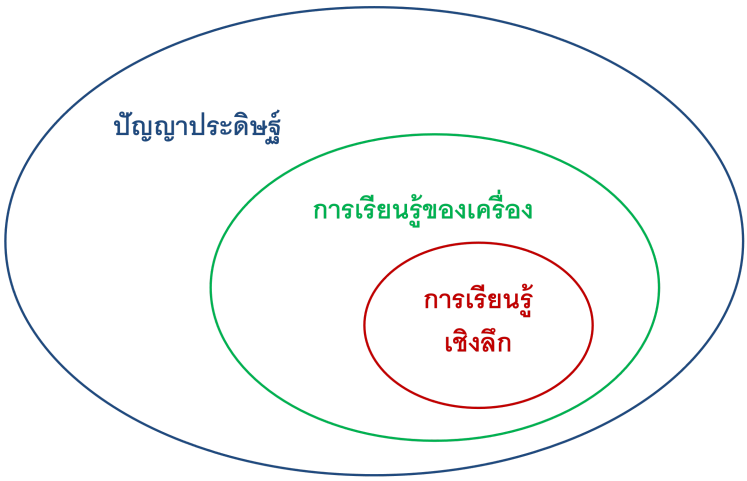
1. บทนำ

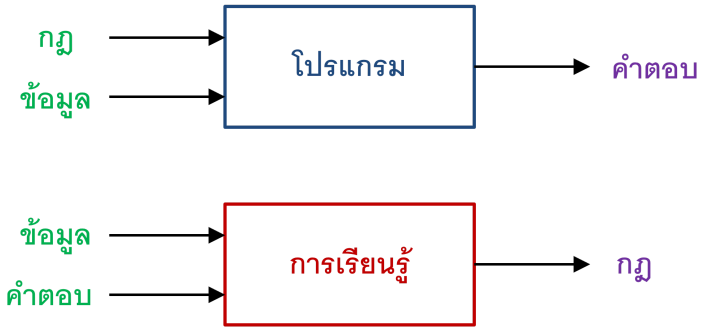
ก่อนจะเข้าสู่เนื้อหาหลัก เราลองหาคำจำกัดความของการเรียนรู้เชิงลึก (deep learning) เริ่มต้นจากโดเมนใหญ่สุด จากคำบรรยายในวิกิพิเดีย *ปัญญาประดิษฐ์ (artificial intelligence)* หรือที่นิยมเรียกโดยย่อว่า AI คือเชาว์ปัญญาที่แสดงให้เห็นในเครื่องจักรกล แตกต่างจากปัญญาธรรมชาติจากสมองของมนุษย์หรือสัตว์ ดังนั้น AI จะครอบคลุมนวัตกรรมทั้งหมดที่ช่วยทำให้คอมพิวเตอร์มีความชาญฉลาดเข้าสู่ปัญญาของมนุษย์ ซึ่งมีขอบเขตที่กว้างมาก *การเรียนรู้ของเครื่อง (machine learning)* คือเซตย่อยของเอไอที่เน้นการศึกษาขั้นตอนวิธีทางคอมพิวเตอร์ในการเรียนรู้และปรับตัวจากข้อมูล ซึ่งสามารถทำได้หลายแนวทางเช่นการหาค่าเหมาะที่สุด ทฤษฎีกราฟ แต่ในหนังสือนี้จะศึกษาวิธีการใช้ *โครงข่ายประสาทเทียม (artificial neural network)* ซึ่งต่อไปจะอ้างถึงโดยตัวย่อ ANN ในการเรียนรู้ กล่าวได้ว่า ANN คือระบบคอมพิวเตอร์หรือโมเดลทางคณิตศาสตร์ที่ใช้ในการจำลองสมองทางชีวภาพ โดยอาศัยการเรียนรู้หรือเรียกว่าการฝึก (training) โดยข้อมูลตัวอย่าง เมื่อเขียนภาพรวมความสัมพันธ์ของขอบเขตปัญหาที่กล่าวมาจะได้ดังแสดงในรูปที่ 1.1



รูปที่ 1.1 ความสัมพันธ์ของปัญญาประดิษฐ์ การเรียนรู้ของเครื่อง และการเรียนรู้เชิงลึก

ความแตกต่างที่สำคัญระหว่างแนวทางการเรียนรู้ของเครื่องกับการเขียนโปรแกรมคอมพิวเตอร์ทั่วไปแสดงได้ในรูปที่ 1.2 แผนภาพด้านบนคือการเขียนโปรแกรมในรูปแบบที่เราคุ้นเคย สมมุติว่าต้องการเขียนฟังก์ชันอย่างง่ายเพื่อบวกเลขสองจำนวน คำสั่งในโปรแกรมคือการดำเนินการบวก และอาจมีการตรวจสอบว่าตัวแปรที่ได้รับเป็นประเภทตัวเลข การดำเนินการและเงื่อนไขเหล่านี้คือกฎของฟังก์ชัน และตัวเลขที่เป็นอาร์กิวเมนต์ของฟังก์ชันคือข้อมูล สิ่งที่คืนค่าจากฟังก์ชันหรือเอาต์พุตก็คือผลลัพธ์ของการบวก ซึ่งจะได้

คำตอบเป็นตัวเลข ซึ่งสำหรับฟังก์ชันง่ายๆ นี้อาจมองไม่เห็นปัญหา แต่ลองคิดดูเล่นๆ ว่าถ้าต้องการขยายฟังก์ชันนี้ให้รับอินพุตรูปแบบอื่น เช่นถ้าอินพุตเป็นตัวแปรสตริง ให้ต่อข้อความเข้าด้วยกัน หรืออาจเป็นวัตถุอื่นเช่นมะละกอฝานรวมกับมะเขือเทศเป็นส่วนหนึ่งของส้มตำ กฎของโปรแกรมจะต้องถูกเพิ่มเงื่อนไขมากขึ้นจนในที่สุดมีความซับซ้อนเกินกว่าจะจัดการได้ หรืออาจเกิดปัญหาความขัดแย้งกันระหว่างกฎ พิจารณาอีกตัวอย่างหนึ่งที่เกี่ยวข้องกับ AI มากขึ้น เราต้องการสร้างอุปกรณ์ที่ช่วยผู้พิการในกิจกรรมต่างๆ ในชีวิตประจำวัน ตั้งกฎหนึ่งไว้เพื่อความปลอดภัยว่าของมีคม จะต้องอยู่ไม่ใกล้อวัยวะของผู้ใช้เกินค่าที่กำหนด ผลคืออุปกรณ์นี้ไม่สามารถโกนหนวดให้กับเขาได้เพราะวิเคราะห์ว่าเป็นอันตราย ความขัดแย้งของกฎในลักษณะนี้ทำให้แนวทางปัญญาประดิษฐ์ที่ใช้ *ระบบผู้เชี่ยวชาญ (expert systems)* ไม่ประสบความสำเร็จเท่าที่ควรและส่งผลให้การวิจัยในด้านนี้หยุดชะงักไปช่วงเวลาหนึ่ง เรียกกันว่าเป็นช่วงฤดูหนาวของ AI



รูปที่ 1.2 ความแตกต่างระหว่างการโปรแกรมทั่วไปกับการเรียนรู้ของเครื่อง

เมื่อพิจารณากระบวนทัศน์ของการแก้ปัญหาในแนวทางการเรียนรู้ของเครื่องในแผนภาพด้านล่างของรูปที่ 1.2 จะเห็นว่าอินพุตของระบบคือ ข้อมูลและคำตอบ ซึ่งโดยทั่วไปจะมีจำนวนมากและมีความหลากหลาย คอมพิวเตอร์ใช้อินพุตนี้ในการฝึกฝนตัวเองจนกระทั่งได้กฎที่สอดคล้องกับความต้องการของผู้ใช้มากที่สุด ตัวอย่างที่มีการใช้งานมากสุดคือการจำแนกภาพ เช่นต้องการจำแนกภาพสัตว์เลี้ยงระหว่างสุนัขกับแมว ข้อมูลสำหรับฝึกคือภาพของสุนัขและแมวในอริยาบทต่างๆ จำนวนมากเพียงพอ และคำตอบของภาพนั้นที่เรียกว่า *เลเบล (label)* หรือ *ความจริงมูลเหตุ (ground truth)* หากการฝึกประสบผลสำเร็จโมเดลที่ได้จะสามารถจำแนกภาพที่เป็นข้อมูลทดสอบ หรือภาพจากกล้องที่ไม่เคยใช้ในการฝึก โดยมีความแม่นยำขึ้นกับหลายปัจจัยเช่น จำนวนข้อมูลในการฝึก สถาปัตยกรรมของโมเดล ความยากง่ายของภาพที่ต้องการพยากรณ์ การเรียนรู้เชิงลึกโดยใช้ ANN เป็นวิธีการหนึ่งที่ใช้ในปัญหาการจำแนกได้อย่างมีประสิทธิภาพ โดยผลจากการแข่งขันมีความแม่นยำเหนือกว่าแนวทางอื่น ทำให้ AI กลับมาได้รับความสนใจอีกครั้งจนถึงปัจจุบัน

1.1 ประวัติโดยย่อของโครงข่ายประสาทเทียม

แท้จริงแล้วการจำลองสมองทางชีวภาพโดย ANN มิใช่เป็นวิธีการใหม่ แต่ได้เริ่มต้นมาตั้งแต่ ค.ศ 1943 โดยนักประสาทวิทยา Warren McCulloch และนักคณิตศาสตร์ Waler Pitts [MP43] เขียนบทความเกี่ยวกับการทำงานของเซลล์ประสาทที่สามารถสร้างโมเดลจำลองเป็นวงจรไฟฟ้า ต่อมาในปี ค.ศ. 1949 Donald Hebb ได้เขียนหนังสือชื่อเรื่อง The Organization of Behavior [Heb49] เสริมแนวคิดการทำงานของเซลล์ประสาท ต่อมาในยุคเริ่มต้นของคอมพิวเตอร์ช่วงทศวรรษ ที่ 1950 เริ่มมีการโมเดลทฤษฎีเหล่านี้และจำลองโครงข่ายประสาทเทียมครั้งแรกโดย Nathanial Rochester ณ ห้องปฏิบัติการวิจัยของไอบีเอ็ม ซึ่งไม่ประสบผลสำเร็จ ในช่วงเวลาดังกล่าวความสนใจในการโปรแกรมคอมพิวเตอร์เป็นไปในแนวทางดั้งเดิมมีมากกว่า แม้ในปี ค.ศ. 1956 จะเกิดโครงการ Darthmouth Summer Research Project on Artificial Intelligence ที่ช่วยกระตุ้นให้เกิดงานวิจัยทั้งด้าน ANN และ AI มากขึ้น

