. 6, the variance nullity pruning (VNP) from Ref. 7, and EPNet from Ref. 14.) In Table 6, the Spartan net achieves the minimum number of hidden units with better generalization than the VNP. However, we do not compare the Spartan net with the VNP on other data sets used in Ref. 7 since the VNP prunes away both input and hidden units in the networks of those benchmarks. Notice that difference to our approach is that the VNP initializes weights each time a hidden unit is pruned away. The network must be retrained from the scratch to meet a criterion. This may lead to a local minimum if the weight initialization is not properly executed.[12]

Since our approach starts pruning at the point when generalization deteriorates and continues training with the current weights, one concern is that the network may be in a local minimum. In fact, if a hidden unit is deleted, as well as its corresponding weights, the pruned network will be stirred around its error surface giving a possibility for the network to get out of this local minimum. In addition, we use the current weights because the deleted weights are redistributed over the remaining weights, resulting in the reduction of the retraining time.

Table 7 shows the comparison on the iris problem. The Spartan simplicity algorithm has the lowest values on both the number of hidden units and the test error rate. Although the DNDM averages the results from mutual exchange of the training

1st Reading

September 4, 2008 18:42 WSPG/123-JCSC 00451

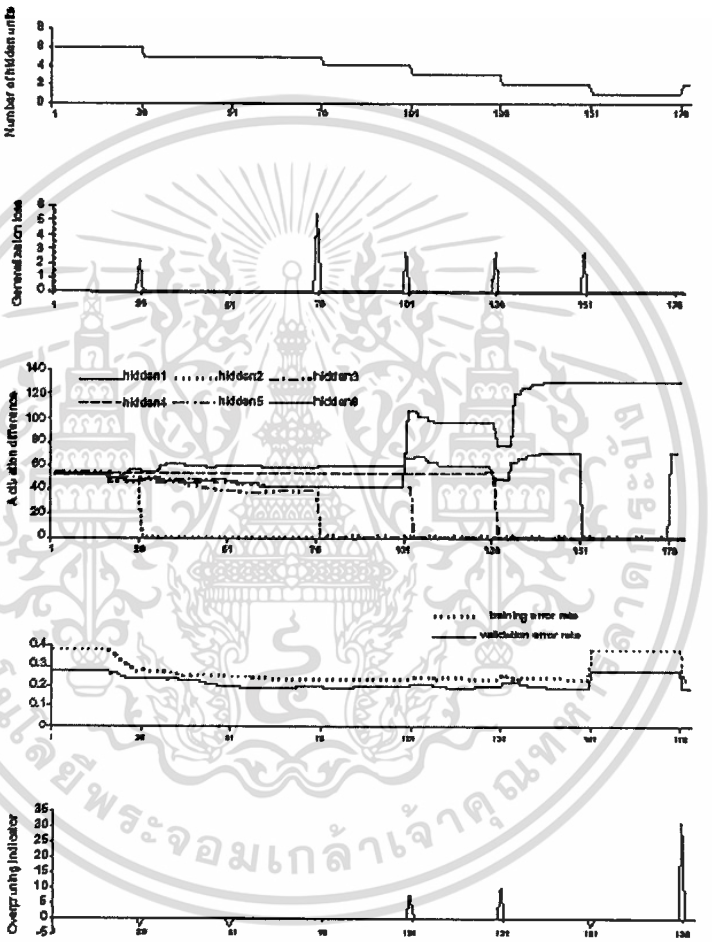


Fig. 9. Learning process for the diabetes problem.

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Table 6. Comparison between the Spartan simplicity algorithm and the VNP.

	Final architecture		Testing MSE		Testing accuracy	
	Spartan	VNP	Spartan	VNP	Spartan	VNP
F1	1-2-1	1-2-1	0.0318	0.0362	n/a	n/a
F2	1-5-1	1-5-1	0.0255	0.0299	n/a	n/a
AC	2-3-1	2-3-1	n/a	n/a	97%	95%

Table 7. Comparison between the Spartan simplicity algorithm and others on the iris problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization	DNDM
# Hidden units	2.25	4	4	4	3.5	2.67
Test error rate	0.0146	0.02	0.02	0.02	0.013	0.0159

data, validation data, and test data, its average result is not yet as good as the Spartan simplicity algorithm.

Table 8 illustrates one sample of the final network achieved by the Spartan simplicity algorithm on the iris problem. This is a compact-sized network with two hidden units and a relatively low test error rate. If the number of 38 test patterns (from Table 1) of the iris data set is passed through the Spartan net, the 0.0146 test error rate will be observed. Table 9 shows the comparison on the breast cancer problem. The number of hidden units of the Spartan net is much lower than that of others, but close to that of the DNDM.

One of the most difficult problems in machine learning is the diabetes problem due to its relatively small size and high noise level. The Spartan simplicity algorithm yet produces the smallest number of hidden units with the test error rate close to the minimum test error rate produced by the MBP, as shown in Table 10. For each of other two larger benchmarks (heart disease and wine problems), the proposed

Table 8. Spartan net for the iris problem.

Inputs	I_1	I_2	I_3	I_4	Bias	Output1	Output2	Output3
Hidden1	2.3809	2.3005	−3.6304	−2.8563	1.4823	−4.3191	2.3457	2.2092
Hidden2	−0.3017	−1.6731	2.5088	1.2970	−0.3560	1.8007	2.4811	−4.2425
Bias	—	—	—	—	—	0.3071	−3.8836	−0.0124

Table 9. Comparison between the Spartan simplicity algorithm and others on the breast cancer problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization	DNDM
# Hidden units	2.43	8	7	7	6	2.33
Test error rate	0.020	0.075	0.100	0.058	0.065	0.018

Table 10. Comparison between the Spartan simplicity algorithm and others on the diabetes problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization	DNDM	EPNet
# Hidden units	2.16	3.6	3.7	4.5	3.5	2.73	3.4
Test error rate	0.2144	0.1962	0.1988	0.1910	0.210	0.2234	0.224

Table 11. Comparison between the Spartan simplicity algorithm and others on the heart disease problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization	DNDM	EPNet
# Hidden units	2.27	4.5	4.5	5.2	3.2	1.83	4.1
Test error rate	0.1751	0.1578	0.1578	0.1478	0.1631	0.166	0.168

Table 12. Comparison between the Spartan simplicity algorithm and others on the wine problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization
# Hidden units	4.53	5	5	7	5
Test error rate	0.0088	0.028	0.028	0.014	0.029

method again produces a compact network with a high generalization, as shown in Tables 11 and 12.

Another difficult problem is the credit card assessment. The comparison in Table 13 shows that the Spartan net achieves the smallest number of hidden units, while the test error rate is still competitive to others.

Table 14 shows the comparison on the isolet problem. The data set contains the voices of 150 people saying the name of each letter of the alphabet twice. It is a very good domain for testing scalability of an algorithm due to its relatively large number of attributes and output classes. Our algorithm reduces the hidden units from 78 to 20.28, while the test error rate is still competitive to others.

Table 13. Comparison between the Spartan simplicity algorithm and others on the credit card problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization	EPNet
# Hidden units	3.25	4.9	4.85	6.8	4.9	4.83
Test error rate	0.1119	0.113	0.116	0.09	0.117	0.115

Table 14. Comparison between the Spartan simplicity algorithm and others on the isolet problem.