ปีต่อมาหลังจากโครงการนี้ John Von Neumann แนะนำการเลียนแบบฟังก์ชันของเซลล์ประสาทอย่างง่ายโดยใช้อุปกรณ์รีเลย์โทรเลขและหลอดสูญญากาศ ในขณะที่นักชีววิทยาด้านเซลล์ประสาทชื่อ Frank Rosenblatt ณ มหาวิทยาลัยคอร์เนลเริ่มต้นงานเกี่ยวกับ *เพอร์เซปตรอน (Perceptron)* [Ros58] โดยได้ความสนใจจากการทำงานของตาของแมลงวัน ซึ่งเป็นตัวหลักในการประมวลผลให้บินหนีจากภัยอันตราย เพอร์เซปตรอนได้มาจากผลการวิจัยนี้และได้ถูกสร้างบนฮาร์ดแวร์ จัดว่าเป็น ANN เก่าแก่ที่สุดที่ยังมีการใช้งานในปัจจุบัน อยางไรก็ตาม เพอร์เซปตรอนในช่วงเวลานี้นเป็นแบบชั้นเดียวและมีข้อจำกัด โดยในปี 1969 Masrvin Minsky และ Seymour Papert ได้เขียนหนังสือ Perceptrons [MP69] เพื่อพิสูจน์ข้อจำกัดนั้น

ในปี ค.ศ. 1959 Bernard Widrow และ Marcian Hoff มหาวิทยาลัยสแตนฟอร์ดได้พัฒนาโมเดลมีชื่อเรียกว่า ADALINE และ MADALINE (Multiple ADAptive LINear Elements) ซึ่งเป็น ANN ตัวแรกที่นำมาประยุกต์ใช้กับงานจริง โดยใช้เป็นตัวกรองแบบปรับตัวที่สามารถขจัดเสียงสะท้อนในสายโทรศัพท์ และยังมีการใช้งานเชิงพานิชย์ในปัจจุบัน

ความสำเร็จในช่วงต้นนี้ทำให้มีการอวดอ้างศักยภาพของ ANN เดินความเป็นจริงโดยเฉพาะในยุคที่อิเล็กทรอนิกส์ยังไม่พัฒนามาก สร้างความผิดหวังเมื่อ ANN ไม่สามารถทำงาน ได้ตามเป้าหมายและสร้างผลร้ายกับงานวิจัย ทำให้การให้ทุนหยุดชะงักไปช่วงเวลาหนึ่งจนถึงปี 1981 จนกระทั่งในปี 1982 John Hopfield สังกัด ม. Caltech นำเสนอบทความไปยัง National Academy of Sciences [Hop82] โดยแนวทางของเขาไม่เพียง

แต่โมเดลสมองมนุษย์ แต่ยังสร้างอุปกรณ์ที่มีประโยชน์ โดยแสดงการวิเคราะห์ทางคณิตศาสตร์ อย่างชัดเจนว่าโครงข่ายทำงานอย่างไรและทำงานอะไรได้บ้าง อย่างไรก็ตาม ความสำเร็จของ Hopfiled ส่วนใหญ่มาจากความสามารถในการพูดเพื่อโน้มน้าวผู้ฟัง ขณะที่นวัตกรรมของเขายังไม่ใช่จุดเด่น ในขณะเดียวกันในงานประชุมที่ประเทศญี่ปุ่นที่แสดงความก้าวหน้าของประเทศคู่แข่งทำให้สหรัฐอเมริกากลัวว่าจะตามไม่ทัน จึงเกิดการกระตุ้นของทุนวิจัยทางด้านนี้อีกครั้งหนึ่ง

ปี ค.ศ. 1989 American Institute of Physics ได้จัดงานซึ่งกลายเป็นการประชุมประจำปีชื่อว่า Neural Networks for Computing และในปี 1987 Institute of Electrial and Electronic Engineer (IEEE) จัดงาน International Conference on Neural Networks ซึ่งมีผู้ร่วมงานกว่า 1,800 คน ในปีเดียวกันนี้ ความสำเร็จในการประยุกต์ใช้ ANN ในเชิงปฏิบัติเกิดขึ้นที่ห้องปฏิบัติการเบลล์ เมื่อ Yann LeCun ผสมผสานแนวคิดของการสังวัตนาการกับการแพร่กระจายย้อนหลังเพื่อสร้างโครงข่าย LeNet [LBBH98]สำหรับจำแนกตัวเลขที่เขียนด้วยลายมือ ซึ่งกรมไปรษณีย์ของสหรัฐอเมริกาได้นำไปใช้งานในช่วงทศวรรษที่ 1990 ในการอ่านรหัสไปรษณีย์บนซองจดหมาย

หลังจากนั้นความสนใจด้าน ANN กลับซบเซาลงอีกครั้งเมื่อมีการนำเสนอวิธีการอื่นในการเรียนรู้ของเครื่อง เช่น *Support Vector Meahine (SVM)* ที่พัฒนาโดย Vladimir Vapnik และ Corinna Cortes [CV95]จากห้องปฏิบัติการเบลล์ในช่วงต้นทศวรรษ 1990 โดยในช่วงนั้น SVM มีสมรรถนะเยี่ยมยอดสำหรับปัญหาการจำแนกพื้นฐาน เป็นหนึ่งในวิธีการด้านการเรียนรู้ของเครื่องจำนวนไม่มากนักที่รองรับโดยทฤษฎีและการวิเคราะห์ทางคณิตศาสตร์อย่างครอบคลุม ทำให้สามารถเข้าใจได้เป็นอย่างดี อย่างไรก็ตาม SVM เริ่มไม่ได้ผลดีสำหรับข้อมูลที่มีขนาดใหญ่เช่นการจำแนกภาพจำนวนมาก ในการใช้ SVM สำหรับปัญหาการรับรู้ดังกล่าวจำเป็นต้องมีการจัดการกับข้อมูลด้วยมือเรียกว่า *วิศวกรรมลักษณะเด่น (feature engineering)* ซึ่งเป็นกรรมวิธีที่ยากและเปราะบาง

ในขณะที่สังคมด้านวิทยาศาสตร์เบนความสนใจจาก ANN ไป ในช่วงทศวรรษ 2010 ได้มีกลุ่มที่ยังคงเกาะติดอยู่กับงานด้านนี้และค้นพบความก้าวหน้าที่สำคัญ ประกอบด้วย Geoffrey Hinton (University of Toronto) Yoshua Bengio (University of Montreal) Yann LeCun (New York University) และ IDSIA (Switzerland) ในปี ค.ศ. 2011 Dan Ciresan จาก IDSIA ชนะการแข่งขันการจำแนกภาพโดยใช้โครงข่ายประสาทเทียมเชิงลึกที่ฝึกหัดโดยใช้ Graphics Processing Unit (GPU) นับเป็นครั้งแรกของความสำเร็จสำหรับการเรียนรู้เชิงลึกสมัยใหม่ ความสำเร็จครั้งใหญ่ตามมาในปี 2012 เมื่อกลุ่มของ Hinton เข้าแข่งขันการจำแนกภาพที่เรียกว่า ILSVRC (ImageNet Large Scale Visual Recognition Challenge) ซึ่งเป็นปัญหาที่ยากมากในขณะนั้น โจทย์คือการจำแนก

ภาพสีความละเอียดสูงออกเป็น 1000 ประเภทที่แตกต่างกัน หลังจากการฝึกโดยภาพจำนวน 1.4 ล้านภาพ ซึ่งในปีก่อนหน้านั้น ความแม่นยำของผู้ชนะโดยวิธีการอื่นด้านการรับรู้ภาพโดยคอมพิวเตอร์อยู่ที่ 74.3 % แต่ในปี 2012 ทีมแข่งขันที่นำโดย Alex Krizhevsky โดยมี Geoffrey Hinton เป็นที่ปรึกษาสามารถทำความแม่นยำได้ถึง 83.6% หลังจากปีนั้นการแข่งขันนี้ผู้แข่งที่ใช้โครงข่ายประสาทเทียมเชิงลึกชนะมาโดยตลอด ในปี 2015 ผู้ชนะได้ความแม่นยำถึง 96.4% และทำให้การจำแนกภาพ ImageNet ถูกจัดว่าเป็นปัญหาที่สามารถหาคำตอบได้โดยสมบูรณ์

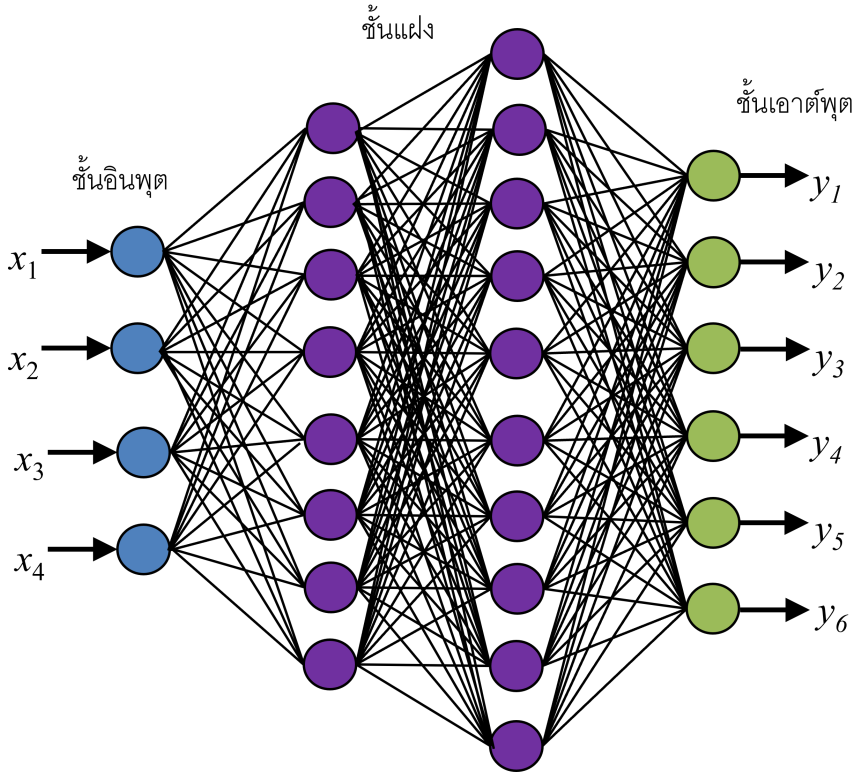
1.2 โครงข่ายประสาทเทียมเชิงลึก

มักมีความเข้าใจผิดกับคำว่า “เชิงลึก” ในการเรียนรู้เชิงลึก ซึ่งมิได้มีความหมายในเชิงปรัชญา หรือแปลว่าการเรียนรู้อย่างถ่องแท้แต่อย่างใด แต่เป็นคำคุณศัพท์ที่ขยายความเกี่ยวกับจำนวนชั้นของโครงข่ายประสาทเทียม โดย ANN ที่มีเฉพาะชั้นที่เป็นอินพุตและเอาต์พุตเรียกว่าเป็นแบบตื้น (shallow) ส่วน *โครงข่ายประสาทเทียมเชิงลึก (deep neural network)* ซึ่งต่อไปจะเรียกโดยย่อว่า DNN นอกจากชั้นอินพุตและเอาต์พุตแล้วยังประกอบด้วย *ชั้นแฝง (hidden layer)* อย่างน้อยหนึ่งชั้น ตัวอย่างดังในรูปที่ 1.3

> **หมายเหตุ:**
>
> ในบางแหล่งข้อมูลใช้ตัวย่อ DNN แทน *โครงข่ายประสาทหนาแน่น (dense neural network)* ซึ่งหมายความว่าเซลล์ทั้งหมดในแต่ละชั้นมีการเชื่อมต่อถึงกัน ในด้านการใช้งานทางปฏิบัติไม่มีความแตกต่างกัน