Algorithm	Spartan	OBD	OBS	MBP	Skeletonization
# Hidden units	20.28	30.55	30.40	35.5	32.5
Test error rate	0.0510	0.0527	0.0507	0.0555	0.0510

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

In order to get a feel for the potential improvement realizable of our technique, we compare the Spartan simplicity algorithm with the intuitive constructive technique (ICT.) The data sets and conditions are the same as those used in our experiments. We do not prune but start with a minimal structure with a none-hidden unit. The network is trained by a standard back-propagation algorithm. We use a 1-of-n representation with the winner-takes-all to designate the class and a sliding window on the validation error to stop training early. If we do not observe a new minimum on the validation error within 25 epochs, we will stop the training and reset the weights back to those at the epoch where we see the minimum validation error. Then, we add another hidden unit and continue training in the same manner as before. The process terminates when the new minimum validation error is larger than the minimum validation error of the previous architecture. In addition, for each structure, we experiment with various settings for the initial weights and learning rate parameters in order to find the best result. Then we choose the best network on the validation set from among 10 different structures.

Table 15 illustrates the comparison between the average results from the Spartan simplicity algorithm and the best results from the ICT. The number of hidden units and the test error rate are denoted by #HD and Err, respectively. It is obvious that our method outperforms the best result of the ICT on every benchmark.

5. Discussions

Based on the fact that a validation set represents an unseen data more truly than the training patterns, let us show the experimental verifications of the improvement by using the validation set in pruning decision. In addition, it is worth considering here that the sensitivities of some well-known pruning algorithms may still underestimate the irrelevance of hidden unit even though the validation set is used in measuring the sensitivity.

Let us first illustrate the comparative results of the Spartan simplicity algorithm using different data sets to calculate the sensitivity. Table 16 reports the results of the first pruning phase when using the validation set to calculate the sensitivity, i.e., the AD_j. Each cell in Table 16 shows the AD_j (upper value) and the validation error rate (lower value) immediately after the hidden unit is removed. The algorithm chooses the hidden unit which has the smallest AD_j as the unit to be pruned away. The bold-font number represents the unit to be removed. Notice that the validation error rate is minimal when compared with other unit removals.

On the other hand, Table 17 reports the results when using the training set to calculate the AD_j under the same initial conditions as experimented in Table 16. There are two disadvantages of using the training set to calculate the AD_j. First, during the early phase of pruning, the network may be too fit to the training set that the AD_j approaches zero (as shown in the *italic* number for F2, AC, and Diabetes.) This is a vulnerable situation because the validation error rates are different even though all the AD_js are zero. At best, the algorithm can randomly choose any

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Table 15. Comparison between the Spartan simplicity algorithm and the intuitive constructive technique.

	AC		Iris		Cancer		Diabetes		Heart		Wine		Credit	
	#HID	Err	#HID	Err	#HID	Err	#HID	Err	#HID	Err	#HID	Err	#HID	Err
Spartan	3	0.000	2.25	0.0145	2.43	0.020	2.16	0.2144	2.27	0.1751	4.53	0.0068	3.25	0.1119
ICT	4	0.000	5	0.030	8	0.100	6	0.3210	4	0.2110	7	0.0220	5	0.1220

Table 16. AD_2 and valid. Error rate immediately after pruning each hidden unit when using the validation set.

	F1		F2		AC		Iris		Cancer		Diabetes		Heart		Wine		Credit	
Hiddn1	4.849	14.149	7.233	11.028	19.0549	47.1713	15.0441	30.9345	18.1131									
	0.0027	0.0776	0.0400	0.2573	0.0686	0.2448	0.1974	0.0682	0.1103									
Hiddn2	3.9234	2.8180	23.119	1.0287	18.0512	47.1480	20.0434	6.0347	18.1026									
	0.0023	0.0037	0.1400	0.0370	0.1029	0.2448	0.2629	0.1364	0.1160									
Hiddn3	6.2165	12.788	20.4321	2.023	10.0672	52.1574	20.0517	6.0418	18.1101									
	0.0302	0.0310	0.0400	0.0541	0.0571	0.2604	0.2632	0.1364	0.1169									
Hiddn4	4.9793	2.6525	6.7686	13.026	11.0599	53.1964	15.0440	22.0409	19.1076									
	0.0165	0.0025	0.0000	0.3514	0.0629	0.2760	0.1974	0.1000	0.1934									
Hiddn5	25.370	18.324	43.0466	3.0274	11.0516	47.1402	18.0513	8.0365	39.1076									
	0.2421	0.1315	0.4000	0.0611	0.0629	0.2448	0.2368	0.1136	0.2539									
Hiddn6	1.4119	2.8654	19.2218	1.0290	19.0543	53.1765	17.0475	2.0320	18.1130									
	0.0099	0.0037	0.1800	0.0370	0.0686	0.2865	0.2337	0.0682	0.1169									
	..... next pruning phase																	

Table 17. AD_1 and valid. Error rate immediately after pruning each hidden unit when using the training set.

	F1	F2	AO	Iris	Chess	Diabetes	Heart	Wine	Classic
Hidden1	10.090	0.0000	0.0000	13.0000	30.0000	0.0000	18.5185	0.0000	46.3457
	0.0483	0.0000	0.0000	0.0011	0.0000	0.2441	0.1074	0.1111	0.1160
Hidden2	30.613	0.0000	0.0000	4.0000	21.1852	0.0000	16.6044	34.0720	60.0000
	0.0439	0.0074	0.0000	0.0370	0.1259	0.2441	0.3333	0.1556	0.1160
Hidden3	31.370	0.0000	0.0000	3.0182	18.71111	0.0000	15.0000	18.0000	60.3438
	0.0457	0.0441	0.0000	0.0370	0.0371	0.3044	0.0532	0.1111	0.1160
Hidden4	28.450	0.0000	0.0000	13.0335	15.0000	0.0000	18.0004	30.0741	57.3419
	0.0448	0.0000	0.0000	0.0011	0.0000	0.2760	0.1074	0.0000	0.1534
Hidden5	28.114	0.0000	0.0000	3.0225	15.0371	0.0000	21.0667	24.0019	63.3853
	0.0503	0.0441	0.0000	0.0441	0.0000	0.2441	0.2308	0.1852	0.3563
Hidden6	28.4700	0.0000	0.0000	37.0407	30.1075	0.0000	20.0772	12.0000	61.3537
	0.0441	0.0445	0.0000	0.0704	0.0000	0.2441	0.2297	0.0000	0.1560

..... best pruning phase .....

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

hidden unit, causing a possibility of the worst validation error rate. Second, the smallest AD_j may not reflect the lowest validation error rate. The situation can be shown in bold number. For example, in the breast cancer problem, the fourth hidden unit has the smallest AD_j (13.0997) with the validation error rate of 0.0629. However, this validation error rate is not the lowest value, comparing with the third hidden unit which has the error rate of 0.0571. Therefore, using the validation set to calculate the sensitivity in our approach is more accurate than using the training set.

In order to see the impact of using the validation set to calculate the sensitivity upon other classical methods, we perform the same experiment to the OBD, the OBS, and Skeletonization algorithms. Here, we modify these pruning algorithms in the SNNS to use the validation set instead of the training set in calculating the sensitivity. The experiments are run under the same conditions as explained in Sec. 4.4. The comparison between the original and the modified algorithms are given in Figs. 10–15. We choose iris, credit card, and isolet to represent small, medium, and large problems, respectively. In every case, using the validation set obviously results in a lower number of hidden units with fairly good generalization. Notice that the test error rates on iris and credit card problems slightly deteriorate; however, they are still comparatively competitive. Nevertheless, these improvements are arrived at by increasing the absolute computational time by the size of the validation set.