นอกจากการเพิ่มชั้นแฝงในโมเดลแล้ว องค์ประกอบสำคัญอีกประการหนึ่งคือที่เซลล์เอาต์พุตของชั้นก่อนหน้าจะต้องมี *ฟังก์ชันกระตุ้น (activation function)* แบบไม่เป็นเชิงเส้น หากไม่มีส่วนนี้ ชั้นแฝงที่เพิ่มเข้ามาจะไม่มีประโยชน์เพราะสามารถใช้การดำเนินการเชิงเส้นเพื่อยุบรวมกับชั้นก่อนหน้านั้นได้

นอกจากนั้นแต่ละชั้นที่เพิ่มให้กับโครงข่ายอาจมีคุณสมบัติที่แตกต่างกันไป เช่นการใช้ชั้นสังวัตนาการในส่วนหน้าเพื่อเพิ่มสมรรถนะในการเรียนรู้ภาพ หรือชั้นที่ออกแบบเฉพาะสำหรับแปลงข้อความเป็นตัวเลข โมเดลการเรียนรู้เชิงลึกที่ใช้งานจริงอาจประกอบด้วยชั้นจำนวนหลายร้อยชั้น



รูปที่ 1.3 ตัวอย่างโครงข่ายประสาทเทียมเชิงลึก (DNN)

อย่างไรก็ตามถึงแม้ว่าโครงข่ายประสาทเทียมจะมีแนวคิดหรือแรงจูงใจมาจากสมองทางชีวภาพของมนุษย์ ก็ยังไม่มีหลักฐานยืนยันว่าสมองมนุษย์มีกลไกการเรียนรู้ในรูปแบบเดียวกันนี้ ดังนั้นหากจะนิยามโดยไม่เป็นนิยายทางวิทยาศาสตร์เกินความจริงไป การเรียนรู้เชิงลึกเป็นเพียงกรอบการทำงานเชิงคณิตศาสตร์สำหรับการเรียนรู้จากข้อมูล โดยอาศัยหน่วยประมวลผลที่จัดรูปเป็นชั้นและเชื่อมต่อกันโดยค่าน้ำหนัก ปัญญาของโครงข่ายได้มาจากการปรับค่าน้ำหนักผ่านการฝึก จนกระทั่งสามารถทำงานตามต้องการโดยมีความแม่นยำอยู่ในเกณฑ์ที่ยอมรับได้

1.3 รูปแบบการเรียนรู้

เราสามารถแยกประเภทของการเรียนรู้ได้เป็น 3 รูปแบบดังนี้

1. การเรียนรู้แบบมีผู้สอน (supervised learning) เป็นลักษณะการเรียนรู้ที่เข้าใจได้ง่ายสุด เพราะเหมือนกับชีวิตมนุษย์ที่ตั้งแต่เกิดมีพ่อแม่ วัยเรียนมีครูชี้แนะจนถึงอาจารย์สอน พ้นจากมหาวิทยาลัยก็ยังมีหัวหน้างาน กูเกิลและยูทูบ ดังนั้นในการเรียนรู้แบบนี้จะมีข้อมูลและเป้าหมาย (นิยมเรียกว่าเลเบลหรือความจริงมูลเหตุ) ตัวอย่างเช่นในการฝึกโมเดลเพื่อจำแนกภาพสุนัขและแมว ข้อมูลในการฝึกคือ X = ภาพ และ

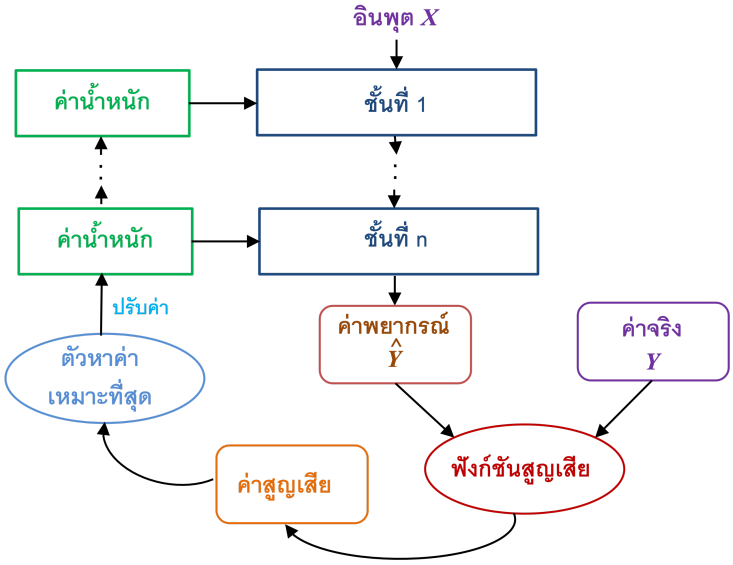
Y = เลเบลบอกว่าภาพนั้นคือสุนัขหรือแมว หลังจากการฝึกเสร็จสิ้น เราจึงทดสอบโดยการป้อนภาพที่ไม่ได้ใช้ในการฝึก เพื่อให้โมเดลพยากรณ์ว่าเป็นประเภทใด

2. การเรียนรู้แบบไม่มีผู้สอน (unsupervised learning) ในการเรียนรู้ประเภทนี้จะไม่มีเลเบลให้ในข้อมูล ดังนั้นโมเดลจะต้องพยายามหาแบบรูปหรือโครงสร้างที่แฝงอยู่ในข้อมูลนั้นเอง ยกตัวอย่างเช่น *การจัดกลุ่ม (clustering)* โดยการรวมข้อมูลตัวอย่างที่มีลักษณะเด่นคล้ายกันเข้าด้วยกัน หรือ *การวิเคราะห์องค์ประกอบหลัก (principal component analysis)* ที่ขั้นตอนวิธีจะหาทางบีบอัดข้อมูลโดยเลือกเฉพาะลักษณะเด่นที่มีประโยชน์ และขจัดส่วนที่เหลือทิ้งโดยอัตโนมัติ
3. การเรียนรู้แบบเสริมกำลัง (reinforcement learning) ซึ่งในบางแหล่งข้อมูลเช่นวิกิพิเดียจะจัดเป็นการเรียนรู้แบบไม่มีผู้สอน อย่างไรก็ตามการเรียนรู้แบบเสริมกำลังมีข้อแตกต่างในวิธีการเรียนรู้ โดยโมเดลไม่ได้พยายามหาโครงสร้างที่แฝงในข้อมูล แต่เลือกการกระทำที่จะทำให้ได้รางวัลเป็นผลตอบแทนมากที่สุด โดยอาศัยเครื่องมือทางคณิตศาสตร์เช่น *กระบวนการมาร์คอฟ (Markov process)* และ *สมการของเบลแมน (Bellman equation)*

1.3.1 แผนภาพการเรียนรู้เชิงลึก

จากภาพรวมการทำงานของ DNN จะเห็นว่าขั้นตอนสำคัญคือการเรียนรู้ ซึ่งก็คือการฝึกโมเดลจากข้อมูล รูปที่ 1.4 แสดงแผนภาพการเรียนรู้แบบมีผู้สอนโดย DNN โดยตัวโมเดลประกอบด้วยชั้นของเซลล์ที่เชื่อมต่อกันโดยค่าน้ำหนัก (รวมค่าเอนเอียง) จากข้อมูลอินพุตแต่ละตัวอย่าง โมเดลจะคำนวณเอาต์พุตและส่งให้กับฟังก์ชันสูญเสียเพื่อเปรียบเทียบกับเลเบลของตัวอย่างนั้นเพื่อคำนวณค่าสูญเสีย ที่เป็นตัววัดความผิดพลาดระหว่างการพยากรณ์กับความจริงมูลเหตุ ค่าสูญเสียนี้จะถูกป้อนให้กับตัวหาค่าเหมาะที่สุดเพื่อปรับค่าน้ำหนักในทิศทางที่ลดค่าสูญเสียให้เหลือน้อยสุด ดังนั้นเมื่อมองในแง่ของขั้นตอนวิธีการฝึกเพื่อเรียนรู้ของ DNN คือการพยากรณ์เอาต์พุตและปรับค่าน้ำหนักแบบวนรอบ จนกระทั่งโมเดลสามารถทำงานได้ตามวัตถุประสงค์โดยมีความแม่นยำอยู่ในเกณฑ์ที่ยอมรับ

โดยทั่วไปการฝึก DNN ขนาดใหญ่โดยข้อมูลจำนวนมากจะใช้เวลานาน ดังนั้นนิยมใช้การประมวลผลแบบขนานเข้าช่วย เช่นการใช้ตัวประมวลผลกราฟิก นอกจากนั้นกรรมวิธีการเตรียมข้อมูลเช่นการจัดกลุ่ม (batching) และการปรับค่าไฮเปอร์พารามิเตอร์ต่างๆ สามารถปรับปรุงการฝึกให้ดีขึ้น ซึ่งจะได้กล่าวถึงต่อไปในหนังสือนี้



รูปที่ 1.4 แผนภาพการเรียนรู้แบบมีผู้สอนโดย DNN

1.4 ซอฟต์แวร์สำหรับพัฒนา

ถึงแม้ว่าในการพัฒนาการเรียนรู้เชิงลึกสามารถกระทำได้โดยภาษาคอมพิวเตอร์หลายภาษา รวมถึงการใช้โปรแกรมสำเร็จรูปเช่น MATLAB แต่กล่าวได้ว่าภาษาที่ได้รับความนิยมอย่างมากคือไพธอน (Python) เนื่องจากเป็นภาษาเชิงวัตถุระดับสูงที่เริ่มใช้มาตั้งแต่ ค.ศ 1991 มีระบบนิเวศน์ที่ค่อนข้างสมบูรณ์ เช่นแพ็กเกจสนับสนุน ชุมชนออนไลน์ที่สามารถขอความช่วยเหลือเมื่อเกิดปัญหา ทางเลือกหนึ่งที่น่าสนใจคือภาษาที่เริ่มต้นไม่นานแต่มีสมรรถนะดีและมีจุดเด่นหลายประการคือ จูเลีย (Julia) ที่เริ่มมีการใช้งานในมหาวิทยาลัยชั้นนำของโลก อย่างไรก็ตาม จูเลียยังเสียเปรียบเรื่องความหลากหลายและสมบูรณ์ของแพ็กเกจสนับสนุน การเปลี่ยนแปลงบ่อยครั้งของไวยากรณ์คำสั่ง และการช่วยเหลือจากชุมชนยังมีน้อยกว่า

ในหนังสือนี้จะเน้นการโปรแกรมภาษาไพธอนโดยอาศัยแพ็กเกจ *เทนเซอร์โฟลว์ (TensorFlow)* ซึ่งเป็นไลบรารีโอเพนซอร์สด้านการเรียนรู้ของเครื่องและโครงข่ายประสาทเทียมที่ใช้งานง่าย เป็นผลงานการวิจัยและผลิตภัณฑ์ของกูเกิล เดิมถูกพัฒนาโดยทีมกูเกิลเบรน (Google brain) เพื่อใช้ภายในบริษัทกูเกิล แต่ต่อมาเปิดให้สาธารณชนได้ใช้งานภายใต้ Apache License 2.0 เมื่อวันที่ 9 พฤศจิกายน ค.ศ. 2015 เทนเซอร์โฟลว์มีการขยายการรองรับไปยังภาษาอื่นด้วยเช่น ซีพลัสพลัส (C++) จาวาสคริป (JavaScript) รวมถึง จูเลีย ถึงแม้ว่ายังไม่สมบูรณ์เท่าการใช้งานบนไพธอน