Although the validation set is used in other pruning techniques, it is interesting that the Spartan simplicity algorithm still achieves the performance better than other algorithms do. As shown in Table 18, the Spartan simplicity algorithm achieves the smallest average number of hidden units and the lowest average test error rate in every benchmark. In addition, our algorithm spends the least average

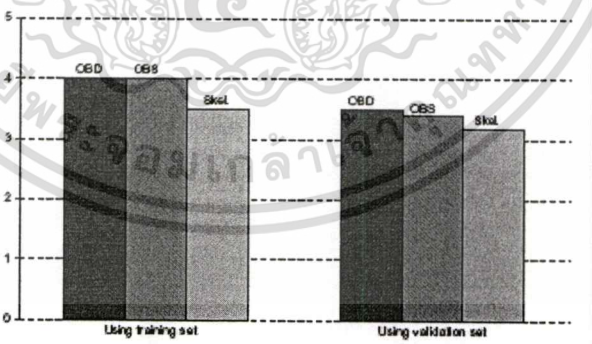


Fig. 10. The number of hidden units for iris.

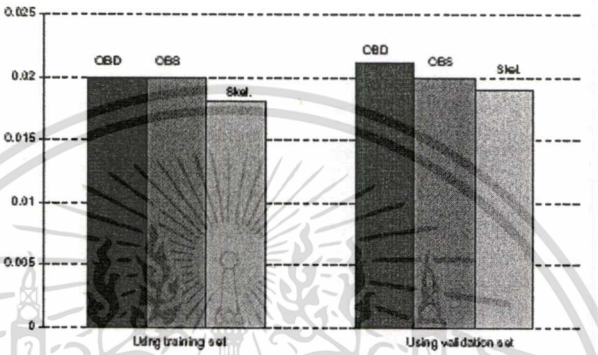


Fig. 11. The test error rate for iris.

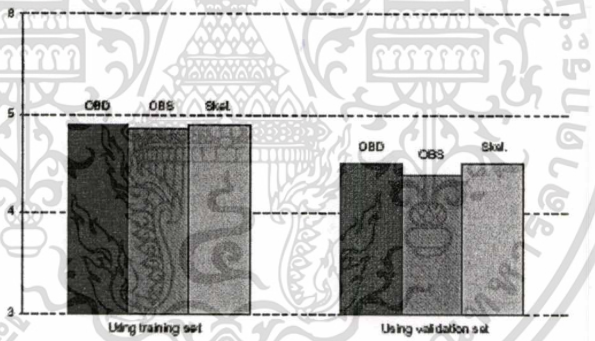


Fig. 12. The number of hidden units for credit card.

CPU time due to its simplicity, while the OBS consumes the largest time due to its full Hessian matrix calculations.

Before explaining further why the Spartan simplicity algorithm outperforms other pruning algorithms even if the validation set is used in measuring the sensitivity, let us review how other methods obtain their sensitivities. The sensitivities of the OBD, the OBS, and Skeletonization are measured by computing the (first or second) derivative of the error function when pruning away a certain parameter. The parameter with the smallest sensitivity is removed. For one example of a classification problem in Fig. 16, consider a particular pruning phase of the network

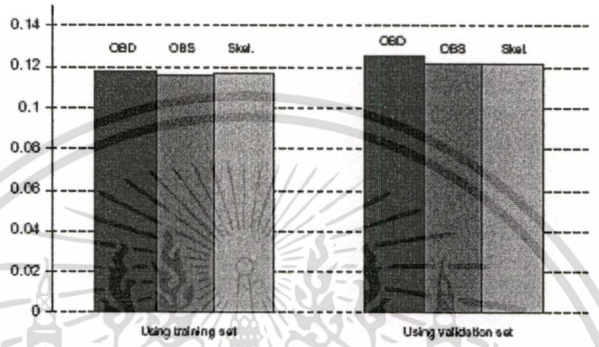


Fig. 13. The test error rate for credit card.

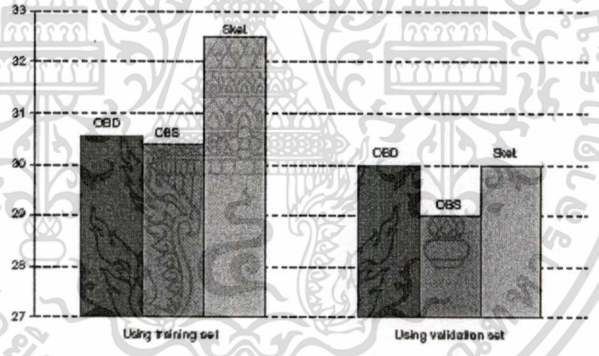


Fig. 14. The number of hidden units for isolet.

having three output units. Let ρ_n be the sensitivity when unit n is removed. It is obvious that $\rho_i < \rho_j$ since the error after removing unit i is smaller than the error when unit j is removed. Therefore, these three algorithms choose hidden unit i and prune it away. However, by using winner-takes-all classification, this pattern is misclassified because the pruned network generates output $\langle 1, 0, 0\rangle$ instead of $\langle 0, 0, 1\rangle$. On the other hand, if they choose to remove hidden unit j, the pruned network will correctly classify this pattern since it generates $\langle 0, 0, 1\rangle$. Therefore, the sensitivities of three well-known pruning algorithms underestimate the irrelevance of the hidden unit. This problem can be resolved by taking into account the number

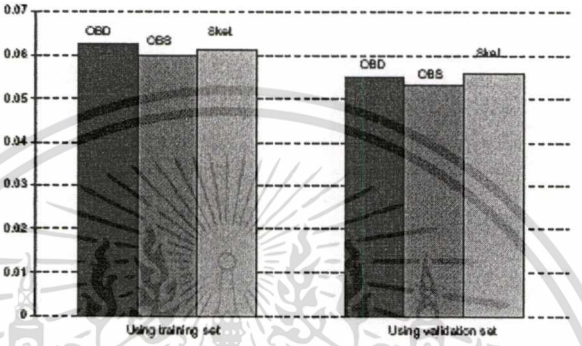


Fig. 13. The test error rate for isolet.

of misclassified validation patterns as the first priority. As a result, the number of misclassified patterns is introduced into Eq. (13) as the main concern. In order to see how often algorithms underestimate the irrelevance of hidden unit, Table 19 illustrates the number of wrong hidden unit removals from a total of 100 hidden unit removals randomly chosen from the previous experiment. The Spartan simplicity algorithm never removes the wrong hidden unit, while the OBD has the largest number of wrong hidden unit removals. If the wrong hidden unit is eliminated, the output error will be large, and the network will be difficult to converge. Thus, there will be unnecessary hidden units left in place and those units prevent the network from having good generalization.

One concern about our approach is a possibility to prune more than one unit at a time. Here, we modify the Spartan simplicity algorithm to support such adaptation as in the following: For hidden units whose AD_j exceeds a certain threshold, we delete them and retrain the network. If the overpruning occurs, we restore the deleted unit which has the largest AD_j compared to other units we recently deleted, and repeat the training process. The restoration of the deleted units continues until the overpruning indicator disappears, and then the algorithm returns the final architecture. Table 20 compares results between the modified Spartan and the original one from Table 18. The results of both versions on the final average number of hidden units and the average error rate are not significantly different. On the other hand, the average CPU time of the modified Spartan is reduced almost by a half in every data set. The modified version saves the retraining time by pruning multiple hidden units at a time. However, we need to design a systematic way to determine a suitable threshold for AD_j since the results of the modified Spartan in Table 20 use the *ad hoc* threshold from the preliminary run. This could be one of the future works that requires further elaboration.

1st Reading

September 4, 2008 18:42 WSPC/123-JCSC 00451

Table 18. Comparison between the Spartan net and other techniques when the validation set is used in calculating the sensitivity.