> **หมายเหตุ:**
>
> 1. เพื่อความกระชับของเนื้อหา ในหนังสือนี้จะใช้ตัวย่อ TF แทนชื่อไลบรารีเทนเซอร์โฟลว์
> 2. ไลบรารีสำหรับการเรียนรู้เชิงลึกสำหรับไพธอนเดิมรู้จักกันในชื่อ Keras (https://keras.io/) แต่ใน TF เวอร์ชัน 2 ขึ้นไปจะรวม Keras อยู่ในแพ็กเกจแล้ว ไม่จำเป็นต้องติดตั้งเพิ่มเติม

คำว่า "เทนเซอร์" คือวัตถุทางคณิตศาสตร์ที่ใช้ในการเก็บข้อมูลหลายมิติ ปกติเราจะคุ้นเคยกับแอเรย์และเมทริกซ์ที่มีสมาชิกเป็นหนึ่งและสองมิติ ข้อมูลที่ใช้สำหรับโมเดลเชิงลึกอาจมีมิติมากกว่านั้น เช่นข้อมูลภาพประกอบด้วยความกว้าง ความสูง จำนวนตัวอย่าง และจำนวนช่องหรือแชนเนล (channel) ข้อมูล 4 มิตินี้จะถูกจัดเก็บอย่างเหมาะสมโดยเทนเซอร์ที่มีค่าลำดับชั้น (rank) เท่ากับ 4

เราจะแนะนำการใช้งาน TF จากตัวอย่างที่กล่าวได้ว่าเป็นการทักทายโลก (Hello world) ของการเรียนรู้เชิงลึก เริ่มต้นโดยการนำเข้าแพ็กเกจที่ต้องการใช้งาน และตรวจสอบเวอร์ชันของ TF ในหนังสือนี้จะต้องการเวอร์ชันตั้งแต่ 2.0 ขึ้นไป

```
import tensorflow as tf
import numpy as np
#tf.__version__
```

ตัวอย่าง 1.1

สมมุติว่าเราอยากทราบความสัมพันธ์ทางคณิตศาสตร์ระหว่างแอเรย์ของอินพุตและเอาต์พุตชุดหนึ่ง

```
xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0],
                                     dtype=float)
ys = np.array([-5.0, -3.0, -1.0, 1.0, 3.0, 5.0],
                                     dtype=float)
```

เนื่องจากเป็นโจทย์ที่ไม่ยาก ผู้อ่านคงสามารถคิดในใจเพื่อหาคำตอบได้ว่า

$$y = 2x - 3 \qquad (1.1)$$

เป็นความสัมพันธ์ระหว่าง `xs` และ `ys` ตามข้อมูลในเซลล์ด้านบน

แต่ในปัญหาการเรียนรู้เชิงลึก โมเดลที่สร้างขึ้นจะไม่ทราบสมการคำตอบ แต่อาศัยการเรียนรู้จากชุดข้อมูล `xs` และ `ys` และเมื่อผ่านการฝึกแล้วสามารถที่จะพยากรณ์เอาต์พุตที่ถูกต้องหรือใกล้เคียงที่สุดสอดคล้องกับความสัมพันธ์ (1.1) เมื่อป้อนอินพุตเป็นตัวเลขค่าหนึ่ง

โจทย์ข้อนี้ต้องการเพียงโมเดลอย่างง่ายที่สุด คือประกอบด้วยหนึ่งอินพุต หนึ่งเอาต์พุต และหนึ่งเซลล์ ใช้คำสั่งสร้างได้ดังนี้

```
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])])
```

ในขั้นตอนการคอมไพล์ ต้องกำหนดฟังก์ชันสูญเสียและตัวหาค่าเหมาะที่สุด สำหรับปัญหารูปแบบนี้ฟังก์ชันสูญเสียเลือกเป็นแบบ *ค่าผิดพลาดเฉลี่ยกำลังสอง (mean squared error)* และตัวหาค่าเหมาะที่สุดแบบ *ลดค่าเกรเดียนต์สโทแคสติก (stochastic gradient descent)*

```
model.compile(optimizer='sgd', loss='mse')
#loss='mean_squared_error')
```

ในกรรมวิธีการฝึกจะใช้ฟังก์ชัน `model.fit()` โดยป้อนข้อมูล `xs, ys` จากเซลล์ด้านบน ทดลองฝึกเป็นจำนวน 500 รอบ

```
model.fit(xs, ys, epochs=500)
```

> 💡 **ตัวอย่างเอาต์พุต**
>
> Epoch 1/500
> 1/1 [==============================] - 0s 245ms/step - loss: 6.1546
> :
> Epoch 500/500
> 1/1 [==============================] - 0s 3ms/step - loss: 1.7463e-04

สังเกตจากค่าสูญเสียที่พิมพ์ออกในแต่ละรอบของการฝึกจะเห็นว่ามีค่าลดลง แสดงว่าโมเดลกำลังเรียนรู้ความสัมพันธ์ระหว่างอินพุต xs และเอาต์พุต ys และปรับปรุงค่าน้ำหนักในทิศทางที่ลดการสูญเสีย หลังจากการฝึก 500 รอบเสร็จสิ้น ต้องการทดสอบโมเดลกับค่าตัวเลขอินพุตใหม่ที่ไม่ได้ใช้ในการฝึก ตัวอย่างเช่น เมื่อป้อนอินพุต $x = 10$ ท่านคิดว่าค่าของเอาต์พุต y ควรเป็นเท่าไร? คิดคำตอบไว้ในใจก่อนรันเซลล์ด้านล่างนี้

```
print(model.predict([10.0]))
```

```
[[16.96028]]
```

จากสมการ (1.1) ค่าที่ถูกต้องควรเป็น 17 แต่ค่าที่พยากรณ์จากโมเดลจะมีความผิดพลาดไปเล็กน้อย โมเดลโครงข่ายประสาทเทียมไม่ทราบสมการที่แท้จริง แต่ทำงานภายใต้หลักการของความน่าจะเป็น ซึ่งในกรณีนี้เราใช้ข้อมูลการฝึกเพียง 6 ตัวอย่างเท่านั้น ท่านอาจทดลองเพิ่มจำนวนตัวอย่างดูว่าได้ความแม่นยำเพิ่มขึ้นหรือไม่

ตัวอย่างนี้เป็นเพียงปัญหาของเล่นที่แนะนำการใช้ไลบรารี TF เท่านั้น โมเดลที่ใช้ไม่ได้เป็นแบบเชิงลึกแต่อย่างใด ในการใช้งานทั่วไปโจทย์จะมีความซับซ้อนมากกว่านี้ ซึ่งจำเป็นต้องใช้โมเดลที่มีเซลล์และจำนวนชั้นมากขึ้น รวมถึงการเลือกใช้ฟังก์ชันสูญเสียและตัวหาค่าเหมาะที่สุดที่สอดคล้องกับปัญหานั้น

ตัวอย่าง 1.2

ในตัวอย่างนี้จะสาธิตการใช้สถาปัตยกรรมโครงข่ายประสาทเทียมเชิงสังวัตนาการ (convolutional neural network) หรือเรียกย่อว่า CNN ที่จะได้กล่าวถึงโดยละเอียดในบทที่ 4 โจทย์ปัญหาของตัวอย่างนี้คือการจำแนกภาพตัวเลข 0 - 9 ที่เขียนด้วยลายมือ โดยใช้ชุดข้อมูลจาก MNIST (Modified National Institute of Standards and Technology) ที่สามารถโหลดได้จากคำสั่งเมื่อติดตั้ง TF แล้ว ตัวอย่างนี้นอกจาก TF แล้วจะใช้แพ็กเกจสนับสนุนเพิ่มเติม ใช้คำสั่งเพื่อนำเข้าดังนี้

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#%matplotlib inline

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten,
                                    Conv2D, MaxPooling2D
```

> **หมายเหตุ :**
>
> สำหรับสองบรรทัดสุดท้ายในเซลล์ด้านบนเป็นคำสั่งนำเข้าเพื่อให้ใช้งานง่ายขึ้น ตัวอย่างเช่นการพิมพ์เพียง `Sequential()` แทนคำสั่งเต็มรูปแบบ `tf.keras.models.Sequential()`

ชุดข้อมูลของ MNIST ประกอบด้วยภาพตัวเลข 0-9 ที่เขียนด้วยลายมือจำนวน 60,000 ภาพ พร้อมเลเบลสำหรับการฝึก และอีก 10,000 ภาพสำหรับชุดทดสอบ เริ่มต้นโดยการโหลดข้อมูล โดยคอมพิวเตอร์ต้องเชื่อมต่ออินเทอร์เน็ต หากเป็นการโหลดครั้งแรกจะเห็นเอาต์พุตจากเซลล์แสดงความก้าวหน้า แต่ถ้าเคยโหลดแล้วจะไม่มีเอาต์พุต เพราะไฟล์ข้อมูลถูกเก็บอยู่อยู่บนคอมพิวเตอร์ของเราแล้ว

```
mnist_data = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels)
                                  = mnist_data.load_data()
```

ก่อนที่จะใช้ข้อมูลกับโมเดล จะต้องมีการประมวลผลล่วงหน้า (preprocessing) โดยปรับมาตราส่วนของพิกเซลให้อยู่ในช่วง 0 - 1 เขียนเป็นฟังก์ชัน `scale_data()` และเรียกใช้กับข้อมูลใน `train_images` และ `test_images` ได้เอาต์พุตอยู่ในเทนเซอร์ `scaled_train_images, scaled_test_images` ตามลำดับ

```
def scale_data(train_images, test_images):
    train_images = train_images/255.
    test_images = test_images/255.
    return train_images, test_images
```

```
scaled_train_images, scaled_test_images =
                scale_data(train_images, test_images)
```

ทดลองตรวจสอบรูปร่างของเทนเซอร์ `scaled_train_images` พบว่ามีมิติหรือแกน (axis) เท่ากับ 3 คือ จำนวนตัวอย่าง ความกว้างและความสูงของภาพตามลำดับ

```
scaled_train_images.shape
```

```
(60000, 28, 28)
```

ในการใช้งานกับ CNN ต้องการแกนที่ 4 เป็นแชนเนล ใช้คำสั่งเพิ่มแกนใหม่ดังนี้

```
scaled_train_images = scaled_train_images[...,
                                          np.newaxis]
scaled_test_images = scaled_test_images[..., np.newaxis]
```

เมื่อจัดรูปข้อมูลแล้ว เราพร้อมที่จะสร้างโมเดล CNN สำหรับฝึกการจำแนกข้อมูล ในตัวอย่างนี้จะใช้โมเดล 6 ชั้นที่มีโครงสร้างและข้อกำหนดดังนี้

- ใช้อาร์กิวเมนต์ `input_shape` เพื่อกำหนดขนาดของอินพุตที่ป้อนให้กับชั้นแรก
- ชั้นที่สองเป็นแบบเชิงสังวัตนาการที่มีเคอร์เนลขนาด 3 x 3 และตัวกรอง 8 ตัว ใช้ *การเสริมเต็ม (padding)* แบบ `'SAME'` (คือการเพิ่มค่าศูนย์เพื่อให้ขนาดความกว้างและความสูงของเอาต์พุตคงเดิม) และฟังก์ชันการกระตุ้นแบบ ReLU
- ชั้นที่สามเป็นแบบ *ชั้นพูลลิงแบบค่ามากสุด (max pooling)* ขนาดหน้าต่าง 2 x 2 ช่วงก้าว (stride) ใช้ค่าโดยปริยาย
- *ชั้นแบน (flatten)* เพื่อคลี่อินพุตออกเป็นเทนเซอร์มิติเดียว
- ชั้นต่อจากนี้ไปเป็น DNN ประกอบด้วยชั้นแฝง 2 ชั้น แต่ละชั้นมี 64 เซลล์และฟังก์ชันกระตุ้นแบบ ReLU
- ชั้นเอาต์พุตมีเซลล์ 10 หน่วยเพื่อจำแนกตัวเลข 0 - 9 และฟังก์ชันกระตุ้นแบบ *ซอฟต์แมกซ์ (softmax)*

ทั้งหมดเขียนได้เป็นฟังก์ชัน `get_model()` ดังนี้

```
def get_model(input_shape):
    model = Sequential([
        Conv2D(8,(3,3), activation='relu',
            padding='same', input_shape=input_shape),
        MaxPooling2D([2,2]),
        Flatten(),
        Dense(64,activation='relu'),
        Dense(64,activation='relu'),
        Dense(10,activation='softmax')
    ])
    return model
```

เรียกใช้ฟังก์ชันเพื่อสร้างโมเดล

```
model1 = get_model(scaled_train_images[0].shape)
```

ขั้นตอนต่อไปคือการคอมไพล์โมเดลโดยวิธี `compile()` กำหนดประเภทของตัวหาค่าเหมาะที่สุด ฟังก์ชันสูญเสีย และตัววัดสมรรถนะของโมเดล ในตัวอย่างนี้จะใช้ตัวหาค่าเหมาะที่สุด `'adam'` ฟังก์ชันสูญเสีย `'sparse_categorical_crossentropy'` และวัด

สถานะโดย 'accuracy' คือความแม่นยำในการจำแนก

```
model1.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
```

ในการฝึกโมเดลโดยใช้ชุดข้อมูล MNIST จะใช้วิธี `fit()` ของวัตถุ `model` ตั้งค่ารอบการฝึกเท่ากับ 10 และคืนค่าประวัติการฝึกเพื่อใช้ในการพล็อตกราฟการเรียนรู้

```
history = model1.fit(scaled_train_images, train_labels,
                             epochs=10,batch_size=256)
```

> 💡 **ตัวอย่างเอาต์พุต**
>
> Epoch 1/10
> 235/235 [====...====] - 3s 10ms/step - loss: 0.4457 - accuracy: 0.8785
> :
> Epoch 10/10
> 235/235 [====...====] - 2s 10ms/step - loss: 0.0275 - accuracy: 0.9918

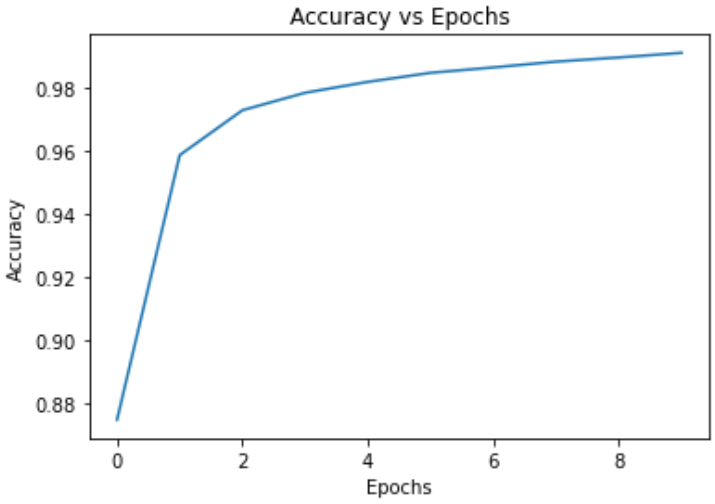
จากเอาต์พุตที่แสดงในแต่ละรอบการฝึก จะพบว่าค่าสูญเสียมีค่าน้อยลงขณะที่ความแม่นยำสูงขึ้นแสดงถึงการเรียนรู้ของโมเดล หลังจากการฝึก 10 รอบได้ค่าสูญเสียประมาณ 0.03 และความแม่นยำประมาณ 99% ผู้อ่านอาจได้เอาต์พุตที่แตกต่างกันบ้างเล็กน้อยในการรันแต่ละครั้ง

ต้องการพล็อตกราฟของความแม่นยำและค่าสูญเสียในการฝึกแต่ละรอบ ในบทนี้จะสาธิตการใช้แพ็กเกจ `panda` โดยโหลดค่า `history` ที่ได้จากโมเดลเข้าใน `DataFrame` และใช้วิธี `plot` เพื่อแสดงกราฟ ผลที่ได้ดังแสดงในรูปที่ 1.5 และ 1.6

```
frame = pd.DataFrame(history.history)
```

```
acc_plot = frame.plot(y="accuracy",
        title="Accuracy vs Epochs", legend=False)
acc_plot.set(xlabel="Epochs", ylabel="Accuracy")
```

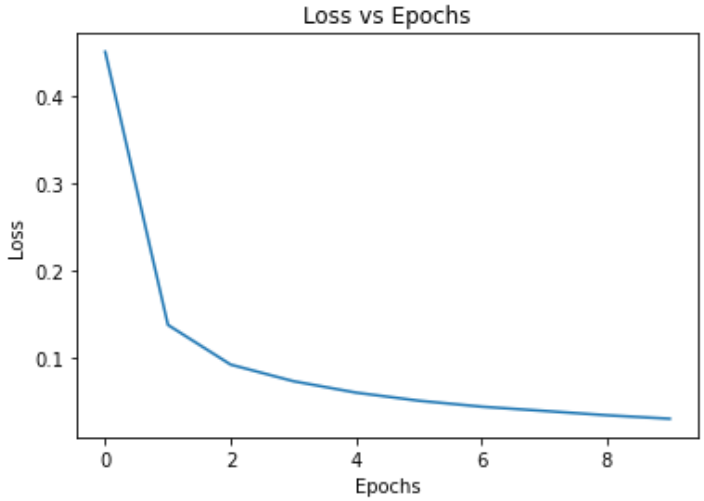
```
[Text(0.5, 0, 'Epochs'), Text(0, 0.5, 'Accuracy')]
```



รูปที่ 1.5 กราฟความแม่นยำเทียบกับจำนวนรอบของการฝึก

```
loss_plot = frame.plot(y="loss",
        title = "Loss vs Epochs",legend=False)
loss_plot.set(xlabel="Epochs", ylabel="Loss")
```

```
[Text(0.5, 0, 'Epochs'), Text(0, 0.5, 'Loss')]
```



รูปที่ 1.6 กราฟความสูญเสียเทียบกับจำนวนรอบของการฝึก

สุดท้ายคือต้องการประเมินสมรรถนะของโมเดลโดยใช้สชุดข้อมูลทดสอบ เริ่มโดยการใช้วิธี `model.evaluate()`

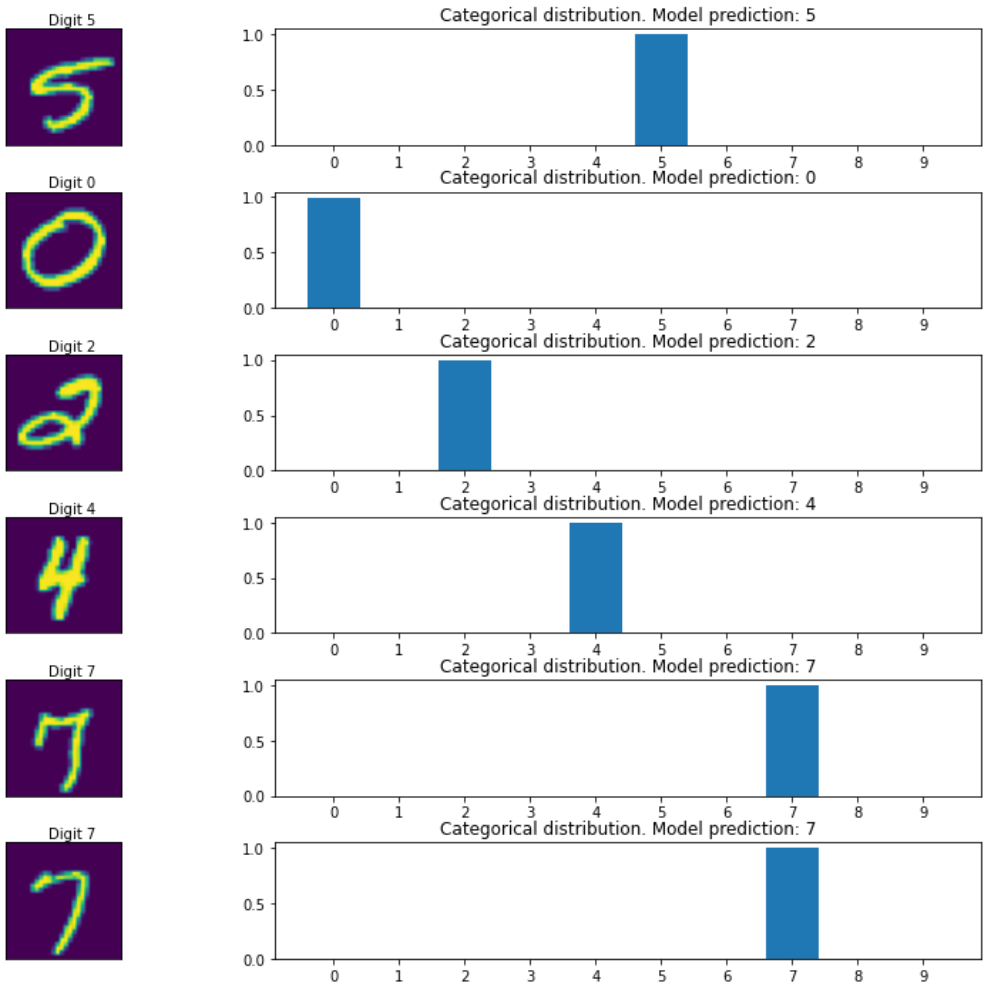
```
test_loss, test_accuracy =
    model1.evaluate(scaled_test_images,test_labels)
print(f"ค่าสูญเสียจากข้อมูลทดสอบ: {test_loss}")
print(f"ความแม่นยำจากข้อมูลทดสอบ: {test_accuracy}")
```

```
313/313 [==============================] - 5s 17ms/step
- loss: 0.0584 - accuracy: 0.9811
ค่าสูญเสียจากข้อมูลทดสอบ: 0.05840187892317772
ความแม่นยำจากข้อมูลทดสอบ: 0.9811000227928162
```

จะพบว่าค่าที่ประเมินได้มีค่าสูญเสียมากกว่าและความแม่นยำน้อยกว่าตัวเลขที่ได้จากข้อมูลการฝึกเล็กน้อย ทั้งนี้เนื่องจากเราได้ใช้ภาพทดสอบที่โมเดลไม่เคยเห็นมาก่อน อย่างไรก็ตามค่าเหล่านี้อยู่ในเกณฑ์ที่ยอมรับได้

สำหรับการตรวจสอบผลการพยากรณ์ของโมเดล ในรูปที่ 1.6 จะสุ่มเลือก 6 ภาพตัวเลขจากชุดข้อมูลทดสอบ พร้อมแสดงภาพและเลเบลของแต่ละภาพทางด้านซ้าย ส่วนทางด้านขวาคือกราฟแท่งแสดงความน่าจะเป็น โดยกราฟแท่งที่มีค่าสูงสุดคือตัวเลขที่โมเดลพยากรณ์ ทดลองรันเซลด้านล่างนี้ตามจำนวนครั้งที่ต้องการ ตรวจสอบว่ามีภาพใดที่โมเดลทายผิด และลองพิจารณาหาสาเหตุที่ภาพนั้นสร้างความสับสนให้กับโมเดล ตัวอย่างเช่นเลข 0 อาจเขียนไม่ครบวง หรือเลข 7 ที่อาจมีการเขียนคล้ายเลข 9

```
num_test_images = scaled_test_images.shape[0]
random_inx = np.random.choice(num_test_images, 6)
random_test_images = scaled_test_images[random_inx, ...]
random_test_labels = test_labels[random_inx, ...]
predictions = model1.predict(random_test_images)
fig, axes = plt.subplots(6, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)
for i, (prediction, image, label) in enumerate(zip(
                    predictions, random_test_images,
                                 random_test_labels)):
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(len(prediction)),
                                       prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution.
            Model prediction: {np.argmax(prediction)}")
plt.show()
```



รูปที่ 1.6 ผลการพยากรณ์เปรียบเทียบกับเลเบลจริง

อีกวิธีการหนึ่งในการสร้างโมเดลบน TF คือใช้ API เชิงฟังก์ชัน ซึ่งในตัวอย่างง่ายนี้จะไม่เห็นข้อได้เปรียบและยุ่งยากขึ้น อย่างไรก็ตามวิธีนี้มีประโยชน์ในกรณีที่ต้องการเข้าถึงเอาต์พุตหรือค่าพารามิเตอร์ในแต่ละชั้นของโมเดลได้อย่างสะดวก หรือการสร้างโมเดลที่มีหลายอินพุตหลายเอาต์พุต หรือมีการสร้างทางลัดระหว่างชั้น (เนื้อหาในบทที่ 4)

โมเดลในตัวอย่าง 1.2 สร้างโดยใช้ API เชิงฟังก์ชันได้ดังนี้

```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
```

```
input_shape=scaled_train_images[0].shape
inputs = Input(input_shape)
x = Conv2D(8, (3,3), activation='relu', padding='same')
                                                (inputs)
x = MaxPooling2D([2,2])(x)
x = Flatten()(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
outputs = Dense(10, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)
```

1.5 การใช้งาน TF เบื้องต้น

ในหัวข้อนี้จะอธิบายและสาธิตการใช้งานฟังก์ชันพื้นฐานของไลบรารี TF เพื่อเสริมความเข้าใจก่อนการศึกษาและพัฒนาในบทต่อไปของหนังสือ

1.5.1 ค่าคงที่ ตัวแปร และเทนเซอร์

ในการใช้งานไลบรารี TF นอกเหนือจากการรันตัวอย่างอย่างง่าย มักจะต้องมีการสร้างตัวแปรที่สามารถทำงานร่วมกับโมเดล หรือว่าในโครงสร้างภายในของโมเดลเองก็จะมีการรับอินพุตและส่งผ่านตัวแปรระหว่างชั้นจนถึงการส่งออกเอาต์พุต ตัวแปรของ TF อาจมีลักษณะคล้ายตัวแปรที่สร้างโดยไลบรารี `numpy` แต่บางครั้งอาจต้องมีการแปลงระหว่างไลบรารีจึงสามารถใช้งานได้ เรามักเรียกตัวแปรใน TF โดยรวมว่าเทนเซอร์ ซึ่งครอบคลุมถึงข้อมูล 3 มิติขึ้นไป

ในที่นี้ตั้งสมมุติฐานว่าได้นำเข้าเทนเซอร์โฟลว์โดยตั้งชื่อว่า `tf` การสร้างเทนเซอร์ค่าคงที่ทำได้โดยคำสั่ง `tf.constant()` เช่น

```
c = tf.constant([[[1,-2],[0,5]],[[3,2],[7,-4]]])
print(c)
```

```
tf.Tensor(
[[[ 1 -2]
  [ 0  5]]

 [[ 3  2]
  [ 7 -4]]], shape=(2, 2, 2), dtype=int32)
```

สมาชิกในเทนเซอร์ที่สร้างโดย `tf.constant()` จะไม่สามารถเปลี่ยนแปลงค่าได้ การแปลงค่าเป็นตัวแปร `numpy` ทำได้โดยใช้เมธอด `numpy()`

```
c.numpy()
```

```
array([[[ 1, -2],
        [ 0,  5]],

       [[ 3,  2],
        [ 7, -4]]], dtype=int32)
```

สำหรับค่าคงที่ที่มีการใช้งานบ่อย คือมีสมาชิกเท่ากับ 0 หรือ 1 ทั้งหมด ใช้คำสั่ง `tf.zeros()` หรือ `tf.ones()` ตามลำดับ

```
c0 = tf.zeros(shape=(2,3))
print(c0)
```

```
tf.Tensor(
[[0. 0. 0.]
 [0. 0. 0.]], shape=(2, 3), dtype=float32)
```

```
c1 = tf.ones(shape=(3,2))
print(c1)
```

```
tf.Tensor(
[[1. 1.]
 [1. 1.]
 [1. 1.]], shape=(3, 2), dtype=float32)
```

หากต้องการเปลี่ยนแปลงค่าในเทนเซอร์ต้องใช้เมธอด `tf.Variable()`

```
v = tf.Variable([[[1,-2],[0,5]],[[3,2],[7,-4]]])
print(v)
```

```
<tf.Variable 'Variable:0' shape=(2, 2, 2) dtype=int32,
numpy=
array([[[ 1, -2],
        [ 0,  5]],

       [[ 3,  2],
        [ 7, -4]]], dtype=int32)>
```

สามารถใส่อาร์กิวเมนต์เพิ่มเติมได้ เช่นถ้าต้องการกำหนดประเภทของตัวแปรให้ใส่เป็นอาร์กิวเมนต์ `dtype` หรือกำหนดชื่อโดย `name=''`

```
v = tf.Variable([[[1,-2],[0,5]],[[3,2],[7,-4]]],
            dtype=tf.float32, name='my_variable')
print(v)
```

```
<tf.Variable 'my_variable:0' shape=(2, 2, 2)
dtype=float32, numpy=
array([[[ 1., -2.],
        [ 0.,  5.]],

       [[ 3.,  2.],
        [ 7., -4.]]], dtype=float32)>
```

การเปลี่ยนแปลงค่าตัวแปรภายหลังทำได้โดยเมธอด `assign()` เช่น

```
v.assign([[[1,-2],[-3,4]],[[-5,6],[7,-8]]])
```

```
<tf.Variable 'UnreadVariable' shape=(2, 2, 2)
dtype=float32, numpy=
array([[[ 1., -2.],
        [-3.,  4.]],

       [[-5.,  6.],
        [ 7., -8.]]], dtype=float32)>
```

การแปลงเป็นตัวแปร `numpy` ทำได้เช่นเดียวกับค่าคงที่

```
v.numpy()
```

```
array([[[ 1., -2.],
        [-3.,  4.]],

       [[-5.,  6.],
        [ 7., -8.]]], dtype=float32)
```

ตัวแปรที่ใช้เป็นอินพุตของโมเดลและส่งผ่านระหว่างชั้นไปยังเอาต์พุตจะเป็นแบบเทนเซอร์ ยกตัวอย่างผังการคำนวณอย่างง่าย คือการบวกตัวแปรเดิมด้วยเทนเซอร์อีกตัวหนึ่ง ผลลัพธ์ที่ได้จะถูกปรับเป็น `tf.tensor` โดยอัตโนมัติ

```
v + [[[1,0],[0,1]],[[0,1],[1,0]]]
```

```
<tf.Tensor: shape=(2, 2, 2), dtype=float32, numpy=
array([[[ 2., -2.],
        [-3.,  5.]],

       [[-5.,  7.],
        [ 8., -8.]]], dtype=float32)>
```

เช่นเดียวกับเอาต์พุตของแต่ละชั้นที่สร้างโดยวิธี API เชิงฟังก์ชันจะเป็นเทนเซอร์ประเภท `KerasTensor`

```
input_shape=scaled_train_images[0].shape
inputs = Input(input_shape)
x = Conv2D(8, (3,3), activation='relu', padding='same')
                                                (inputs)
print(x)
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28,
8), dtype=tf.float32, name=None),
name='conv2d_5/Relu:0', description="created by layer
'conv2d_5'")
```

1.5.2 การเข้าถึงชั้นของโมเดล

ยกตัวอย่างการสร้างโมเดลโดยวิธี API เชิงฟังก์ชัน (สังเกตว่ามีการใช้อาร์กิวเมนต์เพื่อระบุชื่อให้กับแต่ละชั้นด้วย)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense,
                        Flatten, Conv1D, MaxPooling1D

input_shape=scaled_train_images[0].shape
inputs = Input(shape=(32,1), name='input_layer')
x = Conv1D(3,5, activation='relu',name='conv_layer')
                                          (inputs)
x = MaxPooling1D(3, name='pooling_layer')(x)
x = Flatten(name='flatten_layer')(x)
x = Dense(32, activation='relu', name='dense_layer')(x)
outputs = Dense(1, activation='sigmoid',
                    name='output_layer')(x)

model = Model(inputs=inputs, outputs=outputs)
```

ใช้คำสั่งเพื่อพิมพ์ข้อมูลของแต่ละชั้นออกเอาต์พุต

```
print(model.layers)
```

```
[<keras.engine.input_layer.InputLayer object at
0x7fd6c1d9e070>, <keras.layers.convolutional.Conv1D
object at 0x7fd6c18eaac0>,
<keras.layers.pooling.MaxPooling1D object at
0x7fd6c18fe0d0>, <keras.layers.core.flatten.Flatten
object at 0x7fd6c1f98310>,
<keras.layers.core.dense.Dense object at
0x7fd6c1b156d0>, <keras.layers.core.dense.Dense object
at 0x7fd6c14b0c10>]
```

โดยชั้นจะเรียงลำดับจากตัวชี้ 0 ถึง L-1 โดย L คือจำนวนชั้นทั้งหมด เราสามารถเข้าถึงแต่ละชั้นได้โดยใช้ตัวชี้

```
print(model.layers[1])
```

```
<keras.layers.convolutional.Conv1D object at
0x7fd6c18eaac0>
```

การเข้าถึงชั้นของโมเดลในลักษณะนี้ทำให้สามารถเจาะลึกข้อมูลของชั้นที่ต้องการตรวจสอบได้ เช่นค่าน้ำหนักและค่าเอนเอียง โดยค่าที่แสดงเป็นค่าที่กำหนดเริ่มต้นเนื่องจากยังไม่ได้ผ่านการฝึก

```
print(model.layers[1].weights)
```

```
[<tf.Variable 'conv_layer/kernel:0' shape=(5, 1, 3)
dtype=float32, numpy=
array([[[ 0.38343966, -0.4432994 , -0.24529245]],

       [[-0.37581664,  0.4583447 ,  0.18242645]],

       [[ 0.42750007,  0.44005865, -0.24128446]],

       [[-0.44238713,  0.20108515, -0.0286926 ]],

       [[ 0.07358408, -0.22557601, -0.5402513 ]]],
dtype=float32)>, <tf.Variable 'conv_layer/bias:0' shape=
(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>]
```

สามารถเปลี่ยนจาก `weights` เป็น `kernel` หรือ `bias` หากต้องการเฉพาะค่าน้ำหนักหรือค่าเอนเอียง หากต้องการค่าน้ำหนักและค่าเอนเอียงในรูปของแอเรย์ `numpy` ใช้เมธอด `get_weights()`

```
print(model.layers[1].get_weights())
```

```
[array([[[ 0.38343966, -0.4432994 , -0.24529245]],

       [[-0.37581664,  0.4583447 ,  0.18242645]],

       [[ 0.42750007,  0.44005865, -0.24128446]],

       [[-0.44238713,  0.20108515, -0.0286926 ]],

       [[ 0.07358408, -0.22557601, -0.5402513 ]]],
dtype=float32), array([0., 0., 0.], dtype=float32)]
```

หากในการสร้างโมเดลมีการตั้งชื่อแต่ละชั้นไว้ วิธีการหนึ่งที่สะดวกในการเข้าถึงแต่ละชั้นของโมเดลคือระบุชื่อ

```
print(model.get_layer('conv_layer').bias)
```

```
<tf.Variable 'conv_layer/bias:0' shape=(3,)
dtype=float32, numpy=array([0., 0., 0.], dtype=float32)>
```

1.5.3 การเข้าถึงเทนเซอร์ในโมเดล

สังเกตว่าค่าของพารามิเตอร์ในโมเดลจะเป็นชนิด `tf.Variable` แตกต่างจากข้อมูลที่ถ่ายทอดจากอินพุตสู่เอาต์พุตในโมเดล จะเป็นชนิดเทนเซอร์

```
print(model.get_layer('conv_layer').input)
print(model.get_layer('conv_layer').output)
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 32, 1),
dtype=tf.float32, name='input_layer'),
name='input_layer', description="created by layer
'input_layer'")
KerasTensor(type_spec=TensorSpec(shape=(None, 28, 3),
dtype=tf.float32, name=None), name='conv_layer/Relu:0',
description="created by layer 'conv_layer'")
```

การเข้าถึงข้อมูลเทนเซอร์ที่ไหลในโมเดลมีประโยชน์ในกรณีที่ต้องการสร้างโมเดลใหม่ โดยให้อินพุตเป็นค่าจากเอาต์พุตของบางชั้นในอีกโมเดลหนึ่ง ซึ่งเป็นวิธีการที่สนับสนุนการถ่ายโอนการเรียนรู้ในบทที่ 4 ในส่วนนี้จะสาธิตวิธีการโดยสังเขปเท่านั้น

จากโมเดลด้านบน สมมุติว่าต้องการเปลี่ยนจากเอาต์พุตการจำแนกทวิภาคในด้านบนเป็นเอาต์พุตที่สามารถจำแนกได้ 10 ประเภท โดยชั้นที่อยู่เหนือขึ้นไปยังคงเดิม ทำได้โดยเข้าถึงเอาต์พุตของชั้นก่อนหน้าเอาต์พุตดังนี้

```
dense_output = model.get_layer('dense_layer').output
```

สร้างโมเดลใหม่ชื่อ `model2` ประกอบด้วยชั้นทุกชั้นของ `model` ยกเว้นชั้นเอาต์พุต

```
model2 = Model(inputs=model.input, outputs=dense_output)
```

หลังจากนั้นสร้าง `model3` ประกอบด้วยทุกชั้นของ `model2` และเพิ่มชั้นเอาต์พุตใหม่สำหรับการจำแนก 10 ประเภท

```
model3 = Sequential([
    model2,
    Dense(10, activation='softmax',
          name='new_output_layer')
])
```

หรือหากใช้ API เชิงฟังก์ชัน สามารถสร้างได้โดยคำสั่งดังนี้

```
new_outputs = Dense(10, activation='softmax',
                    name='new_output_layer')(model2.output)
model3 = Model(inputs=model2.input, outputs=new_outputs)
```

1.5.4 พื้นฐานการถ่ายโอนการเรียนรู้

การถ่ายโอนการเรียนรู้ (transfer learning) คือวิธีการที่มีการใช้งานอย่างแพร่หลาย ยกตัวอย่างผู้พัฒนาโมเดล CNN ที่มีความลึกมาก และใช้ข้อมูลขนาดใหญ่เพื่อฝึกโมเดลเป็นเวลานานโดยใช้ทรัพยากรของห้องปฏิบัติการที่มีสมรรถนะสูง สมมุติว่าโมเดลที่ผ่านการฝึกเรียกว่าโมเดล A สามารถจำแนกวัตถุบนโลกได้ 1000 ประเภท และผู้พัฒนาได้ยินยอมให้กับบุคคลทั่วไปสามารถดาวน์โหลดโมเดลไปใช้งานได้ เราเป็นนักพัฒนาสำหรับโครงงานขนาดเล็กกว่าที่ต้องการจำแนกวัตถุเพียง 10 ประเภท มีข้อมูลสำหรับฝึกไม่มากนักและเครื่องคอมพิวเตอร์ก็มีสมรรถนะไม่สูง แทนที่จะสร้างโมเดลเองและฝึกโดย

ข้อมูลจำนวนน้อย เราสามารถนำเอาโมเดล A มาใช้งานโดยเปลี่ยนชั้นส่วนเอาต์พุตเป็นการจำแนก 10 ประเภท หรืออาจมีการเปลี่ยนชั้นก่อนหน้าเอาต์พุตจำนวนหนึ่ง และฝึกเฉพาะชั้นส่วนที่เปลี่ยนแปลง โดยไม่ต้องการแตะต้องพารามิเตอร์ส่วนใหญ่ของโมเดล A ซึ่งผ่านการฝึกมาอย่างดีแล้ว ในหัวข้อย่อยนี้จะกล่าวเฉพาะพื้นฐานการดำเนินการเพื่อสนับสนุนการถ่ายโอนการเรียนรู้ คือการปิดกั้นพารามิเตอร์ของโมเดลในส่วนที่ไม่ต้องการฝึก

เพื่อความต่อเนื่องในการอธิบายจะคัดลอกโค้ดการสร้างโมเดลในตัวอย่าง 1.2 มาใส่ในเซลล์ด้านล่างนี้เพื่อใช้เป็นตัวอย่าง โดยเพิ่มอาร์กิวเมนต์เป็นชื่อของแต่ละชั้นเพื่อความสะดวกในการเข้าถึง

```
input_shape=scaled_train_images[0].shape
inputs = Input(input_shape)
x = Conv2D(8, (3,3), activation='relu', padding='same',
                            name='conv2d_layer')(inputs)
x = MaxPooling2D([2,2], name='maxpool2d_layer')(x)
x = Flatten(name='flatten_layer')(x)
x = Dense(64, activation='relu', name='dense_layer1')(x)
x = Dense(64, activation='relu', name='dense_layer2')(x)
outputs = Dense(10, activation='softmax',
                             name='output_layer')(x)

model = Model(inputs=inputs, outputs=outputs)
```

สมมุติว่าต้องการปิดกั้นชั้น `conv2d_layer` และ `dense1_layer` ไม่ให้มีการฝึกพารามิเตอร์ ทำได้โดยเพิ่มอาร์กิวเมนต์ `trainable=False` ให้กับชั้นที่ไม่ต้องการฝึก (สำหรับชั้น `maxpool2d_layer` และ `flatten_layer` ไม่มีพารามิเตอร์การเรียนรู้)

```
x = Conv2D(8, (3,3), activation='relu', padding='same',
           name='conv2d_layer',trainable=False)(inputs)
x = MaxPooling2D([2,2], name='maxpool2d_layer')(x)
x = Flatten(name='flatten_layer')(x)
x = Dense(64, activation='relu', name='dense_layer1',
                                    trainable=False)(x)
x = Dense(64, activation='relu', name='dense_layer2')(x)
outputs = Dense(10, activation='softmax',
                             name='output_layer')(x)

model = Model(inputs=inputs, outputs=outputs)
```

ในกรณีที่ต้องการปิดกั้นการเรียนรู้ของบางชั้นหลังจากสร้างโมเดลแล้ว ใช้เมธอด `get_layer()` เพื่อเข้าถึงชั้นนั้นและตั้งค่า `trainable=False` ได้ดังนี้

```
model.get_layer('conv2d_layer').trainable=False
model.get_layer('dense_layer1').trainable=False
```

เมื่อใช้คำสั่ง `model.summary()` จะเห็นในส่วนท้ายว่าพารามิเตอร์ 100,416 ตัวจะถูกปิดกั้นการฝึก โดยจะมีค่าเท่ากับจำนวนพารามิเตอร์ของชั้น `conv2d_layer` และ `dense_layer1` รวมกัน

```
model.summary()
```

```
Model: "model_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_layer (Conv2D)       (None, 28, 28, 8)         80

 maxpool2d_layer (MaxPooling  (None, 14, 14, 8)        0
 2D)

 flatten_layer (Flatten)     (None, 1568)              0

 dense_layer1 (Dense)        (None, 64)                100416

 dense_layer2 (Dense)        (None, 64)                4160

 output_layer (Dense)        (None, 10)                650

=================================================================
Total params: 105,306
Trainable params: 4,810
Non-trainable params: 100,496
_________________________________________________________________
```

หลังจากนี้เราสามารถตัดชั้นเอาต์พุตและชั้นก่อนหน้าเอาต์พุตออก และเพิ่มชั้นใหม่ที่มีจำนวนเอาต์พุตการจำแนกตามต้องการ การดำเนินการทิ้งให้เป็นแบบฝึกหัด

1.5.5 การบันทึกและโหลดโมเดล

หลังจากการฝึกโมเดลจนได้ผลเป็นที่น่าพอใจ เราสามารถบันทึกโมเดลลงบนดิสก์รูปแบบมี 2 ประเภทคือของ TF (ค่าโดยปริยาย) หรือรูปแบบ *hdf5 (hierarchical data format)* เมื่อใส่นามสกุล `.h5` สมมุติว่าต้องการบันทึกโมเดลชื่อ `model` ใช้คำสั่งดังนี้

```
model.save('model_filename') # TF format
model.save('model_filename.h5') # hdf5 format
```

ไฟล์จะถูกบันทึกในไดเรคทอรีที่ใช้งานอยู่ โดยกรณีรูปแบบของ TF จะสร้างไดเรคทอรีย่อยตามชื่อไฟล์ที่ใส่เป็นอาร์กิวเมนต์และบันทึกข้อมูลเป็นกลุ่มของไฟล์ ส่วนรูปแบบ hdf5 จะบันทึกเป็นไฟล์เดี่ยว

สำหรับการโหลดโมเดลจะต้องนำเข้าฟังก์ชัน `load_model` และเรียกใช้โดยใส่อาร์กิวเมนต์เป็นชื่อของโมเดลที่บันทึกไว้

```
from tensorflow.keras.models import load_model
new_model = load_model('model_filename')
new_model2 = load_model('model_filename.h5')
```

เราสามารถตั้งค่าฟังก์ชันเรียกกลับให้บันทึกโมเดลในขณะฝึกตามเงื่อนไขที่กำหนดได้ด้วย โดยเลือกบันทึกโมเดลทั้งหมดหรือเฉพาะค่าพารามิเตอร์ก็ได้ ศึกษาเพิ่มเติมได้จากเพจอ้างอิงของ TF

สำหรับข้อมูลการฝึกที่อยู่ในไฟล์ `history` สามารถจัดเก็บได้ในรูปแบบไบนารีหรือ json ในที่นี้เราจะแสดงโค้ดสำหรับอย่างหลัง

```
import json
with open('history.json', 'w') as file:
    json.dump(history.history, file)
```

ส่วนการโหลดเข้าสู่ตัวแปรดิกชันนารีใช้โค้ดดังนี้

```
with open('history.json') as json_file:
    saved_history = json.load(json_file)
```

สังเกตว่าในการใช้งานดิกชันนารีที่โหลดเข้ามาคือ `saved_history` จะเหมือนกับ `history.history` จากการฝึก

1.6 โครงสร้างของหนังสือ

เนื้อหาในหนังสือนี้เป็นการศึกษาการเรียนรู้เชิงลึกในการแก้ปัญหาหลายรูปแบบ โดยจำกัดขอบเขตเฉพาะการเรียนรู้แบบมีผู้สอน หลังจากการแนะนำเบื้องต้นในบทนี้แล้ว บทที่ 2 กล่าวถึงพื้นฐานโครงข่ายประสาทเทียมตั้งแต่แบบไม่มีชั้นแฝงที่รู้จักกันในชื่อโมเดลการถดถอยเชิงเส้นและการถดถอยลอจิสติก ซึ่งทำให้เข้าใจหลักการของผังการไหลด้านหน้าและการแพร่กระจายย้อนกลับ การเรียนรู้ของโมเดลแท้จริงแล้วคือการปรับค่าพารามิเตอร์ในทิศทางที่ทำให้ค่าสูญเสียต่ำสุด ประโยชน์ของฟังก์ชันกระตุ้นแบบไม่เป็นเชิงเส้น หลังจากการปรับปรุงโดยเพิ่มชั้นแฝงทำให้เป็นโมเดลที่มีความลึกเรียกว่า DNN เพื่อความเข้าใจกลไกภายใน ในบทนี้จะกล่าวถึงการเขียนโค้ดภาษาไพธอนโดยไม่ใช้ไลบรารี TF สำหรับ DNN ที่มีความซับซ้อนไม่มาก แต่ในกรณีทั่วไปแนะนำให้ใช้ไลบรารี เมื่อทดสอบโมเดล DNN ในการจำแนกภาพพบว่าได้ความแม่นยำไม่สูงมาก ทั้งนี้เนื่องจาก DNN ไม่มีการใช้ข้อมูลเชิงพื้นที่อันเป็นองค์ประกอบสำคัญของข้อมูลภาพ

ในบทที่ 3 กล่าวถึงการปรับปรุงโครงข่ายประสาทเทียมโดยปรับแต่งไฮเปอร์พารามิเตอร์เพื่อให้ได้ประสิทธิภาพที่ดี เริ่มต้นจากการอธิบายค่าเอนเอียงและความแปรปรวนของข้อมูลที่เป็นมูลเหตุสำคัญของการฟิตเกิน วิธีที่นิยมใช้แก้ปัญหาคือคือการทำเรกูลาร์ไรเซชันและดรอปเอาต์ การหยุดฝึกตั้งแต่ช่วงต้น การจัดการข้อมูลก่อนเริ่มต้นการฝึกช่วยทำให้การฝึกโมเดลทำได้ง่ายขึ้น คือการทำอินพุตให้เป็นบรรทัดฐาน และการกำหนดค่าเริ่มต้นของพารามิเตอร์การเรียนรู้ สำหรับการเลือกตัวหาค่าเหมาะที่สุดมีส่วนช่วยในการลดค่าเกรเดียนต์ ตั้งแต่วิธีการพื้นฐานคือ SGD ที่สามารถเพิ่มโมเมนตัม วิธีการ RMSprop และ Adam การปรับอัตราการเรียนรู้ในการฝึกทำได้ทั้งแบบเลือกค่าคงที่หรือปรับค่าอัตโนมัติ โดยทั่วไปจะให้ค่าลดลงตามจำนวนรอบการฝึก การทำกลุ่มให้เป็นบรรทัดฐานเป็นอีกวิธีการหนึ่งที่สามารถช่วยในการฝึกโมเดล

จากการทดสอบโมเดล DNN พบว่ามีความแม่นยำในการจำแนกข้อมูลภาพไม่สูงมาก สถาปัตยกรรมที่เหมาะสมกับการใช้ในงานที่เกี่ยวข้องกับข้อมูลภาพคือ CNN ซึ่งเป็นเนื้อหาของบทที่ 4 ในส่วนต้นของบทกล่าวถึงการทำสังวัตนาการที่ช่วยในการดึงลักษณะเด่นของภาพ เช่นขอบของวัตถุ ซึ่งหัวใจสำคัญของสถาปัตยกรรม CNN คือการเพิ่มชั้นสังวัตนาการให้กับโครงข่ายเพื่อประสิทธิภาพในการจำแนกข้อมูลภาพได้ดีขึ้น โดยมักใช้งานร่วมกับชั้นพูลลิงที่สามารถลดขนาดของข้อมูลลง อย่างไรก็ตามโมเดล CNN ที่เพิ่มชั้นสังวัตนาการจำนวนมากทำให้มีความลึกมากและประสบปัญหาในการฝึก การใช้บล็อกส่วนตกค้างใน ResNets ช่วยบรรเทาอุปสรรคในการฝึกโมเดล ส่วนมอดูลอินเซปชันเป็นตัวช่วยสร้างคอขวดเพื่อลดจำนวนการดำเนินการทางคณิตศาสตร์ลงได้ เป็นองค์ประกอบหลักของโมเดลอินเซปชัน ในกรณีที่มีข้อมูลจำนวนไม่มากหรือไม่ต้องการฝึกโมเดลเป็นเวลา

นาน วิธีที่เรียกว่าการถ่ายโอนการเรียนรู้ช่วยให้เราสามารถนำโมเดลที่ถูกพัฒนาและฝึกแล้วมาใช้ในปัญหาของเราได้ โดยแทนที่บางชั้นในส่วนท้ายของโมเดลและฝึกเฉพาะส่วนนั้น ในส่วนท้ายของบทนี้เป็นการประยุกต์ใช้โมเดล CNN โดยกล่าวแนะนำขั้นตอนวิธี YOLO เพื่อหาตำแหน่งของวัตถุในภาพจากกล้องแบบเรียลไทม์ ซึ่งเป็นการขยายการสังวัตนาการในโมเดล CNN จนถึงชั้นเอาต์พุต ทำให้สามารถตรวจจับวัตถุได้ในครั้งเดียวของการประมวลผล ส่วนงานอีกรูปแบบหนึ่งที่กล่าวถึงคือการรู้จำใบหน้าของบุคคล

การใช้งานการเรียนรู้เชิงลึกเพื่อแก้ปัญหาที่เกี่ยวข้องกับข้อมูลลำดับเป็นอีกแนวทางหนึ่งที่ได้รับความสนใจอย่างมาก เช่นการประมวลผลภาษาธรรมชาติ การแปลภาษาโดยเครื่อง การกำเนิดบทกวีหรือดนตรี การจำแนกอารมณ์ของบทความ การสั่งงานโดยเสียง การวิเคราะห์ข้อมูลฐานเวลา ลักษณะงานเหล่านี้ต้องการการส่งผ่านข้อมูลสถานะตามขั้นเวลา ในบทที่ 5 เป็นการศึกษาโมเดลลำดับตั้งแต่สถาปัตยกรรมพื้นฐานเรียกว่า RNN จนถึงโมเดลที่มีพัฒนาให้มีความจำระยะยาว คือ GRU และ LSTM

ภาคผนวก A รวบรวมหลักการและวิธีการที่สนับสนุนเนื้อหาหลักของหนังสือแต่มีรายละเอียดมากเกินกว่าจะแทรกในตัวบท ส่วนภาคผนวก B กล่าวถึงการติดตั้งซอฟต์แวร์ที่ใช้ในหนังสือ

1.7 สรุปท้ายบท

ในบทแรกนี้เป็นการกล่าวแนะนำการเรียนรู้เชิงลึก ซึ่งเป็นเซตย่อยของการเรียนรู้ของเครื่องและปัญญาประดิษฐ์ โดยสมรรถนะของฮาร์ดแวร์คอมพิวเตอร์ที่พัฒนาขึ้นอย่างต่อเนื่องผนวกกับปัญหาปัจจุบันที่มักเกี่ยวข้องกับข้อมูลขนาดใหญ่ ทำให้โครงข่ายประสาทเทียมและการเรียนรู้เชิงลึกถูกนำมาใช้ได้อย่างมีประสิทธิภาพและประสบผลสำเร็จในหลายสาขา ในส่วนท้ายของบทนี้กล่าวถึงซอฟต์แวร์ที่ใช้เป็นกรอบการพัฒนา ในหนังสือจะใช้ไลบรารี TF ซึ่งจากสถิติปัจจุบันพบว่า ได้รับความนิยมอย่างสูง ได้สาธิตการสร้างตัวแปรประเภทต่างๆ และโมเดลพื้นฐานโดย TF ซึ่งจะได้ขยายไปสู่โมเดลที่ซับซ้อนขึ้นต่อไป

โจทย์ปัญหา

1-1 จากตัวอย่าง 1.1 ทดลองเปลี่ยนฟังก์ชันในสมการ (1.1) เช่น $y = -3x + 2$ สร้างแอเรย์อินพุตและเอาต์พุต 10 ค่าเพื่อฝึกโมเดล ตรวจสอบผลการพยากรณ์

1-2 ดัดแปลงโมเดลในตัวอย่าง 1.1 ให้สามารถพยากรณ์สมการ 2 ตัวแปรอิสระ เช่น $y = x_1 - 2x_2 + 4$

1-3 ทดสอบการสร้างโมเดลในตัวอย่าง 1.2 โดยใช้ API เชิงฟังก์ชัน

1-4 สร้างเทนเซอร์ค่าคงที่มีมิติเท่ากับ 2 x 3 x 4 x 5 สมาชิกทั้งหมดมีค่าเท่ากับ 6

1-5 สร้างเทนเซอร์ที่มีค่าและมิติดังเช่นโจทย์ข้อ 1-4 แต่เป็นแบบตัวแปร หลังจากนั้นเปลี่ยนค่าสมาชิกตำแหน่ง (1,1,3,2) ให้มีค่าเท่ากับ 10 (ค่าดัชนีของไพธอนเริ่มจาก 0)

1-6 หากต้องการดูค่าน้ำหนักของชั้น `Dense(64)` ชั้นแรกของโมเดลในตัวอย่าง 1.2 ต้องแก้ไขฟังก์ชัน `get_model()` อย่างไร?

1-7 จากโมเดลในหัวข้อ 1.5.4 หลังจากปิดกั้นการฝึกของชั้น `conv2d_layer` และ `dense_layer1` แล้ว เขียนโค้ดเพื่อถอดชั้น `dense_layer2` และ `output_layer` ออก แทนที่ด้วยชั้น `output_layer` ใหม่ที่เป็นการจำแนกทวิภาค