	Spartan			OBD			OBS			Skeletonization		
	Avg. #HD	Avg. err. rate	Avg. CPU time (s)	Avg. #HD	Avg. err. rate	Avg. CPU time (s)	Avg. #HD	Avg. err. rate	Avg. CPU time (s)	Avg. #HD	Avg. err. rate	Avg. CPU time (s)
Iris	2.25	0.015	535	3.5	0.021	980	3.4	0.020	2000	3.2	0.019	945
Credit card	3.25	0.112	1900	4.5	0.126	2005	4.4	0.122	5835	4.5	0.121	1987
Isolet	20.28	0.051	12610	30	0.025	12870	29	0.053	54810	30	0.055	12020

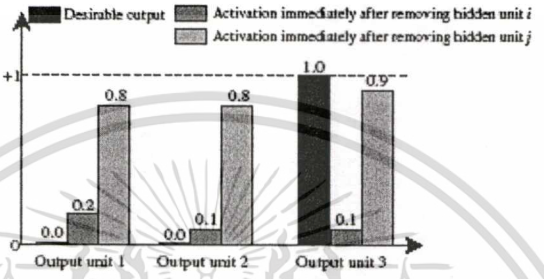


Fig. 16. The desirable outputs and the activations immediately after either the hidden unit i or j is removed.

Table 19. The number of wrong hidden unit removals on three classification problems.

	Spartan	OBD	OBS	Skeletonization
Iris	0	3	2	2
Credit card	0	7	3	5
Isolet	0	13	9	14

Table 20. Comparison between the modified Spartan net and the original Spartan net.

	Modified Spartan			Original Spartan		
	Avg. #HD	Avg. err. rate	Avg. CPU time (s)	Avg. #HD	Avg. err. rate	Avg. CPU time (s)
Iris	2.21	0.018	509	2.25	0.015	935
Credit card	3.20	0.120	1018	3.25	0.112	1950
Isolet	20.21	0.060	6983	20.28	0.051	12610

6. Conclusions

In this paper, we proposed a sensitivity analysis pruning algorithm for the single-hidden layer feed-forward neural networks using the validation set. Our approach is simple and easy to implement, yet produces a very good result. There are four factors behind the success of the Spartan simplicity algorithm. First, the sensitivity (AD_j) is derived by the absolute difference between the desirable output (in the validation set) and the activation of the output unit when the hidden unit is removed. Second, the sensitivity calculation is executed on the fly by using the validation set without increasing the asymptotic computational complexity of the back-propagation algorithm. Third, for a classification problem, the algorithm measures the sensitivity by considering the number of misclassified validation patterns, when

each hidden unit is removed, as the main concern. This resolves the situation when the sensitivity underestimates the irrelevance of hidden unit, even if the validation set is used in measuring the sensitivity. At last, the algorithm does not reinitialize weights when a particular unit is removed. This prevents the small-sized network from being relatively sensitive to initial conditions. We have shown the effectiveness of the algorithm on three artificial and seven real-world benchmarks. The correctness of the pruning results is illustrated by using problems for which the minimum number of hidden units are known, e.g., F1, F2, and AC problems. The robustness ability is tested by the high-noised data sets, e.g., diabetes, heart disease, and credit card problems. The scalability of our algorithm is verified by the experiment on the isolet problem. A limitation of our algorithm is the assumption of having sufficient validation patterns (at least 25% of the data set) that truly represent the test set. Otherwise, the technique will begin to pay too much attention to the specific details of the validation set and then not generalize as well to new data. There are two future improvements to the Spartan simplicity algorithm. First, we may detect the overpruning early in order to avoid the network restoration. Notice that during the last pruning phase, the AD_j of each hidden unit will be large and relatively close to the others. Some statistical approach can be used to determine if the AD_j value of a unit is not significantly smaller than that of other units. If that is the case, the algorithm can stop early before the overpruning occurs. Second, as mentioned in Sec. 5, we could design a systematic way to find a suitable threshold for AD_j in order to perform a multiple-unit deletion and reduce the pruning time.

Acknowledgments

The authors are grateful to Dr Matthew Dailey (AIT), Dr Vasin Punyakanok, Samarth Swarup (UIUC), anonymous reviewers from the *Journal of Circuits, Systems and Computers* for sharing valuable ideas and giving constructive comments, and Noppawan Jearanaitanakij for providing proofs of reading on the early draft.

References

1. K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (1989) 359–366.
2. M. C. Mozer and P. Smolensky, *Advances in Neural Information Processing Systems*, Vol. 1 (Morgan Kaufman, San Mateo, CA, 1989), pp. 107–115.
3. Y. LeCun, J. S. Denker and S. A. Solla, *Advances in Neural Information Processing Systems*, Vol. 2 (Morgan Kaufman, San Mateo, CA, 1990), pp. 598–605.
4. B. Hassibi and D. G. Stork, Second order derivatives for network pruning: Optimal brain surgeon, *Adv. Neural Inform. Process. Sys.* 5 (1993) 162–171.
5. K. Murase, Y. Matsunaga and Y. Nakade, A back-propagation algorithm which automatically determines the number of association units, *Proc. IEEE Int. Conf. Neural Networks* 1 (1991) 783–788.
6. Md. Shahjahan and K. Murase, A dynamic node decaying method for pruning artificial neural networks, *IEICE Trans. Inf. Sys.* E86-D (2003) 736–751.

7. A. P. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, *IEEE Trans. Neural Networks* 12 (2001) 1386–1399.
8. P. Lauret, E. Fock and T. A. Mara, A node pruning algorithm based on a Fourier amplitude sensitivity test method, *IEEE Trans. Neural Networks* 17 (2006) 273–293.
9. L. Prechelt, Proben1 — A set of neural network benchmark problems and benchmarking rules, Technical Report, September 1994, Karlsruhe, Germany.
10. Y. Chauvin, Generalization performance of overtrained backpropagation networks, in *Neural Networks Proc. EUROSIP Workshop*, eds. L. B. Almeida and C. J. Wellekens, (1990), pp. 46–55.
11. R. Reed, Pruning algorithms — A survey, *IEEE Trans. Neural Networks* 4 (1993) 740–747.
12. S. Haykin, *Neural Networks: A Comprehensive Foundation* (Macmillan College Publishing Company, New York, 1994).
13. Y. LeCun, *Efficient Learning and Second-order Methods, A Tutorial* (NIPS, Denver, 1993).
14. X. Yao and Y. Liu, A new evolutionary system for evolving artificial neural networks, *IEEE Trans. Neural Networks* 8 (1997) 694–713.
15. M. Hagiwara, Removal of hidden units and weights for backpropagation networks, in *Proc. IEEE Int. Joint Conf. Neural Networks* (1993), 351–354.

ภาคผนวก ข.

ตารางสัญลักษณ์

สัญลักษณ์	ความหมาย
α	อัตราในการเรียนรู้
θ	ค่าเทรชโฮลประจำนิวรอน
γ	จำนวนรอบในการฝึกโครงข่ายประสาทเทียม
φ	แอ็คติเวชั่นฟังก์ชั่น
β	ค่าเทรชโฮลสำหรับตัวชี้โอเวอร์พรุนนิ่ง

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายเกียรติกูล เจียรนัยธนะกิจ
วัน เดือน ปีเกิด	14 เมษายน 2516
ประวัติการศึกษา	2538 วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
	2542 Master of Computer Science
	Oregon State University, Corvallis, USA

ความชำนาญเฉพาะด้าน:

1) โครงข่ายประสาทเทียม
2) ปัญญาประดิษฐ์
3) ทฤษฎีการคำนวณ

ประสบการณ์การทำงานและผลงานวิจัย:

พ.ศ.2538-2539	ตำแหน่งวิศวกรระบบบริษัท โทเท็ล แอ็คเซส คอมมิวนิเคชั่น จำกัด
พ.ศ.2539-2540	ตำแหน่งโปรแกรมเมอร์ศูนย์ฝึกอบรมคอมพิวเตอร์ทบวงมหาวิทยาลัย
พ.ศ.2542 – ปัจจุบัน	อาจารย์ประจำภาควิชาวิศวกรรมคอมพิวเตอร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง