Machine Learning for Quantum Chemistry

การเรียนรู้ของเครื่องสำหรับเคมีควอนตัม

รังสิมันต์ เกษแก้ว

การเรียนรู้ของเครื่องสำหรับเคมีควอนตัม

Machine Learning for Quantum Chemistry

รังสิมันต์ เกษแก้ว

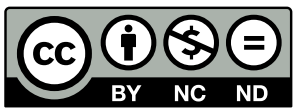
ฉบับพิมพ์ครั้งที่ 1

รังสิมันต์ เกษแก้ว

การเรียนรู้ของเครื่องสำหรับเคมีควอนตัม
Machine Learning for Quantum Chemistry

ปกด้านหน้า: แขนกลและโมเลกุล
ปกด้านหลัง: ปริภูมิเคมีของชุดข้อมูล QM9

ฉบับพิมพ์ครั้งที่ 1 ปรับปรุงล่าสุด 21 ธันวาคม พ.ศ. 2565



ซอร์สโค้ดของหนังสือเล่มนี้ถูกเขียนขึ้นโดยใช้ภาษา LaTeX เผยแพร่ที่ https://github.com/rangsimanketkaew/ml-qm-book และไฟล์ PDF ถูกสร้างขึ้นโดยใช้ XeLaTeX เผยแพร่ที่ https://rangsimanketkaew.github.io/ml-qm-book

หากต้องการติดต่อผู้เขียน กรุณาส่งอีเมลมาที่ rangsiman1993[at]gmail[dot]com

สารบัญ

คำนำ

ในปัจจุบันได้มีการนำการเรียนรู้ของเครื่อง (Machine Learning) ไปใช้ประโยชน์ในหลากหลายด้าน เช่น คอมพิวเตอร์วิทัศน์, คอมพิวเตอร์กราฟฟิก, ภาษาศาสตร์, เศรษฐศาสตร์, อุตสาหกรรม, เกษตรกรรม รวมไปถึงวิทยาศาสตร์และวิศวกรรม โดยสาขาเคมีนั้นก็เป็นอีกหนึ่งศาสตร์ที่ได้มีการนำ Machine Learning เข้ามาประยุกต์ใช้มาเป็นระยะเวลานานนับตั้งแต่ช่วงปี ค.ศ. 1990 โดยนักวิจัยได้พัฒนาเทคนิค Machine Learning เพื่อใช้ในการศึกษาคุณสมบัติของโมเลกุลอินทรีย์และอนินทรีย์ ศึกษาโครงสร้างโปรตีน ออกแบบโมเลกุลยา รวมไปถึงศึกษาปฏิกิริยาเคมีและตัวเร่งปฏิกิริยา

ผู้เขียนเล็งเห็นว่าการประยุกต์ใช้ Machine Learning กับเคมีควอนตัมนั้นเป็นสิ่งใหม่ที่หลายคนต่างก็ให้ความสนใจ ทั้งนักศึกษา อาจารย์ และนักวิจัย ดังนั้นผู้เขียนจึงได้เรียบเรียงหนังสือเล่มนี้ขึ้นมาเพื่อใช้เป็นแนวทางสำหรับการศึกษาและทำงานวิจัยทางด้านนี้ โดยหนังสือเล่มนี้ครอบคลุมทฤษฎีและเทคนิค Machine Learning, ทฤษฎีโครงสร้างเชิงอิเล็กทรอนิกส์ซึ่งเป็นสาขาหนึ่งของเคมีควอนตัมที่ช่วยให้นักวิจัยเข้าใจองค์ความรู้พื้นฐานของอะตอมและโมเลกุล, หัวข้อเคมีควอนตัมที่สามารถนำการเรียนรู้ของเครื่องไปประยุกต์ใช้ได้ รวมไปถึงรายละเอียดเชิงลึกในการพัฒนาวิธีคำนวณแบบใหม่เพื่อปรับปรุงประสิทธิภาพการพยากรณ์ของแบบจำลอง Machine Learning

ผู้เขียนหวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยให้ผู้อ่านทุกท่านได้รับความรู้และความเข้าใจที่ครบถ้วนเกี่ยวกับ Machine Learning สำหรับเคมีควอนตัม

รังสิมันต์ เกษแก้ว
9 ตุลาคม พ.ศ. 2565

กิตติกรรมประกาศ

ความรู้และแรงบันดาลใจในการเขียนหนังสือเล่มนี้ของผู้เขียนมาจากแรงผลักดันและการสนับสนุนของบุคคลหลายท่าน การเขียนหนังสือเล่มนี้จะไม่เกิดขึ้นหรือสำเร็จไม่ได้ถ้าหากขาดบุคคลดังต่อไปนี้

ครอบครัวของผู้เขียนที่สนับสนุนให้ผู้เขียนได้ทำตามความฝันในการเรียนต่อระดับอุดมศึกษา ทั้งในระดับปริญญาโทและปริญญาเอก โดยเฉพาะการเห็นคุณค่าของการเรียนและการทำงานวิจัยทางด้านวิทยาศาสตร์พื้นฐาน

รศ.ดร. ยุทธนา ตันติรุ่งโรจน์ชัย บุคคลผู้เป็นต้นแบบด้านการเรียนและเป็นผู้สร้างแรงบันดาลใจให้ผู้เขียนเรียนต่อต่างประเทศและทำงานวิจัยทางด้านเคมีทฤษฎีและเคมีคอมพิวเตอร์

อาจารย์และเพื่อน ๆ ในช่วงมัธยมศึกษา (ตอนต้น-ปลาย) ที่โรงเรียนพนัสพิทยาคาร และช่วงปริญญาตรี-โท ที่มหาวิทยาลัยธรรมศาสตร์ สำหรับความทรงจำอันดีงามและความเป็นกัลยาณมิตรที่ดีเสมอมา

เพื่อน ๆ ที่เมืองซูริค ประเทศสวิตเซอร์แลนด์ สำหรับมิตรภาพอันดีงาม รอยยิ้มและเสียงหัวเราะที่เกิดขึ้น รวมไปถึงกิจกรรมที่ได้ทำร่วมกันในระหว่างที่ผู้เขียนกำลังศึกษาปริญญาเอกซึ่งเป็นช่วงเวลาเดียวกันกับที่ผู้เขียนกำลังเขียนหนังสือเล่มนี้

เพื่อนร่วมงานทั้งนักศึกษาปริญญาโท ปริญญาเอก และนักวิจัยหลังปริญญาเอกของกลุ่มวิจัยของผู้เขียนที่ภาควิชาเคมี มหาวิทยาลัยแห่งซูริค สำหรับการแลกเปลี่ยนความรู้ ไอเดียใหม่ ๆ และการช่วยเหลือเกี่ยวกับงานวิจัยทางด้านเคมีทฤษฎี

รังสิมันต์ เกษแก้ว

คำแนะนำในการอ่านหนังสือเล่มนี้

ผู้เขียนเรียบเรียงหนังสือเล่มนี้ขึ้นมาเพื่อให้บุคคลที่สนใจ Machine Learning และเคมีควอนตัมได้ศึกษาเพื่อเป็นแนวทางในการทำงานวิจัยในสาขานี้ต่อไป โดยผู้เขียนได้ศึกษาจากหนังสือต่างประเทศรวมถึงบทความงานวิจัยที่ตีพิมพ์ในวารสารวิชาการชั้นนำทางด้านปัญญาประดิษฐ์และเคมีทฤษฎีที่ได้รับการยอมรับ

หนังสือเล่มนี้เน้นไปที่การอธิบายทฤษฎีประกอบกับสมการทางคณิตศาสตร์ที่ใช้ในเคมีควอนตัมอย่างกระชับ โดยผู้เขียนพยายามเลือกใช้คำและสำนวนที่ไม่เป็นทางการมากนักในการอธิบายเนื้อหาที่ซับซ้อนเพื่อให้ผู้อ่านสามารถเข้าใจได้ง่ายขึ้น ดังนั้นสไตล์การเขียนของผู้เขียนจึงเป็นในเชิงที่ใช้ภาษาพูด โดยมีการใส่ความคิดเห็นส่วนตัว แนวคิดและมุมมองของผู้เขียนที่ได้จากการอภิปรายกับเพื่อนร่วมงานและผู้เชี่ยวชาญในสาขาที่คิดว่าเหมาะสมเข้าไปด้วย

หนังสือเล่มนี้ประกอบไปด้วยเนื้อหาสามส่วน ดังนี้

1. **การเรียนรู้ของเครื่อง**: ในส่วนแรกผู้อ่านจะได้ศึกษาที่มาและความสำคัญของ Machine Learning และการเชื่อมโยงเพื่อนำไปใช้กับเคมีควอนตัมโดยเฉพาะในการทำงานวิจัย ผู้อ่านจะได้ศึกษาอัลกอริทึม Machine Learning แบบต่าง ๆ ทั้งการเรียนรู้แบบมีผู้สอนและแบบไม่มีผู้สอนซึ่งเป็นอัลกอริทึมมาตรฐานที่นักวิจัยใช้เพื่อพัฒนาโมเดลสำหรับการทำนายเอาต์พุตที่ต้องการ นอกจากนี้ผู้อ่านจะได้เรียนรู้ปัญหาของ Machine Learning ที่มักจะพบเจอได้บ่อยและขั้นตอนหรือเทคนิคการแก้ปัญหาดังกล่าว และการเลือกโมเดล Machine Learning เพื่อให้เหมาะสมกับโจทย์ปัญหาที่ต้องการแก้

2. **การทำนายคุณสมบัติเชิงเคมีควอนตัม**: ในส่วนที่สองจะเป็นการอธิบายทฤษฎีของโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ของอะตอมและโมเลกุล ผู้อ่านจะได้ศึกษาระบบอิเล็กตรอน วิธีการคำนวณที่อ้างอิงกับฟังก์ชันคลื่นและความหนาแน่นของอิเล็กตรอน รวมถึงทฤษฎีของคุณสมบัติเชิงโมเลกุลแบบต่าง ๆ ซึ่งเป็นสิ่งที่นักวิจัยเคมีทฤษฎีให้ความสนใจและเกี่ยวข้องโดยตรงกับหัวข้อถัดไปนั่นก็คือทฤษฎีของตัวอธิบายเชิงอิเล็กทรอนิกส์ (Electronic-based Representation) แบบต่าง ๆ ที่ได้รับการพัฒนาตั้งแต่อดีตจนถึงปัจจุบันและถูกนำมาใช้สำหรับการคำนวณลักษณะเฉพาะ (Feature) ของโมเลกุล นอกจากนี้ผู้อ่านจะได้ศึกษาชุดข้อมูลทางเคมีควอนตัมที่เราจะนำมาใช้เป็นอินพุตสำหรับการฝึกสอนโมเดล ผู้อ่านจะได้ศึกษาการพัฒนาโมเดล Machine Learning รวมไปถึงโมเดลเฉพาะทางแบบต่าง ๆ สำหรับเคมีควอนตัมที่ได้รับการพัฒนาในช่วงสิบปีที่ผ่านมาและการวิเคราะห์ผลการทำนายของโมเดลซึ่งจะเป็นประโยชน์ในการทำงานวิจัยและเผยแพร่ผลงานวิชาการต่อไป

3. **ภาคผนวก**: ในส่วนที่สามเป็นภาคผนวกซึ่งจะรวบรวมหัวข้อพื้นฐานที่เป็นประโยชน์ต่อการทำความเข้าใจเนื้อหาหลักในสองส่วนแรก ประกอบไปด้วยพื้นฐานพีชคณิตเชิงเส้นซึ่งเกี่ยวข้องกับเมทริกซ์ การเขียนโปรแกรม Machine Learning โดยผู้อ่านจะได้ฝึกการเขียนโปรแกรมของโครงข่ายประสาท (Neural Network) ด้วยไลบรารี่ TensorFlow เป็นต้น นอกจากนี้ยังมีการแนะนำโปรแกรมเคมีเชิงคำนวณที่ผู้อ่านสามารถใช้งานเพื่อคำนวณคุณสมบัติเชิงโครงสร้างและเชิงอิเล็กทรอนิกส์ของโมเลกุลได้อีกด้วย

ในการอธิบายทฤษฎีนั้นผู้เขียนได้ใส่โค้ดเพื่อให้ผู้อ่านสามารถนำไปเขียนโปรแกรมและทดสอบได้ด้วยตนเองเข้าไปด้วย โดยสามารถดาวน์โหลดโค้ดได้ที่ https://github.com/rangsimanketkaew/ml-qm-book-code ซึ่งจากประสบการณ์ส่วนตัวของผู้เขียนพบว่าการเขียนโปรแกรมนั้นสามารถช่วยให้เข้าใจทฤษฎีต่าง ๆ ทางเคมีควอนตัมได้ง่ายขึ้น รวมไปถึงเข้าใจวิธีคิดในการคำนวณอย่างเป็นขั้นเป็นตอน ดังนั้นผู้เขียนจึงขอแนะนำให้ผู้อ่านศึกษาทฤษฎีควบคู่ไปพร้อมกับโค้ดอย่างละเอียด

ในกรณีที่ผู้อ่านมีข้อเสนอแนะหรือพบข้อผิดพลาดของหนังสือสามารถแจ้งผู้เขียนได้โดยกรอกแบบฟอร์มที่ https://cutt.ly/ml-qm-book-feedback สุดท้ายนี้ผู้เขียนขอให้ผู้อ่านมีความสุขกับการอ่านหนังสือเล่มนี้ครับ :)

รังสิมันต์ เกษแก้ว

If I have seen further, it is by standing on the shoulders of Giants.

- Sir Isaac Newton PRS (1643 - 1726)

ส่วนที่ 1

การเรียนรู้ของเครื่อง

บทที่ 1

การเรียนรู้ของเครื่อง

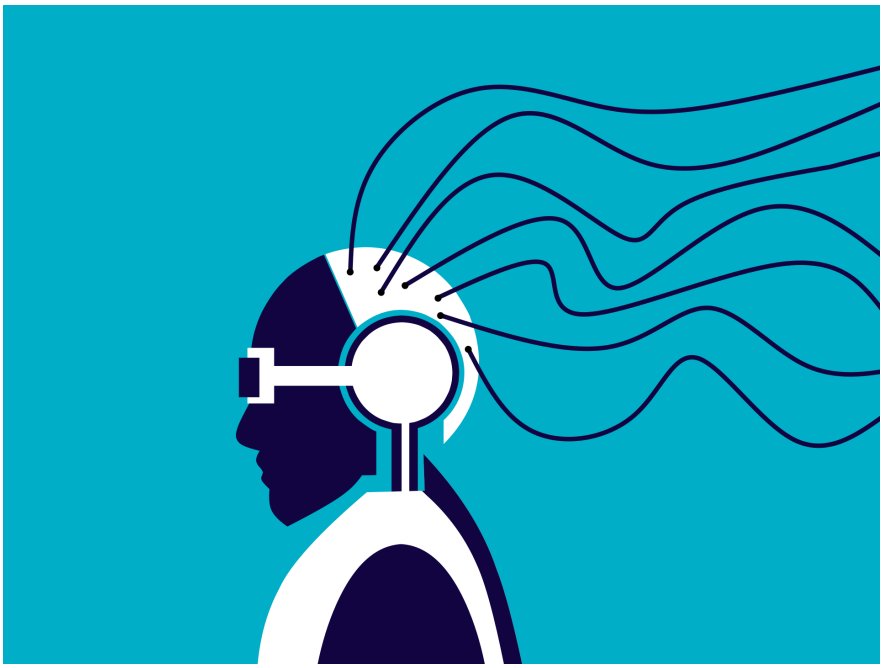
1.1 ความสำคัญของการเรียนรู้ของเครื่อง

การเรียนรู้ของเครื่อง (Machine Learning หรือ ML)[1] เป็นศาสตร์ที่เชื่อมโยงวิชาสถิติ คณิตศาสตร์และวิทยาการคอมพิวเตอร์เข้าด้วยกัน อธิบายง่าย ๆ คือมนุษย์พยายามทำให้เครื่องจักรนั้นมี “สติปัญญา” หรือ “ความฉลาด” และมีความสามารถในการเรียนรู้สิ่งต่าง ๆ ได้ด้วยตัวเอง โดยกระบวนการดังกล่าวนั้นจะเกี่ยวข้องกับการฝึกสอนเครื่องจักรจนกระทั่งเครื่องจักรสามารถเรียนรู้และค่อย ๆ พัฒนาการทำงานต่าง ๆ ได้โดยไม่ต้องพึ่งการตั้งโปรแกรม ซึ่งในบริบทนี้เครื่องจักรที่เรารู้จักกันดีก็คืออุปกรณ์ที่มีหน่วยประมวลผล (Processing Unit) เช่น คอมพิวเตอร์ ที่ถูกสั่งงานหรือควบคุมผ่านซอฟต์แวร์ในรูปแบบของโปรแกรมนั่นเอง โดยวิธีการก็คือเราป้อนข้อมูลเข้าไปให้กับโปรแกรม โปรแกรมจะทำการสร้างแบบจำลองหรือโมเดล (Model) ที่สามารถอธิบายความสัมพันธ์ในข้อมูลที่เราป้อนเข้าไปได้ ซึ่งขั้นตอนที่เกิดขึ้นระหว่างการเรียนรู้ก็คือการฝึกสอนโมเดล (Model Training) ซึ่งโปรแกรมสามารถแปลงข้อมูลทั้งหมดเป็นโมเดลที่ปรับปรุงได้ นั่นหมายความว่า ML สามารถทำให้คอมพิวเตอร์เรียนรู้วิธีทำงานของมนุษย์ได้โดยเฉพาะการลอกเลียนแบบ (Imitation) กิจกรรมที่ทำซ้ำหรือเกิดขึ้นแบบเดิมจนมีแบบแผน (Pattern) ที่ชัดเจน ยกตัวอย่างเช่น ถ้าเรามีข้อมูล 2 มิติซึ่งอธิบายได้ด้วยตัวแปร (x, y) โดยที่ x คืออินพุตและ y คือเอาต์พุต เราสามารถใช้ ML เพื่อหาฟังก์ชันที่สามารถเชื่อมโยง (Correlate) ความสัมพันธ์ระหว่างสองตัวแปรนี้ได้ $f : x \rightarrow y$

ML ได้ถูกพัฒนาเรื่อยมาเป็นระยะเวลาหลายทศวรรษ จุดเริ่มต้นที่นับได้ว่าสำคัญที่สุดของปัญญาประดิษฐ์เกิดขึ้นในปี ค.ศ. 1943 โดยนักตรรกศาสตร์ Walter Pitts และนักประสาทวิทยา Warren McCulloch ได้ตีพิมพ์ผลงานวิจัยที่เสนออัลกอริทึมคณิตศาสตร์ของโครงข่ายประสาท (Neural Network) และในปี ค.ศ. 1950 ศาสตราจารย์ด้านคณิตศาสตร์และนักถอดรหัสชื่อดัง Alan M. Turing[2] ได้เสนอการทดสอบ

[1]Arthur L. Samuel นักคอมพิวเตอร์ชาวอเมริกันและหนึ่งในผู้บุกเบิกสาขาปัญญาประดิษฐ์ เป็นคนแรกที่เริ่มใช้คำว่า Machine Learning ตั้งแต่ปี ค.ศ. 1959 เป็นต้นมา[1]

[2]Alan M. Turing เป็นอัจฉริยะด้านคณิตศาสตร์และการคำนวณ จบการศึกษาปริญญาตรีจาก University of Cambridge



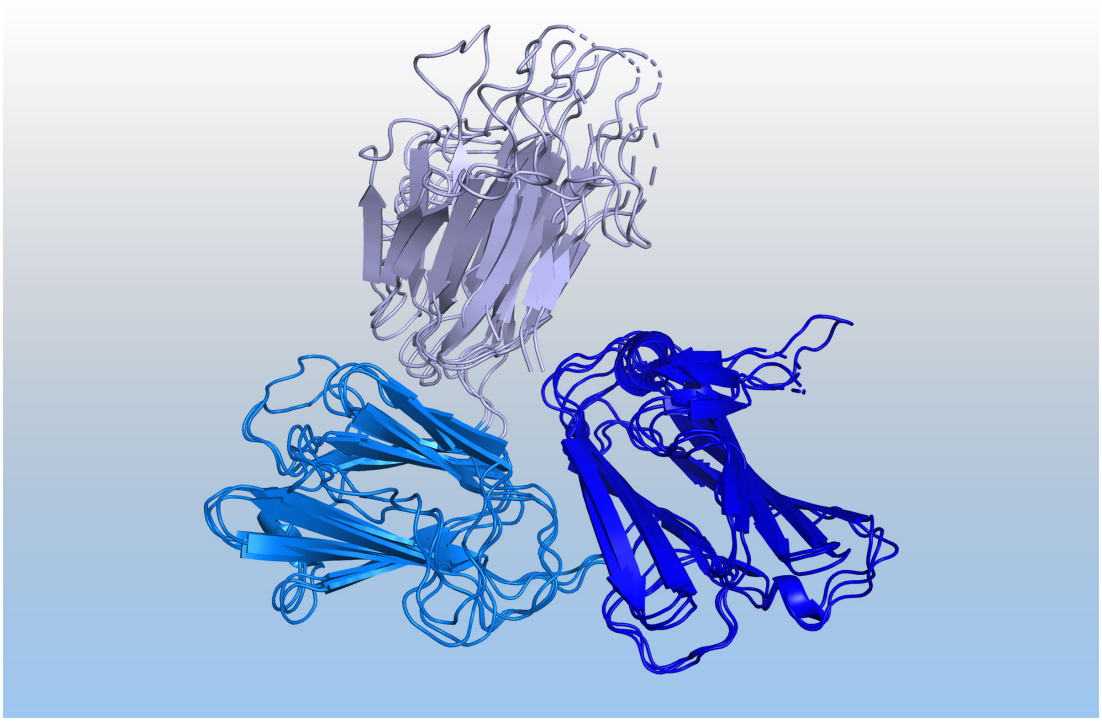
ภาพ 1.1 หุ่นยนต์ที่มีสมองเหมือนกับมนุษย์ที่มีความนึกคิดและสติปัญญา (เครดิตภาพ: https://pixabay.com)

ของทัวริง (The Turing Test)[2] ซึ่งเป็นแนวคิดในการทดสอบความมีสติปัญญาของเครื่องจักร โดยหนึ่งในการทดสอบอันโด่งดังก็คือเกมเลียนแบบ (Imitation Game)[1] หลังจากนั้นได้มีเหตุการณ์สำคัญเกิดขึ้นอีกมากมาย เช่น Arthur L. Samuel ได้เขียนโปรแกรมเล่นหมากฮอสโปรแกรมแรกของโลกให้กับ IBM ในปี ค.ศ. 1952 และอัลกอริทึม Nearest Neighbor ถูกคิดค้นขึ้นในช่วงปี ค.ศ. 1967 และในปี ค.ศ. 1996 IBM ได้พัฒนาโปรแกรมที่ชื่อว่า Deep Blue ที่สามารถเอาชนะนักหมากรุกมือวางอันดับหนึ่งของโลกได้สำเร็จ

นับตั้งแต่ปี ค.ศ. 2000 เป็นต้นมานั้นเรียกได้ว่าเป็นช่วงของการพัฒนาปัญญาประดิษฐ์แบบสมัยใหม่ก็ว่าได้ จุดเปลี่ยนที่น่าสนใจที่ทำให้คนหันมาสนใจและให้ความสำคัญกับ ML ก็คือในช่วงระยะเวลา 10 ปีที่ผ่านมาได้มีเหตุการณ์สำคัญที่สร้างแรงกระเพื่อมแก่มวลมนุษยชาติ เช่น ในปี ค.ศ. 2011 Apple ได้ปล่อยผลิตภัณฑ์ที่ชื่อ Siri ซึ่งเป็นผู้ช่วยเสมือน (Intelligent Virtual Assistant) อันแสนฉลาดออกมา ในปี ค.ศ. 2016 บริษัท DeepMind ซึ่งเป็นบริษัทลูกของ Alphabet ก็พัฒนาโมเดลปัญญาประดิษฐ์ AlphaGo ที่สามารถเล่นหมากล้อมได้อย่างชาญฉลาด และในปีเดียวกันนั้น DeepMind ได้เริ่มต้นพัฒนา AlphaFold ซึ่งเป็นโมเดล ML สำหรับการทำนายโครงสร้างสามมิติของโปรตีน และในปี ค.ศ. 2021 DeepMind ก็ได้ตีพิมพ์บทความที่นำเสนอ AlphaFold 2 ออกมา[3] ซึ่งผู้เขียนขอแนะนำให้ผู้อ่านลองอ่านบทความงานวิจัยฉบับเต็มครับ แต่ถ้าหากไม่มีเวลาอ่านหรือว่าอ่านแล้วแต่ยังไม่ค่อยเข้าใจก็สามารถดูวิดีโออธิบายบน YouTube ได้ โดยค้นหาวิดีโอที่

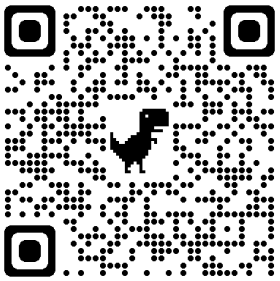
และปริญญาเอกจาก Princeton University โดยในช่วงสงครามโลก Alan ได้สร้างเครื่องถอดรหัสที่ชื่อว่า Bombe เพื่อมาต่อสู้กับเครื่องเข้ารหัส Enigma ซึ่งได้มีการประเมินไว้ว่าผลงานของ Alan ได้ช่วยชีวิตไว้ได้หลายล้านคน

[1]ผู้อ่านสามารถอ่านบทความงานวิจัยต้นฉบับของ Alan Turing ได้ที่ https://academic.oup.com/mind/article/LIX/236/433/986238 หรือดูภาพยนต์เรื่อง The Imitation Game https://www.imdb.com/title/tt2084970



ภาพ 1.2 แบบจำลองโครงสร้างสามมิติการพับของโปรตีน Pfs48/45 ซึ่งเป็นองค์ประกอบสำคัญของปรสิตมาลาเรีย โครงสร้างนี้ทำนายด้วยโมเดล ปัญญาประดิษฐ์ AlphaFold ซึ่งถูกพัฒนาโดย DeepMind (เครดิตภาพ: DeepMind)

ชื่อว่า “DeepMind’s AlphaFold 2 Explained! AI Breakthrough in Protein Folding! What we know (& what we don’t)” ที่อธิบายโดย Yannic Kilcher หรือสแกน QR Code ตามภาพที่ 1.3



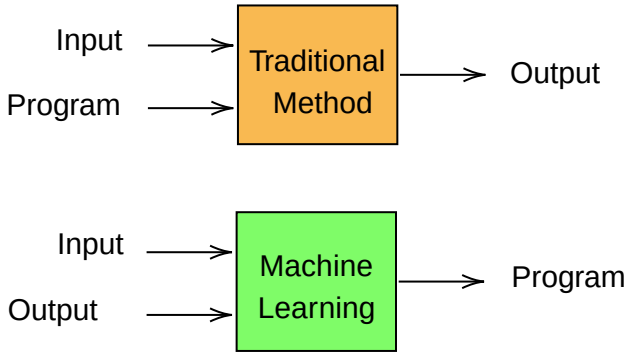
ภาพ 1.3 ลิงก์วิดีโอ DeepMind’s AlphaFold 2 Explained! AI Breakthrough in Protein Folding! What we know (& what we don’t) https://www.youtube.com/watch?v=B9PL__gVxLI

ช่วงปลายปี ค.ศ. 2022 ทีม DeepMind ก็ได้สั่นสะเทือนวงการปัญญาประดิษฐ์อีกครั้งด้วยอัลกอริทึม ML แบบใหม่ที่สามารถหาอัลกอริทึมที่สามารถคูณเมทริกซ์ได้โดยมีความเร็วมากกว่าอัลกอริทึมแบบดั้งเดิมที่ใช้กันมายาวนานกว่า 50 ปี โดยโมเดล ML ที่ทาง DeepMind ได้สร้างขึ้นมานั้นมีชื่อว่า AlphaTensor โดยใช้ Transformer[4] เป็นโมเดลหลักในการฝึกสอน[1] โดยอัลกอริทึม ML ที่ถูกค้นพบโดย AlphaTensor สามารถทำการคูณเมทริกซ์ขนาด 4×4 ได้โดยใช้การคูณทั้งหมดแค่ 47 ครั้ง ซึ่งน้อยกว่าการใช้อัลกอริทึมของสตราซเซนแบบสองระดับ (Strassen’s Two-level Algorithm) ซึ่งใช้การคูณทั้งหมด 49 ครั้ง[5] และค่าความซับซ้อนเชิงการคำนวณ (Computational Complexity) ของอัลกอริทึมใหม่นี้อยู่ที่ประมาณ $\mathcal{O}(N^{2.778})$

จากตัวอย่างข้างต้นนั้นเราสามารถสรุปได้อย่างไม่ต้องลังเลเลยว่า ML นั้นสามารถปฏิวัติวงการต่าง ๆ ได้อย่างน่าเหลือเชื่อ ไม่เพียงเฉพาะวงการคณิตศาสตร์และวิทยาศาสตร์เท่านั้น แต่รวมไปถึงวงการอื่น ๆ ด้วย ที่เทคนิคเหล่านี้เข้าไปมีบทบาท ซึ่งไม่เพียงแคในเฉพาะปัจจุบันเท่านั้นแต่ว่าในอนาคตนั้นเราอาจจะได้เห็นสิ่งใหม่ ๆ ที่ ML นั้นสามารถสร้างสรรค์ขึ้นมาได้เอง

แผนภาพที่ 1.4 แสดงการเปรียบเทียบอินพุต (Input) และเอาต์พุต (Output) ระหว่างโปรแกรมคอมพิวเตอร์ทั่วไปแบบดั้งเดิมที่เราเขียนโค้ดกันอยู่ในทุกวันนี้และโมเดลของปัญญาประดิษฐ์ โดยการเขียนโปรแกรมแบบทั่วไปนั้นเราจะต้องทำการเขียนโปรแกรมเริ่มต้นขึ้นมาและทำการป้อนอินพุตเข้าไปเพื่อให้ได้มาซึ่งเอาต์พุตหรือคำตอบที่เราต้องการ โดยกระบวนการของการเขียนโปรแกรมแบบดั้งเดิมนั้นจะเป็นกระบวนการแบบที่ทำแล้วจบ อธิบายง่าย ๆ คือโปรแกรมของเราถูกกำหนดมาเพื่อรับอินพุตและคำนวณเอาต์พุตรูปแบบเดียว ซึ่งเราไม่สามารถนำโปรแกรมที่เขียนขึ้นมานี้ไปใช้ต่อได้ (Non-transferable) แต่สำหรับกรณีของปัญญาประดิษฐ์หรือ ML นั้นจะตรงข้ามกันนั่นคือเราจะมีการป้อนทั้งอินพุตและเอาต์พุตเข้าไปให้กับปัญญาประดิษฐ์แล้วให้ปัญญาประดิษฐ์ทำการสร้างโปรแกรมหรือโมเดลโดยใช้เทคนิค ML ที่สามารถอธิบายความสัมพันธ์ของชุดข้อมูลที่มีตัวแปรต้นหรือตัวแปรอิสระ (Independent Variable) และตัวแปรตาม (Dependent Variable) ได้ ซึ่งเราสามารถนำโมเดล ML ที่ถูกสร้างขึ้นมานี้ไปใช้งานต่อกับชุดข้อมูลอื่นได้โดยเราเรียกความ

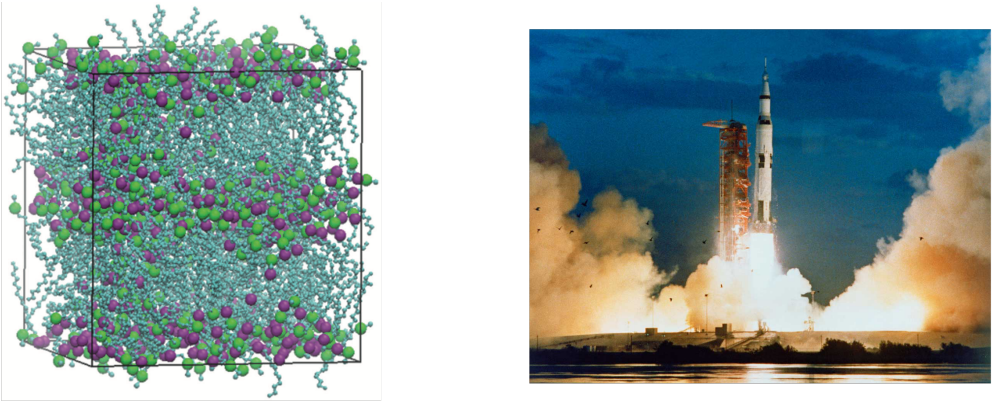
[1]โมเดลการเรียนรู้เชิงลึกแบบหนึ่งซึ่งใช้ Self-attention พัฒนาโดย Google Brain สำหรับงานทางด้านการประมวลผลภาษาธรรมชาติ (Natural Language Processing) หรือ NLP และเผยแพรในปี ค.ศ. 2017



ภาพ 1.4 แผนภาพเปรียบเทียบการทำงานของโปรแกรมแบบดั้งเดิมกับการเรียนรู้ของเครื่อง

สามารถนี้ของโมเดลว่า “ความสามารถในการส่งต่อหรือส่งผ่าน (Transferability)”

1.2 เมื่อการเรียนรู้ของเครื่องมาเจอกับเคมี



2 ms molecular dynamics simulation
= 20,000 GPU days
= **500 gigajoules**

Launch a Saturn V rocket and deliver
50-ton payload to lunar orbit
= **1,500 gigajoules**

ภาพ 1.5 เปรียบเทียบพลังงานที่ใช้ในการจำลองระบบโมเลกุลขนาดใหญ่ด้วยวิธีพลวัตเชิงโมเลกุล (Molecular Dynamics) กับพลังงานที่ใช้ในการส่งจรวด Saturn V เพื่อไปโคจรรอบดวงจันทร์

เคมีเป็นศาสตร์ที่เสมือนกับเป็นสะพานเชื่อมโยงฟิสิกส์ ชีววิทยา และคณิตศาสตร์เข้าด้วยกัน เราใช้ความรู้ทางฟิสิกส์โดยเฉพาะทฤษฎีกลศาสตร์ทั้งแบบดั้งเดิมและสมัยใหม่ในการทำความเข้าใจอนุภาคขนาดเล็ก อะตอม โมเลกุลอินทรีย์และอนินทรีย์ สารประกอบที่มีความซับซ้อน พอลิเมอร์ที่มีขนาดใหญ่ องค์ประกอบและหน่วยวัสดุต่าง ๆ รวมไปถึงสารชีวโมเลกุล เช่น โปรตีน, ลิพิด, และสารพันธุกรรม ซึ่งเป็นหน่วย

ย่อยขั้นพื้นฐานที่เป็นองค์ประกอบของสิ่งมีชีวิต โดยการที่เราต้องทำความเข้าใจคุณสมบัติขั้นพื้นฐานของโมเลกุลหรือหน่วยย่อยขั้นพื้นฐานเหล่านี้จำเป็นที่จะต้องใช้องค์ความรู้ทางด้านเคมีเชิงฟิสิกส์ซึ่งจะต้องใช้ความรู้ทางคณิตศาสตร์ในการเข้ามาช่วยแก้ปัญหาด้วยนั่นเอง

สำหรับปัญญาประดิษฐ์ซึ่งมี ML เป็นหัวใจสำคัญนั้นก็ถือว่าเป็นสาขาหนึ่งของสถิติเชิงข้อมูลและเกี่ยวข้องกับวิทยาการคอมพิวเตอร์ โดยการนำเทคนิค ML เข้ามาใช้ในการแก้ปัญหาบางอย่างทางเคมีนั้นถือว่าสมเหตุสมผลมาก นั่นก็เพราะว่านักเคมีมีข้อมูลที่ได้จากการทดลองอย่างมากมายมหาศาล มีทั้งผลการทดลองที่เป็นบวกและผลการทดลองที่เป็นลบ ซึ่งข้อมูลเหล่านี้มีข้อมูลเชิงลึกที่สำคัญแฝงอยู่ ดังนั้น ML จึงเข้ามามีบทบาทอย่างมากในการสกัด (Extract) หรือเปิดเผยสิ่งที่ซ่อนอยู่ภายในข้อมูลที่เรามีออกมาและทำให้เราเข้าใจข้อมูลทางเคมีมากขึ้นและนำไปสู่การค้นพบหรือการทำนายสิ่งใหม่ ๆ ที่จะช่วยต่อยอดให้การทำงานวิจัยทางด้านเคมีนั้นเป็นไปอย่างรวดเร็วอย่างที่เรียกว่าก้าวกระโดดเลยทีเดียว[6,7]

ภาพที่ 1.5 แสดงการเปรียบเทียบพลังงานที่ใช้ในการจำลองระบบโมเลกุลขนาดใหญ่ด้วยวิธีพลวัตเชิงโมเลกุล (Molecular Dynamics หรือ MD)[1] กับพลังงานที่ใช้ในการส่งจรวด Saturn V เพื่อไปโคจรรอบดวงจันทร์ โดยจะเห็นได้ว่าพลังงานที่ใช้ในการคำนวณ MD Simulation นั้นเป็นหนึ่งในสามของพลังงานที่ใช้ในการส่งจรวด ถ้าหากเราต้องการที่จะศึกษาโมเลกุลใหม่หรือปรับแก้พารามิเตอร์ของการคำนวณเราจะต้องคำนวณการจำลอง MD ใหม่ทุกครั้งซึ่งสิ้นเปลืองพลังงานเป็นอย่างมาก ดังนั้น ML จึงเข้ามาบทบาทอย่างมากในเคมี (โดยเฉพาะเคมีเชิงคำนวณ) เพราะว่าโมเดล ML ที่ผ่านการฝึกสอนมาแล้วนั้นสามารถนำไปใช้ในการศึกษาโมเลกุลอื่น ๆ ที่โมเดลไม่เคยเห็นมาก่อนได้เพราะว่าโมเดล ML นั้นมีคุณสมบัติของการส่งต่อความสามารถในการทำนาย (Transferability) นั่นเอง

1.3 บทบาทของการเรียนรู้ของเครื่องต่อเคมีควอนตัม

เคมีควอนตัม (Quantum Chemistry) เป็นแขนงหนึ่งของเคมีเชิงฟิสิกส์ (Physical Chemistry) ซึ่งเป็นการผสมผสานระหว่างกลศาสตร์ควอนตัม (Quantum Mechanics) และการศึกษาโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ของอะตอมและโมเลกุลเข้าด้วยกัน ซึ่งจะเรียกอีกอย่างว่ากลศาสตร์ควอนตัมโมเลกุลก็ได้เช่นกัน (Molecular Quantum Mechanics) อธิบายได้ง่าย ๆ คือเป็นการนำความรู้ทางกลศาสตร์ควอนตัมที่เป็นการศึกษาอันตรกิริยาระหว่างอนุภาคพื้นฐานในอะตอม (สนใจเฉพาะอิเล็กตรอนและโปรตอน) มาศึกษาคุณสมบัติโดยรวมของโมเลกุลที่เราสนใจ ซึ่งนักวิทยาศาสตร์ได้ศึกษาและค้นคว้างานวิจัยศาสตร์ด้านนี้มากว่าหนึ่งศตวรรษแล้วนับตั้งแต่ช่วงต้นปี ค.ศ. 1920 โดยได้มีการพัฒนาทฤษฎีต่าง ๆ มากมาย แต่สิ่งที่ผู้เขียนคิดว่าน่าสนใจก็คือจุดเปลี่ยนสำคัญของเคมีควอนตัมยุคใหม่ (Modern Quantum Chemistry) นั่นคือ “ทฤษฎีฟังก์ชันนอลความหนาแน่น” หรือ “Density Functional Theory (DFT)”[8] ซึ่งถูกพัฒนาและใช้งานอย่างต่อเนื่องมามากกว่าครึ่งศตวรรษแล้ว โดยนักวิทยาศาสตร์ได้ใช้ DFT ในงานวิจัยทางด้านเคมี ฟิสิกส์ ชีววิทยาและวัสดุศาสตร์ ถ้าหากใครที่เคยเรียนวิชาเคมีเชิงฟิสิกส์หรือฟังการนำเสนอผลงานวิจัยตามงานประชุม

[1]Molecular Dynamics เทคนิคการจำลองด้วยคอมพิวเตอร์สำหรับศึกษาโครงสร้างเฉลี่ยหรือโครงสร้างที่เปลี่ยนแปลงตามเวลาของระบบเชิงโมเลกุล

วิชาการเคมีหรือฟิสิกส์ก็น่าจะเคยได้ยินชื่อทฤษฎีนี้กันมาบ้าง

DFT เป็นทฤษฎีที่เรานำมาใช้ในการศึกษาคุณสมบัติของโมเลกุลไม่ว่าจะเป็นโมเลกุลขนาดเล็ก อย่างเช่น สารประกอบอินทรีย์และอนินทรีย์ หรือจะเป็นโมเลกุลขนาดใหญ่ อย่างเช่น โปรตีน, วัสดุโลหะ, และพอลิเมอร์ นั่นก็เพราะว่าการคำนวณด้วยวิธี DFT ให้ผลแม่นยำในระดับที่ยอมรับได้ (การพิจารณาความแม่นยำของวิธีการคำนวณนั้นขึ้นอยู่กับหลายปัจจัย) และใช้เวลาในการคำนวณที่ไม่นานมาก นั่นจึงทำให้ทฤษฎี DFT ได้รับการยกย่องเชิดชูเกียรติด้วยรางวัลโนเบลสาขาเคมีในปี ค.ศ. 1998 โดยผู้รับรางวัลได้แก่ศาสตราจารย์ Walter Kohn (University of California, Santa Barbara, CA, USA) สำหรับการพัฒนาทฤษฎี DFT และศาสตราจารย์ John Pople (Northwestern University, Evanston, IL, USA) สำหรับการพัฒนาวิธีการคำนวณสำหรับเคมีควอนตัม[1] อย่างไรก็ตามในปัจจุบันก็ได้มีงานวิจัยที่ศึกษาทฤษฎี DFT แล้วพบว่าในความเป็นจริงนั้น DFT ไม่ได้ให้ผลการคำนวณที่แม่นยำสูงมากนักเมื่อเทียบกับวิธีฟังก์ชันคลื่นหรือ Wavefunction Theory (WFT)[9,10] และยังไม่สามารถคำนวณคุณสมบัติของบางระบบได้ จึงทำให้ในปัจจุบันนั้นได้มีการพัฒนาระเบียบวิธีใหม่ ๆ ขึ้นมาเพื่อปรับปรุงประสิทธิภาพหรือความสามารถของ DFT ให้เทียบเท่ากับวิธี WFT (ผู้เขียนมีความเห็นส่วนตัวว่าทฤษฎี DFT ได้รับรางวัลโนเบลนั้นไม่ใช่เพราะว่า DFT นั้นให้ผลการคำนวณที่มีความถูกต้องที่สูงมาก แต่เป็นเพราะไอเดียของตัวทฤษฎี โดยที่ยังมีวิธีอื่น ๆ อีกหลายวิธีที่ให้ผลการคำนวณที่ถูกต้องมากกว่า DFT)

ในขณะเดียวกันนั้น ML ก็ถูกนำมาใช้ประโยชน์ในงานวิจัยเคมีมานานกว่า 30 ปีแล้ว เนื่องจากในปัจจุบันนั้นเทคโนโลยีต่าง ๆ เช่น การประมวลผลขั้นสูง (High Performance Computing), การประมวลผลบนก้อนเมฆหรือคลาวด์ (Cloud Computing) และหน่วยประมวลผลภาพกราฟิก (Graphics Processing Unit หรือ GPU) ได้เข้ามามีบทบาทอย่างมากในวิทยาศาสตร์เชิงคำนวณ (Computational Science) โดยเฉพาะเคมีเชิงคำนวณ (Computational Chemistry) จึงทำให้มีจุดเปลี่ยนที่ทำให้ความสนใจของนักวิจัยในช่วง 10 ปีที่ผ่านมานี้ในหันมาทำงานวิจัยโดยใช้ ML กันมากขึ้น อีกเหตุผลหนึ่งก็คือในปัจจุบันเราสามารถศึกษาและใช้ง่าย ML ได้ง่ายขึ้นเมื่อเทียบกับในอดีต ทุกวันนี้เราไม่จำเป็นต้องมานั่งเขียนโค้ดเพื่อสร้างโมเดล ML แบบเริ่มจากศูนย์กันแล้ว ในปัจจุบันเรามีภาษาโปรแกรมที่ศึกษาและใช้งานได้ง่าย เช่น ภาษา Python และมีไลบรารี่แบบโอเพนซอร์ซ (Open-source) ที่เราสามารถใช้งานได้ฟรีหรือนำมาพัฒนาต่อได้มากมาย เช่น NumPy,[11] Scikit-learn,[12] TensorFlow,[13] PyTorch,[14] Flux[15] หรือแม้แต่ Matlab[16] ที่ก็มีฟังก์ชันสำเร็จรูปที่ให้เราสามารถเลือกใช้โมเดล ML ต่าง ๆ ได้ตามต้องการ

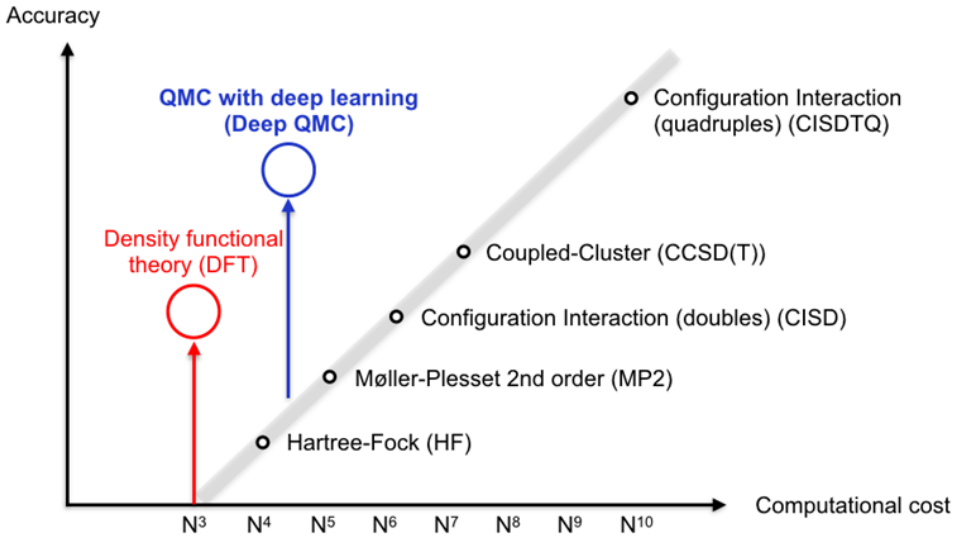
ผู้เขียนขอยกตัวอย่างหัวข้องานวิจัยหนึ่งที่ตอนนี้กำลังมาแรง (อย่างน้อย ๆ ก็ ณ วันที่ผู้เขียนกำลังเขียนหนังสือเล่มนี้) นั่นคือการพัฒนาโมเดล ML เพื่อเรียนรู้ฟังก์ชันนอลการแลกเปลี่ยนและสหสัมพันธ์ (Exchange-Correlation Functional หรือ XC Functional)[17] ซึ่ง XC Functional คือเทอมที่อธิบายอันตรกิริยาระหว่างอิเล็กตรอนและถือได้ว่าเป็นพารามิเตอร์ที่สำคัญที่สุดของ DFT ในการนำไปศึกษาระบบต่าง ๆ โดยถ้าหากว่า XC Functional นั้นมีความสามารถในการอธิบายระบบทางเคมีได้หลากหลายระบบ เราจะเรียกสิ่งที่มีความสามารถในการทำอะไรได้หลาย ๆ อย่างว่า “สารพัดประโยชน์ (General Purpose)” หรืออาจจะเรียกอีกอย่างว่าฟังก์ชันนอลนั้นมีความเป็นสากล (Universality) นั่นเอง ถ้าหากเราพัฒนาโมเดล XC Functional ด้วย ML ที่มีประสิทธิภาพที่ดีมาก ๆ ได้สำเร็จ เราก็จะมีโมเดล XC Functional ที่สามารถนำไปใช้ในการ

[1]รายละเอียดของรางวัลโนเบลสาขาเคมี ปี ค.ศ. 1998 ดูได้ที่ https://www.nobelprize.org/prizes/chemistry/1998/summary

ทำนายคุณสมบัติต่าง ๆ ของโมเลกุลรวมไปถึงสารประกอบและวัสดุทางเคมีได้อย่างถูกต้องและแม่นยำ แต่ในความเป็นจริงนั้น XC Functional ก็เปรียบเสมือนเป็นกล่องดำ (Black Box) ของทฤษฎี DFT เพราะว่าไม่มีใครที่รู้หน้าตาสมการหรือผลเฉลยทั่วไปที่แน่นอนของ XC Functional ดังนั้นนับตั้งแต่อดีตจนถึงปัจจุบัน เราจึงทำได้เพียงแค่หารูปแบบของ XC Functional โดยใช้วิธีการประมาณและการเทียบพารามิเตอร์กับผลการทดลอง

ตรงจุดนี้เองที่ ML ก็เข้ามามีบทบาทและทำให้งานวิจัยในช่วงระยะหลังตั้งแต่ ปี ค.ศ. 2010 เป็นต้นมามีการพัฒนาอัลกอริทึม ML สำหรับเคมีควอนตัมเยอะมาก โดยหนึ่งในนั้นก็คือการนำนำ ML มาช่วยในการหาหน้าตาสมการของ XC Functional นั่นก็เพราะว่า ML นั้นเป็นการวิเคราะห์เชิงปริมาณที่ใช้หลักการทางสถิติเข้ามาช่วยในการหาความสัมพันธ์ระหว่างของสองสิ่งซึ่งให้ความแม่นยำสูงในการประมาณค่าที่สูงและให้คำตอบใกล้ที่ใกล้เคียงกับการใช้วิธีแบบคลาสสิคหรือวิธีดั้งเดิมในการคำนวณแต่มีความสิ้นเปลืองในเชิงการคำนวณที่น้อยกว่ามาก ถึงแม้ว่าตอนนี้การนำ ML เข้ามาช่วยในการทำวิจัยทางด้านเคมีควอนตัม (และสาขาอื่น ๆ ด้วย) จะยังอยู่ในขั้นของการพัฒนา แต่สิ่งหนึ่งที่เราเห็นได้ชัดเลยก็คือ ML สามารถช่วยลดระยะเวลาและต้นทุนในการศึกษาคุณสมบัติเชิงอิเล็กทรอนิกส์ (Electronic Properties) ของโมเลกุลได้เยอะมาก สำหรับรายละเอียดเพิ่มเติมของ XC Functional ผู้อ่านสามารถศึกษาได้ที่บทที่ 7

$$\hat{H}\psi(\mathbf{r}_1,\ldots,\mathbf{r}_N) = E\psi(\mathbf{r}_1,\ldots,\mathbf{r}_N)$$



ภาพ 1.6 แผนภาพแสดง Scaling ของวิธีทางเคมีควอนตัม (เครดิตภาพ: https://www.chemistryworld.com)

ตารางที่ 1.1 แสดงค่าความซับซ้อนของการคำนวณ (Compuational Complexity) ของแต่ละวิธี โดยจะเห็นได้ว่าวิธี DFT นั้นมีความซับซ้อนคือ $\mathcal{O}(N^3)$ นั่นคือมันเป็นสัดส่วนโดยตรงกับจำนวนของอิเล็กตรอนของระบบ (N) ยกกำลังสาม ซึ่งมาจากการที่เราจะต้องทำการทำให้แฮมิลโทเนียน (Hamiltonian) เกิดเมทริกซ์รูปทแยง (Diagonalization)[1] ซึ่งความซับซ้อนของการทำ Diagonalization สำหรับเมทริกซ์จตุรัส

[1]เป็นวิธีการเปลี่ยนฐานของปริภูมิเวกเตอร์เพื่อให้ได้เมทริกซ์การแปลงเชิงเส้นที่อยู่ในรูปเมทริกซ์ทแยง เพื่อที่จะนำไปคำนวณ

ตาราง 1.1 ตารางเปรียบเทียบความซับซ้อนเชิงคำนวณของวิธีทางเคมีควอนตัม[18] โดย N คือจำนวนของอิเล็กตรอนหรือจำนวนของฟังก์ชันพื้นฐาน (Basis Function)

ตัวย่อ	วิธี	Runtime
FCI	Full Configuration Interaction (CISDTQ)	$\mathcal{O}(N^{10})$
CC	Coupled Cluster (CCSDT)	$\mathcal{O}(N^8)$
CC	Coupled Cluster (CCSD(T))	$\mathcal{O}(N^7)$
CC	Coupled Cluster (CCSD)	$\mathcal{O}(N^6)$
FCI	Full Configuration Interaction (CISD)	$\mathcal{O}(N^6)$
MP2	Mllor-Plesset second order perturbation theory	$\mathcal{O}(N^5)$
QMC	Quantum Monte Carlo	$\mathcal{O}(N^3) - \mathcal{O}(N^4)$
HF	Hartree-Fock	$\mathcal{O}(N^3) - \mathcal{O}(N^4)$
DFT	Density Functional Theory (Kohn-Sham)	$\mathcal{O}(N^3)$
TB	Tight Binding	$\mathcal{O}(N^3)$
MM	Molecular Mechanics	$\mathcal{O}(N^2)$

ขนาด $n \times n$ คือ $\mathcal{O}(n^3)$

ตาราง 1.2 ตารางเปรียบเทียบความซับซ้อนเชิงคำนวณของวิธีทางเคมีควอนตัม[19] โดย n คือจำนวนของข้อมูล, p คือจำนวน Feature, n_{trees} คือจำนวนของต้นไม้ (Trees), n_{sv} คือจำนวนของ Support Vectors, n_{L_i} คือจำนวนของ Neuron หรือ Node ของชั้นที่ i และ t คือจำนวนของ Epochs ที่ใช้ในการฝึกฝนโมเดล

อัลกอริทึม	Runtime	
	การฝึกฝนโมเดล	การทำนายเอาต์พุต
Decision Tree	$\mathcal{O}(n^2p)$	$\mathcal{O}(p)$
Random Forest	$\mathcal{O}(n^2pn_{trees})$	$\mathcal{O}(pn_{trees})$
Gradient Boosting (n_{trees})	$\mathcal{O}(npn_{trees})$	$\mathcal{O}(pn_{trees})$
Linear Regression	$\mathcal{O}(p^2n + p^3)$	$\mathcal{O}(p)$
SVM (Kernel)	$\mathcal{O}(n^2p + n^3)$	$\mathcal{O}(n_{sv}p)$
Neural Network	$\mathcal{O}(npt * (n_{L_1}n_{L_2} + n_{L_2}n_{L_3} + \dots)$	$\mathcal{O}(pn_{L_1} + n_{L_1}n_{L_2} + \dots)$

ตารางที่ 1.2 แสดงค่าความซับซ้อนของการคำนวณของอัลกอริทึม ML แบบต่าง ๆ เช่นเดียวกับตารางที่ 1.1 โดยอัลกอริทึมที่แสดงนั้นถูกใช้กับโจทย์ปัญหา Classification และ Regression (ยกเว้น Linear Regression ที่ใช้สำหรับ Regression เท่านั้น) จะเห็นได้ว่าความซับซ้อนของวิธี ML นั้นจะขึ้นอยู่กับจำนวนของข้อมูลและจำนวน Feature ของข้อมูลแต่ละตัวเป็นหลัก ยกเว้นกรณีของอัลกอริทึม Neural Network (ซึ่งเราสนใจเฉพาะการเรียนรู้เชิงลึกหรือ Deep Learning เท่านั้น) ที่ความซับซ้อนของอัลกอริทึมจะขึ้นอยู่กับจำนวนรอบที่ใช้ในการฝึกสอนโมเดลและวามซับซ้อนของโครงข่าย เช่น จำนวนชั้นของโครงข่ายและจำนวนหน่วยการเรียนรู้ของแต่ละชั้น

ในขั้นตอนต่อไปได้อย่างสะดวกมากขึ้น

ถ้าหากเปรียบเทียบแล้วจะพบว่าวิธี ML นั้นมีความซับซ้อนน้อยกว่าวิธี QM อย่างมีนัยสำคัญ อย่างน้อย ๆ ก็ในระดับหลายเท่าตัว และยิ่งไปกว่านั้น ความซับซ้อนเชิงการคำนวณในการทำนายค่าเอาต์พุตของโมเดลที่ผ่านการฝึกสอนมาแล้วนั้นต่ำมาก ซึ่งโดยส่วนใหญ่แล้วจะอยู่ในรูปของผลคูณเชิงเส้นแบบดีกรี 1 ระหว่างจำนวนของข้อมูลกับจำนวนของ Feature นี่จึงเป็นเหตุผลที่ทำให้งานวิจัยทางด้านเคมีโดยเฉพาะด้าน QM ในช่วงระยะเวลา 10 ปีที่ผ่านมานั้นนักวิจัยเริ่มให้ความสนใจกับการพัฒนาและประยุกต์ใช้ ML เพื่อมาใช้ในการสร้างโมเดลสำหรับประมาณหรือทำนายค่าต่าง ๆ ทางควอนตัมนั่นก็เพราะว่า ML เป็นวิธีที่มีความสิ้นเปลืองในเชิงของระยะเวลาในการคำนวณน้อยกว่ามาก[20,21,22]

1.4 ทักษะที่จำเป็นสำหรับผู้เริ่มต้นศึกษาการเรียนรู้ของเครื่อง

การมีความรู้พื้นฐานก่อนเริ่มศึกษา ML อย่างจริงจังนั้นเป็นสิ่งสำคัญมาก ผู้เขียนได้สรุป 5 สิ่งสำคัญที่ผู้ที่เริ่มต้นศึกษาควรจะต้องศึกษา ดังต่อไปนี้

1. **พีชคณิตเชิงเส้นและแคลคูลัสแบบหลายตัวแปร** : ทั้งสองวิชานี้ถือว่าเป็นรากฐานของ ML เลยก็ว่าได้ เพราะว่าโมเดลทุกรูปแบบของ ML นั้นต่างก็ล้วนแต่เป็นคณิตศาสตร์ ถ้าหากเราต้องการที่จะพัฒนาอัลกอริทึมใหม่ ๆ หรือปรับปรุงอัลกอริทึมที่มีอยู่แล้ว เราจะต้องอาศัยความรู้พีชคณิตเชิงเส้น (เวกเตอร์และเมทริกซ์) และแคลคูลัส (การหาอนุพันธ์) แต่ถ้าหากว่าเราเน้นไปทางสายแอพพลิเคชันเราก็อาจจะไม่จำเป็นต้องรู้แบบลึกหรือละเอียดมากก็ได้ เพราะว่าในปัจจุบันมีเครื่องมือและไลบรารี่สำเร็จรูปให้เราเลือกใช้มากมาย

2. **สถิติ** : เนื่องจากว่าในขั้นตอนก่อนที่จะเริ่มสร้างและฝึกสอนโมเดล ML นั้น เราจะต้องใช้เวลาส่วนใหญ่ (อาจจะมากถึง 80%) ไปกับการรวบรวมข้อมูล ทำความสะอาดข้อมูล การศึกษาการกระจายตัวของข้อมูล การตั้งและทดสอบสมมติฐาน การทำการถดถอย (Regression) หรือการแยกประเภท (Classification) เราจึงจำเป็นที่จะต้องใช้สถิติเข้ามาช่วยเพื่อให้เข้าใจถึงรายละเอียด ของชุดข้อมูลที่เรากำลังจะเล่นกับมัน หมายความว่ายิ่งเราเข้าใจข้อมูลมากเท่าไหร่ ก็ยิ่งช่วยให้เราสามารถเลือกใช้โมเดล ML ได้เหมาะสมเท่านั้น

3. **โปรแกรมมิ่ง** : สิ่งสำคัญลำดับถัดมาคือทักษะในการเขียนโปรแกรมหรือเขียนโค้ด ถึงแม้ว่าเราจะมีความรู้ด้านทฤษฎีที่แม่นยำ แต่ถ้าหากเราไม่สามารถเขียนโปรแกรมได้เราก็ไม่สามารถสร้างโมเดลหรือนำ ML มาใช้งานจริงได้เลย ดังนั้นเราควรจะต้องเรียนรู้การเขียนโปรแกรม ให้ได้อย่างน้อยสัก 1 ภาษา ซึ่งภาษาที่ได้รับความนิยมมากที่สุดสำหรับงานทางด้านวิทยาศาสตร์ข้อมูล ณ ตอนนี้คือภาษา Python นั่นก็เพราะตัวภาษาเองมีไวยากรณ์ (Syntax) ที่เข้าใจง่าย มีไลบรารี่เสริมให้เลือกใช้เยอะ มี Community ที่ใหญ่มาก ไม่ต้องกลัวเลยว่าถ้าหากมีปัญหาเกี่ยวกับการเขียน Python แล้วจะไม่มีคนช่วยหรือหาวิธีแก้ปัญหาไม่ได้ เว็บไซต์ที่มีการถามตอบคำถามเกี่ยวกับการเขียนโปรแกรมที่ได้รับความนิยมเป็นอันดับหนึ่งก็คือ Stack Overflow (https://stackoverflow.com)

4. **แนวคิดของ ML** : แนวคิดหรือ Concept ทางด้าน ML เป็นสิ่งที่สำคัญมากเช่นเดียวกัน เราควรจะต้องทราบความหมายของคำศัพท์เฉพาะทาง (Terminology), ประเภทและอัลกอริทึมแบบต่าง ๆ

ของ ML, รวมถึงแนวทางการนำ ML มาใช้ (Best Practice) การเรียนรู้แนวคิดของ ML นั้นไม่สามารถทำได้ในระยะเวลาอันสั้นเพราะว่าเป็นสิ่งที่เกิดจากการสั่งสมประสบการณ์ การลองผิดลองถูกซ้ำไปซ้ำมาจนตกผลึกได้เป็นความเข้าใจของเราเอง ซึ่งผู้อ่านสามารถอ่านหนังสือ ML ประกอบเพื่อใช้เป็นแนวทางในการศึกษาแนวคิดของ ML ได้

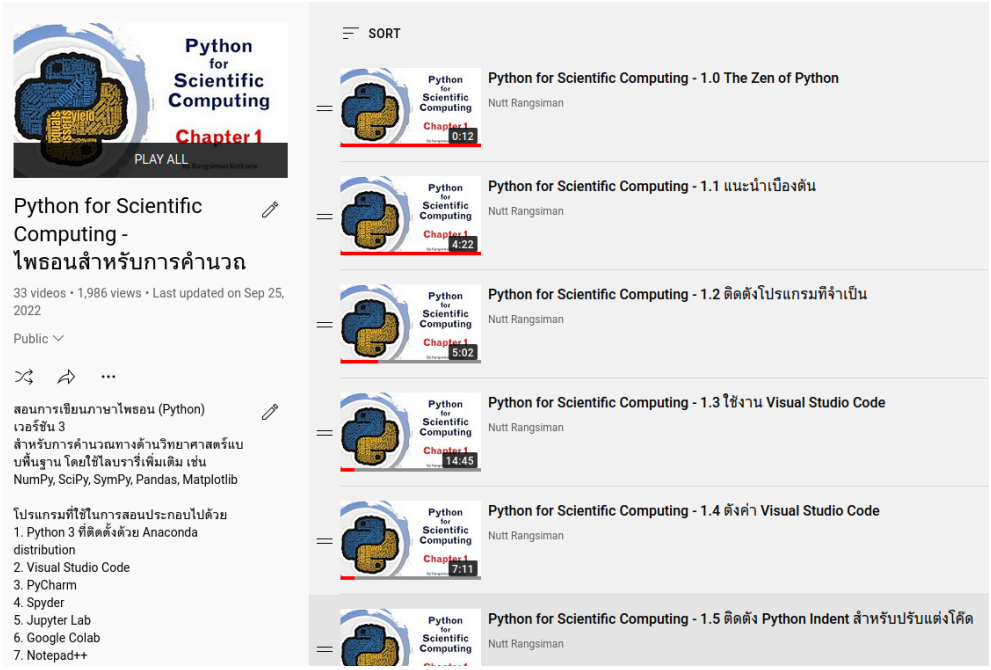
5. **ฝึกทำโจทย์จริง** : ตัวช่วยที่ดีที่สุดให้ทำเราเรียนรู้ ML ได้อย่างรวดเร็วนั่นก็คือการฝึกฝนการเขียนโค้ดและวิเคราะห์โจทย์แบบต่าง ๆ ผู้อ่านสามารถหาโจทย์ตามหนังสือหรือเว็บไซต์มาฝึกทำ หรืออาจจะลองเก็บเกี่ยวประสบการณ์โดยเข้าร่วมการแข่งขันวิทยาศาสตร์ข้อมูล หรือ Datathon ซึ่งเป็นการแข่งขัน Hackathon แบบหนึ่งที่เน้นไปที่การแก้ปัญหาเชิงข้อมูล ซึ่งในปัจจุบันก็มีการจัดแข่งขันบ่อยมาก ๆ โดยสนามสำหรับฝึกฝนวิทยายุทธที่ได้รับความนิยมนั้นก็คือ Kaggle (https://www.kaggle.com/competitions) และ Alcrowd (https://www.aicrowd.com) โดยหนึ่งในการแข่งขันที่ให้ผู้เข้าแข่งใช้ ML ในการทำนายคุณสมบัติเชิงแม่เหล็กของโมเลกุลนั่นก็คือ “Predicting Molecular Properties” ซึ่งจัดโดย CHAMPS (CHemistry And Mathematics in Phase Space) โดยมีเงินรางวัลรวมมากถึง 30,000 US dollars[1]

1.5 แนวทางสำหรับการศึกษาการเรียนรู้ของเครื่อง

ผู้เขียนได้สรุปแนวทางศึกษา ML สำหรับงานวิจัยทางด้านเคมี (เน้นเคมีควอนตัม) สำหรับผู้เริ่มต้นตามด้านล่างต่อไปนี้

1. ฝึกเขียนภาษาคอมพิวเตอร์ให้คล่อง โดยภาษาที่ผู้เขียนแนะนำคือภาษา Python (เวอร์ชัน 3.6 เป็นต้นไป) โดยสามารถเรียนได้จากชุดวิดีโอที่ผู้เขียนได้จัดทำไว้ “Python for Scientific Computing - ไพธอนสำหรับการคำนวณทางวิทยาศาสตร์” ซึ่งเน้นไปที่การเขียน Python สำหรับงานทางด้านวิทยาศาสตร์เชิงคำนวณโดยใช้ Interactive Python โดยดูได้บน YouTube

[1] https://www.kaggle.com/c/champs-scalar-coupling



ภาพ 1.7 เพลลิสต์ไพธอนสำหรับการคำนวณทางวิทยาศาสตร์ (Python for Scientific Computing) https://youtube.com/playlist?list=PLt-twymrmZ2eUPDfuXP6A7fbiCZygd-sa

2. เริ่มจากปรับพื้นฐานพีชคณิตเชิงเส้น (โดยเน้นไปที่เมทริกซ์) และแคลคูลัสแบบหลายตัวแปร เช่น การหาอนุพันธ์ย่อยและเวกเตอร์แคลคูลัส รวมไปถึงวิธีวิเคราะห์ทางสถิติ

3. ดูวิดีโอบน YouTube ที่อธิบายปัญญาประดิษฐ์เบื้องต้นตาม QR Code หรือลิงก์ในภาพที่ 1.8

ภาพ 1.8 ลิงก์วิดีโอ Machine Learning Basics | What Is Machine Learning? | Introduction To Machine Learning https://www.youtube.com/watch?v=ukzFI9rgwfU

หลังจากนั้นให้เรียนคอร์ส ML โดยคอร์สที่ผมแนะนำคือคอร์สออนไลน์ของศาสตราจารย์ Andrew Ng (Stanford University) บน Coursera โดยมีสองคอร์สหลักคือ

(a) Machine Learning Specialization เป็นคอร์ส ML ที่ได้รับความนิยมมากที่สุดในโลก[1]

[1] https://www.coursera.org/specializations/machine-learning-introduction

(b) Deep Learning Specialization เป็นคอร์สที่ได้รับความนิยมไม่แพ้กัน โดยจะเน้นไปที่ Neural Network[1]

และคอร์สของ Stanford University ซึ่งมีศาสตราจารย์ Andrew Ng นำทีมสอนเช่นเดียวกัน

(a) CS229: Machine Learning[2]

(b) CS230: Deep Learning[3]

4. ทำแบบฝึกหัดตามหนังสือ “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems” และหนังสือ “Python Machine Learning” เพื่อจะได้ทำความคุ้นเคยกับ Framework ในการเขียนโค้ด ML และ Neural Network

5. ศึกษาการประยุกต์ ML (โดยเฉพาะ Neural Network) สำหรับเคมีด้วยเว็บไซต์ https://dmol.pub ที่จัดทำโดยกลุ่มวิจัยของศาสตราจารย์ Andrew White (University of Rochester) ซึ่งจะมีโจทย์ทางเคมีและโค้ดให้ฝึกเขียนตาม เช่น การทำนายคุณสมบัติที่สำคัญและการทำนายพลังงานรวมของโมเลกุล

6. ทบทวนงานวิจัย (Literature Review) ในวารสารวิชาการที่อ่านได้ง่ายและไม่ลงรายละเอียดเชิงเทคนิคมากเกินไป เช่น Chemical Reviews หรือ Science Advances ที่เกี่ยวกับ ML สำหรับเคมีควอนตัมและเคมีสาขาอื่น ๆ

7. เลือกอ่านบทความงานวิจัยจากวารสารเฉพาะทางเน้นไปที่เคมีทฤษฎีและการประยุกต์ โดยเลือกหัวข้องานวิจัยเคมีที่เราสนใจและต้องการนำ ML ไปประยุกต์ใช้กับหัวข้อนั้น ๆ

อย่างไรก็ตาม แนวทางทั้ง 7 ข้อตามด้านบนนั้นเป็นความคิดเห็นส่วนตัวของผู้เขียนซึ่งอ้างอิงตามประสบการณ์จริง นอกจากแหล่งความรู้ที่ผู้เขียนได้แนะนำไว้แล้วจริง ๆ ยังมีแหล่งความรู้อื่น ๆ อีกมากมายที่ผู้อ่านสามารถศึกษาตามได้ด้วยตัวเอง

1.6 คำศัพท์เฉพาะทางด้านการเรียนรู้ของเครื่อง

Accuracy

ค่าความถูกต้องของการทำนายของโมเดล ส่วนใหญ่เรามักจะรายงานค่าความถูกต้องเป็นเปอร์เซ็นต์

Algorithm

วิธีหรือขั้นตอนกระบวนการคิดคำนวณทางคณิตศาสตร์เพื่อให้ได้ผลลัพธ์ออกมา

[1] https://www.coursera.org/specializations/deep-learning

[2] https://cs229.stanford.edu

[3] https://cs230.stanford.edu

Attribute
ปริมาณที่ได้จากการสังเกตซึ่งบ่งบอกถึงคุณลักษณะของสิ่งที่สนใจ เช่น สี, ขนาด, และความสูง พูดง่าย ๆ คือถ้าเป็นชุดข้อมูล Attribute ก็คือชื่อของแต่ละคอลัมน์นั่นเอง

Bias ความแตกต่างระหว่างค่าที่ได้จากการทำนายและค่าอ้างอิง กล่าวคือ Bias เป็นการอ้างถึงความถูกต้อง (Accuracy)

Categorical Variables
ตัวแปรจัดกลุ่ม ไม่มีความต่อเนื่อง เป็นข้อมูลประเภทแบบแยกออกจากกันและไม่ขึ้นต่อค่าอื่น ๆ เช่น เพศ เชื้อชาติ พันธ์สุนัข ชนิดของผลไม้ ชนิดของหมู่ฟังก์ชันในโมเลกุล

Classification
การจำแนกข้อมูลหรือการทำนายค่าที่มีความไม่ต่อเนื่อง เช่น ประเภทของยานพาหนะ ชนิดของผลไม้

Clustering
การจัดกลุ่มข้อมูล

Confusion matrix
เมทริกซ์ที่แสดงการประเมินผลลัพธ์ของการทำนาย

- True Positives (TP): ทำนายว่าจริง และสิ่งที่เกิดขึ้นก็จริง
- True Negatives (TN): ทำนายว่าไม่จริง และสิ่งที่เกิดขึ้นก็ไม่จริง
- False Positives (FP): ทำนายว่าจริง แต่สิ่งที่เกิดขึ้นคือไม่จริง
- False Negatives (FN): ทำนายว่าไม่จริง แต่สิ่งที่เกิดขึ้นคือจริง

Continuous Variables
ตัวแปรต่อเนื่อง เป็นตัวแปรเชิงปริมาณที่มีค่าในช่วงที่กำหนด เช่น น้ำหนัก จำนวนรถ ความยาวพันธะ

Convergence
สถานะของโมเดลเมื่อการเปลี่ยนของค่า Loss ระหว่าง Interation มีขนาดน้อยมาก ๆ

Descriptor
วิธีที่ใช้ในการแปลงข้อมูล เช่น ข้อมูลของโครงสร้างโมเลกุล ให้เป็นข้อมูลเวกเตอร์ที่เป็น Feature

Dataset หรือ Data Set
ชุดข้อมูลที่มีคุณสมบัติเหมือนกันโดยถูกจัดเป็นชุดให้ถูกต้องตามลักษณะโครงสร้างข้อมูล

Epoch
จำนวนครั้งหรือจำนวนรอบที่อัลกอริทึมมองเห็นหรือได้รับชุดข้อมูลทั้งหมดเข้ามาเพื่อทำการเรียนรู้

Extrapolation
เป็นการทำนายที่อยู่นอกเหนือจากชุดข้อมูลที่ใช้ในการฝึกสอนโมเดล

Feature
คุณลักษณะเฉพาะของสิ่งที่สนใจ โดยเป็นข้อมูลแบบ 1 มิติที่ถูกสร้างโดย Descriptor

Hyperparameter
พารามิเตอร์ขั้นสูงที่กำหนดคุณสมบัติของโมเดล เช่น ความเร็วในการเรียนรู้ ความซับซ้อนของโมเดล ซึ่งผู้ใช้งานจะต้องทำการกำหนดพารามิเตอร์เหล่านี้ก่อนทำการสร้างหรือฝึกสอนโมเดล

item[Input] อินพุตเป็นข้อมูลที่เรานำเข้าหรือป้อนเข้าไปให้กับโปรแกรมคอมพิวเตอร์

Learning Rate
อัตราเร็วในการเรียนรู้ของโมเดล หรือในทางทฤษฎีคือขนาดของก้าวในการปรับปรุง (Update Step) ในกระบวนการ Optimization โดยการใช้อัลกอริทึม เช่น Gradient Descent

Loss ค่าความคลาดเคลื่อนหรือความแตกต่างระหว่างค่าที่ได้จากการทำนาย (Prediction) และค่าอ้างอิง (Ground Truth หรือ Reference) ยิ่ง Loss มีค่าน้อย หมายความว่าโมเดลของเรายิ่งมีประสิทธิภาพสูง โดยค่า Loss จะถูกคำนวณในระหว่างการฝึกสอนโมเดล

Model
ชุดคำสั่งหรือโปรแกรมที่ถูกสร้างขึ้นมาโดยมีความสามารถในการคำนวณ ประมวลผลและตัดสินใจ

Noise
ข้อมูลที่มีความผิดปกติและไม่มีความเกี่ยวข้องกับข้อมูลที่เราสนใจ รวมไปถึงค่าที่เกิดจากการสุ่ม (Randomness)

Normalization
การทำให้เป็นปกติ เป็นการกำหนดหรือบังคับค่าของน้ำหนัก (Weights) ในการทำ Regression เพื่อป้องกันปัญหา Overfitting (การ Fit ข้อมูลที่ดีเกินไป) และเพิ่มความเร็วในการคำนวณ

Outlier
ค่าที่ผิดปกติไปจากข้อมูลตัวอื่นในชุดข้อมูล

Output
เอาต์พุตเป็นข้อมูลที่ถูกส่งออกมาจากโปรแกรมคอมพิวเตอร์

Overfitting
คือการที่โมเดลที่ถูกฝึกสอนด้วยชุดข้อมูล Training Set มีค่าความถูกต้องในการบ่งบอกคลาสเป้าหมายสูง แต่เมื่อนำไปใช้กับข้อมูลทดสอบ Test Set กลับได้ค่าความถูกต้องต่ำ กล่าวอีกนัยหนึ่งคือตัวโมเดลที่ได้สามารถเรียนรู้ข้อมูลจาก Training Set ได้ดีมาก แต่ไม่สามารถนำไปใช้กับข้อมูลใหม่ที่ไม่เคยพบมาก่อน (Unknown Data) ได้ดี

Parameter
คุณสมบัติของข้อมูลที่ถูกเรียนรู้โดยโมเดลปัญญาประดิษฐ์ โดยพารามิเตอร์จะถูกปรับให้มีความเหมาะสมตามอัลกอริทึมที่ใช้ เช่น Optimization ตัวอย่างของพารามิเตอร์มีดังนี้

- น้ำหนัก (Wegiths) ในเทคนิค Neural Network
- เวกเตอร์ค้ำยัน (Support Vectors) ในเทคนิค Support Vector Machine

- ค่าสัมประสิทธิ์ (Coefficients) ในเทคนิค Linear Regression

Precision

ความแม่นยำของการทำนาย มีสมการในการคำนวณดังนี้

$$P = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1.1}$$

Prediction

กระบวนการทำนายค่าของโมเดลโดยจะทำนายค่าเอาต์พุตของข้อมูลใหม่ที่ถูกป้อนเข้าไป

Regression

การทำนายค่าที่มีความต่อเนื่อง เช่น ราคาสินค้า ปริมาณน้ำมัน ถ้าในบริบทของเคมีก็จะเป็นคุณสมบัติของโมเลกุล เช่น พลังงานภายในของโมเลกุล, พลังงานอิสระของโมเลกุล และพลังงานกระตุ้น

Regularization

การทำให้ถูกต้อง เป็นเทคนิคสำหรับการแก้ปัญหา Overfitting โดยการเพิ่มพจน์พิเศษที่มีความซับซ้อนเข้าไปใน Loss function โดยเทคนิคนี้มีประโยชน์อย่างมากในการสร้างโมเดลที่มีความซับซ้อน

Reinforment Learning

การเรียนรู้แบบเสริมแรง เป็นอัลกอริทึมการเรียนรู้ที่ความสามารถในการเรียนรู้นั้นเกิดขึ้นมาจากการปฏิสัมพันธ์ (Interaction) ระหว่างผู้เรียนรู้ (Agent) กับสิ่งแวดล้อม (Environment)

Representation

Feature ของโมเลกุลในรูปแบบสัญลักษณ์ (Symbolic) เช่น SMILES, InChI, สูตรโมเลกุล

Segmentation

กระบวนการหรือขั้นตอนการแบ่งส่วนของชุดข้อมูลออกเป็นชุดข้อมูลย่อยหลาย ๆ ชุด

Specificity

ความจำเพาะเจาะจง เป็นพารามิเตอร์ที่วัดประสิทธิภาพของโมเดลในการจำแนกกรณีที่เป็นเท็จหรือไม่เป็นจริง (Negative) กล่าวอีกอย่างคือ เมื่อคำตอบคือ Negative พารามิเตอร์ตัวนี้จะบอกเราว่าการทำนายของโมเดลของเรานั้นถูกต้องมากน้อย (บ่อย) แค่ไหน โดยมีสมการที่ในการคำนวณคือ

$$S = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \tag{1.2}$$

Supervised Learning

การเรียนรู้ของโมเดลแบบมีผู้สอน (Output) โดยเราจะทำการป้อนข้อมูลอินพุตและเอาต์พุตให้กับโมเดล

Target / Output / Class / Label

คำตอบหรือเป้าหมายที่ต้องการคำนวณ (Calculate) ประมาณค่า (Approximate) หรือทำนาย (Predict)

Test Set

ชุดข้อมูลที่ใช้ทดสอบความถูกต้องและแม่นยำของ Model

Training

กระบวนการสร้างและฝึกสอนโมเดลโดยใช้ Training set

Training Set

ชุดข้อมูลที่นำมาใช้ในการสอนคอมพิวเตอร์เพื่อสร้าง Model

Transfer Learning

การเรียนรู้แบบส่งต่อ เป็นการนำโมเดล ML ที่ผ่านการฝึกสอนแล้วและสามารถทำงานอย่างหนึ่งได้อยู่แล้วมาฝึกสอนอีกครั้งหนึ่งเพื่อให้สามารถทำงานที่สองได้ โดยวิธีการทำ Transfer Learning ก็คือเรานำค่าพารามิเตอร์ที่ถูกปรับโดยผ่านการฝึกสอนแล้ว นำมาใช้ต่อในการทำนายค่าอื่น ๆ นั่นเอง ตัวอย่างเช่นถ้าเป็น Neural Network พารามิเตอร์ที่เรามาสามารถนำมาใช้ได้ก็จะเป็นน้ำหนัก (Weights) ของแต่ละ Layer เป็นต้น

Unsupervised Learning

การเรียนรู้ของโมเดลแบบไม่มีผู้สอน (Output-free) ซึ่งยังสามารถแบ่งออกได้เป็นสองประเภทคือ 1. Binary Classification กับ 2. Multi-class Classification

Validation Set

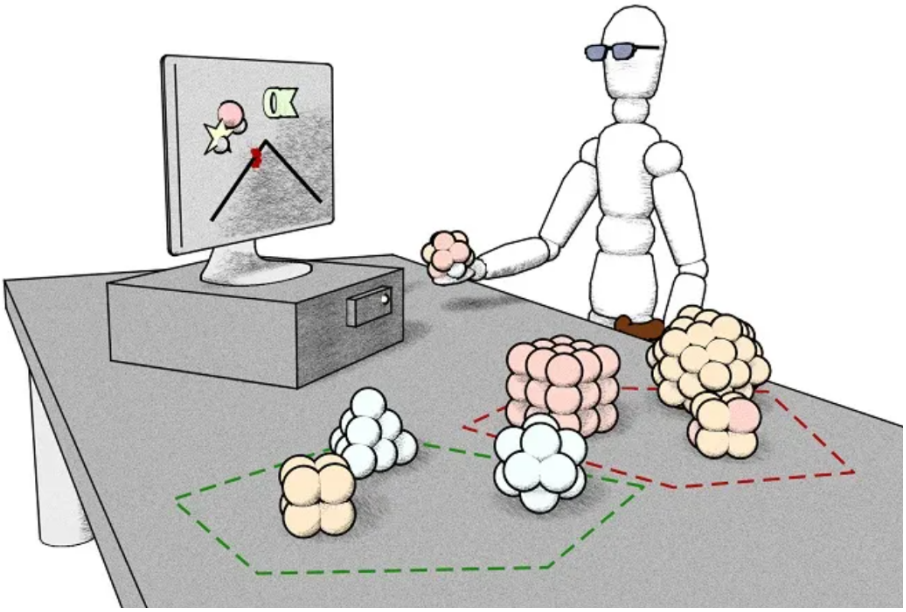
ชุดข้อมูลสำหรับประเมินประสิทธิภาพของโมเดลก่อนที่จะนำไปทดสอบกับ Test Set จริง โดย Dataset ประเภทนี้มักจะถูกนำมาใช้ในการทำ Cross-validation

Variance

ความแตกต่างระหว่างค่าที่ได้จากการทำนายและค่าเฉลี่ยของการทำนายนั้น ๆ กล่าวคือ Variance เป็นการอ้างถึงความเป็นไปในทางเดียวกัน (Consistency)

บทที่ 2

การเรียนรู้แบบมีผู้สอน



ภาพ 2.1 หุ่นยนต์กำลังเรียนรู้หาความเชื่อมโยงระหว่างโครงสร้างกับคุณสมบัติของโมเลกุล (เครดิตภาพ: https://puentesdigitales.com)

การเรียนรู้แบบมีผู้สอนหรือ Supervised Learning เป็นเทคนิคแรก ๆ ที่ถูกพัฒนาขึ้นมาในช่วงยุคเริ่มต้นของ ML ซึ่งเป็นแนวคิดที่ใช้อินพุตและเอาต์พุตในการฝึกสอนโมเดล ซึ่งโมเดลที่ได้ออกมานั้นจะเก็บข้อมูลที่อธิบายความสัมพันธ์ระหว่างอินพุตและเอาต์พุตนั่นเอง (เปรียบเสมือนฟังก์ชันทางคณิตศาสตร์ $f(x)$) ซึ่งผู้เขียนมีความคิดเห็นส่วนตัวว่าการสร้างโมเดลประเภทนี้ง่ายกว่าประเภทอื่นทั้งในแง่ทฤษฎีของอัลกอริทึม การเรียนรู้ของผู้เริ่มต้นศึกษาและการนำไปใช้จริง โดยเทคนิคนี้ได้รับความนิยมมากที่สุดนั่นก็เพราะว่าสามารถนำไปประยุกต์ใช้งานกับโจทย์ที่หลากหลาย

Supervised ML เป็นเทคนิคที่เข้าใจได้ง่ายที่สุดเพราะว่าเป็นการฝึกให้โมเดลมีความสามารถในการเรียนรู้ Target อย่างตรงไปตรงมา จริง ๆ แล้วนิยามของ Supervised ML นั้นมีเยอะมากขึ้นอยู่กับว่าเราต้องการนิยามในเชิงปรัชญา เชิงคณิตศาสตร์ หรือเชิงปฏิบัติ ทุกครั้งที่มีคนถามผู้เขียนว่า Supervised ML คืออะไร ผู้เขียนก็มักจะตอบไปสั้น ๆ แบบไม่จริงจังว่า "*Supervised ML คือการ Fit Curve*" (จริง ๆ แล้ว ML ทุกอัลกอริทึมเลยก็ว่าได้) ซึ่งการให้นิยามแบบนี้เป็นการอธิบายในเชิงปฏิบัติ ตัวอย่างเช่น กำหนดให้มีข้อมูลในตารางที่ 2.1 เป็นความสัมพันธ์ระหว่างโดเมน (Domain) และเรนจ์ (Range) ของฟังก์ชันเลขชี้กำลังง่าย ๆ ดังต่อไปนี้

ตาราง 2.1 ตัวอย่างข้อมูลอินพุตกับเอาต์พุตของฟังก์ชันเลขชี้กำลัง (Exponential Function)

Input (x)	**Output** (y)
1	5
2	25
3	125
4	625
5	?

ถ้าหากถามว่ากรณีที่อินพุต (x) เท่ากับ 5 แล้วเอาต์พุต (y) มีค่าเท่ากับเท่าไร ผู้อ่านก็คงตอบได้ทันทีเลยว่าเท่ากับ 3125 เพราะว่ามนุษย์นั้นมองเห็นรูปแบบที่เกิดขึ้นระหว่าง x กับ y ซึ่งการเปลี่ยนของ y นั้นก็คือเพิ่มขึ้นครั้งละ 5 เท่า โดยสัมพันธ์กับการเปลี่ยนแปลงของ x ที่เพิ่มขึ้นครั้งละ 1 ดังนั้นจากกรณีที่ x เปลี่ยนจาก 4 เป็น 5 ค่าของ y นั้นก็จะต้องเพิ่มขึ้นจาก 625 เป็น 625 $\times$ 5 = 3125 นั่นเอง สำหรับความสัมพันธ์นี้เราสามารถสรุปฟังก์ชันคณิตศาสตร์ได้เป็น $y = 5^x$

ประเด็นที่น่าสนใจก็คือถ้าหากเราถามคำถามเดียวกันนี้กับคอมพิวเตอร์หรือเครื่องจักรว่าคำตอบของ y จะมีค่าเป็นเท่าไรเมื่อ $x = 5$ แน่นอนว่าการรับรู้ของเครื่องจักรนั้นไม่สามารถเทียบเท่ากับการรับรู้ของมนุษย์ถึงแม้ว่าเครื่องจักรจะประมวลผลได้เร็วกว่ามากก็ตาม (ความสามารถในการรับรู้กับความสามารถในการประมวลนั้นต่างกันนะครับ) ดังนั้นสิ่งที่เราต้องการให้เครื่องจักรมีเหมือนมนุษย์ก็คือความสามารถในการเรียนรู้รูปแบบของข้อมูลโดยคำนวณออกมาเป็นฟังก์ชันคณิตศาสตร์ แต่แน่นอนว่าถ้าหากมนุษย์เจอโจทย์หรือข้อมูลที่มีความซับซ้อน เช่น ความสัมพันธ์ที่ไม่เป็นเชิงเส้น ก็ยากที่จะหาคำตอบหรือฟังก์ชันออกมาได้เช่นเดียวกัน ดังนั้นข้อดีของเครื่องจักรก็คือนำความสามารถหรือความเร็วในการคำนวณมาใช้ในการปรับปรุงความสามารถในการเรียนรู้หรือที่เราเรียกว่าการฝึกสอนหรือเทรน (Train) โมเดลนั่นเอง

2.1 การถดถอยเชิงเส้น

เทคนิคของการเรียนรู้แบบมีผู้สอนที่พื้นฐานที่สุดและได้รับความนิยมอย่างมากในช่วงยุคแรกของปัญญาประดิษฐ์ก็คือ การถดถอยแบบเชิงเส้น (Linear Regression) สมมติว่าเราพิจารณาชุดข้อมูลที่มีตัวแปรต้น 2 ตัว $(x_1$ และ $x_2)$ และมีตัวแปรตาม 1 ตัว (y) ซึ่งตัวแปรตามในที่นี้ก็คือคำตอบหรือเป้าหมายที่เรา

ต้องการทำนายนั่นเอง โดยยกตัวอย่างเช่น กำหนดให้ x_1 เป็นจำนวนพันธะเดี่ยวในโมเลกุล x_2 เป็นจำนวนวงอะโรมาติก (Aromatic) ในโมเลกุล และ y เป็นค่าพลังงานรวมของโมเลกุล เราพบว่าเราสามารถสร้างหรือกำหนดสมการที่อธิบายความสัมพันธ์ระหว่างตัวแปรทั้งสามตัวนี้ได้แบบง่าย ๆ ดังนี้

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{2.1}$$

โดยที่ x ในที่นี้คือเวกเตอร์แบบสองมิติในปริภูมิ $\mathbb{R}^2$ และ θ_i คือพารามิเตอร์หรือเรียกว่าน้ำหนัก (Weights) ก็ได้ ซึ่งจะเป็นตัวแปรที่ปรับความเชื่อมโยง (Mapping) ระหว่าง x_i และ y ซึ่งเราสามารถเขียนให้อยู่ในรูปทั่วไปได้ดังนี้

$$h(x) = \sum_{i=0}^{d} \theta_i x_i \tag{2.2}$$

$$= \theta^\top x \tag{2.3}$$

โดยสมการด้านบนนั้นจะเขียนในรูปของผลคูณระหว่างเวกเตอร์ของพารามิเตอร์ $(\theta^\top)$ และเวกเตอร์ x

ลำดับถัดมาคือเราจะทำการปรับพารามิเตอร์ θ อย่างไรเพื่อให้ได้ชุดพารามิเตอร์ที่ทำการ Mapping ได้ดีที่สุด คำตอบก็คือเราสามารถทำได้โดยการกำหนดฟังก์ชันที่จะเป็นตัววัดพารามิเตอร์ θ_i ทีละตัว ซึ่งเรากำหนดและเรียกฟังก์ชันที่จะมาช่วยเราว่า Cost Function (Loss Function) โดยมีรูปสมการทั่วไปดังต่อไปนี้

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{2.4}$$

ซึ่งจะมีความคล้ายกันกับ Ordinary Least Square นั่นเอง โดยในหัวข้อต่อไปเราจะมาดูรายละเอียดของเทคนิคที่เราสามารถนำมาใช้ในการแก้ปัญหาของ Cost Function

ขออธิบายเสริมครับ: สำหรับฟังก์ชันที่มีความเป็นเชิงเส้นนั้นจะต้องสอดคล้องกับเงื่อนไขดังต่อไปนี้

$$f(\vec{x} + \vec{y}) = f(\vec{x}) + f(\vec{y}) \tag{2.5}$$

สำหรับ $\vec{x}$ และ $\vec{y}$ ทุกค่า และเงื่อนไขที่สองคือ

$$f(s\vec{x}) = sf(x) \tag{2.6}$$

ถ้าหากฟังก์ชันไม่สอดคล้องกับเงื่อนไขทั้งสองข้อด้านบน ฟังก์ชันนั้นจะมีความไม่เป็นเชิงเส้น (Nonlinearity)

2.1.1 การถดถอยแบบง่าย

เรามาดูตัวอย่างของกรณีแรกของการถดถอย นั่นก็คือการถดถอยแบบง่าย (Simple Regression) โดยพิจารณาข้อมูลในตาราง 2.2 ดังต่อไปนี้

ตาราง 2.2 แสดงเงินที่ใช้ในการลงทุนการโฆษณาของบริษัทต่าง ๆ กับยอดขายรายปี

บริษัท	วิทยุ	ยอดขาย (ต่อหน่วย)
Amazon	37.8	22.1
Google	39.3	10.4
Facebook	45.9	18.3
Apple	41.3	18.5

ตารางที่ 2.2 แสดงความสัมพันธ์ระหว่างเงินที่ใช้ในการลงทุนในสื่อวิทยุของบริษัทต่าง ๆ กับยอดขายรายปีต่อหน่วยการลงทุน โดยเราสามารถกำหนดตัวแปรได้เป็นตัวแปร x กับ y ซึ่งเป็นอินพุตและเอาต์พุตตามลำดับ ดังสมการต่อไปนี้

$$y = mx + b \tag{2.7}$$

โดยที่ m คือความชันหรือน้ำหนัก (Weight) และ b คือจุดตัดแกน y หรือความโน้มเอียง (Bias) นั่นเอง สำหรับ Loss Function ที่เราจะเลือกใช้นั้น คือ Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2 \tag{2.8}$$

ในการ Optimize ฟังก์ชัน MSE ด้านบนนั้นเราจะใช้เทคนิค Gradient Descent ซึ่งเป็นเทคนิคที่เราจะคำนวณหา Gradient ซึ่งสามารถใช้สมการที่ (2.9) ในการคำนวณได้

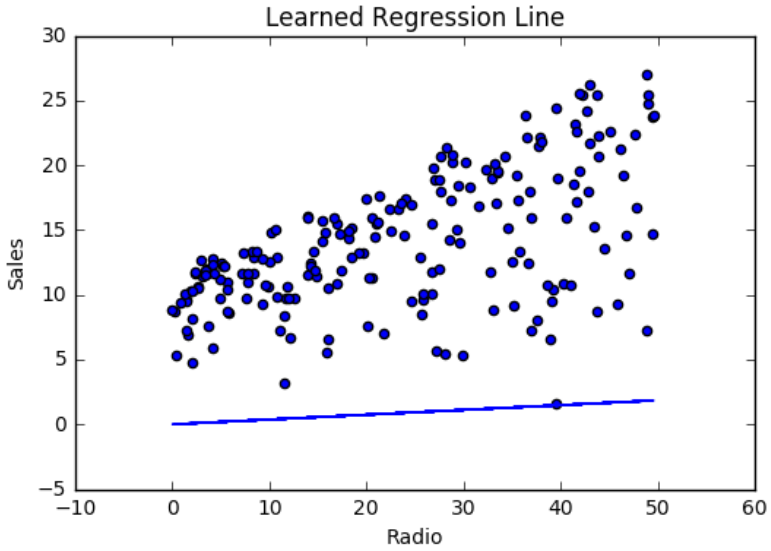
$$f'(m,b) = \begin{bmatrix} \frac{\mathrm{d}f}{\mathrm{d}m} \\ \frac{\mathrm{d}f}{\mathrm{d}b} \end{bmatrix} \tag{2.9}$$

$$= \begin{bmatrix} \frac{1}{N}\sum -x_i \cdot 2(y_i - (mx_i + b)) \\ \frac{1}{N}\sum -1 \cdot 2(y_i - (mx_i + b)) \end{bmatrix} \tag{2.10}$$

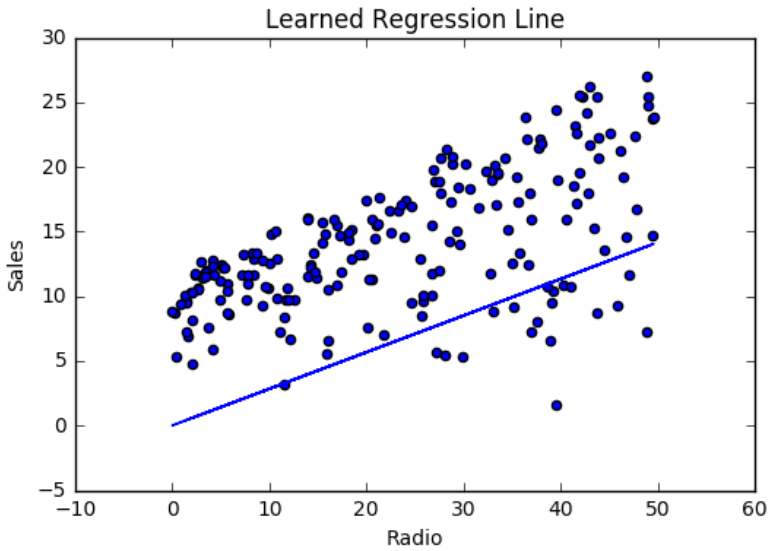
$$= \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N}\sum -2(y_i - (mx_i + b)) \end{bmatrix} \tag{2.11}$$

หลังจากนั้นเราจะทำการฝึกสอนโมเดลโดยการใช้วิธีวนซ้ำเพื่อปรับค่าพารามิเตอร์ต่าง ๆ ทั้ง Weight

และ Bias โดยภาพด้านล่างแสดงการเปลี่ยนแปลงของการทาบเส้นตรงกับข้อมูล (Fitting) ระหว่างการฝึกสอน เราจะพบว่าเส้นตรง (Linear Line) ของเรานั้นจะลากผ่านข้อมูลที่อยู่ในช่วงบริเวณตรงกลางได้ดีขึ้นเรื่อย ๆ



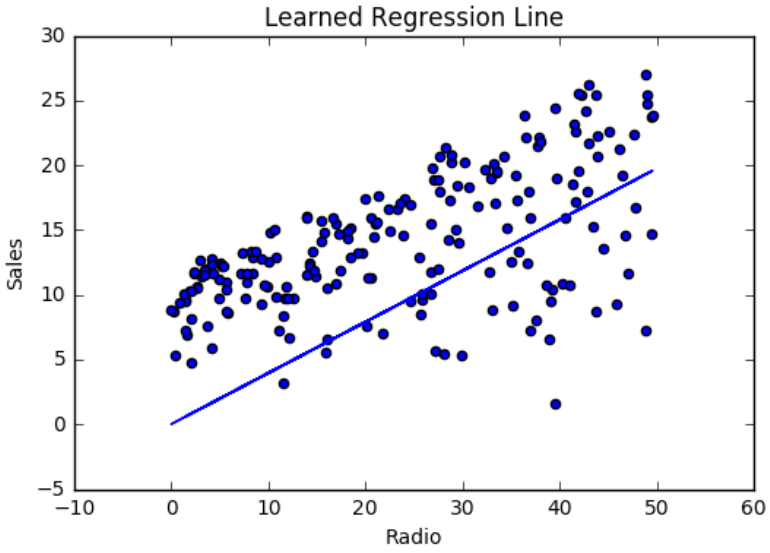
(a) ครั้งที่ 1



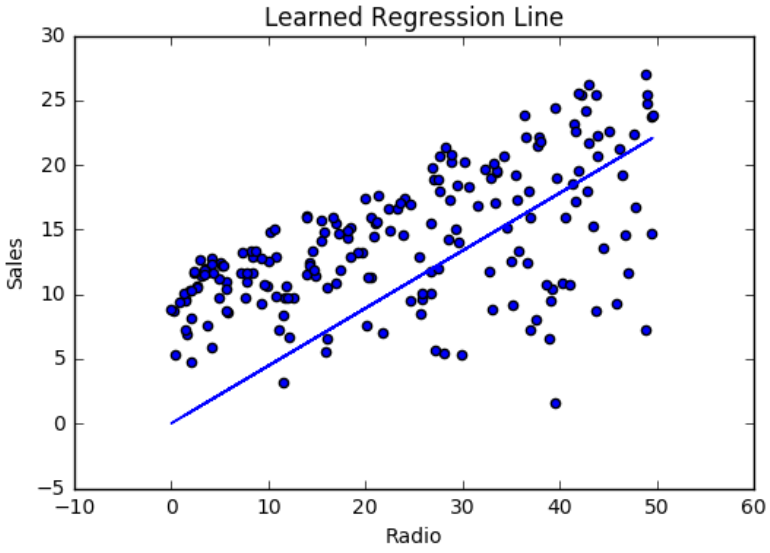
(b) ครั้งที่ 2

2.1.2 การถดถอยแบบหลายตัวแปร

สำหรับกรณีที่เรามีอินพุตหรือ Feature มากกว่าหนึ่งตัว เช่น ข้อมูลในตาราง 2.3 ด้านล่างที่เป็นการนำข้อมูลในตาราง 2.2 มาเพิ่มข้อมูลเงินที่ใช้ในการลงทุนสำหรับการโฆษณาทางสื่อโทรทัศน์และหนังสือพิมพ์



(c) ครั้งที่ 3



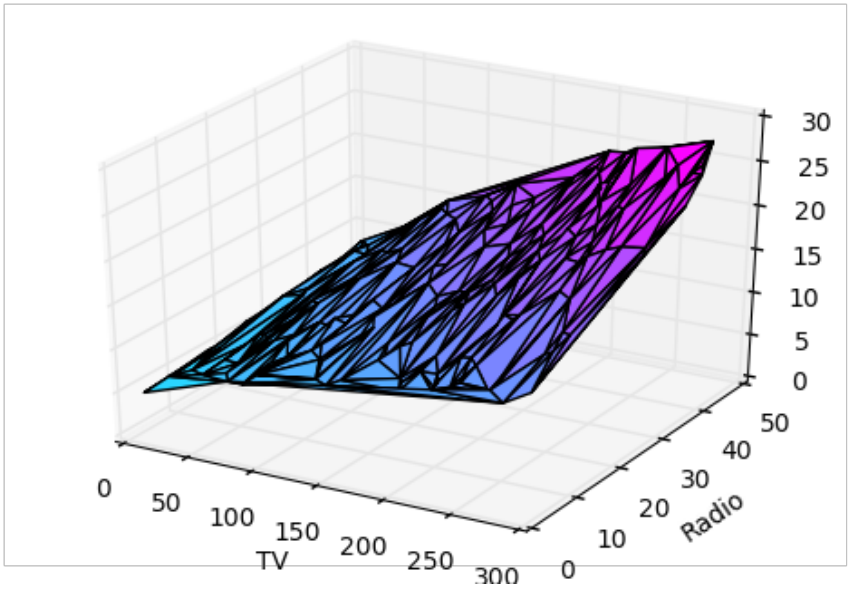
(d) ครั้งที่ 4

ภาพ 2.2 การเปลี่ยนแปลงของเส้นตรงที่ถูกทาบ (Fitting) เข้ากับชุดข้อมูลอย่างง่าย

เข้าไป (คอลัมน์ที่ 3 กับ 4)

ตาราง 2.3 แสดงเงินที่ใช้ในการลงทุนการโฆษณาของบริษัทต่าง ๆ กับยอดขายรายปี

บริษัท	วิทยุ	โทรทัศน์	หนังสือพิมพ์	ยอดขาย (ต่อหน่วย)
Amazon	37.8	230.1	69.1	22.1
Google	39.3	44.5	23.1	10.4
Facebook	45.9	17.2	34.7	18.3
Apple	41.3	151.5	13.2	18.5



ภาพ 2.3 ความสัมพันธ์ของข้อมูลหลายตัวแปร (Multivariables Data)

โดยในกรณีที่ข้อมูลมีความซับซ้อนมากขึ้นแบบนี้ เราไม่สามารถใช้สมการเส้นตรงแบบง่าย ๆ ที่เราใช้ไปก่อนหน้านี้มาอธิบายความสัมพันธ์ระหว่าง Feature ได้ ดังนั้นเราจะต้องมีการกำหนด Loss Function ขึ้นมาใหม่ โดยตอนนี้เราจะต้องมีการกำหนดค่า Weight ขึ้นมา 3 ค่า นั่นคือจากที่เราเคยมีฟังก์ชัน $mx + b$ ก็จะกลายเป็นฟังก์ชัน $W_1x_1 + W_2x_2 + W_3x_3$ โดยจะได้สมการ Loss Function ใหม่ดังนี้

$$\text{MSE} = \frac{1}{2N}\sum_{i=1}^{n}(y_i - (W_1x_1 + W_2x_2 + W_3x_3))^2 \tag{2.12}$$

สำหรับสมการที่เราจะมาใช้ในการหา Gradient ของกรณีนี้สามารถพิสูจน์ได้โดยใช้กฎลูกโซ่ (Chain Rule) เช่นเดียวกับกรณีก่อนหน้านี้

$$f'(W_1) = -x_1(y - (W_1x_1 + W_2x_2 + W_3x_3)) \tag{2.13}$$
$$f'(W_2) = -x_2(y - (W_1x_1 + W_2x_2 + W_3x_3)) \tag{2.14}$$
$$f'(W_3) = -x_3(y - (W_1x_1 + W_2x_2 + W_3x_3)) \tag{2.15}$$

2.2 การจำแนกประเภท

ในหัวข้อนี้จะเป็นการศึกษาโจทย์ปัญหาแบบการจำแนกประเภท (Classification) ซึ่งก็คล้าย ๆ กับโจทย์แบบ Regression แต่ว่าจะต่างกันตรงที่ค่า y ที่เราต้องการทำนายนั้นจะมีความไม่ต่อเนื่อง (Discrete Data) ซึ่งจะตรงข้ามกับ Regression ที่ค่า y จะมีความต่อเนื่อง (Continuous Data) โดยเริ่มต้นเราจะสนใจกรณี Classification แบบง่ายก่อน นั่นก็คือมีประเภทของข้อมูลที่เราจะจำแนกเพียงแค่ 2 ประเภท เรียกว่าโจทย์ปัญหา Binary Classification ซึ่งค่า y จะมีค่าได้แค่ 0 กับ 1 เท่านั้น ซึ่งในภายหลังเราจึงค่อยมาพิจารณากรณีที่มีประเภทมากกว่า 2 ประเภท (Multiple-class Case)

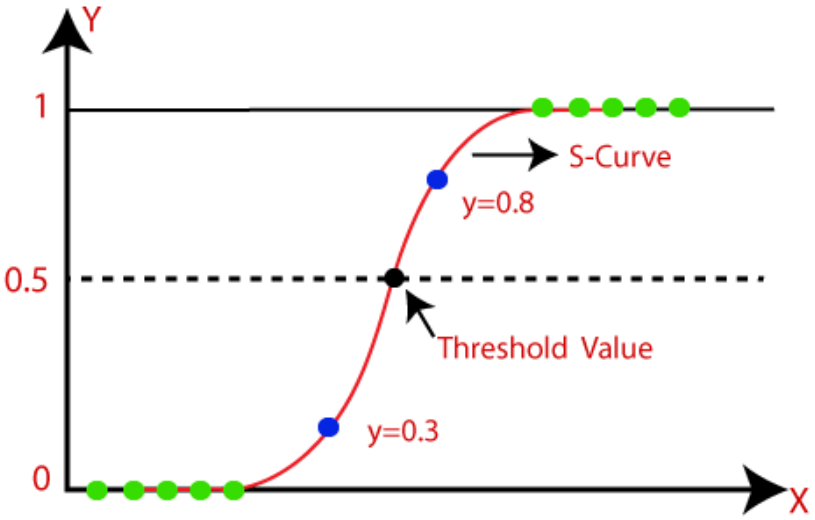
สำหรับการระบุชื่อของประเภทหรือคลาส (Class) นั้น เราจะเรียกคลาส 0 ว่าเป็น Negative Class และเรียกคลาส 1 ว่า Positive Class ซึ่งบ่อยครั้งเรามักจะเจอการใช้เครื่องหมาย - และ + แทนการเขียน 0 กับ 1 โดยที่เราจะกำหนดให้ y^i คือ Label ของข้อมูลลำดับที่ i สำหรับตัวอย่างการฝึกสอน

2.3 การถดถอยแบบโลจิสติค

การวิเคราะห์การถดถอยโลจิสติค (Logistic Regression) เป็นการวิเคราะห์ที่มีเป้าหมายเพื่อประมาณค่าหรือทำนายเหตุการณ์ที่สนใจว่าจะเกิดหรือไม่เกิดเหตุการณ์นั้นภายใต้อิทธิพลของตัวปัจจัยโดยอาศัยฟังก์ชันโลจิสติค (Logistic Function) ที่สร้างขึ้นจากชุดตัวแปรทำนายที่เป็นตัวแปรที่มีข้อมูลอยู่ในระดับช่วงเป็นอย่างน้อย โดยที่ระหว่างตัวแปรทำนายจะต้องมีความสัมพันธ์กันต่ำและในการวิเคราะห์จะต้องใช้ขนาดตัวแปรทำนายไม่ต่ำกว่า 30 ตัวแปร Logistic Regression จัดเป็นเครื่องมือวิเคราะห์ข้อมูลในการศึกษาวิจัยที่มีวัตถุประสงค์เพื่อทำนายเหตุการณ์หรือประเมินความเสี่ยง (เช่น “เสี่ยง” หรือ “ไม่เสี่ยง”) จึงมีการประยุกต์ใช้ในงานวิจัยหลากหลายสาขา ทั้งสาขาทางการแพทย์ วิศวกรรมศาสตร์ นิเวศวิทยา เศรษฐศาสตร์และสังคมศาสตร์

นอกจากการทำนายการเกิดเหตุการณ์ที่สนใจว่าเกิด (0) หรือไม่เกิด (1) ได้แล้ว Logistic Regression ยังสามารถทำนายค่าความน่าจะเป็นของเหตุการณ์ได้ด้วย (ค่าระหว่าง 0 กับ 1) จริง ๆ แล้วเทคนิค Logistic Regression นั้นคล้ายกับ Linear Regression มาก โดยทั้งสองเทคนิคนี้ต่างกันตรงที่การนำไปใช้งาน โดยเราใช้ Linear Regression สำหรับการแก้ปัญหาการถดถอยแต่ Logistic Regression สำหรับการแก้ปัญหาการแยกคลาสหรือจัดกลุ่มของข้อมูล

การทำ Logistic Regression นั้นเราไม่ได้ทำการ Fit เส้น Regression กับข้อมูลแต่จะเป็นการ Fit กับ



ภาพ 2.4 Logistic Function หรือ Sigmoid Function

Logistic Function (ภาพที่ 2.4) แทนซึ่งจะเป็นตัวที่ทำนายค่าออกมา โดย Logistic Function ที่เราใช้นั้นจริง ๆ แล้วก็คือเส้นโค้งตัว S หรือ Sigmoid Function นั่นเอง โดยมีนิยามทางคณิตศาสตร์ดังต่อไปนี้

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{2.16}$$

สำหรับกรณีที่กำหนดให้ $k = 1$, $x_0 = 0$, และ $L = 1$ เราจะได้สมการดังต่อไปนี้

$$\begin{aligned} f(x) &= \frac{1}{1 + e^{-x}} \\ &= \frac{e^x}{e^x + 1} \\ &= \frac{1}{2} + \frac{1}{2} \tanh\left(\frac{x}{2}\right) \end{aligned} \tag{2.17}$$

ซึ่งเราเรียกสมการที่ (2.17) นี้ว่าฟังก์ชันโลจิสติคมาตรฐาน (Standard Logistic Function)

โดยเทคนิคนี้ถูกนำมาใช้เยอะมากใน ML เพราะว่ามีความสามารถในการทำนายค่าความน่าจะเป็นและแยกข้อมูลโดยใช้ชุดข้อมูลที่มีความต่อเนื่องหรือแบบไม่ต่อเนื่องก็ได้ หนังสือบางเล่มหรือบทความวิจัยบางฉบับเรียก Logistic Regression ว่า Maximum-entropy Classification (MaxEnt) หรือ Log-linear Classifier เพราะว่าถูกนำมาใช้กับโจทย์ปัญหา Classification มากกว่า Regression ตามที่ได้อธิบายไว้

โค้ดด้านล่างคือตัวอย่างการเรียกใช้ฟังก์ชัน `LogisticRegression` ของไลบรารี่ Scikit-Learn สำหรับการฝึกสอนโมเดลด้วย Logistic Regression โดยใช้ข้อมูลตัวอย่างที่สมมติขึ้นมา

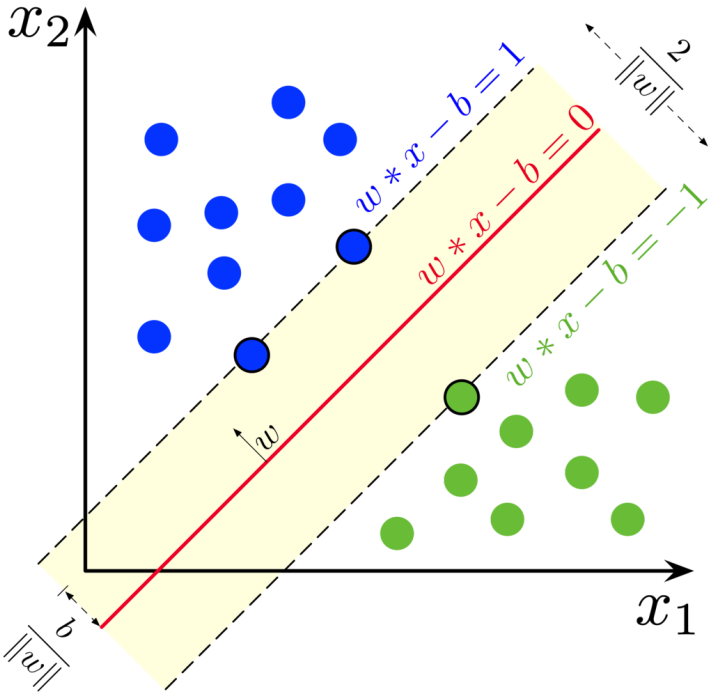
```python
import numpy as np
from sklearn.linear_model import LogisticRegression

# Create dataset
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Create a logistic regression model
model = LogisticRegression(solver='liblinear', random_state=0)

# Train the model
model.fit(x, y)

# Get results
model.classes_
# Output
array([0, 1])
model.intercept_
# Output
array([-1.04608067])
model.coef_
# Output
array([[0.51491375]])
```

เราสามารถแสดงเมทริกซ์ของค่าความน่าจะเป็น (Probability Matrix) ของข้อมูลแต่ละตัวได้ด้วย ดังนี้

```python
model.predict_proba(x)
# Output
array([[0.74002157, 0.25997843],
       [0.62975524, 0.37024476],
       [0.5040632 , 0.4959368 ],
       [0.37785549, 0.62214451],
       [0.26628093, 0.73371907],
       [0.17821501, 0.82178499],
       [0.11472079, 0.88527921],
       [0.07186982, 0.92813018],
       [0.04422513, 0.95577487],
       [0.02690569, 0.97309431]])
```

2.4 เครื่องเวกเตอร์ค้ำยัน

เครื่องเวกเตอร์ค้ำยัน (Support Vector Machine หรือ SVM) เป็นวิธีเคอร์เนลแบบหนึ่งที่มีความคล้ายกับ GPR หรือ KRR เป็นอย่างมาก โดย SVM จะทำการทำนายค่าโดยทำการเปรียบเทียบข้อมูลใหม่กับข้อมูลอ้างอิงด้วยฟังก์ชัน $k(x_i, x_j)$ และคำนวณค่าความเหมือน (Similarity) ระหว่างจุดสองจุด ซึ่งเราเรียกสิ่งนี้ว่าเคอร์เนล (Kernel) โดยความซับซ้อนของวิธีนี้นั้นไม่มีกฎเกณฑ์ที่แน่นอนในการกำหนด (Arbitrarily) ดังนั้นเราจะต้องทำการปรับ Hyperparameters เพื่อให้มีความเหมาะสมและสามารถควบคุมความซับซ้อนของวิธี SVM ซึ่งเราเรียกวิธีการปรับนี้ว่า Regularization เพื่อทำการหลีกเลี่ยงปัญหา Overfit นั่นเอง ผู้อ่านสามารถศึกษาเคอร์เนลเพิ่มเติมได้ในหัวข้อที่ 3.1



ภาพ 2.5 Maximum Margin Hyperplan และ Margi สำหรับการฝึกสอนโมเดลของชุดข้อมูลตัวอย่างที่มี 2 คลาสด้วย Support Vector Machine (เครดิตภาพ: https://en.wikipedia.org/wiki/Support_vector_machine)

ภาพที่ 2.5 แสดงชุดข้อมูลตัวอย่างที่มี 2 คลาส (สีน้ำเงินกับสีเขียว) โดยมีระนาบระยะห่างที่มากที่สุด (Maximum Margin Hyperplan หรือ MMH) เป็นตัวแบ่งข้อมูลซึ่งอ้างอิงโดยจุดข้อมูลที่อยู่ใกล้กับ Hyperplan ซึ่งจุดข้อมูลเหล่านี้มีชื่อเรียกว่าเวกเตอร์ค้ำยัน (Support Vector) โดยเราคำนวณ Support Vector จากช่องว่าง (Margin) ระหว่างคลาสทั้ง 2 คลาสโดยใช้ระยะห่างที่น้อยที่สุด ดังนั้นเป้าหมายของการฝึกสอนโมเดลด้วย SVM ก็คือการหา Hyperplan ที่สามารถแบ่งข้อมูลทั้ง 2 คลาสออกจากกันได้ดีที่สุด

โค้ดด้านล่างคือตัวอย่างการเรียกใช้ฟังก์ชัน `svm` ของไลบรารี่ Scikit-Learn สำหรับการฝึกสอนโมเดล

ด้วย SVM โดยใช้ข้อมูลตัวอย่างที่สมมติขึ้นมา

```
import numpy as np
from sklearn import svm

# Create dataset
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
x_test = x + 1.2

# Create a SVM classifier using linear kernel
clf = svm.SVC(kernel='linear')

# Train the model
clf.fit(x, y)

# Predict the response for test dataset
clf.predict(x_test)
# Output
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

2.5 เทคนิคการเรียนรู้แบบมีผู้สอนแบบอื่น ๆ

2.5.1 Partial Least Squares (PLS)

วิธีกำลังสองน้อยที่สุดบางส่วน (Partial Least Squares หรือ PLS) เป็นวิธีเชิงสถิติที่ใช้สำหรับการวิเคราะห์หลายตัวแปรเพื่อสร้างตัวแบบความสัมพันธ์ระหว่างกลุ่มของตัวแปรทำนาย (Predictor Variable) โดยอาศัยตัวแปรแฝง (Latent variable) ซึ่งเทคนิคนี้มีความคล้ายกับ Principle Component Analysis (PCA) ซึ่งจะเป็นการลดจำนวนมิติของข้อมูล[23] ในช่วงยุคเริ่มต้นที่มีการใช้ปัญญาประดิษฐ์ในงานด้านเคมีนั้น เทคนิคนี้ได้ถูกนำมาใช้อย่างแพร่หลาย เช่น นำมาใช้สำหรับการระบุ Vibrational Bands สำหรับ Vibrational Spectra และนำผลที่ได้มาเปรียบเทียบกับค่าการทำนายที่ได้จากวิธีอื่น เช่น ANN และ PCA-ANN

2.5.2 Gaussian Process Regression (GPR)

การถดถอยของกระบวนการเกาส์เซียน (Gaussian Process Regression หรือ GPR) เป็นวิธีการถดถอยของเบส์แบบหนึ่งโดยใช้ Kernel Function ที่สามารถบ่งบอกหรือแสดงค่าความแปรปรวน (Covariance) ในขั้นตอน Gaussian Process ได้[24] โดย GPR จะทำการสร้างโมเดลแบบ Non-parametric และสามารถคำนวณ

ค่าความเชื่อมั่น (Confidence Intervals) ไปพร้อม ๆ กับการทำนาย รายละเอียดเพิ่มเติมของ GPR สามารถศึกษาได้ในหัวข้อ 3.5

2.5.3 Random Forest

การสุ่มป่าไม้ (Random Forest หรือ RF) เป็นวิธีหนึ่งในกลุ่มของโมเดลที่เรียกว่าการเรียนรู้แบบกลุ่มก้อน (Ensemble Learning) ที่มีหลักการคือการฝึกสอนโมเดลที่เหมือนกันหลาย ๆ ครั้ง (Multitude) บนข้อมูลชุดเดียวกัน โดยแต่ละครั้งของการเทรนจะเลือกส่วนของข้อมูลที่ฝึกสอนไม่เหมือนกัน แล้วนำการตัดสินใจของโมเดลเหล่านั้นมาโหวตเลือกกันว่า Class ไหนถูกเลือกมากที่สุด[25,26]

ตัวอย่างการเขียนโค้ดโมเดล Random Forest สำหรับการทำ Regression

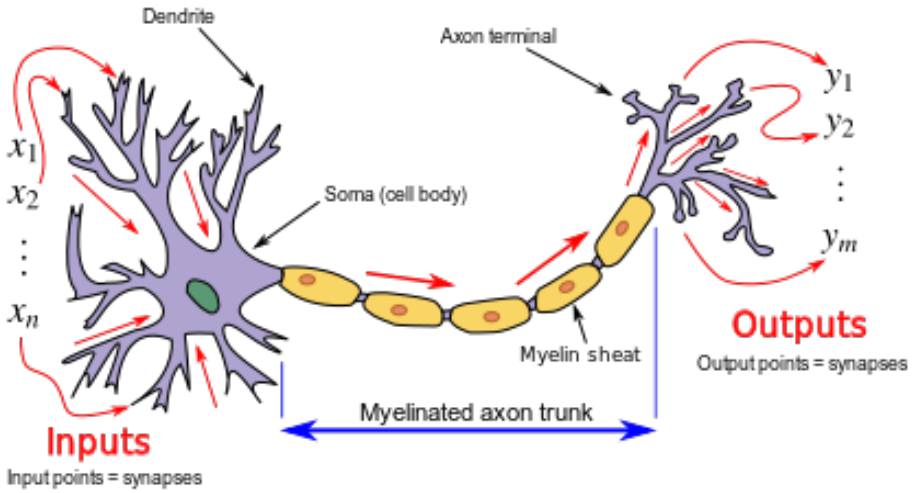
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression

X, y = make_regression(n_features=4, n_informative=2,
                       random_state=0, shuffle=False)
regr = RandomForestRegressor(max_depth=2, random_state=0)
regr.fit(X, y)

print(regr.predict([[0, 0, 0, 0]]))
# Output
[-8.32987858]
```

ตัวอย่างการเขียนโค้ดโมเดล Random Forest สำหรับการทำ Classification

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X, y)

print(clf.predict([[0, 0, 0, 0]]))
# Output
[1]
```

2.5.4 Artificial Neural Network

โครงข่ายประสาทเทียมประดิษฐ์ (Artificial Neural Network หรือ ANN) หรือเรียกว่าโครงข่ายประสาทเทียม (Neural Network หรือ Neural Net) เป็นอัลกอริทึมรูปแบบหนึ่งที่เลียนแบบการทำงานของสมองมนุษย์ โดยทำการสร้างโมเดลเรียนรู้ที่ประกอบไปด้วยชั้นเรียนรู้ระหว่างกลาง (Hidden Layer) และหน่วยย่อยที่เกิดการเรียนรู้ (Node หรือ Artificial Neuron หรือ Unit)



ภาพ 2.6 การรับส่งข้อมูลภายในเซลล์ประสาท

จริง ๆ แล้ว Neural Network ก็คือการจำลองสมองมนุษย์โดยพยายามสร้างองค์ประกอบต่าง ๆ ให้มีความคล้ายกันให้มากที่สุด เช่น ในสมองมีเซลล์ประสาท (Neurons) และจุดประสานประสาท (Synapses) แต่ละเซลล์ประสาทประกอบด้วยปลายในการรับกระแสประสาทเรียกว่า “เดนไดรท์” (Dendrite) ซึ่งเป็นอินพุตและปลายในการส่งกระแสประสาทเรียกว่าแอคซอน (Axon) ซึ่งเปรียบเหมือนเป็นเอาต์พุตของเซลล์

โดยโมเดล Neural Network ที่มีการนำไปใช้มากที่สุดคือเครือข่ายประสาทแบบป้อนไปหน้า (Feed-forward Network) และโมเดล Neural Network ยังสามารถแบ่งออกได้เป็นหลายประเภท ดังนี้

- เพอร์เซ็ปตรอนชั้นเดียว (Single-layer Perceptron)
- เพอร์เซ็ปตรอนหลายชั้น (Multi-layer Perceptron)
- โครงข่ายแบบวนซ้ำ (Recurrent Neural Network)
- แผนผังจัดระเบียบเองได้ (Self-organizing Map)
- เครื่องจักรโบลทซ์แมน (Boltzmann Machine)
- กลไกแบบคณะกรรมการ (Committee of Machines)

- โครงข่ายความสัมพันธ์ (Associative Neural Network)
- โครงข่ายกึ่งสำเร็จรูป (Instantaneously Trained Networks)
- โครงข่ายแบบยิงกระตุ้น (Spiking Neural Networks)

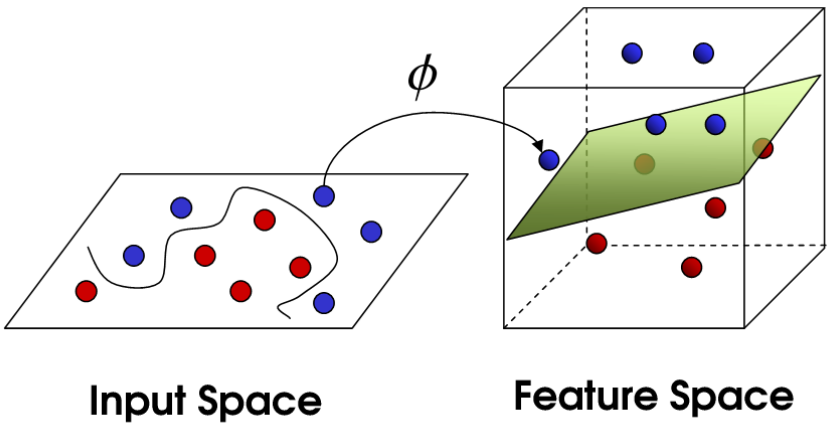
โดยในหนังสือเล่มนี้จะอธิบายเฉพาะ Neural Network แบบเพอร์เซ็ปตรอนชั้นเดียวและเพอร์เซ็ปตรอนหลายชั้น (บทที่ 4) สำหรับผู้อ่านที่สนใจศึกษารายละเอียดของ Neural Network ประเภทอื่น ๆ นั้น สามารถศึกษาได้จากหนังสือเฉพาะทางด้าน Neural Network เช่น “Deep Learning” เขียนโดย Ian Goodfellow, Yoshua Bengio และ Aaron Courville[27] รายละเอียดเพิ่มเติมดูได้ที่เว็บไซต์ https://www.deeplearningbook.org

บทที่ 3

วิธีเคอร์เนล

3.1 เคอร์เนลคืออะไร

ในบทที่แล้วเราได้เรียนรู้เทคนิค Linear Regression (การถดถอยแบบเส้นตรง) กันไปแล้ว ซึ่งเป็นกรณีที่เราเจอปัญหาที่เกี่ยวข้องกับความสัมพันธ์ของตัวแปรสองตัว โดยเราสามารถทำการ Fit สมการเชิงเส้นของอินพุต (x) ให้เข้ากับชุดข้อมูล (Training Data) แล้วถ้าหากค่าเอาต์พุต (y) ที่เราต้องการทำนายนั้นสามารถถูกทำนายหรืออธิบายได้ดีกว่าด้วยสมการไม่เชิงเส้น (Nonlinear Function) ของตัวแปร x นั้น เราสามารถใช้สิ่งที่เรียกว่าเคอร์เนล (Kernel) ในการหาความสัมพันธ์ระหว่างอินพุตกับเอาต์พุตได้ ซึ่งเราจะมาทำความรู้จักและเรียนรู้เคอร์เนลกันในบทนี้



ภาพ 3.1 การทำ Mapping จากปริภูมิ 2 มิติไปเป็นปริภูมิ 3 มิติซึ่ง Observation ของเราสามารถถูกแยกได้โดยฟังก์ชันเชิงเส้น

ถ้าสมมติว่าผู้เขียนเริ่มต้นด้วยการยกสมการทางคณิตศาสตร์มาอธิบายว่าเคอร์เนลคืออะไร ผู้อ่านหลายท่านก็อาจจะสับสนได้ ดังนั้นผู้จึงขอยกเริ่มต้นด้วยการอธิบายง่าย ๆ แบบนี้ว่าถ้าสมมติเรามีชุดข้อมูลที่อยู่ในปริภูมิ 2 มิติ (Two-dimensional Space) หรือที่เรียกว่า Input Space ดังที่แสดงในภาพที่ 3.1 เราจะพบว่าเราไม่สามารถหาฟังก์ชันเชิงเส้นที่สามารถแยกข้อมูลจุดสีแดงออกจากสุดสีน้ำเงินได้ แต่ถ้าหากเราแปลงข้อมูลจากปริภูมิ 2 มิติให้ไปอยู่ในปริภูมิ 3 มิติซึ่งเราจะเรียกปริภูมินี้ว่า Feature Space ตามรูปทางด้านขวา เราพบว่าข้อมูลของเราหลังจากที่ถูกแปลงนั้นจะสามารถถูกแยกออกจากกันได้ด้วยฟังก์ชันเชิงเส้น (ในที่นี้คือระนาบที่แบ่งข้อมูลทั้งสองคลาสออกจากกัน) ซึ่งกระบวนการนี้เรียกว่า Mapping หลังจากนั้นให้เราทำ Mapping อีกครั้งหนึ่งโดยแปลงย้อนกลับไปสู่ปริภูมิ 2 มิติ ซึ่งผลลัพธ์ที่ได้จะเป็นฟังก์ชันแบบไม่เชิงเส้นที่อยู่ในมิติที่ต่ำลงนั่นเอง

สรุปสั้น ๆ คือวิธีเคอร์เนลเป็นการประยุกต์ใช้ฟังก์ชันเชิงเส้นกับปัญหาที่เป็นแบบไม่เชิงเส้นโดยการแปลงข้อมูลให้อยู่ในปริภูมิที่มีมิติที่สูงขึ้น (อาจจะ 3 มิติหรือสูงกว่านี้ก็ได้) โดยที่เราไม่จำเป็นที่จะต้องเข้าใจปริภูมิมิติสูงเหล่านั้น (จริง ๆ แล้วถ้าเราเข้าใจได้ก็ดี แต่ว่าการอธิบายปริมาณทางคณิตศาสตร์ที่มีจำนวนมิติมากกว่าสามมิตินั้นเป็นสิ่งที่เข้าใจได้ยาก)

3.2 คณิตศาสตร์ของเคอร์เนล

ก่อนที่เราจะลงลึกไปที่ตัวทฤษฎีของเคอร์เนลนั้น เราควรจะต้องมาทำความเข้าใจสิ่งที่เป็นพื้นฐานกันก่อนนั่นก็คือ Feature Map ซึ่งเป็นสิ่งที่ทำการเชื่อมโยง Attribute ให้เข้ากับ Feature ซึ่งเราเรียกกระบวนการนี้ว่า Mapping ตามที่ได้อธิบายไปก่อนหน้านี้

เริ่มต้นด้วยการพิจารณาการ Fit ฟังก์ชันแบบ Cubic Function โดยสมการดังนี้

$$y = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 \tag{3.1}$$

จะเห็นได้ว่าเราสามารถที่จะมอง Cubic Function ด้านบนเป็นสมการเชิงเส้นง่าย ๆ ซึ่งสมการ (3.1) นั้นจะขึ้นอยู่กับ Feature Variables (x) ที่เรากำหนดไว้นั่นเอง คราวนี้เราลองมากำหนดฟังก์ชันใหม่โดยอ้างอิงสมการเดิม ซึ่งเราจะมีการกำหนดเซตของตัวแปร x ตัวใหม่ขึ้นมานั่นคือ

$$\begin{aligned} y &= \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 \\ &= \theta^\top \phi(x) \end{aligned} \tag{3.2}$$

โดยที่ ϕ คือเวกเตอร์ของตัวแปรอินพุต ดังนี้

$$\phi = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \tag{3.3}$$

และ $\theta^\top$ เป็นเวกเตอร์ของพารามิเตอร์ θ_i ดังนี้

$$\theta^\top = \begin{bmatrix} \theta_o & \theta_1 & \theta_2 & \theta_3 \end{bmatrix} \tag{3.4}$$

ซึ่งทั้งสองเวกเตอร์นี้ก็คูณกันแบบ Dot Product ตามสมการที่ (3.2)

คำถามที่ตามมาก็คือเราจะแยกความแตกต่างระหว่างสมการ (3.1) กับ (3.2) ได้อย่างไร คำตอบก็คือเราสามารถทำได้โดยการกำหนดให้ตัวแปรอินพุต x ของเรานั้นเป็น Attribute เมื่อเราทำการ Map หรือเชื่อมโยงตัวแปร x ของเราไปยังปริมาณตัวใหม่ที่เป็น $\phi(x)$ เราจะเรียกปริมาณตัวนี้ว่า Feature และฟังก์ชันที่เราใช้ในการ Mapping นั้นเราเรียกว่า Feature Map (ϕ) ซึ่งเป็นตัวที่ทำการเชื่อมโยงความสัมพันธ์ของ Attribute ไปยัง Feature ตามที่ได้เกริ่นไว้ก่อนหน้านี้

ดังนั้นโจทย์ของเราในการทำ Regression ก็คือการหาอัลกอริทึม Gradient Descent ที่จะนำมาใช้ในการ Fitting โมเดลของเรา (อันที่จริงแล้ว $\theta^\top\phi(x)$ ก็คือโมเดลของเรานั่นเอง) โดยหนึ่งในอัลกอริทึมที่สามารถนำมาใช้ในการ Fitting ได้อย่างมีประสิทธิภาพก็คือ Stochastic Gradient Descent ซึ่งมีสมการดังต่อไปนี้

$$\theta := \theta + \alpha(y^i - \theta^\top\phi(x^i))\phi(x^i) \tag{3.5}$$

โดยที่ α คือขนาดของการก้าวเดิน (Step Size) หรืออัตราเร็วของการเรียนรู้ (Learning Rate) ซึ่งเป็นพารามิเตอร์ที่จะปรับความเร็วในการปรับความเหมาะสม (Optimize) เกรเดียนต์ (สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับการพิสูจน์สมการที่ (3.5) ผู้อ่านสามารถอ่านได้จากหนังสือปัญญาประดิษฐ์) อย่างไรก็ตาม ปัญหาอย่างหนึ่งของ Stochastic Gradient Descent นั้นก็คือไม่สามารถที่จะหาผลเฉลยได้ง่าย จึงทำให้การฝึกสอนโมเดลนั้นมีความสิ้นเปลืองในการคำนวณเป็นอย่างมาก (Computationally Expensive) โดยเฉพาะอย่างยิ่งเมื่อ Feature ของเรา $(\phi(x))$ นั้นมีจำนวนมิติที่เยอะมาก ๆ (ผู้อ่านสามารถศึกษารายละเอียดของ Stochastic Gradient Descent ได้ในหัวข้อที่ 4.2.2)

สำหรับนิยามของเคอร์เนล (K) นั้นจริง ๆ แล้วมีหลายนิยามมาก ขึ้นอยู่กับว่าต้องการจะให้นิยามในบริบททางคณิตศาสตร์หรือสถิติ โดยส่วนตัวของเขียนนั้นคิดว่าเคอร์เนลเป็นวิธีทางสถิติที่เกี่ยวข้องกับการหาความเชื่อมโยงระหว่างตัวแปรสองตัว ซึ่งความเชื่อมโยงในที่นี้ก็คือความคล้ายคลึงกัน (Similarity) ผู้อ่านบางท่านอาจจะเคยได้ยินหรือเคยใช้วิธี เช่น Jaccard Similarity หรือ Cosine Similarity มาก่อนบ้าง

คราวนี้เรามาดูนิยามทางคณิตศาสตร์ของเคอร์เนลกันครับ เริ่มต้นกำหนดเคอร์เนลให้อยู่ในรูปของ Feature Map (ϕ) ซึ่งเป็นฟังก์ชันที่ทำการ Mapping ปริภูมิของตัวแปรอินพุต x ที่ได้อธิบายไว้ก่อนหน้านี้ $(\chi \times \chi \to \mathbb{R})$ ดังนี้

$$K(x,z) = \langle \phi(x), \phi(z) \rangle \tag{3.6}$$

ซึ่งเราสามารถคำนวณ $\langle \phi(x), \phi(z) \rangle$ ได้โดยการใช้สมการต่อไปนี้

$$\begin{aligned}
\langle \phi(x), \phi(z) \rangle =& 1 + \sum_{i=1}^{d} x_i z_i + \sum_{i,j \in \{1,\ldots,d\}} x_i x_j z_i z_j \\
&+ \sum_{i,j,k \in \{1,\ldots,d\}} x_i x_j x_k z_i z_j z_k && (3.7) \\
=& 1 + \sum_{i=1}^{d} x_i z_i + \left(\sum_{i=1}^{d} x_i z_i \right)^2 + \left(\sum_{i=1}^{d} x_i z_i \right)^3 && (3.8) \\
=& 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 && (3.9)
\end{aligned}$$

อธิบายง่าย ๆ ก็คือเราจะคำนวณพจน์แรก $\langle x, z \rangle$ ของสมการ (3.9) ก่อน หลังจากนั้นจึงคำนวณพจน์อื่น ๆ ที่เหลือ โดยกำหนดให้ i, j เป็นสมาชิกของเซต $\{1, \ldots, n\}$

3.3 ฟังก์ชันเคอร์เนลและคุณสมบัติของเคอร์เนล

ในหัวข้อนี้เราจะมาดูรายละเอียดของเคอร์เนลกันว่า $K(x,z)$ มีคุณสมบัติอะไรที่น่าสนใจบ้าง สำหรับสัญลักษณ์ที่เราจะกำหนดขึ้นมาเพื่ออธิบายเคอร์เนลนั้นจะเป็น $K(\cdot,\cdot)$ หรือเรียกง่าย ๆ ว่าเป็นฟังก์ชันเคอร์เนล (Kernel Function) ก็ได้ ผู้เขียนขอยกตัวอย่างของฟังก์ชันเคอร์เนลแบบเรียบง่ายให้ดูกันก่อน ดังนี้

$$K(x,z) = (x^\top z)^2 \tag{3.10}$$

ซึ่งเราสามารถจัดรูปสมการใหม่ได้เป็น

$$\begin{aligned}
K(x,z) &= \left(\sum_{i=1}^{d} x_i z_i \right) \left(\sum_{j=1}^{d} x_j z_j \right) && (3.11) \\
&= \sum_{i=1}^{d} \sum_{j=1}^{d} x_i x_j z_i z_j && (3.12) \\
&= \sum_{i,j=1}^{d} (x_i x_j)(z_i z_j) && (3.13)
\end{aligned}$$

ซึ่งจะเห็นได้ชัดเลยว่าจริง ๆ แล้วนั้น $K(x,z) = \langle \phi(x), \phi(z) \rangle$ เป็นฟังก์ชันเคอร์เนลที่สอดคล้องกับ Feature Mapping (ϕ) โดยตัวอย่างเช่น กรณีที่ $d = 3$ นั้นมีสมการเป็น

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix} \tag{3.14}$$

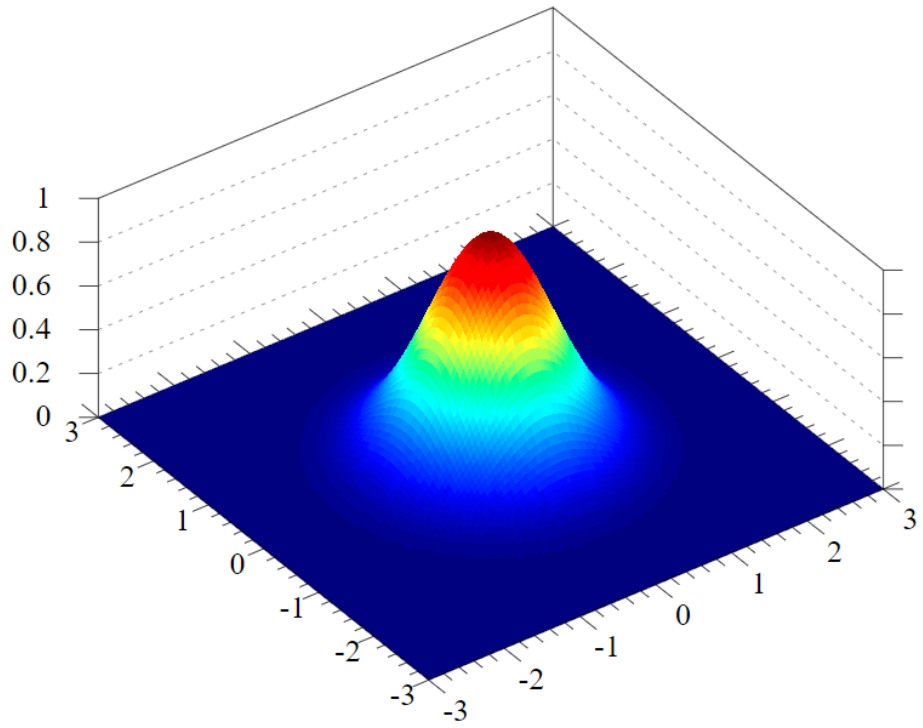
เราลองมาดูตัวอย่างที่สองของ $K(\cdot,\cdot)$ ซึ่งถูกกำหนดด้วยฟังก์ชันเชิงเส้นดังต่อไปนี้

$$K(x,z) = (x^\top z + c)^2 \tag{3.15}$$

$$= \sum_{i,j=1}^{d} (x_ix_j)(z_iz_j) + \sum_{i=1}^{d} \left(\sqrt{2c}x_i\right)\left(\sqrt{2c}z_i\right) + c^2 \tag{3.16}$$

โดยที่ฟังก์ชันเคอร์เนลด้านบนนี้ก็จะคล้าย ๆ กับก่อนหน้านี้แต่จะมีความแตกต่างตรงที่มีการเพิ่มพารามิเตอร์ c เข้ามา ซึ่งเป็นสิ่งที่กำหนดการถ่วงน้ำหนัก (Weighting) ระหว่าง x_i และ x_ix_j โดยที่เรามองได้ง่าย ๆ ก็คือ เคอร์เนล $K(x,z) = (x^\top z + c)^2$ นั้นจะมีความสอดคล้องกับ Feature Mapping ไปยังปริภูมิของ $\binom{d+k}{k}$ โดยที่ Feature Mapping ของกรณีนี้ที่ $d = 3$ นั้นมีสมการดังต่อไปนี้

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix} \tag{3.17}$$



ภาพ 3.2 ตัวอย่างพื้นผิวการกระจายตัวเกาส์เซียน (Gaussian Distribution) ในปริภูมิ 3 มิติ

คราวนี้เราลองมามองเคอร์เนลให้เป็นเมตริกหรือตัววัดความคล้ายคลึงกันระหว่าง Feature Mapping (Similarity Metrics) เราเริ่มต้นด้วยสมมติฐานที่ว่าถ้ากรณีที่ $\phi(x)$ กับ $\phi(z)$ บนปริภูมินั้นมีความใกล้กันมาก ๆ เราอาจจะคาดการณ์ได้ว่า $K(x,z) = \phi(x)^\top \phi(z$ จะมีขนาดที่ใหญ่มากเพราะว่ามีการซ้อนทับกันเยอะ (เรานิยามให้การซ้อนทับกันหรือ Overlap นั้นเป็นความคล้ายคลึงกัน) แต่ในกรณีที่ตรงข้ามกัน ถ้าหาก $\phi(x)$ กับ $\phi(z)$ อยู่ห่างกันมากก็จะทำให้การ Overlap นั้นมีน้อย จึงทำขนาดของเคอร์เนล $K(x,z) = \phi(x)^\top \phi(z)$ มีขนาดที่เล็กลงตามไปด้วย ซึ่งการที่เราสามารถนิยามเคอร์เนลให้เป็นมาตรวัดความคล้ายคลึงกันระหว่าง x และ z นั้นมีประโยชน์อย่างมาก เพราะเราสามารถนำไปแก้ปัญหาหลาย ๆ อย่างได้ แต่ว่าฟังก์ชันที่เราเลือกมาใช้ในการอธิบายความแตกต่างของทั้งสองตัวแปรนั้นจะต้องมีความสมเหตุสมผล โดยฟังก์ชันที่ได้รับความนิยมในการนำมาใช้เป็นฟังก์ชันเคอร์เนลนั้นก็คือฟังก์ชันเกาส์เซียน (Gaussian Function) หรือเรียกอีกอย่างว่า Radial BasisFunction (RBF) ตามภาพที่ 3.2 ซึ่งมีสมการดังต่อไปนี้

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) \tag{3.18}$$

โดยที่ σ คือไฮเปอร์พารามิเตอร์ (Hyperparameter) ที่กำหนดความเรียบเนียน (Smoothness) ของขอบเขตการตัดสินใจ (Decision Boundary) และ $\|x-z\|^2$ คือระยะห่างยูคลิเดียนยกกำลังสอง (Squared Euclidean Distance) ระหว่าง Feature Vector x และ z ซึ่งสามารถหาค่าได้โดยใช้สมการดังต่อไปนี้

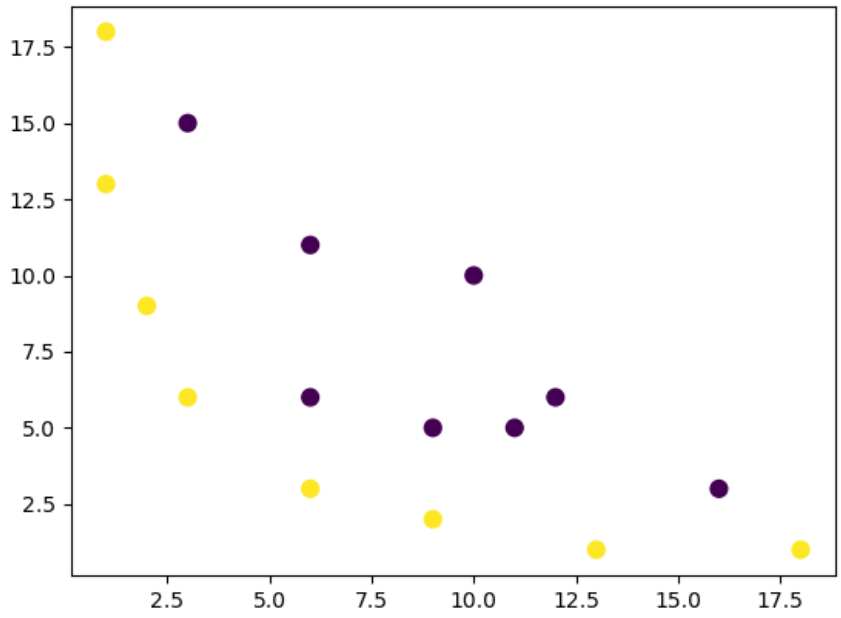
$$d(x_i, x_k) = \sqrt{(x_i^{(1)} - x_k^{(1)})^2 + (x_i^{(2)} - x_k^{(2)})^2 + \cdots + (x_i^{(N)} - x_k^{(N)})^2} \tag{3.19}$$

$$= \sqrt{\sum_{n=1}^{N} (x_i^{(n)} - x_k^{(n)})^2} \tag{3.20}$$

ฟังก์ชันในสมการ (3.18) นั้นเมื่อถูกนำมาใช้เป็นเคอร์เนลแล้วเราจะเรียกเคอร์เนลนี้ว่าเคอร์เนลเกาส์เซียน (Gaussian Kernel) ซึ่งเป็นฟังก์ชันที่เหมาะสมมาก ๆ เพราะว่ามีความสมมาตร มีความต่อเนื่องตลอดช่วงของปริภูมิ และมีค่าเข้าใกล้ 1 เมื่อ x และ z นั้นอยู่ใกล้กัน แต่จะมีค่าเข้าใกล้ 0 เมื่อ x และ z อยู่ห่างกัน

เรามาดูการเขียนโค้ดสำหรับการ Mapping ด้วยวิธีเคอร์เนลกันครับ เริ่มต้นด้วยโค้ดที่กำหนดชุดข้อมูลตัวอย่างซึ่งเป็นแบบไม่เป็นเชิงเส้น ดังนี้

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # Kernel
5  x = np.array([1,1,2,3,3,6,6,6,9,9,10,11,12,13,16,18])
6  y = np.array([18,13,9,6,15,11,6,3,5,2,10,5,6,1,3,1])
7  label = np.array([1,1,1,1,0,0,0,1,0,1,0,0,0,1,0,1])
8
9  # Plot
10 fig = plt.figure()
11 plt.scatter(x, y, c=label, s=60)
12 plt.show()
```



ภาพ 3.3 ข้อมูลที่จะใช้ในการ Mapping

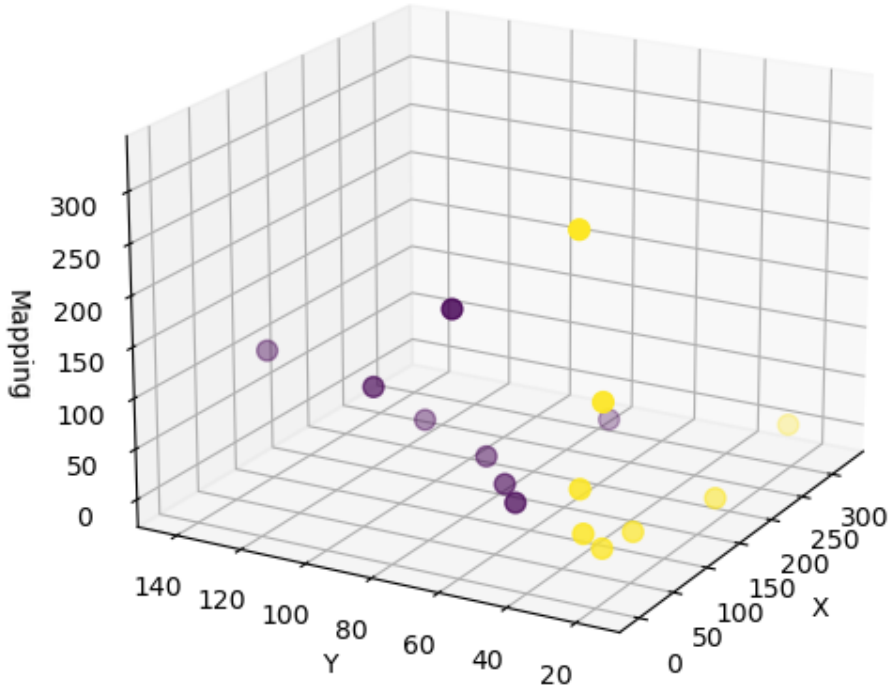
เขียนฟังก์ชันสำหรับการ Mapping ดังนี้

```
def mapping(x, y):
    x = np.c_[(x, y)]
    if len(x) > 2:
        x_1 = x[:,0]**2
        x_2 = np.sqrt(2)*x[:,0]*x[:,1]
        x_3 = x[:,1]**2
    else:
        x_1 = x[0]**2
        x_2 = np.sqrt(2)*x[0]*x[1]
        x_3 = x[1]**2
    trans_x = np.array([x_1, x_2, x_3])
    return trans_x
```

แล้วทำการ Mapping, แสดงผลลัพธ์ และพล็อตข้อมูลหลังจาก Mapping

```
# Mapping
x_1  = mapping(x, y)
x_1.shape
```

```
# Output
(3, 15)

# Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_1[0], x_1[1], x_1[2], c=label, s=60)
ax.view_init(30, 185)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Mapping')
plt.show()
```



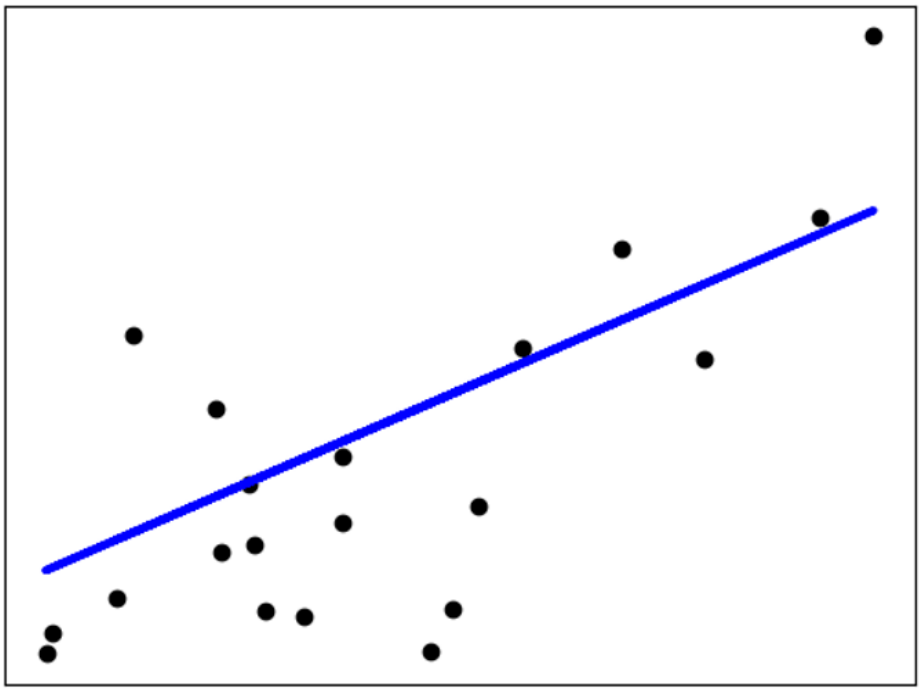
ภาพ 3.4 ข้อมูลที่ผ่านการ Mapping

โดยสรุปก็คือเคอร์เนลเป็นเทคนิคอย่างหนึ่งที่ช่วยให้เราสามารถทรานฟอร์มหรือแปลงข้อมูล (Transform) จากปริภูมิมิติต่ำ (Low-dimensional Space) ไปยังปริภูมิมิติสูง (High-dimensional Space) ซึ่งฟังก์ชันที่เหมาะสมที่สุดที่เราสามารถเลือกมาใช้เป็นฟังก์ชันเคอร์เนลได้นั้นไม่มีใครรู้ว่ามีหน้าตาเป็นอย่างไร ดังนั้นการที่เราจะทำ Mapping โดยเลือกใช้ทุกฟังก์ชันนั้นจึงแทบจะเป็นไปไม่ได้เลยเพราะว่ามีขีดจำกัดด้านการคำนวณ

3.3.1 การถดถอยแบบเชิงเส้น

กรณีแบบแรกของการถดถอย (Regression) ก็คือการถดถอยแบบเชิงเส้น (Linear Regression) ซึ่งเราได้ศึกษากันไปแล้วในหัวข้อที่ 2.1 ของบทที่ 2 ซึ่งเราใช้สมการดังต่อไปนี้ในการแก้ปัญหาการปรับค่าลงให้ต่ำที่สุด (Minimization)

$$\min_{w} \|X_w - y\|_2^2 \tag{3.21}$$



ภาพ 3.5 เส้นตรงที่ถูก Fit (สมดุล) ให้ผ่านจุดในชุดข้อมูล แกน x คืออินพุตและแกน y คือเอาต์พุต (เครดิตภาพ: https://scikit-learn.org)

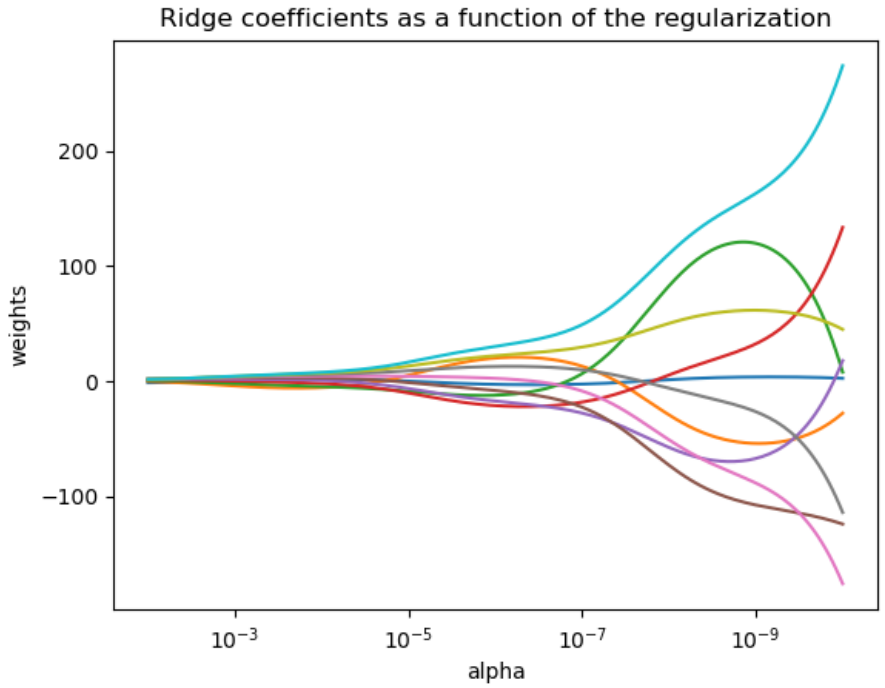
รูปที่ 3.5 แสดงเส้นตรงที่เกิดจากการทำ Minimization ของสมการถดถอยแบบเชิงเส้น เส้นตรงสีน้ำเงินเส้นนี้ถูกปรับระยะห่างเฉลี่ยระหว่างจุดทุกจุดในชุดข้อมูลโดยที่มีความสมดุลมากที่สุด โดยสังเกตด้วยตาเปล่าได้คร่าว ๆ ว่าแนวโน้มของจุดนั้นจะมีแนวโน้มที่เป็นแบบเอียงและชันขึ้นจากทางด้านซ้ายไปยังด้านขวา ดังนั้นเส้นตรงที่ได้จากการ Fit นั้นจึงมีแนวโน้มไปในทางเดียวกันซึ่งมีความชันเป็นบวกนั่นเอง

3.3.2 การถดถอยแบบริดจ์

สำหรับวิธีการถดถอยแบบริดจ์ (Ridge Regression) นั้นอาจจะมองได้ว่าเป็นการยกระดับ (Upgrade) หรือปรับปรุง Linear Regression ในกรณีที่เป็นแบบสามัญ (Ordinary) ให้มีประสิทธิภาพมากขึ้น นั่นก็เพราะว่ากรณีที่เราจะต้องทำการ Fit ข้อมูลที่มีจำนวนหลายตัวแปรและมีการกระจายในแบบที่ไม่สามารถอธิบายได้ด้วยสมการเส้นตรงนั้น เรามีเทคนิคก็คือการใส่พจน์พิเศษเข้าไป ซึ่งวิธีนี้เรียกว่าเป็นการลงโทษ (Penalize) ซึ่งเป็นหนึ่งในรูปแบบของการทำ Regularization โดยมีรูปสมการทั่วไปดังต่อไปนี้

$$\min_{w} \|X_w - y\|_2^2 + \alpha \|w\|_2^2 \tag{3.22}$$

โดยพระเอกของเราใน Ridge Regression นั้นก็คือพจน์สุดท้ายซึ่งมีศัพท์ทางเทคนิคที่เรียกว่า $L2$ Regularization โดยมีพารามิเตอร์ที่สำคัญนั่นก็คือ α ซึ่งเป็นตัวปรับจำนวนการหด (Schrinkage) ของฟังก์ชัน ซึ่งคูณอยู่กับค่าขนาดของน้ำหนัก (Weight) ยกกำลังสอง ซึ่งการยกกำลังนี้เป็นที่มาของการเรียกว่า $L2$ นั่นเอง สำหรับวิธีการปรับค่า α และผลกระทบที่เกิดขึ้นนั้นสามารถดูได้ตามรูปที่ 3.6



ภาพ 3.6 สัมประสิทธิ์ของริดจ์ที่เป็นฟังก์ชันกับ Regularization (เครดิตภาพ: https://scikit-learn.org)

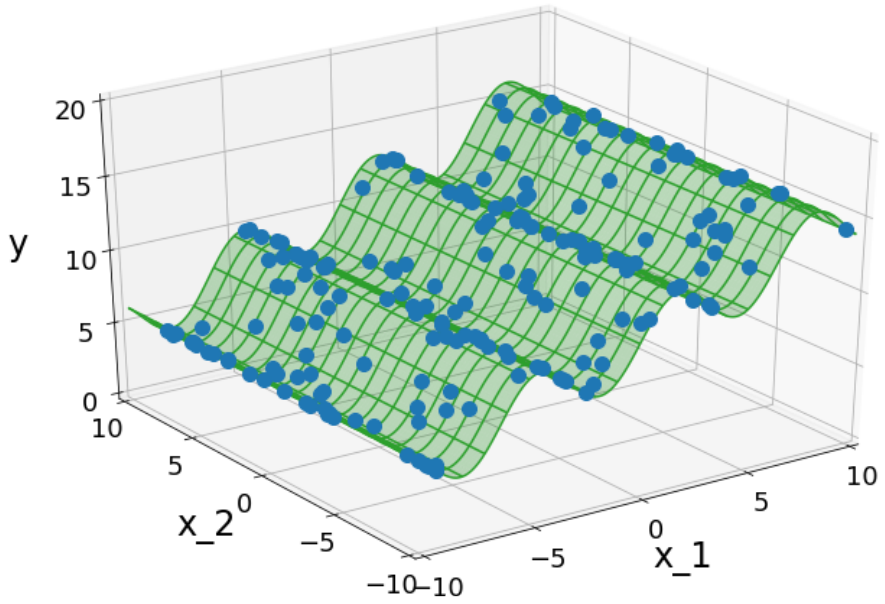
โดยเราจะพบว่ายิ่ง α มีค่ามากเท่าไหร่ จำนวนของการขดของฟังก์ชันหรือเส้นโค้งก็จะมีจำนวนมากตามไปด้วย (เช่น เส้นสีส้ม) ดังนั้นค่าสัมประสิทธิ์ที่มีความซับซ้อนขึ้นนั้นก็จะยิ่งส่งผลดีต่อการนำไปอธิบายชุดข้อมูลที่มีความไม่เป็นเส้นตรงสูง

อธิบายเสริม: กรณีที่การทำ Regularization ไม่ได้ใช้การยกกำลังสองของค่าขนาดของน้ำหนักแต่ใช้เพียงแค่ยกกำลังหนึ่งนั้น เราจะเรียกเทคนิคนี้ว่า LASSO ซึ่งย่อมาจาก Least Absolute Shrinkage and Selection Operator หรือเรียกสั้น ๆ ว่า $L1$ โดยมีสมการดังนี้

$$\min_{w} \|X_w - y\|_2^2 + \alpha \|w\|_1 \tag{3.23}$$

3.4 การถดถอยแบบริดจ์ด้วยเคอร์เนล

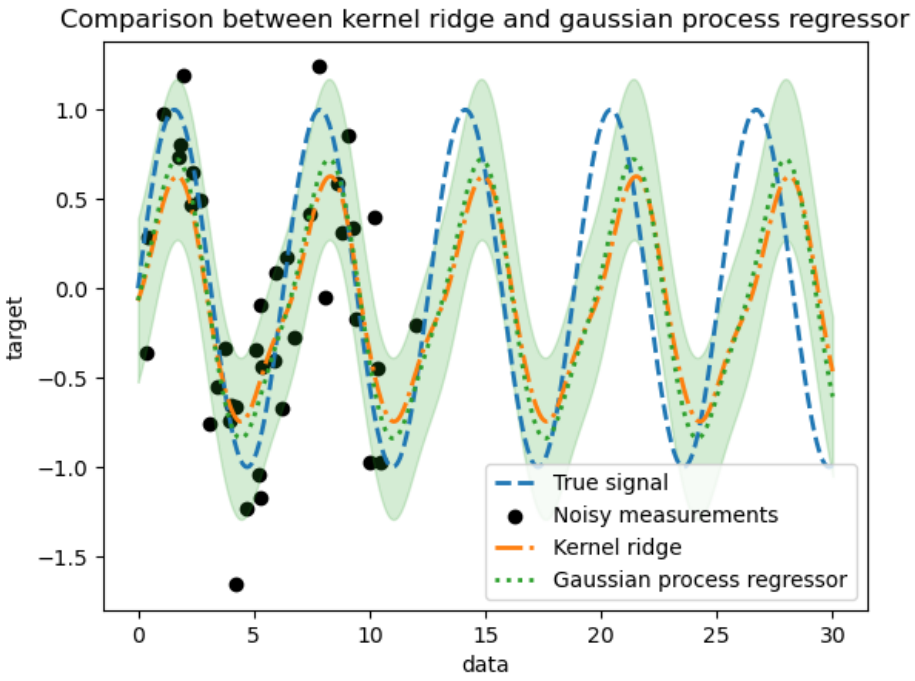
การถดถอยแบบริดจ์ด้วยเคอร์เนล (Kernel Ridge Regression หรือ KRR) เป็นการต่อยอดจาก Ridge Regression หรืออธิบายง่าย ๆ ว่า KRR ก็คือ RR ในเวอร์ชันที่เป็น Nonlinear Problem ซึ่งมีการผสม Kernel Trick เข้าไปด้วย (Kernel + Ridge Regression) โดยรูปแบบของโมเดลที่ถูกสอนให้เรียนรู้โดย KRR นั้นก็ยังมีรูปแบบอื่น ๆ แยกย่อยไปอีกหลายเทคนิค เช่น เทคนิค Support Vector Regression (SVR) โดยความแตกต่างระหว่าง KRR กับ SVM ก็คือการใช้ Loss Function ที่ต่างกัน โดย KRR จะใช้ค่า Loss Error ยกกำลังสอง ในขณะที่ SVR จะใช้ ϵ-incentive Loss



ภาพ 3.7 ตัวอย่างของปริภูมิ Kernel Ridge Regression

นอกจากนี้ยังมีเทคนิคอื่น ๆ อีกที่อาศัยหลักการของ Kernel Trick โดยหนึ่งในนั้นก็คือ Gaussian Process Regression ซึ่งถูกนำมาใช้ในการพัฒนาเทคนิค Gaussian Approximation Potential (GAP) ซึ่งเป็นเทคนิคที่มีการใช้อย่างแพร่หลายโดยเฉพาะการศึกษาการทำนายพลังงานรวมของโมเลกุล[28,29]

3.5 การถดถอยแบบกระบวนการเกาส์เซียน



ภาพ 3.8 เปรียบเทียบการเรียนรู้ Target ระหว่างเทคนิค KRR และ GPR (เครดิตภาพ: https://scikit-learn.org)

การถดถอยแบบกระบวนการเกาส์เซียน (Gaussian Process Regression หรือ GPR) เป็นเทคนิคที่มีความคล้ายกับ KRR นั่นก็คือทำการเรียนรู้ ฟังก์ชันคำตอบ (Target Function) ของโดยการใช้ Kernel Trick เหมือนกัน แต่ว่าจะมีความแตกต่างกันก็คือในกรณีของ KRR นั้นจะทำการเรียนรู้ฟังก์ชันเชิงเส้นในในปริภูมิที่ถูกสร้างขึ้นมาใหม่ด้วยเคอร์เนลที่เรากำหนดเข้าไปซึ่งจะสอดคล้องหรือเชื่อมโยงกับฟังก์ชันแบบไม่เป็นเชิงเส้นในปริภูมิเดิม ซึ่งฟังก์ชันเชิงเส้นในปริภูมิของเคอร์เนลนั้นก็จะขึ้นอยู่กับ Loss Function (ในกรณีทั่วไปคือ Mean Square Error) กับ Ridge Regularization ในกรณีของ GPR นั้นจะเป็นการใช้เคอร์เนลในการกำหนดความแปรปรวนร่วม (Covariance) ของการแจกแจงก่อน (Prior Distribution) ซึ่ง Covariance นี้เป็นพารามิเตอร์ที่สามารถบ่งบอกถึงแนวโน้มของข้อมูลว่ามีการเปลี่ยนแปลงไปในทิศทางเดียวกันมากน้อยแค่ไหน กล่าวง่าย ๆ ก็คือ GPR จะเป็นการพยายามที่จะมาเล่นกับ Covariance มากกว่าจะเป็นการทำนายฟังก์ชันคำตอบและใช้ชุดข้อมูลที่ใช้ในการฝึกสอนโมเดลมาทำการกำหนดฟังก์ชันที่ควรจะเป็น (Likelihood Function) นอกจากนี้แล้ว GPR ยังใช้หลักการของ Bayes Theorem ซึ่งจะมีการกำหนดการแจกแจงภายหลัง (Posterior Distribution) โดยใช้ Gaussian Function เพื่อนำค่าเฉลี่ยมาใช้ในการทำนายคำตอบอีกด้วย

ภาพที่ 3.8 แสดงการเปรียบเทียบความแม่นยำในการทำนายค่า Target ระหว่าง KRR (เส้นสีส้ม) และ

GPR (เส้นประสีเขียว) และมีค่าอ้างอิง (เส้นประสีฟ้า) เป็นตัวชี้วัดความแม่นยำ โดยสรุปได้ว่า GPR มีความแม่นยำและความถูกต้องในการทำนายเทียบเท่าพอ ๆ กับ KRR ตลอดช่วงของจำนวนข้อมูล เมื่อเราทำการเพิ่มขนาดของชุดข้อมูลจะพบว่าโมเดลทั้งสองอันจะให้ค่าที่ยังสอดคล้องกันแต่จะ Deviate ออกห่างจากค่าอ้างอิงมากขึ้นเรื่อย ๆ

รายละเอียดของ Gaussian Process นั้นมีเยอะมาก ถ้าหากผู้อ่านสนใจศึกษาเพิ่มเติมเกี่ยวกับทฤษฎีเชิงลึกและคณิตศาสตร์ที่ใช้ในการอธิบายวิธีนี้ผู้เขียนแนะนำหนังสือ *Gaussian Processes for Machine Learning* ของ Carl Edward Rasmussen และ Christopher K. I. Williams ซึ่งสามารถอ่านและดาวน์โหลดได้ฟรีที่ http://gaussianprocess.org/gpml

บทที่ 4

การเรียนรู้เชิงลึก

ในบทนี้ผู้อ่านจะได้ศึกษาการเรียนรู้เชิงลึกหรือ Deep Learning ซึ่งเป็นหนึ่งใน Buzzword[1] ที่หลาย ๆ คนต่างก็พูดถึงในช่วงระยะเวลา 10 ปีที่ผ่านมาโดยเฉพาะในช่วงเวลาที่ผู้เขียนกำลังเขียนหนังสือเล่มนี้ ไม่ว่าจะเป็นวงการหรืออาชีพไหน ๆ ต่างก็มีการนำ Deep Learning เข้าไปประยุกต์ใช้ทั้งนั้นและแน่นอนว่าหนึ่งในนั้นก็คือการนำมาประยุกต์ใช้ในงานวิจัยทางด้านเคมี เนื่องจากว่าข้อมูลที่นักเคมีมีอยู่ในมือนั้นมากมายมหาศาล ไม่ว่าจะเป็นข้อมูลเกี่ยวกับสภาวะของปฏิกิริยาเคมี (อุณหภูมิและความดันที่ใช้ เป็นต้น), ตัวเร่งปฏิกิริยาที่มีประสิทธิภาพสูงสำหรับปฏิกิริยาแต่ละประเภท, และหมู่ฟังก์ชันนอลที่นำมาใช้ในการสังเคราะห์โมเลกุลยา ดังนั้นจึงน่าสนใจว่า Deep Learning จะช่วยนักเคมีในการขับเคลื่อนงานวิจัยได้อย่างไรและมากน้อยแค่ไหน

ถ้าจะให้นิยาม Deep Learning แบบเข้าใจง่าย ๆ โดยใช้การยกตัวอย่างก็คือเป็นเทคนิคในการสร้างโมเดลคอมพิวเตอร์ที่ได้จากการฝึกฝนให้สามารถทำงานได้เหมือนมนุษย์แบบที่มีสภาวะในการทำงานที่ซับซ้อนมาก ๆ (ที่มาของคำว่าลึกหรือ Deep) เช่น การสังเกต การจดจำคำพูด การระบุภาพ หรือการคาดการณ์ แทนที่จะจัดระเบียบข้อมูลที่จะคำนวณผ่านทางสมการที่กำหนดไว้ล่วงหน้า Deep Learning จะกำหนดค่าพารามิเตอร์พื้นฐานเริ่มต้นเกี่ยวกับข้อมูล (Hyperparameters) และฝึกให้คอมพิวเตอร์เรียนรู้ด้วยตัวเองโดยการจดจำรูปแบบโดยการใช้การประมวลผลหลายชั้น อีกหนึ่งวัตถุประสงค์ของการพัฒนา Deep Learning นั้นก็คือการสร้าง Neural Network ที่สามารถสกัดหรือดึงสิ่งที่ซ่อนอยู่ภายในข้อมูล (Hidden Features) ออกมาได้ด้วยตัวของโมเดลเองโดยปราศจากการช่วยเหลือจากผู้ใช้งานหรือมนุษย์

จริง ๆ แล้ว Deep Learning นั้นถือว่าเป็นอัลกอริทึมประเภทหนึ่งของ Neural Network ซึ่งเป็นเทคนิคปัญญาประดิษฐ์แบบหนึ่งที่ถูกพัฒนาขึ้นมาเพื่อศึกษารูปแบบที่แน่นอนของข้อมูลเรียกว่า *การรับรู้รูปแบบ (Pattern Recognition)* ซึ่งเป็นกระบวนที่ทำงานเลียนแบบ (Mimic) สมองของมนุษย์ในการแยกแยะความจำเพาะเจาะจงบางอย่างออกมาจากข้อมูลที่ป้อนเข้าไป ภาพที่ 4.1 แสดงเครื่อง WISARD ซึ่งเป็นคอมพิวเตอร์เครื่องแรกของโลกที่ถูกสร้างขึ้นมาในปี ค.ศ. 1981 สำหรับการคำนวณ Deep Learning

[1] สิ่งที่กำลังเป็นที่พูดถึงหรือได้รับความนิยม โดยที่ฟังแล้วก็อาจจะเป็นคำเก๋ ๆ

ภาพ 4.1 เครื่องการเรียนรู้เชิงลึก WISARD เครื่องแรกของโลก ณ พิพิธภัณฑ์วิทยาศาสตร์ กรุงลอนดอน ประเทศอังกฤษ (เครดิตภาพ: รังสิมันต์ เกษแก้ว)

ผู้เขียนขอสรุปสั้น ๆ เพื่อไม่ให้สับสนดังนี้ *"Deep Learning ทุกรูปแบบเป็น Machine Learning แต่ว่าไม่ใช่ Machine Learning ทุกเทคนิคที่จะเป็น Deep Learning"*

ในบทก่อนหน้านี้นั้นเราได้พูดถึง Supervised Learning กันไปแล้ว ซึ่งจะเป็นการทำนายค่า y จากข้อมูลนำเข้า x โดยเป็นการพิจารณาในกรณีของการถดถอยเชิงเส้น ในลำดับถัดมา (ซึ่งก็คือในบทนี้) เราจะมาพิจารณากรณีที่พารามิเตอร์ของเรานั้นมีความไม่เป็นเชิงเส้น (Nonlinear) กันครับ ซึ่งโมเดลปัญญาประดิษฐ์ที่ถูกนำมาใช้อย่างแพร่หลายนั้นก็คือ Neural Network นั่นเอง โดยเฉพาะ Deep Learning ที่ ณ ปัจจุบันได้มีการพัฒนาและปรับปรุงจนมีประสิทธิภาพที่สูงมาก ข้อดีอย่างหนึ่งของ Neural Network ก็คือสามารถจัดการกับข้อมูลที่มีความซับซ้อนที่มีจำนวน Feature หลายร้อยหรือหลายพัน Feature ได้

4.1 การเรียนรู้ของโมเดลที่ไม่เป็นเชิงเส้น

สมมติว่าเรามี $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ ซึ่งเป็นตัวอย่างชุดข้อมูลฝึกสอน (Training Data) และเราจะเริ่มกันด้วยกรณีที่ง่ายที่สุดนั่นก็คือ $y^{(i)} \in \mathbb{R}$ และ $h_\theta(x) \in \mathbb{R}$

เราจะทำการกำหนด Loss Function หรือ Coss Function ขึ้นมาก่อน ซึ่งเราจะใช้ Least Square Cost Function สำหรับข้อมูลลำดับที่ i นั่นก็คือคู่อันดับ $(x^{(i)}, y^{(i)})$ ดังนี้

$$J^{(i)}(\theta) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \tag{4.1}$$

และกำหนด Mean-Square Cost Function สำหรับชุดข้อมูลดังนี้

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J^{(i)}(\theta) \tag{4.2}$$

ซึ่งผู้อ่านอาจจะสังเกตได้ว่าสมการข้างต้นนั้นจะเหมือนกับในกรณีของ Linear Regression เว้นแต่จะต่างกันตรงที่เราได้มีการเพิ่ม $1/n$ เข้าไปในด้านหน้าของ Cost Function ซึ่งเป็นการคูณ Cost Function ด้วยปริมาณสเกลาร์นั่นเอง โดยการคูณแบบนี้จะไม่ทำให้จุดต่ำสุดสัมพัทธ์ (Local minima) และจุดต่ำสุดสัมบูรณ์ (Global Minima) เปลี่ยนไป นอกจากนี้ผู้อ่านจะต้องทำความเข้าใจด้วยว่าการปรับค่าพารามิเตอร์ (Parameterization) ของ $h_\theta(x)$ นั้นจะแตกต่างจากกรณีของ Linear Regression ถึงแม้ว่าเราจะใช้ Cost Function ที่เหมือนกันก็ตาม

4.2 การเคลื่อนลงตามความชัน

หัวข้อถัดมาคือการเคลื่อนลงตามความชัน (Gradient Descent) ซึ่งเป็นเทคนิคที่สำคัญมากเพราะเป็นเทคนิคที่เรานำมาใช้ในการปรับค่าพารามิเตอร์ (Optimization) เพื่อลดความคลาดเคลื่อนระหว่างค่าที่ทำนายออกมากับค่าอ้างอิง ซึ่งเป็นกระบวนการสำคัญที่ทำให้เกิดการเรียนรู้ของโมเดลนั่นเอง ซึ่ง Gradient Descent นี้เป็นเทคนิคสุดคลาสสิกที่เรานำมาใช้เพื่อหาค่าที่เหมาะสมที่สุดให้กับฟังก์ชันหนึ่ง ๆ ซึ่งใน ML นี้เราจะใช้กับ Loss หรือ Coss Function นั่นเอง (ผู้อ่านสามารถศึกษารายละเอียดของ Loss Function เพิ่มเติมได้ในหัวข้อที่ 4.6)

ภาพ 4.2 ยอดเขามัทเทอร์ฮอร์น (Matterhorn) ตั้งอยู่บนแนวของเทือกเขาแอลป์ ประเทศสวิตเซอร์แลนด์ (เครดิตภาพ: https://www.myswitzerland.com)

Gradient เป็นกระบวนการทำซ้ำ (Iterative Process) โดยทำการขยับหรือกระโดดจากจุดหนึ่ง (x^k) ไปยังอีกจุดหนึ่ง (x^{k+1}) ในทิศทางที่ทำให้ค่าของฟังก์ชันมีค่าเพิ่มขึ้นหรือลดลงในปัญหาของการหาค่าสูงสุดหรือค่าต่ำสุดตามลำดับ จนกระทั่งได้ค่าที่เหมาะสมออกมา โดยสิ่งที่ Gradient Descent ทำในการหาพารามิเตอร์ที่เหมาะสมของฟังก์ชันก็คือการวนหาค่าที่ทำให้ Loss Function หรือค่าความคลาดเคลื่อนมีค่าน้อยที่สุด โดยเราจะแทนความคลาดเคลื่อนหรือ Error ที่เกิดขึ้นด้วย J (เช่นสมการที่ (4.1)) หรืออธิบายง่าย ๆ ก็คือหาจุดที่มี J ต่ำที่สุดจากการคำนวณความชัน (Slope) ณ จุดที่เราอยู่แล้วพยายามหาเส้นทางในการขยับจุดไปในทิศทางตรงข้ามกับ Slope โดยให้ลองนึกภาพว่าเรากำลังเดินขึ้นเขา เช่น ตามภาพที่ 4.2 โดยวิธีการเดินเพื่อไปให้ถึงจุดสูงสุดคือเราไต่ขึ้นตามทางที่ชันขึ้นเพื่อไปถึงจุดสูงสุดนั่นเอง ถ้ายิ่งชันเท่าไหร่ก็มีแนวโน้มว่าเราเข้าใกล้จุดสูงสุดมากขึ้นเท่านั้น

Gradient Descent เป็นอัลกอริทึมที่ใช้หาจุดต่ำสุดหรือสูงสุดของฟังก์ชันซึ่งโดยส่วนมากเป็นฟังก์ชันรูปกรวยคว่ำ (Convex) แต่ถ้าลองดูตัวอย่างง่าย ๆ เช่น ฟังก์ชันพาราโบลาหงายที่มีฟังก์ชันเป็น

$$f(x) = x^2 - 4x \tag{4.3}$$

เราจะสามารถหาอนุพันธ์ของสมการที่ (4.3) ได้ง่าย ๆ ซึ่งจะได้ออกมาเป็นสมการที่อธิบายความชัน ดังนี้

$$\begin{aligned} f'(x) &= \nabla f(x) \\ &= 2x - 4 \end{aligned} \tag{4.4}$$

4.2.1 Batch Gradient Descent

คราวนี้เราลองมาดูกรณีที่ Loss Function นั้นเป็นแบบกรณีทั่วไปกันบ้าง โดยเรานิยามให้ Loss Function มีหน้าตาแบบนี้

$$\begin{aligned} J(\theta) &= J(\theta_0, \theta_1) \\ &= \frac{1}{2m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2 \end{aligned} \tag{4.5}$$

เราสามารถหา Gradient ได้ดังนี้

$$\nabla J(\theta) = \begin{bmatrix} \dfrac{\partial J}{\partial \theta_0} \\ \dfrac{\partial J}{\partial \theta_1} \end{bmatrix} \tag{4.6}$$

$$= \begin{bmatrix} \frac{1}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}) \\ \frac{1}{m} \sum_{i=1}^{m} ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)}) x^{(i)} \end{bmatrix} \tag{4.7}$$

ซึ่งเราสามารถเขียนสมการในกรณีที่เราสนใจและอัพเดท θ ได้ดังนี้[1]

$$\theta := \theta - \alpha \nabla_\theta J(\theta) \tag{4.8}$$

[1]สังเกตว่าเราใช้เครื่องหมาย $a := b$ เพื่อบ่งบอกการดำเนินการ (Operation) ซึ่งเป็นการระบุค่าให้กับตัวแปรในโปรแกรมคอมพิวเตอร์

โดยสมการที่ (4.8) นั้นคือ Batch Gradient Descent มีสัมประสิทธิ์ด้านหน้า $\nabla_\theta J$ ก็คือ α คืออัตราเร็วในการเรียนรู้ *Learning Rate* หรืออาจจะเรียกว่า *Step Size* ก็ได้ โดยเรามักจะกำหนดให้ค่า α มีค่ามากกว่าศูนย์ ซึ่งเป็นอัลกอริทึมแบบที่ง่ายที่สุดเพราะว่าใช้ข้อมูลทั้งหมดใน Training Set ในการฝึกสอนโมเดล ดังนั้นผู้อ่านน่าจะพอเดาออกว่าถ้าหากเราใช้ Batch Gradient Descent กับชุดข้อมูลที่มีขนาดใหญ่มากนั้นก็จะใช้ระยะเวลานานในการฝึกสอนโมเดล นั่นก็เพราะว่า Optimization แบบ Batch นั้นช้ามาก ด้านล่างคืออัลกอริทึมของ Batch Gradient Descent

Algorithm 4.1 อัลกอริทึมของ Batch Gradient Descent

Hyperparameter: learning rate α, number of total iteration n_{iter}.
Initialize θ randomly.
for $i = 1$ to n_{iter} **do**

$$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$$

end for

ตัวอย่างของโค้ดของ Batch Gradient Descent มีดังนี้

```
for i in range(nb_epochs):
    params_grad = evaluate_gradient(
        loss_function,
        data,
        params
        )
    params = params - learning_rate * params_grad
```

4.2.2 Stochastic Gradient Descent

เพื่อแก้ปัญหาในกรณีที่ชุดข้อมูลมีขนาดใหญ่มากนั้น จึงได้มีการพัฒนาอัลกอริทึมแบบที่สองขึ้นมา เรียกว่า Stochastic Gradient Descent (SGD) โดย SGD นี้เป็นวิธีที่ง่ายและไม่ซับซ้อนในการนำมาใช้ปรับค่าพารามิเตอร์ให้มีความเหมาะสม โดยในแต่ละครั้งของการคำนวณ Gradient เราจะทำการสุ่มข้อมูลเพียงบางส่วนเพื่อใช้ในการอัพเดทค่าเท่านั้น ไม่ได้ใช้ข้อมูลทั้งหมดเหมือนในกรณี Batch โดยตามทฤษฎีแล้วได้มีการพิสูจน์ว่าเราสามารถใช้ข้อมูลในปริมาณที่เล็กน้อยเพื่อใช้ในการอัพเดทพารามิเตอร์ในแต่ละครั้ง (Iteration) ได้โดยไม่ต้องนำข้อมูลทั้งหมดมาใช้ทีเดียว ซึ่งในท้ายที่สุดแล้วการ Optimization ก็จะลู่เข้าสู่คำตอบที่ใกล้เคียงกัน ซึ่ง SGD นี้ถือว่ามีส่วนสำคัญต่อการฝึกสอนโมเดลที่มีพารามิเตอร์ที่เกี่ยวข้องในปริมาณที่เยอะมาก ๆ เช่น Neural Network การใช้ SGD สามารถลดปัญหาของการที่ Optimization นั้นติดหรือค้างอยู่ในจุดต่ำสุดสัมพันธ์ได้อีกด้วย ซึ่งการปรับค่าด้วย SGD นั้นจะใช้สมการดังต่อไปนี้

$$\theta := \theta - \alpha \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{4.9}$$

ด้านล่างคืออัลกอริทึมของ SGD

Algorithm 4.2 อัลกอริทึมของ Stochastic Gradient Descent

Hyperparameter: learning rate α, number of total iteration n_{iter}.
Initialize θ randomly.
for $i = 1$ to n_{iter} **do**
    Sample j uniformly from $1, \ldots, n$, and update θ by

$$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta)$$

end for

ตัวอย่างของโค้ดของ Stochastic Gradient Descent มีดังนี้

```
for i in range(nb_epochs):
    np.random.shuffle(data)
    for example in data:
        params_grad = evaluate_gradient(
            loss_function,
            example,
            params
            )
        params = params - learning_rate * params_grad
```

นอกจากนี้แล้วในการฝึกสอนโมเดล Neural Network นั้น เรามักนิยมใช้ Stochastic Gradient Descent โดยด้านล่างคือตัวอย่างโค้ดของการเรียกใช้ Stochastic Gradient Descent (SGD) Optimizer ของ TensorFlow

```
import tensorflow as tf

# Create an optimizer with the desired parameters
opt = tf.keras.optimizers.SGD(
    learning_rate=0.01,
    momentum=0.0,
    nesterov=False,
    name='SGD',
    **kwargs
)

# `loss` is a callable that takes no argument and
# returns the value to minimize
loss = lambda: 3 * var1 * var1 + 2 * var2 * var2

```

```
# In graph mode, returns op that minimizes the loss
# by updating the listed variables
opt_op = opt.minimize(loss, var_list=[var1, var2])
opt_op.run()

# In eager mode, simply call minimize to update
# the list of variables
opt.minimize(loss, var_list=[var1, var2])
```

4.2.3 Mini-batch Stochastic Gradient Descent

นอกเหนือจาก Batch และ Stochastic Gradient Descent แล้ว ยังมีอัลกอริทึมแบบที่สามที่เป็นการรวมข้อดีของทั้งสองอัลกอริทึมเข้าไว้ด้วยกัน นั่นคืออัลกอริทึมที่ชื่อว่า Mini-batch Stochastic Gradient Descent โดยแนวคิดก็คือในทางปฏิบัตินั้นการคำนวณ Gradient ของ Batch (B) หลาย ๆ ครั้งสามารถทำพร้อมกันได้เพราะว่าในปัจจุบันเรามีเทคนิคการทำการคำนวณแบบขนาน (Parallelization) สำหรับการปรับค่า θ ซึ่งจะเร็วกว่าการคำนวณ Gradient ของ B แบบแยกกันทีละค่าแน่นอน ซึ่งการที่เราจะทำการคำนวณแบบพร้อม ๆ กันได้นั้นเราจะต้องมีการแบ่งข้อมูลของเราออกเป็นส่วนย่อย ๆ แล้วทำการคำนวณแยกกัน โดยมีสมการดังต่อไปนี้

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{4.10}$$

ในการคำนวณจริงนั้นเราจะต้องมีการปรับอัลกอริทึมเล็กน้อย โดยด้านล่างคืออัลกอริทึมของ Mini-batch SGD ซึ่งได้จากการปรับอัลกอริทึมของ SGD แบบธรรมดา

Algorithm 4.3 อัลกอริทึมของ Mini-batch Stochastic Gradient Descent

Hyperparameter: learning rate α, batch size B, # iteration n_{iter}.
Initialize θ randomly.
for $i = 1$ to n_{iter} **do**
  Sample j uniformly from $1, \ldots, n$, and update θ by
  Sample B examples $j_1, \ldots, j_B$ (without replacement) uniformly from $\{1, \ldots, n\}$, and update θ by

$$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$$

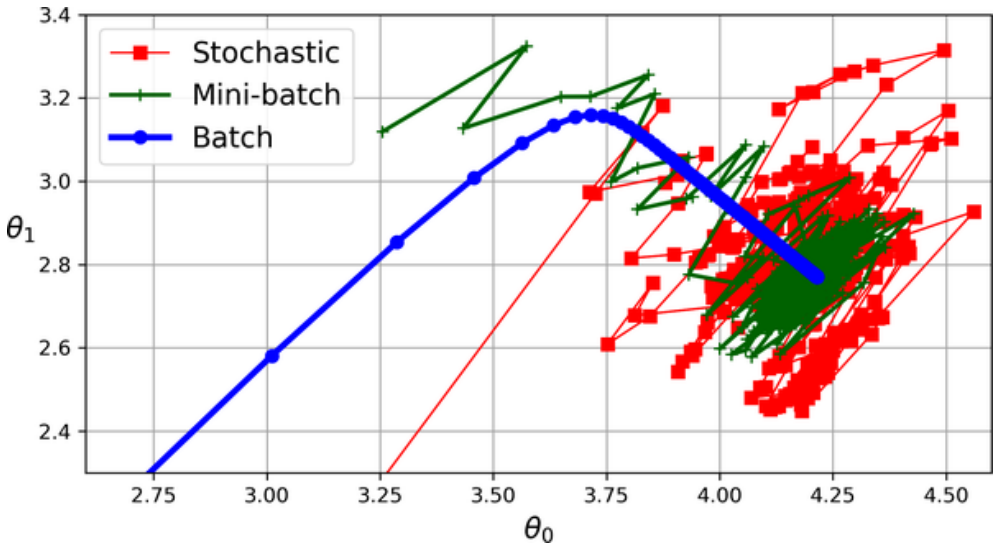
end for

ตัวอย่างของโค้ดของ Mini-batch Stochastic Gradient Descent มีดังนี้

```
for i in range(nb_epochs):
    np.random.shuffle(data)
```

```
    for batch in get_batches(data, batch_size=50):
        params_grad = evaluate_gradient(
            loss_function,
            batch,
            params
            )
        params = params - learning_rate * params_grad
```

- **สรุปเปรียบเทียบอัลกอริทึมของ Gradient Descent**



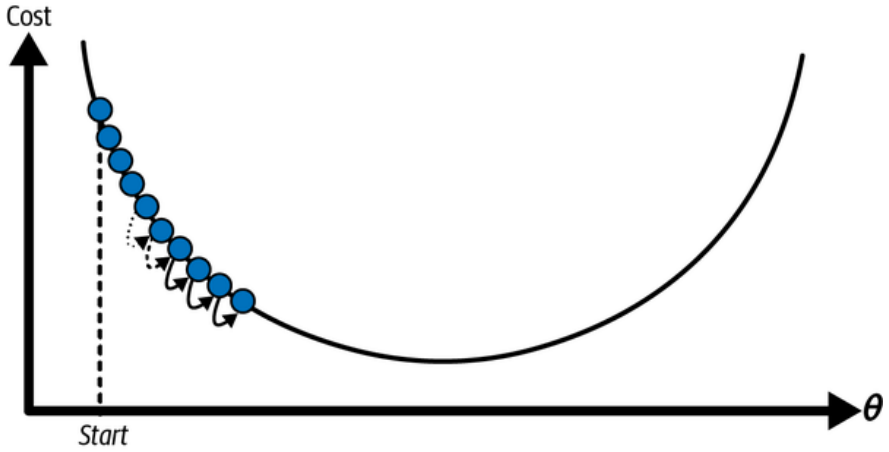
ภาพ 4.3 วิถีของ Gradient Descent ในปริภูมิของพารามิเตอร์ (เครดิตภาพ: https://www.oreilly.com)

ภาพที่ 4.3 แสดงการเปรียบเทียบวิถี (Path) ของการขยับของ Gradient Descent ด้วยอัลกอริทึมที่ต่างกัน ซึ่งแสดงอยู่บนปริภูมิพารามิเตอร์ โดยสรุปได้ว่าทั้งสามอัลกอริทึมนั้นให้ผลลัพธ์ที่เข้าใกล้จุดต่ำสุดแต่ว่าจริง ๆ แล้ว Batch Gradient Descent นั้นหยุดอยู่ที่จุดต่ำสุดพอดีในขณะที่ Stochastic และ Mini-batch Gradient Descent นั้นจะขยับวนไปมาอยู่รอบ ๆ จุดต่ำสุดและยังคงขยับไปมาอยู่เรื่อย ๆ อย่างไรก็ตามเราจะต้องไม่ลืมว่าอัลกอริทึมแบบ Batch นั้นใช้ระยะเวลาในการขยับจากจุดหนึ่งไปยังอีกจุดหนึ่งที่นานมาก แต่อีกสองอัลกอริทึมนั้นใช้เวลาน้อยกว่ามาก ซึ่งถ้าหากเราเลือกใช้อัลกอริทึม Stochastic หรือ Mini-batch ได้อย่างเหมาะสมแล้วทั้งสองอัลกอริทึมนี้ก็สามารถให้ผลลัพธ์ที่ลู่เข้า (Convergence) สู่จุดต่ำสุดได้เช่นเดียวกัน

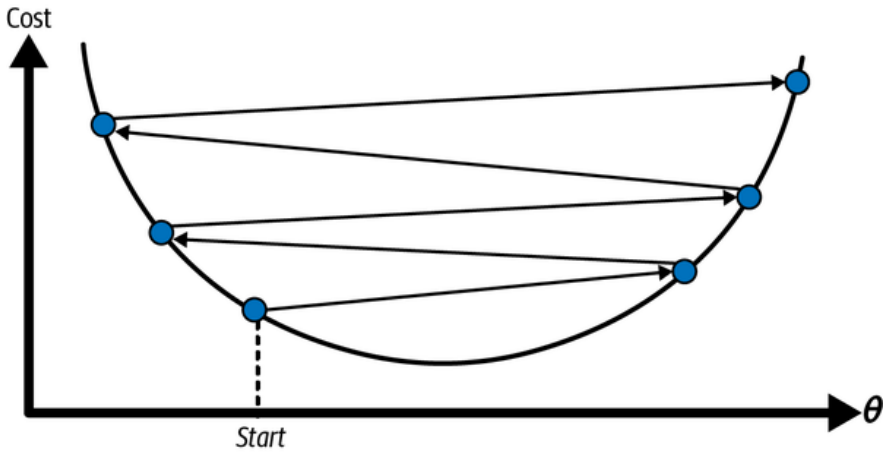
โดยทั่วไปแล้วโมเดล Deep Learning นั้นจะมีการเรียนรู้ซึ่งอาศัยอัลกอริทึมตามด้านบนโดยทำตามขั้นตอนดังต่อไปนี้

1. กำหนด $h_\theta(x)$
2. เขียนอัลกอริทึม Backpropagation เพื่อคำนวณ Gradient ของ Loss Function $J^{(j)}(\theta)$
3. ทำการปรับ Loss Function โดยใช้ SGD หรือ Mini-batch SGD หรือใช้อัลกอริทึมอื่น เช่น Normal Equation หรือ SVD

นอกจากนี้แล้วอีกสิ่งหนึ่งที่เราต้องคำนึงถึงก็คือ Learning Rate ซึ่งเป็นค่าอัตราเร็วของการเรียนรู้ โดยพารามิเตอร์ตัวนี้มีผลทั้งในเชิงประสิทธิภาพของอัลกอริทึมที่ใช้ในการปรับให้การ Optimization นั้นลู่เข้าและมีผลต่อความเร็วหรือความสิ้นเปลืองในการปรับค่าด้วย



ภาพ 4.4 แสดงค่าคลาดเคลื่อน (Error หรือ Cost) เทียบกับการเปลี่ยนแปลงของการเรียนรู้ในกรณีที่กำหนดให้ Learning Rate นั้นมีที่น้อยมาก ๆ (เครดิตภาพ: https://www.oreilly.com)



ภาพ 4.5 แสดงค่าคลาดเคลื่อน (Error หรือ Cost) เทียบกับการเปลี่ยนแปลงของการเรียนรู้ในกรณีที่กำหนดให้ Learning Rate นั้นมีที่สูงมาก ๆ (เครดิตภาพ: https://www.oreilly.com)

ภาพที่ 4.4 และ 4.5 แสดงความสามารถในการขยับหรือปรับค่าของการทำ Gradient Descent จากจุดหนึ่งไปยังอีกจุดหนึ่งโดยใช้อัตราเร็วในการเรียนรู้ที่มีค่าน้อย ๆ และค่าสูง ๆ ตามลำดับ โดยเราสามารถสรุปได้ว่าในกรณีที่เราใช้อัตราการเรียนรู้ที่มีค่าน้อยเกินไปนั้นจะเป็นการขยับจุดแบบช้ามาก ๆ และการขยับไปในแต่ละจุดนั้นจะเป็นแบบก้าวสั้น ๆ แต่ก็ให้ผลลัพธ์ที่มีความแม่นยำและค่อย ๆ ขยับจุดเข้าไปใกล้จุดต่ำสุดตามที่ต้องการ ซึ่งจะตรงข้ามกับกรณีที่ใช้อัตราการเรียนรู้ที่สูงเกินไปนั่นก็คือการขยับจุดนั้นเป็นไปอย่างรวดเร็วและระยะห่างระหว่างจุดหรือขนาดของก้าวแต่ละก้าวนั้นจะกว้างกว่ามาก แต่จะพบว่าการปรับค่าเข้าไปหาจุด

ต่ำสุดนั้นจะทำได้ไม่ค่อยดีเพราะจะเป็นการขยับจุดที่เร็วเกินไปทำให้เกิดการวนซ้ำไปซ้ำมารอบ ๆ จุดต่ำสุด และทำให้ลู่เข้าได้ยากเพราะว่าหาจุดต่ำสุดจริง ๆ ไม่เจอเสียที

สรุปคือเราควรจะต้องกำหนดอัตราเร็วของการเรียนรู้ให้มีความเหมาะสม ถ้าหากอัตราการเรียนรู้ช้าเกินไปก็จะใช้เวลานาน แต่ถ้าหากอัตราการเรียนรู้เร็วเกินไปก็จะทำให้การปรับค่านั้นได้ผลลัพธ์ที่ไม่แม่นยำ นอกจากนี้แล้วเราไม่มีกฎตายตัวในการหาค่าอัตราเร็วในการเรียนรู้ที่ดีที่สุด ดังนั้นการเลือกค่าอัตราเร็วการเรียนรู้หรือ Learning Rate นั้นจึงเป็น Art อย่างหนึ่งของการทำ Deep Learning ซึ่งเป็น Hyperparameter ที่สำคัญมากจนถึงกับมีการกล่าวว่า “ถ้าหากจะต้องเลือกปรับ Hyperparameter ได้เพียงแค่หนึ่งตัว สิ่งที่เราควรจะต้องเลือกนั้นก็คือ Learning Rate”[1]

4.3 โครงข่ายประสาทเทียม

โครงข่ายประสาทเทียม (Neural Network) ถือว่าเป็นโมเดล ML แบบหนึ่งที่มีประสิทธิภาพสูงมากและเป็นสิ่งที่พลิกโฉมหน้าประวัติศาสตร์ของวงการปัญญาประดิษฐ์เลยก็ว่าได้ ตามที่ได้อธิบายไว้คร่าว ๆ ในหัวข้อ 2.5.4 แล้วว่า Neural Network นั้นเป็นการเลียนแบบการทำงานของสมองมนุษย์ ซึ่งในช่วงยุคต้นของการพัฒนาเทคนิคนี้นั้นมีจุดประสงค์คือการแก้ปัญหาแบบเดียวกับที่สมองมนุษย์สามารถทำได้ แต่เมื่อเวลาผ่านไปจุดประสงค์ของการสร้าง Neural Network ก็ได้เปลี่ยนไปเป็นการทำงานที่เฉพาะเจาะจงมากขึ้น และมีการพัฒนาให้สามารถทำงานหรือแก้ปัญหาให้ดีกว่ามนุษย์เสียอีก ซึ่งเป็นการแทนจุดประสงค์เดิมในการสร้างสมองเทียม โดยในปัจจุบันมีการประยุกต์ใช้โ Neural Network กับงานหลากหลายรูปแบบ เช่น คอมพิวเตอร์วิทัศน์ (Computer Vision), การเรียนรู้เสียง (Voice Learning และ Recognition), การแปลภาษา (Translation), การกรองเนื้อหาโซเชียลมีเดีย, การเล่นเกม, การวินิจฉัยโรค และกิจกรรมที่ไม่คิดว่าปัญญาประดิษฐ์จะทำได้ เช่น การวาดภาพ, การประพันธ์เพลง และการประพันธ์บทกวี

Neural Network ที่พบได้ทั่วไปจะมีลักษณะคือประกอบไปด้วยชั้นของเซลล์ประสาทเทียม (Neural Layer) คือ ชั้นที่รับข้อมูลขาเข้าเรียกว่าชั้นอินพุต (Input Layer), ชั้นที่สร้างข้อมูลขาออกเรียกว่าชั้นเอาต์พุต (Output Layer) และชั้นอื่น ๆ ที่อยู่ตรงกลางระหว่างชั้นอินพุตและชั้นเอาต์พุตที่มีส่วนในการช่วยทำการประมวลผลอยู่ภายในเรียกว่าชั้นซ่อน (Hidden Layer) ซึ่งใน Neural Network อาจมี Hidden Layer ได้หลายชั้น นอกจากนี้เราสามารถแบ่งโครงสร้างพื้นฐานตามการไหล (Flow) ของข้อมูลได้ดังต่อไปนี้

- แบบป้อนไปข้างหน้า (Feed-forward) เป็น Neural Network ที่ข้อมูลจะมีการไหลหรือส่งต่อไปในทิศทางเดียวจาก Input Layer ไปยัง Hidden Layer และ Output Layer ตามลำดับ การเชื่อมโยงจะถูกกำหนดขึ้นระหว่างชั้นที่ติดกันโดยจะมีการเชื่อมโยงระหว่างเซลล์ประสาทเทียม (Neuron) ทุกตัว จากชั้นหนึ่ง ๆ ไปยังเซลล์ประสาทเทียมทุกตัวในชั้นต่อไป ในบางสถาปัตยกรรมอาจมีการเชื่อมโยงข้ามชั้นก็ได้ นอกจากนี้แล้วรูปแบบของ Neural Network ประเภทนี้ยังจัดแบ่งได้เป็นสองแบบย่อย คือ แบบมีชั้นของเซลล์ประสาทชั้นเดียว และแบบมีชั้นของเซลล์ประสาทหลายชั้น

[1]ค่า Default ของ Learning Rate ของ Adam Optimizer ที่ถูกกำหนดไว้ใน TensorFlow คือ 0.001

- แบบมีการป้อนไปเวียนกลับ (Recurrent) เป็น Neural Network ที่การเชื่อมโยงที่ถูกกำหนดขึ้นระหว่างเซลล์ประสาทเทียมในชั้นหนึ่ง ๆ นั้นย้อนกลับไปยังชั้นอื่น ๆ ก่อนหน้านั้นได้หรือแม้แต่ภายในชั้นเดียวกันเองโดยผ่านการวนลูปรอบ ๆ เซลล์ประสาทนั้น ๆ โดยการไหลของข้อมูลนั้นสามารถเกิดได้สองทิศทาง ทั้งทิศทางที่ไปข้างหน้าและย้อนกลับ สถาปัตยกรรมแบบนี้ยังมีชื่อเรียกอีกชื่อว่า Feedback Neural Network

สำหรับในหัวข้อนี้เราจะมาดูการเรียนรู้เชิงลึก (Deep Learning) รูปแบบที่มาตรฐานที่สุดคือการเรียนรู้แบบมีผู้สอนด้วยโมเดลแบบไม่เป็นเชิงเส้น (Supervised Learning with Nonlinear Model) นอกจากนี้ Neural Network ยังมีสถาปัตยกรรมที่หลากหลายซึ่งผู้อ่านสามารถศึกษาได้ในหัวข้อที่ 4.9

4.4 การแผ่กระจายการเรียนรู้

การแผ่กระจายการเรียนรู้ (Learning Propagation) เป็นขั้นตอนการเรียนรู้ของโมเดล Neural Network ที่เลียนแบบการทำงานของสมอง

จะเห็นได้ว่าไดอะแกรมด้านบนนั้นมีความซับซ้อนมาก ซึ่งจริง ๆ แล้วถ้าหากเราจะมาทำความเข้าใจองค์ประกอบของ Neural Network นั้น เราควรพิจารณากรณีง่าย ๆ ด้วยโครงสร้างแบบเล็ก ๆ ก่อน ตามภาพที่ 4.7a ซึ่งเป็นชั้นการเรียนรู้ (Learning Layer) ประกอบไปด้วย 3 ส่วน ดังนี้

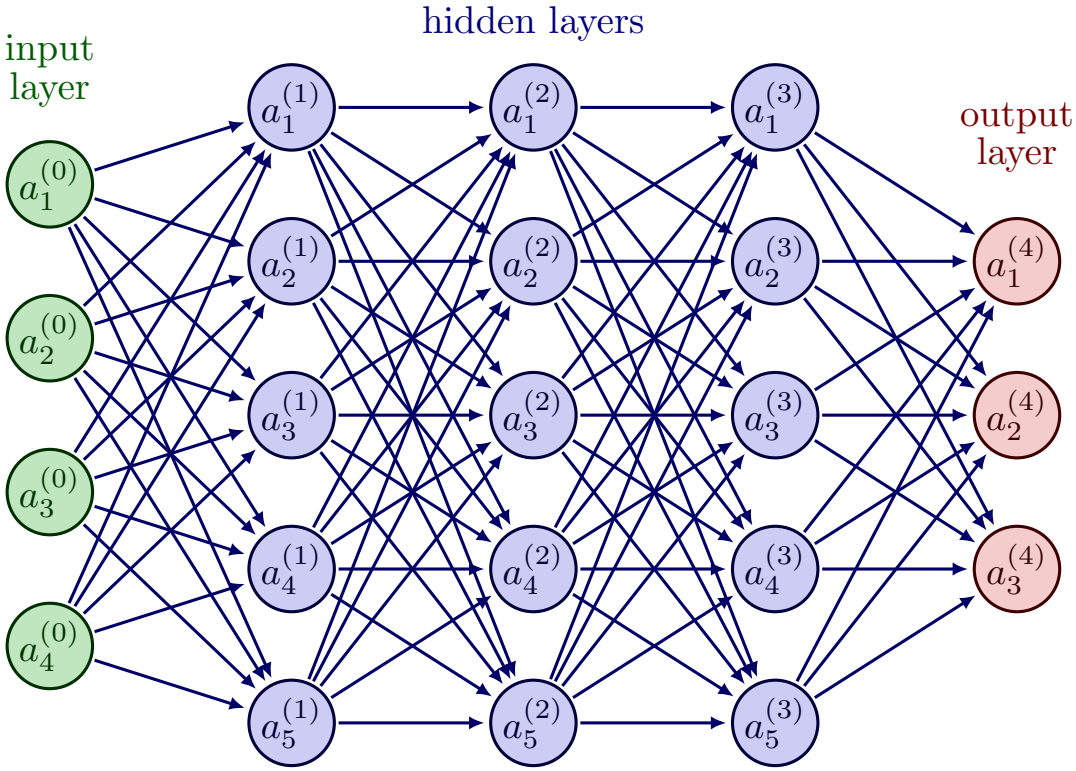
Input Layer ชั้นอินพุต เก็บข้อมูลที่เราจะนำมาใช้ในการฝึกสอนโมเดล โดยในแต่ละหน่วยประสาท (Neuron หรือ Learning Unit หรือ Node) จะเป็นตัวที่เก็บคุณลักษณ์ที่อยู่ในข้อมูล เช่น ความยาวพันธะหรือจำนวนเวเลนซ์อิเล็กตรอนของอะตอม

Hidden Layer ชั้นที่ถูกซ่อนไว้ เป็นชั้นที่อยู่ระหว่างชั้นอินพุตและชั้นเอาต์พุต โดยข้อมูลที่ถูกส่งมาจากชั้นก่อนหน้า (ในที่นี้คือชั้นอินพุต) จะถูกนำไปผ่านฟังก์ชันกระตุ้น (Activation Function) ในชั้นนี้ และหลังจากนั้นจะถูกส่งออกไปชั้นต่อไป (ในที่นี้คือชั้นเอาต์พุต)

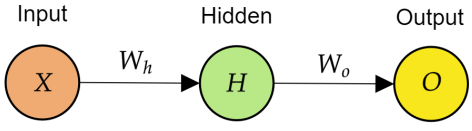
Output Layer ชั้นเอาต์พุต เป็นชั้นสุดท้ายของ Neural Network โดยจะรับข้อมูลหรืออินพุตมาจากชั้นก่อนหน้าซึ่งเป็น Hidden Layer โดยในชั้นนี้อาจจะมีการนำฟังก์ชันกระตุ้นมาใช้หรือไม่ใช้ก็ได้

โดยเราสามารถใช้โค้ดต่อไปนี้ในการสร้าง Neural Network แบบง่าย ๆ ได้

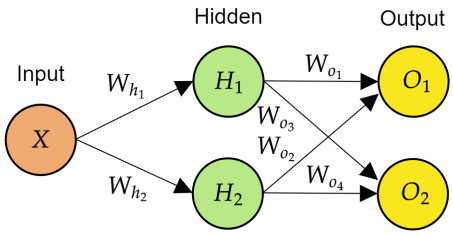
```
1 def relu(z):
2     return max(0,z)
3
4 def feed_forward(x, Wh, Wo):
5     # Hidden layer
```



ภาพ 4.6 โครงข่ายประสาทแบบสมบูรณ์ (Dense Neural Network) มี Notation $a_i^{(j)}$ แทนข้อมูลของหน่วยเรียนรู้หรือนิวรอนที่ i ของชั้นที่ j โดยชั้นที่ 1 กับชั้นที่ 4 ของตัวอย่าง Neural Network นี้คือชั้นอินพุตและชั้นเอาต์พุตตามลำดับ



(a) ชั้นการเรียนรู้ (Learning Layer)



(b) โครงข่ายแบบค่าน้ำหนักหลายตัว

ภาพ 4.7 ตัวอย่างของโครงข่ายประสาทแบบง่าย

```
    Zh = x * Wh
    H = relu(Zh)

    # Output layer
    Zo = H * Wo
    output = relu(Zo)
    return output
```

เมื่อเราพิจารณาภาพที่ 4.7b ซึ่งมีความซับซ้อนมากขึ้นโดยมีการเพิ่มจำนวน Neuron เข้าไปในชั้น Hidden Layer และ Output Layer เราจะพบว่าค่าน้ำหนักที่ถูกคำนวณออกมาจากชั้น Input จะถูกส่งไปยังชั้น Hidden และค่าน้ำหนักที่ออกมาจากชั้น Hidden ก็ถูกส่งต่อไปยังชั้น Output ตามลำดับ โดยจะเห็นว่าทุก ๆ Neuron ของชั้นที่ถูกติดกันนั้นจะมีการแลกเปลี่ยนกันทุก Neuron โดยสมการที่ใช้ในการคำนวณหาเอาต์พุตของแต่ละ Neuron มีดังนี้

- อินพุต 1 ตัว

$$\begin{aligned} Z &= Input \cdot Weight \\ &= XW \end{aligned} \tag{4.11}$$

- อินพุตมากกว่า 1 ตัว

$$\begin{aligned} Z &= \sum_{i=1}^{n} x_i w_i \\ &= x_1 w_1 + x_2 w_2 + x_3 w_3 \end{aligned} \tag{4.12}$$

เราจะสังเกตได้ว่าสมการที่ (4.11) และ (4.12) เป็นสมการที่เหมือนกับที่เราใช้ใน Linear Regression เป๊ะ ๆ เลย ซึ่งจริง ๆ แล้ว Neural Network ที่มีจำนวน Neuron แค่ 1 อันนั้นคือ Linear Regression เลย แต่สิ่งที่ต่างกันก็คือ Neural Network จะมีกระบวนที่เกี่ยวข้องกับค่าน้ำหนักและฟังก์ชันกระตุ้นด้วย สำหรับการกำหนดค่าน้ำหนักในช่วงเริ่มต้นของการฝึกสอนนั้นเราสามารถกำหนดค่าได้โดยใช้การสุ่มค่าตามตัวอย่างโค้ดดังต่อไปนี้

```
import numpy as np

INPUT_LAYER = 1
HIDDEN_LAYER = 2
OUTPUT_LAYER = 2

def init_weights():
    Wh = np.random.randn(INPUT_LAYER, HIDDEN_LAYER) \
            * np.sqrt(2.0/INPUT_LAYER)
    Wo = np.random.randn(HIDDEN_LAYER, OUTPUT_LAYER) \
```

```
11              * np.sqrt(2.0/HIDDEN_LAYER)
```

และเรามักจะกำหนดค่าเริ่มต้นของความโน้มเอียง (Bias) ด้วยค่าน้อย ๆ เช่น 0.1 หรือ 0.2 ดังตัวอย่างต่อไปนี้

```
1 def init_bias():
2     Bh = np.full((1, HIDDEN_LAYER), 0.1)
3     Bo = np.full((1, OUTPUT_LAYER), 0.1)
4     return Bh, Bo
```

4.4.1 การแผ่กระจายแบบไปข้างหน้า

ในขั้นเริ่มต้นของการฝึกสอนโมเดล Neural Network นั้น โมเดลจะยังไม่มีพารามิเตอร์ที่ถูกต้อง ดังนั้นเราจึงต้องสุ่มค่าเริ่มต้นของพารามิเตอร์ขึ้นมาก่อน หลังจากนั้นจึงทำ Forward Propagation รอบที่หนึ่งแล้วก็เปรียบเทียบผลการทำนายกับคำตอบ (Output) ที่โมเดลทราบก่อนหน้านั้นแล้ว ขั้นตอนต่อมาคือการปรับพารามิเตอร์ที่สำคัญอีกสองตัวนั่นคือน้ำหนัก (Weight) และความอคติหรือความโน้มเอียง (Bias) ให้มีค่าที่ถูกต้อง ซึ่งในขั้นตอนนี้เราจะใช้กระบวนการที่ตรงข้ามกันที่เรียกว่า Backward propagation (หรือ Backpropagation) โดยการทำ Propagation ทั้งสองแบบพร้อม ๆ กันครบหนึ่งรอบนั้นจะเรียกว่า 1 Epoch แต่ต้องระวังนะครับว่า Epoch, Batch Size และ Iteration นั้นมีความหมายต่างกัน โดยความแตกต่างมีดังนี้

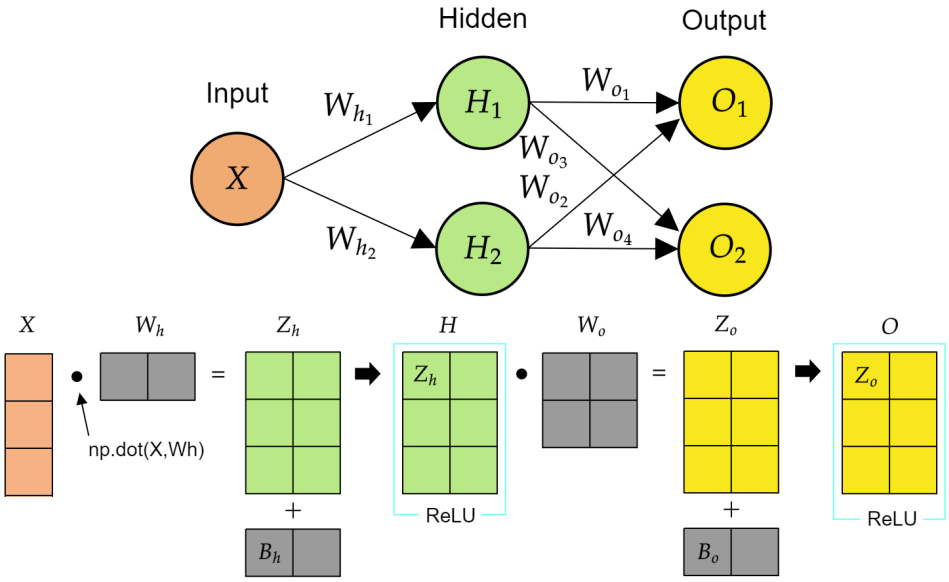
- **1 Epoch** การทำ Forward และ Backward Propagation 1 ครั้ง
- **Batch Size** จำนวนของข้อมูลที่ใช้ในการฝึกสอนในการทำ Forward และ Backward Propagation 1 รอบ
- **Iteration** จำนวนของรอบในการฝึกสอน ซึ่งแต่ละรอบจะใช้ Batch Size ที่ถูกกำหนดไว้ก่อนการฝึกสอน

เพื่อให้เห็นภาพมากขึ้น เราลองมาดูตัวอย่างกันครับ เช่น ถ้าเรามีจำนวนข้อมูลในการฝึกสอน 1,000 ข้อมูลและกำหนด Batch Size เป็น 500 จะได้ว่าโมเดลของเราจะใช้ 2 Iterations สำหรับการฝึกสอน 1 Epoch

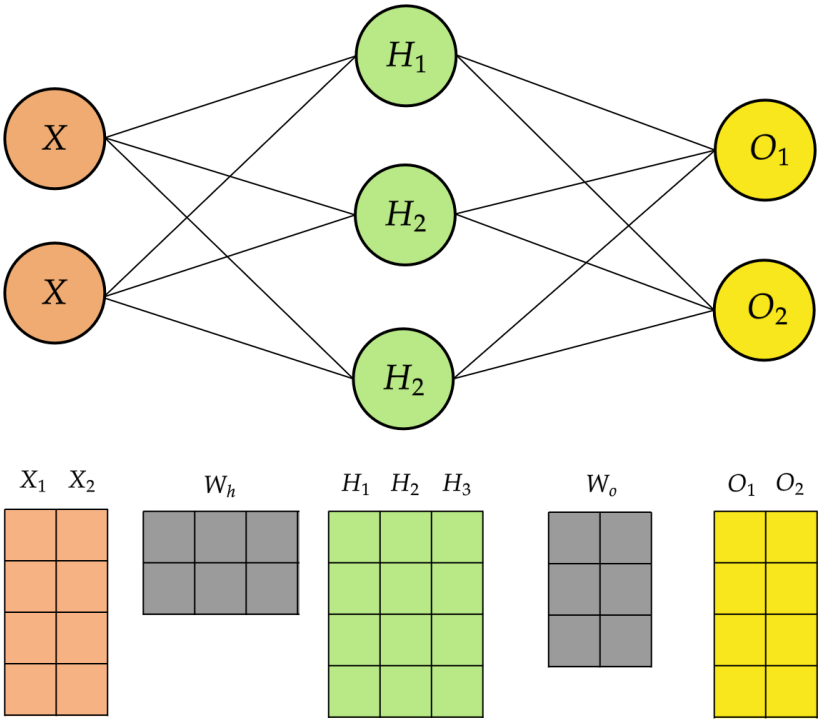
4.4.2 การแผ่กระจายแบบย้อนกลับ

การแผ่กระจายแบบย้อนกลับ (Backpropagation) เป็นหัวใจหลักของ Deep Learning เลยก็ว่าได้[1] นั่นก็เพราะว่าถ้า Neural Network ที่เราสร้างขึ้นนั้นมีแต่การเรียนรู้แบบแผ่ไปข้างหน้าโดยที่พารามิเตอร์ของโมเดลไม่มีการถูกปรับ (Optimization) ให้เหมาะสมนั้น ความสามารถในการทำนายค่าก็จะไม่เพิ่มขึ้น

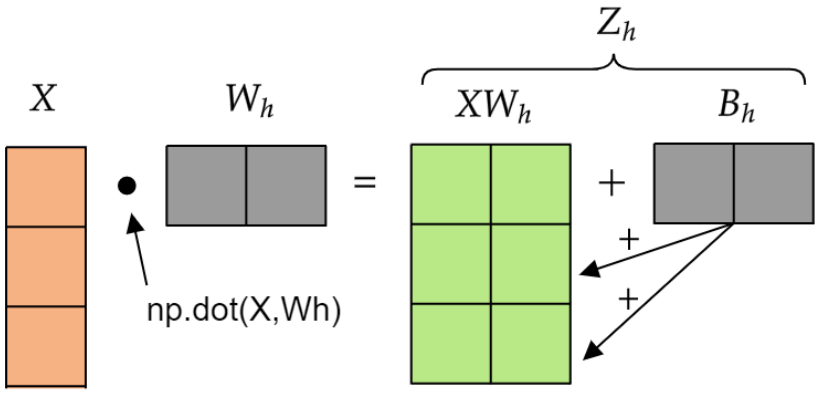
[1]ใน Neural Network เราเน้นที่ Deep Learning



ภาพ 4.8 แผนภาพการดำเนินการคูณเมทริกซ์อินพุนด้วยเมทริกซ์น้ำหนัก

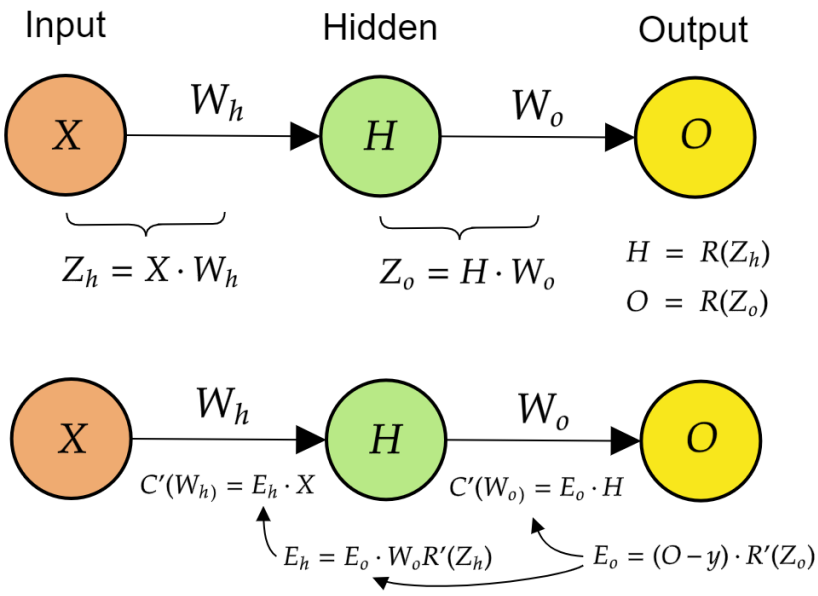


ภาพ 4.9 แผนภาพขนาดของเมทริกซ์ที่สามารถปรับขนาดได้แบบไดนามิกส์



ภาพ 4.10 แผนภาพการดำเนินการคูณเมทริกซ์น้ำหนักและการเพิ่มความโน้มเอียง

ดังนั้นถ้าหากเราต้องการเพิ่มความสามารถในการเรียนรู้ของโมเดล การแผ่กระจายย้อนกลับ (เรียกสั้น ๆ ว่า Backprop) นั้นจึงจำเป็นมาก เพราะการทำ Backprop นั้นเป็นการปรับค่า Weight โดยการเทียบกับ Loss Function ของเรา ซึ่ง Loss Function นี้เองที่เป็นตัวบอกความแตกต่างระหว่างค่าเอาต์พุตหรือค่าที่เราทำนาย (Prediction) กับค่าอ้างอิง (Reference) ดังนั้นถ้าหากเราต้องการที่จะปรับค่า Weight เพื่อให้มี Loss ที่น้อยลงเรื่อย ๆ เราสามารถทำได้โดยการหาอนุพันธ์ของ Loss เทียบกับ Weight แต่ว่าใน Neural Network นั้นมี Weight หลายค่ามาก ดังนั้นเราจึงจำเป็นต้องใช้กฎลูกโซ่เข้ามาช่วยในการหาอนุพันธ์หลายตัวแปร



ภาพ 4.11 การคำนวณการแผ่กระจายแบบย้อนกลับจากชั้นเอาต์พุตไปยังชั้นอินพุน

สำหรับการหาอนุพันธ์ของ Loss เทียบกับ Weight นั้นเราสามารถกระจายอนุพันธ์ออกมาให้อยู่ในรูป

ที่มี Weight จากแต่ละชั้น (Layer) ได้ เมื่อเราทำการหาอนุพันธ์นั้นเราจะต้องทำการหาของ Weight ที่อยู่ในชั้นท้ายสุดไล่ไปหาชั้นแรกสุด (ดูตามภาพที่ 4.11) นั่นจึงเป็นเหตุผลที่เราเรียกการแผ่กระจายแบบนี้ว่าการแผ่กระจายย้อนกลับเพราะว่าเราทำการปรับ Weight ของ Hidden Layer จากหลังไปหน้านั่นเอง

Algorithm 4.4 อัลกอริทึมของ Backpropagation สำหรับ Neural Network ซึ่งแสดงด้วย Computation Graph $G = (V, E)$.

1. For a sample (x_n, y_n^*), propagate the input x_n through the network to compute the outputs $(v_{i_1}, \ldots, v_{i_{|V|}})$ (in topological order).
2. Compute the loss $\mathcal{L}_n := \mathcal{L}(v_{i_{|V|}}, y_n^*)$ and its gradient

$$\frac{\partial \mathcal{L}_n}{\partial v_{i_{|V|}}}. \tag{4.13}$$

3. For each $j = |V|, \ldots, 1$ compute

$$\frac{\partial \mathcal{L}_n}{\partial w_j} = \frac{\partial \mathcal{L}_n}{\partial v_{i_{|V|}}} \prod_{k=j+1}^{|V|} \frac{\partial v_{i_k}}{\partial v_{i_{k-1}}} \frac{\partial v_{i_j}}{\partial w_j}. \tag{4.14}$$

   where w_j refers to the weights in node i_j.

สำหรับผู้อ่านที่สนใจอัลกอริทึมของ Backprop สามารถศึกษาเพิ่มเติมได้ที่หัวข้อ 4.4

4.5 ฟังก์ชันกระตุ้น

ฟังก์ชันกระตุ้น (Activation Function) หรือเรียกอีกชื่อว่า ฟังก์ชันการส่งต่อ (Transfer Function) เป็นหนึ่งใน Hyperparameters ที่สำคัญมาก ๆ ของ Neural Network นั่นก็เพราะว่าฟังก์ชันกระตุ้นนี้จะทำหน้าที่ในการปรับให้ฟังก์ชันที่ใช้อธิบายความสัมพันธ์ระหว่างข้อมูล x และ y นั้นมีความไม่เป็นเส้นตรง (Nonlinearity) อธิบายง่าย ๆ ก็คือ Neural Network ที่ไม่มีการใช้ฟังก์ชันกระตุ้นก็จะไม่ต่างอะไรจากโมเดล Linear Regression แบบเส้นตรงนั่นเอง โดยสิ่งที่ฟังก์ชันกระตุ้นทำก็คือจะทำการคำนวณค่าของเอาต์พุตออกมา ซึ่งรูปแบบของฟังก์ชันกระตุ้นก็จะมีหลากหลายรูปแบบ เนื่องจากปัญหาของข้อมูลในโลกความเป็นจริงมีลักษณะเป็นแบบสมการเส้นตรงน้อยมาก ฟังก์ชันกระตุ้นจึงเป็นเปรียบเสมือนเป็นเจ้าหน้าที่ที่คอยตัดสินใจว่าหน่วยเรียนรู้ (Node) ควรจะถูกกระตุ้นเพื่อให้เกิดการเรียนรู้หรือไม่ โดยพิจารณาจากค่าผลรวมของอินพุต, ค่า Weight, และค่า Bias โดยฟังก์ชันกระตุ้นถูกนำไปใช้ทั้งกับ Node ใน Hidden Layer และใน Output Layer ซึ่ง Node ในทั้งสองชั้นนี้อาจจะใช้ฟังก์ชันกระตุ้นที่เหมือนหรือต่างกันก็ได้ ขึ้นอยู่กับความสามารถและพฤติกรรมของการเรียนรู้ของชั้นนั้น ๆ

โดยส่วนมากแล้วเรามักจะใช้ฟังก์ชันกระตุ้นแบบไม่เป็นเชิงเส้นกับ Hidden Layer เหตุผลก็คือเนื่องจากว่าใน Hidden Layer จะมีการคำนวณแบบการรวมเชิงเส้น ถ้าฟังก์ชันกระตุ้นของ Hidden Layer มีการคำนวณแบบเชิงเส้นอีกก็จะเป็นการเรียนรู้การทำงานที่ซ้ำซ้อนกับการคำนวณแบบการรวมเชิงเส้นใน Output Layer และจะส่งผลให้ผลลัพธ์นั้นเทียบเท่ากับ Logistic Regression

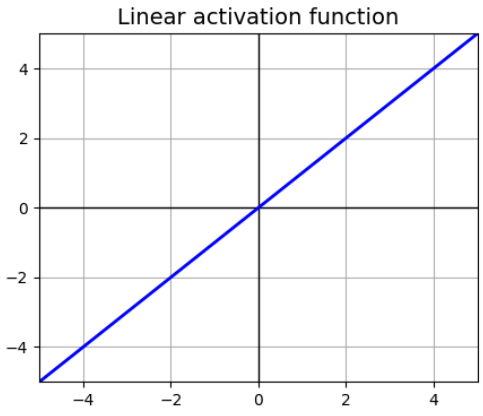
การเลือกฟังก์ชันกระตุ้นนั้นสำคัญมาก ถ้าหากเราเลือกฟังก์ชันกระตุ้นที่ไม่เหมาะสมหรือเลือกผิดชีวิตอาจเปลี่ยนได้เลย เช่น หนึ่งในปัญหาที่มักกวนใจผู้ที่ใช้ Neural Network อยู่เสมอนั่นก็คือ Vanishing Gradient Problem ซึ่งเป็นปัญหาที่เกิดขึ้นในระหว่างการฝึกสอนโมเดลโดยที่ Gradient ของ Loss Function นั้นมีขนาดเล็กลงเรื่อย ๆ จนเท่ากับ 0 ทำให้ Weight ไม่ถูกอัพเดทอีกต่อไป ส่งผลให้การฝึกสอนโมเดลไม่สามารถทำต่อได้ โดยวิธีการแก้ปัญหานั้นก็มีอยู่ด้วยกันหลายวิธี เช่น การทำ Weight Initialization หรือการใช้ Batch Normalization และการใช้ฟังก์ชัน ReLU แทนฟังก์ชัน Sigmoid ก็สามารถแก้ปัญหาได้เช่นเดียวกัน นอกจาก Vanishing Gradient Problem แล้วยังมีปัญหาที่ตรงข้ามกันนั่นก็คือ Exploding Gradient Problem ซึ่งเกิดขึ้นในระหว่างการฝึกสอนโมเดลเช่นเดียวกัน แต่จะเป็นกรณีที่ Gradient ของ Loss Function มีขนาดใหญ่ขึ้นเรื่อย ๆ จนเข้าใกล้ค่าอนันต์ (Infinity) ซึ่งจะถูกกำหนดหรือนิยามเป็น Not a Number (NaN) หมายความว่าตัวเลขมีค่าที่เยอะเกินหน่วยความจำของระบบที่ได้ถูกจัดสรรค์ไว้ (Allocation) ทำให้ไม่สามารถฝึกสอนโมเดลต่อไปได้

ฟังก์ชันกระตุ้นที่ถูกใช้ใน Neural Network มีหลากหลายรูปแบบ ตัวอย่างดังต่อไปนี้

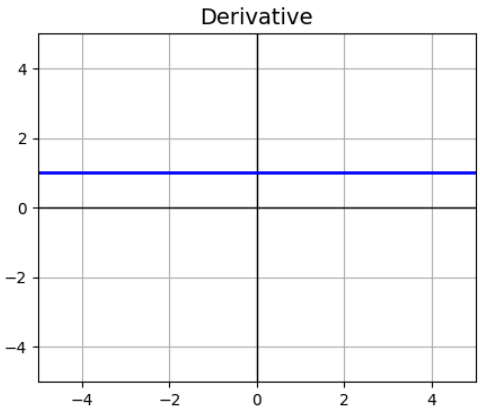
- **Binary Step** ฟังก์ชันไบนารี่เป็นฟังก์ชันกระตุ้นแบบที่ง่ายที่สุด โดยให้ค่าเอาต์พุตเพียงแค่ 2 ค่าเท่านั้นคือ 0 กับ 1 ถ้าอินพุตมีค่าน้อยกว่าหรือเท่ากับ 0 เอาต์พุตที่ได้จะเป็น 0 และถ้าอินพุตมีค่ามากกว่า 0 เอาต์พุตที่ได้ก็จะเป็น 1

$$R(z) = \begin{cases} 0 & for \quad z \leq 0 \\ 1 & for \quad z > 0 \end{cases} \tag{4.15}$$

- **Linear** ฟังก์ชันเส้นตรงโดยที่ Activation นั้นเป็นสัดส่วนโดยตรงกับอินพุต (นำค่า Weights มารวมกันตรง ๆ)



(a)



(b)

$$R(z,m) = \{z * m\} \tag{4.16}$$

$$R'(z,m) = \{m\} \tag{4.17}$$

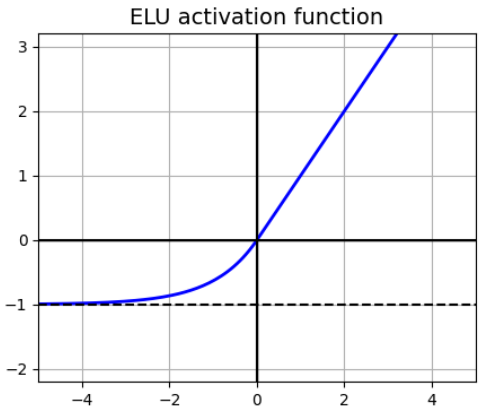
ข้อดี:

- เป็น Activation แบบที่เป็นช่วง (Range) ซึ่งมีประสิทธิดีกว่าฟังก์ชันแบบไบนารี่

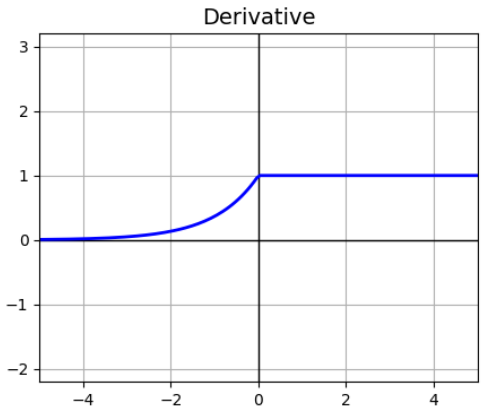
ข้อเสีย:

- อนุพันธ์ของฟังก์ชันนี้เป็นค่าคงที่ นั่นคือเกรเดียนต์ของฟังก์ชันนี้จึงไม่มีความสัมพันธ์กับอินพุต

- **ELU** ย่อมาจาก Exponential Linear Unit เป็นฟังก์ชันที่พยายามทำให้ Cost นั้นลู่เข้าสู่ศูนย์และให้ประสิทธิภาพที่ดีมากขึ้น โดยพารามิเตอร์ที่ควบคุมประสิทธิภาพของ ELU ก็คือ α ซึ่งควรจะต้องมีค่าเป็นบวกเสมอ



(a)



(b)

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha(e^z\text{–}1) & z <= 0 \end{Bmatrix} \quad (4.18)$$

$$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha e^z & z < 0 \end{Bmatrix} \quad (4.19)$$

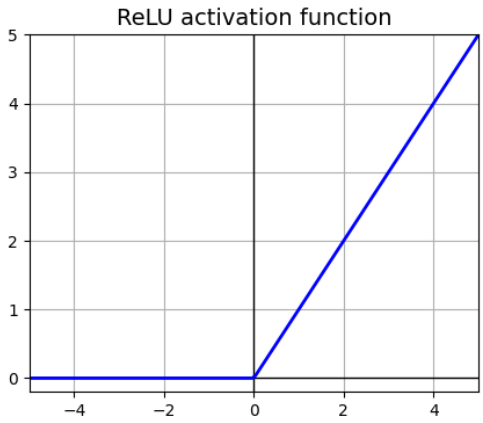
ข้อดี:

- ฟังก์ชัน ELU นั้นมีความราบเรียบ (Smooth) ยอ่างช้า ๆ จนกว่าค่าเอาต์พุตของฟังก์ชันจะมีค่าเท่ากับ $-\alpha$
- เราสามารถใช้ฟังก์ชัน ELU แทน ReLU ได้ในบางกรณีเพราะทั้งสองฟังก์ชันนี้มีความคล้ายกันมาก
- ELU สามารถให้เอาต์พุตที่มีค่าเป็นลบได้ แต่ว่าฟังก์ชัน ReLU นั้นให้เอาต์พุตที่เป็นบวกเสมอ

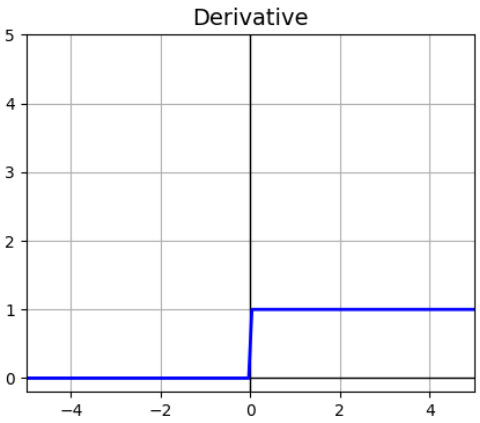
ข้อเสีย:

- สำหรับกรณีที่ $x > 0$ การใช้ฟังก์ชัน ELU อาจจะทำให้เกิดปัญหาได้ (เกิดการระเบิดหรือ Blow Up) จะทำให้เอาต์พุตอยู่ในช่วง $[0, \infty)$

- **ReLU**[30] ย่อมาจาก Rectified Linear Units เป็นฟังก์ชันกระตุ้นที่มีความสามารถในการเรียนรู้ฟังก์ชันหรือความสัมพันธ์ที่ไม่เป็นเชิงเส้นได้ดีมาก (ดีกว่าฟังก์ชัน Sigmoid)



(a)



(b)

$$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix} \quad (4.20) \qquad R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix} \quad (4.21)$$

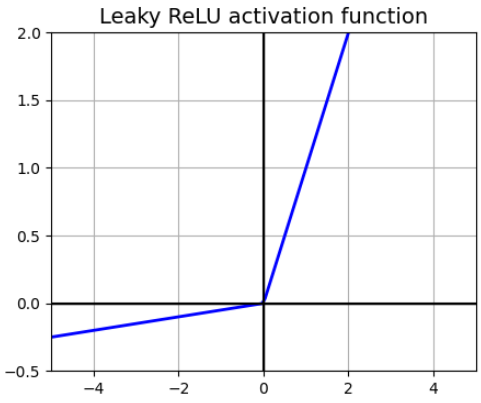
ข้อดี:

- ReLU แก้ปัญหา Vanishing Gradient Problem ได้
- ฟังก์ชัน ReLU นั้นเรียบง่ายกว่า Sigmoid และ Tanh จึงทำให้การคำนวณของฟังก์ชัน ReLU นั้นเร็วกว่าและสิ้นเปลืองน้อยกว่า

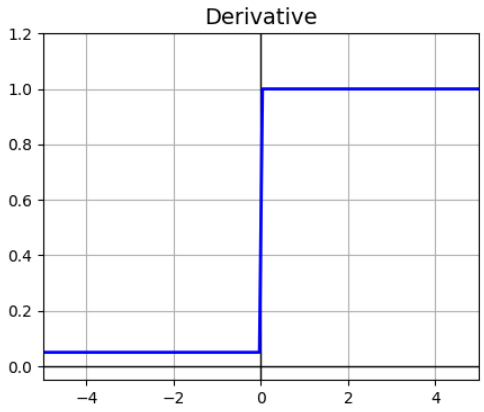
ข้อเสีย:

- ควรใช้ฟังก์ชัน ReLU เฉพาะใน Hidden Layer ของ Neural Network เท่านั้น
- ในระหว่างการฝึกสอนโมเดลด้วย ReLU ของบางกรณีนั้น การคำนวณ Gradient อาจจะมีปัญหาได้
- สำหรับการ Activation ของฟังก์ชัน ReLU ในช่วงที่อินพุต $x < 0$ ค่า Gradient จะเท่ากับ 0 เพราะว่า Weights นั้นจะไม่ถูกปรับค่าในระหว่างการทำ Gradient Descent
- เนื่องจากว่า ReLU นั้นมีความคล้ายกับ ELU ดังนั้นการใช้ฟังก์ชัน ReLU อาจจะทำให้เกิดปัญหาได้เพราะว่ามี Range คือ $[0, \infty)$

- **LeakyReLU**[31] เป็นฟังก์ชันที่ถูกพัฒนาต่อมาจาก ReLU ซึ่งได้มีการปรับปรุงให้มีประสิทธิภาพมากขึ้นโดยทำการปรับค่าเอาต์พุต ในกรณีที่อินพุต $z < 0$ ค่าเอาต์พุตจะไม่เป็น 0 อีกต่อไป แต่จะเป็น αz แทน ซึ่งทำให้มีความ General มากกว่าฟังก์ชัน ReLU ซึ่งโดยปกติแล้วจะกำหนดให้ $\alpha = 0.01$



(a)



(b)

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha z & z <= 0 \end{Bmatrix} \quad (4.22)$$

$$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha & z < 0 \end{Bmatrix} \quad (4.23)$$

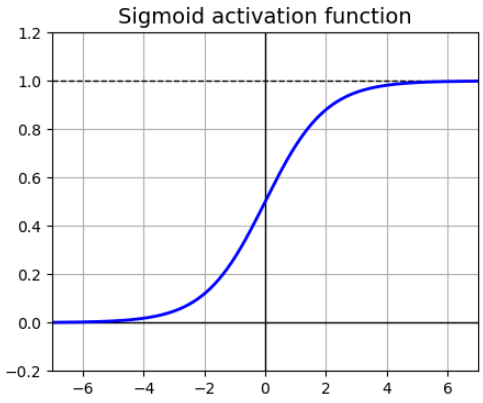
ข้อดี:

- LeakyReLU สามารถแก้ปัญหาของ ReLU ได้โดยการทำให้มีเอาต์พุตเป็น Slope ที่มีความชันน้อย ๆ

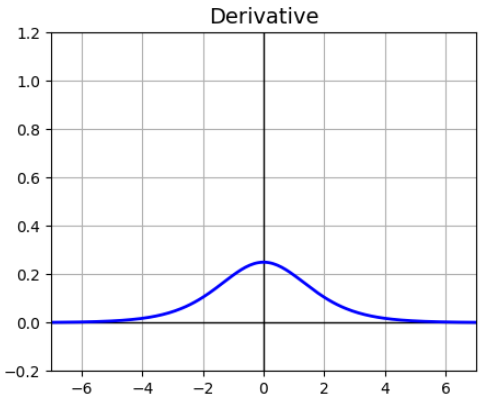
ข้อเสีย:

- เนื่องจากว่า Range ของ LeakyReLU นั้นเป็นเส้นตรง ดังนั้นจึงไม่เหมาะที่จะนำฟังก์ชันนี้มาใช้กับปัญหา Classification ที่ซับซ้อน ซึ่งในกรณีดังกล่าวการใช้ฟังก์ชัน Sigmoid หรือ Tanh จะเหมาะสมกว่า

- **Sigmoid**[32] เป็นหนึ่งในฟังก์ชันกระตุ้นที่มีประสิทธิภาพสูงมากในการนำมาใช้กับ Neural Network โดยฟังก์ชันนี้ทำการนำค่าอินพุตมาคำนวณโดยใช้ฟังก์ชัน Exponential ซึ่งให้ค่าเอาต์พุตที่อยู่ระหว่าง $(0, 1)$ ดังนั้นจึงทำให้ฟังก์ชันนี้มีคุณสมบัติหลายอย่างที่ฟังก์ชันกระตุ้นควรมี เช่น ความไม่เป็นเส้นตรง, มีความต่อเนื่อง, สามารถหาอนุพันธ์ได้ตลอดช่วง, มีความโมโนโทนิค (Monotonic) และมีเอาต์พุตที่อยู่ในช่วงที่แน่นอน



(a)



(b)

$$S(z) = \frac{1}{1 + e^{-z}} \tag{4.24}$$

$$S'(z) = S(z) \cdot (1 - S(z)) \tag{4.25}$$

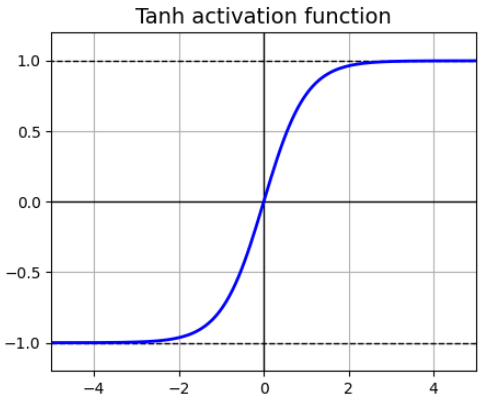
ข้อดี:

- มีความไม่เป็นเส้นตรง
- ฟังก์ชัน Sigmoid มีอนุพันธ์ที่มีความราบเรียบ
- เหมาะสำหรับการนำไปใช้กับโจทย์ Classification
- ฟังก์ชัน Sigmoid มี Range คือ $(0, 1)$ ซึ่งไม่เหมือนกับ Range ของฟังก์ชันเส้นตรงซึ่งมีค่าที่ไม่อยู่ในช่วงที่แน่นอนนั่นคือ $-\infty, \infty$

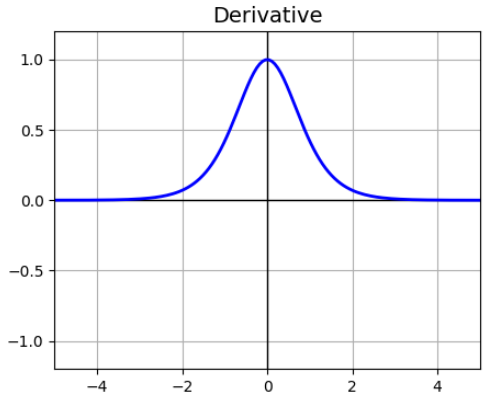
ข้อเสีย:

- กรณีที่ค่าอินพุตมีค่ามาก ๆ ค่าเอาต์พุตของ Sigmoid จะมีการเปลี่ยนแปลงที่น้อยมาก ๆ
- ฟังก์ชัน Sigmoid ทำให้เกิดปัญหา Vanishing Gradient Problem ได้
- เอาต์พุตของฟังก์ชัน Sigmoid มีจุดกึ่งกลางที่ไม่ใช่ 0 (Not Zero-centered) ทำให้การเปลี่ยนแปลงของ Gradient นั้นมีค่าที่อยู่ห่างจากฟังก์ชันเดิมมาก ๆ ซึ่งเป็นสาเหตุที่ทำให้การ Optimization นั้นยากขึ้น
- ในบางกรณีนั้นการใช้ฟังก์ชัน Sigmoid จะทำให้การเรียนรู้ของ Neural Network นั้นทำได้ยากและช้า

- **Tanh** เป็นฟังก์ชันกระตุ้นแบบไม่เป็นเชิงเส้นที่มี Range อยู่ระหว่าง -1 ถึง 1 $(-1, 1)$ สิ่งที่ฟังก์ชัน Tanh ต่างจากฟังก์ชัน Sigmoid ก็คือเอาต์พุตมีจุดกึ่งกลางอยู่ที่ 0 (Zero-centered) จึงทำให้ฟังก์ชันนี้ได้รับความนิยมมากกว่า Sigmoid นั่นเอง



(a)



(b)

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (4.26)$$

$$tanh'(z) = 1 - tanh(z)^2 \quad (4.27)$$

ข้อดี:

- Gradient ของฟังก์ชัน Tanh มีค่าการเปลี่ยนแปลงของ Slope ที่ดีกว่า Gradient ของฟังก์ชัน Sigmoid

ข้อเสีย:

- ฟังก์ชัน Tanh ยังคงมีปัญหาเกี่ยวกับ Vanishing Gradient Problem

- **Softmax** เป็นฟังก์ชันกระตุ้นที่คำนวณ Probability Distribution ของเหตุการณ์ทั้งหมด n เหตุการณ์ที่แตกต่างกัน กล่าวง่าย ๆ คือฟังก์ชันนี้ทำการคำนวณคลาส (Class) ของเป้าหมายของเราให้อยู่ในรูปของค่าความน่าจะเป็นซึ่งจะถูกนำมาใช้ในการกำหนดหรือทำนายคลาสของเหตุการณ์ที่เราสนใจ

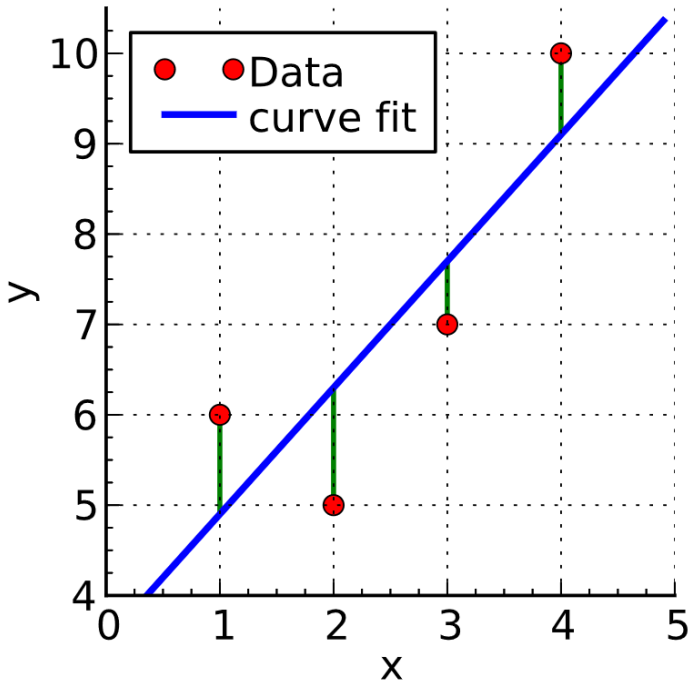
$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \quad (4.28)$$

ถ้าหากผู้อ่านต้องการศึกษาการเขียนโค้ดสำหรับพล็อตกราฟฟังก์ชันกระตุ้นแบบต่าง ๆ สามารถดูได้ที่ https://github.com/siebenrock/activation-functions

4.6 ฟังก์ชันสูญเสีย

4.6.1 ความสำคัญของฟังก์ชันสูญเสีย

ฟังก์ชันสูญเสีย (Loss Function หรือ Cost Function) เป็นฟังก์ชันความคลาดเคลื่อน (Error Function) รูปแบบหนึ่งซึ่งมีความสำคัญมากใน Neural Network (จริง ๆ แล้วเราจะถือว่า Loss Function กับ Error Function นั้นเป็นสิ่งเดียวกันก็ได้) เพราะว่าเป็นฟังก์ชันคณิตศาสตร์ที่ Map ค่าเหตุการณ์ (Event) จากอินพุตหลาย ๆ ตัว ให้ออกมาเป็นค่า Error เพียงแค่เดียวซึ่งเป็นค่าที่ระบุถึง Cost ของเหตุการณ์นั้น ๆ โดย Loss Function ถูกใช้ในการทำ Optimization นั่นก็คือการหาวิธีที่ทำให้ค่าเอาต์พุตของของ Loss Function นั้นมีค่าน้อยที่สุด เรียกว่า Minimization



ภาพ 4.18 แสดงการทำ Least Squares Fitting โดยมีพิกัดตำแหน่งของข้อมูลแต่ละตัว (จุดสีแดง) ดังนี้ $(1,6)$, $(2,5)$, $(3,7)$ และ $(4,10)$ และเส้นตรงที่ได้จากการประมาณค่าด้วยวิธี Least Squares Estimation นั้น (เส้นสีน้ำเงิน) (เครดิตภาพ: https://en.wikipedia.org/wiki/Linear_least_squares)

แนวคิดของ Loss Function ก็คือเราต้องการตัวชี้วัดที่เป็นตัวเลขค่าเดียวที่สามารถบอกได้ว่าโมเดล ML ที่ฝึกสอนมาแล้วนั้นทำงานได้ดีแค่ไหน โดยถ้าหากดูจากภาพที่ 4.18 โดยเปรียบเทียบเอาต์พุตของโมเดลนั่นก็คือ $\hat{y}$ กับข้อมูลเอาต์พุตตัวอย่าง y นั่นก็คือจุดสีแดงและมีค่าเส้นน้ำเงินที่เป็นเส้นที่เกิดจากการ Fitting และมีเส้นสีเขียวที่บ่งบอกว่าจุดสีแดงแต่ละจุดนั้นมีการเบี่ยงเบน (Deviation) ออกจากเส้นสีน้ำเงินมากน้อยเพียงใด

ฟังก์ชันสูญเสียสำหรับโจทย์ประเภท Regression

การทำ Regression นั้นจะเกี่ยวข้องกับการทำนายค่าที่แน่นอนและมีความต่อเนื่อง ดังนั้น Loss Function ที่เหมาะสมจึงจะต้องสามารถที่จะอธิบายความต่อเนื่องของเอาต์พุตของแต่ละจุดในชุดข้อมูลได้ด้วย ตัวอย่างของ Loss Function ที่นิยมใช้ในโจทย์ประเภท Regression ของชุดข้อมูลที่มี n จำนวนข้อมูลและมีค่าเอาต์พุตที่ได้จากการทำนายที่เป็น $\hat{y}$ และคำตอบอ้างอิง (Reference หรือ Label) คือ y มีดังต่อไปนี้

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.29}$$

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{x_i}\right| \times 100 \tag{4.30}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.31}$$

$$\text{MaxAE} = \max\{y_i - \hat{y}_i\}, i = 1, 2, ..., n \tag{4.32}$$

$$\text{MaxAPE} = \max\left\{\left|\frac{y_i - \hat{y}_i}{x_i}\right| \times 100\right\}, i = 1, 2, ..., n \tag{4.33}$$

$$\text{RMSD} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.34}$$

$$\text{GRMSD} = \sqrt[2n]{\prod_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.35}$$

$$\text{GWRMSD} = \sqrt{\frac{\sum_{i=1}^{n}\zeta_i(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}\zeta_i}} \tag{4.36}$$

โดยที่ $\zeta_i = e^{-(y_i - \hat{y}_i)/c}$

$$L_\delta = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |(y - \hat{y})| < \delta \\ \delta((y - \hat{y}) - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{4.37}$$

สำหรับ Loss Function ที่ (4.29) - (4.36) จะมีความคล้ายคลึงกัน แต่จะต่างกันตรงที่การปรับรูปแบบให้ฟังก์ชันที่เป็นความแตกต่างระหว่างค่าอ้างอิงและค่าทำนายนั้นมีความไว (Sensitivity) ต่อ Outlier ที่ต่างกันไป สำหรับสมการที่ (4.37) นั้นคือ Huber Loss ซึ่งจะมีความไวต่อค่าที่ห่างค่าผิดปกติ (Outlier)[1] ที่น้อยกว่ากรณีของ MSE (สมการที่ (4.31)) เพราะว่าใน MSE เทอม $y_i - \hat{y}_i$ ถูกกำลังสองอยู่นั่นเอง

เพิ่มเติม: MAE กับ MSE นั้นจะมีชื่อเรียกอีกอย่างว่า L1 และ L2 ด้วย

ฟังก์ชันสูญเสียสำหรับโจทย์ประเภท Classification

Loss Function สำหรับโจทย์แบบ Classfication นั้นจะแตกต่างจาก Regression โดยสิ้นเชิงเนื่องจากว่าเราไม่ได้ทำการหาค่าระยะห่างระหว่างจุดข้อมูลที่ได้จากการทำนายแล้ว แต่จะเป็นการหาความน่าจะเกิดที่เกิดขึ้นระหว่างข้อมู,ที่มีความไม่ต่อเนื่องกัน (Discrete Class Output) ซึ่งจะออกจากกันโดยสิ้นเชิง กล่าวคือโจทย์ประเภทนี้จะเป็นการแบ่งชุดข้อมูลออกเป็นคลาสหลาย ๆ คลาสที่แตกต่างกันโดยขึ้นอยู่กับพารามิเตอร์ที่ต่างกัน ซึ่งหน้าที่ของ Loss Function สำหรับโจทย์ประเภทนี้คือจะต้องทำการคำนวณความน่าจะเป็นว่าควรจะต้องเพิ่มหรือระบุว่าข้อมูลใหม่นั้นควรจะต้องถูกจัดเข้าไปอยู่ในกลุ่มไหน

- Cross-entropy คือการวัดความแตกต่างระหว่างการกระจายตัวของความน่าจะเป็น 2 กลุ่มของตัวแปรแบบสุ่มของเหตุการณ์ที่เราสนใจ (กลุ่มหรือ Class ในชุดข้อมูล) โดยกรณีที่มีแค่ 2 Class $(M = 2)$ เรามีชื่อเรียกฟังก์ชันประเภทนี้ว่า Binary Cross-entropy และกรณีที่มีมากกว่า 2 Class $(M > 2)$ เราจะเรียกว่า Multiclass Cross-entropy หรือจะเรียกว่า Categorical Cross-entropy ก็ได้ ซึ่งทั้งสองแบบมีสมการดังต่อไปนี้

$$H(p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{4.38}$$

$$H(p) = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{4.39}$$

โดยที่ p คือความน่าจะเป็นของการสังเกต o ใน Class c และ y คือตัวระบุ Class (เช่น 0 กับ 1) โดยที่ Binary

[1] ค่าผิดปกติหรือ Outlier คือค่าที่อยู่ห่างจากค่าส่วนใหญ่ในชุดข้อมูลมากเกินไป

Cross-entropy นี้เป็น Loss Function ที่ได้รับความนิยมมากที่สุดสำหรับโจทย์ Classification ที่มีสองคลาส ซึ่งคำว่า Entropy นี้ก็หมายถึงเป็นการวัดความไม่แน่นอนของการสุ่ม (Randomness) ที่เกิดขึ้นในข้อมูลในขณะที่ถูก Process อยู่นั่นเอง และ Cross Entropy ก็คือการวัดความแตกต่างของ Randomness ระหว่างตัวแปรสุ่มสองตัว

- Negative Log-likelihood (NLL)

$$l(y) = -\log(p(y)) \tag{4.40}$$

- Hinge Loss

$$l(y) = \max(0, 1 - y \cdot \hat{y}) \tag{4.41}$$

Hinge Loss เป็น Loss Function ที่ถูกพัฒนาขึ้นมาเพื่อใช้งานกับ Support Vector Machine โดยเฉพาะสำหรับการคำนวณระยะห่างที่มากที่สุด (Maximum Margin) จาก Hyperplane ถึง Class

- Kullback-Leibler (KL) Divergence

$$KL(\hat{y}||y) = \sum_{c=1}^{M} \hat{y}_c \log\left(\frac{\hat{y}_c}{y_c}\right) \tag{4.42}$$

เป็นการวัดว่าข้อมูลนั้นมีการกระจายตัวห่างไปจากค่าการกระจายตัวอ้างอิงมากน้อยแค่ไหน

- Jensen-Shannon (JS) Divergence

$$JS(\hat{y}||y) = \frac{1}{2}\left(KL\left(y||\frac{y+\hat{y}}{2}\right) + KL\left(\hat{y}||\frac{y+\hat{y}}{2}\right)\right) \tag{4.43}$$

4.6.2 คณิตศาสตร์ของฟังก์ชันสูญเสีย

ลำดับต่อมาเรามาดูกันที่คณิตศาสตร์ที่อยู่เบื้องหลังของ Loss Function กันครับ โดยจะมาดูว่าเราสามารถทำการปรับค่าพารามิเตอร์ต่าง ๆ ใน Neural Network เช่น Weight ได้อย่างไร โดยผู้เขียนจะใช้ตัวอย่างโจทย์ Regression สองแบบนั่นคือ Linear Regression และ Logistic Regression

Linear Regression

เราเริ่มต้นด้วยการกำหนดฟังก์ชันหลักสำหรับการทำ Linear Regression ดังนี้

Linear Equation :	$z = Xw + b$
Activation Function :	None
Prediction :	$\hat{y} = z$
Loss Function :	$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$

เราสามารถเขียนโค้ดของฟังก์ชันด้วยภาษา Python ได้ดังนี้

```
1  import numpy as np
2
3  weights = np.random.normal(size =
       n_features).reshape(n_features, 1)
4  bias = 0
5
6  def linear_regression_inference(inputs):
7      return np.matmul(inputs, weights) + bias
8
9  def calculate_error(x, y):
10     # Mean Squared Error (Ignore taking an average)
11     y_hat = linear_regression_inference(x)
12     return 0.5 * (yhat - y)**2
```

นอกจากนี้เรายังสามารถคำนวณอนุพันธ์ (Derivative) ของ Loss Function เทียบกับ z โดยใช้กฎลูกโซ่ได้ดังนี้

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \tag{4.44}$$

เริ่มต้นด้วยการหาอนุพันธ์ย่อยของ Loss Function เทียบกับค่า Prediction

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y \tag{4.45}$$

ขั้นตอนต่อมาคือเราหาอนุพันธ์ย่อยของ Prediction เทียบกับ Linear Equation แต่เนื่องจากว่า Linear Equation จริง ๆ แล้วก็คือ Prediction นั่นเอง จึงทำให้อนุพันธ์ย่อยนั้นมีค่าเท่ากับ 1

$$\frac{\partial \hat{y}}{\partial z} = 1 \tag{4.46}$$

เมื่อเราทำการคูณสมการที่ (4.45) กับ (4.45) เข้าด้วยกัน เราจะได้อนุพันธ์ของ Loss Function ตามที่ต้องการ

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y \tag{4.47}$$

จากตัวอย่างข้างต้นนี้ผู้อ่านจะพบว่า Linear Regression นั้นง่ายมาก แต่ตัวอย่างต่อไปจะมีความซับซ้อนมากขึ้นครับ

Logistic Regression

ตัวอย่างที่สองคือกรณีของ Logistic Regression โดยเรากำหนดฟังก์ชันต่าง ๆ ดังนี้

Linear Equation :	$z = Xw + b$
Activation Function :	$\sigma(z) = \frac{1}{1+e^{-z}}$
Prediction :	$\hat{y} = \sigma(z)$
Loss Function :	$\mathcal{L} = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

และเขียนโค้ดของฟังก์ชันได้ดังนี้[1]

```
1 import numpy as np
2
3 weights = np.random.normal(size =
     n_features).reshape(n_features, 1)
4 bias = 0
5
6 def sigmoid(x):
```

[1]โดยทั่วไปแล้วในการคำนวณหา Error ของ Logistic Regression ควรจะทำการใส่ค่าคงที่เข้าไปในฟังก์ชัน Log เพื่อป้องกันไม่ให้อินพุตของ Log นั้นเป็น 0

```
    return 1 / (1 + np.exp(-x))

def logistic_regression_inference(x):
    return sigmoid(np.matmul(x, weights) + bias)

def calculate_error(x, y):
    # Binary Cross-Entropy
    y_hat = logistic_regression_inference(x)
    return -(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
```

เราเริ่มต้นด้วยวิธีเดียวกันกับที่เราใช้ในตัวอย่างที่แล้วคือการใช้กฎลูกโซ่ (สมการที่ (4.44)) แล้วทำการหาอนุพันธ์ของแต่ละเทอม ดังนี้

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \tag{4.48}$$

ลำดับต่อไปก็คือการทำอนุพันธ์ย่อยของ Prediction เทียบกับฟังก์ชัน z ซึ่งสามารถทำได้ดังนี้

$$\begin{aligned}
\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z}\left[\frac{1}{1+e^{-z}}\right] \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}-1}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) \\
&= \hat{y}(1-\hat{y})
\end{aligned} \tag{4.49}$$

เมื่อเรามาถึงขั้นตอนนี้แล้ว ขั้นตอนต่อไปคือการรวมสมการอนุพันธ์ย่อยทั้งสองสมการเข้าด้วยกัน โดยสามารถทำได้ดังนี้

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial z} &= \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right)\hat{y}(1-\hat{y}) \\ &= -\frac{y}{\hat{y}}\hat{y}(1-\hat{y}) + \frac{1-y}{1-\hat{y}}\hat{y}(1-\hat{y}) \\ &= -y(1-\hat{y}) + (1-y)\hat{y} \\ &= -y + y\hat{y} + \hat{y} - y\hat{y} \\ &= \hat{y} - y \end{aligned} \tag{4.50}$$

ซึ่งเราจะพบว่าคำตอบของสมการที่ (4.47) และ (4.50) นั้นเท่ากันเลย

4.7 ตัวประเมินโมเดล

สิ่งที่เราใช้ในการประเมินหรือวัดประสิทธิภาพของโมเดลก็คือ Metric ซึ่ง Metric สำหรับโจทย์ประเภท Regression นั้นเราสามารถใช้ฟังก์ชันที่เป็น Loss Function ได้เลย (หัวข้อที่ 4.6.1) โดยด้านล่างคือตัวอย่างของโค้ดสำหรับการใช้ Metric

```
from tensorflow.keras import metrics

model.compile(loss='mse', optimizer='adam',
              metrics=[metrics.mean_squared_error,
                       metrics.mean_absolute_error,
                       metrics.mean_absolute_percentage_error])
                       metrics.categorical_accuracy])
```

แต่กรณีโจทย์ประเภท Classification นั้นเราจะต้องใช้ฟังก์ชันที่ต่างกันออกไป ฟังก์ชันต่อไปนี้คือ Metrics ที่มักจะถูกใช้สำหรับ Classification

- ประเภทวัดความถูกต้องและแม่นยำ

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.51}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.52}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.53}$$

$$\begin{aligned} \text{F1} &= \frac{2 * Precision * Recall}{Precision + Recall} \\ &= \frac{2 * TP}{2 * TP + FP + FN} \end{aligned} \tag{4.54}$$

● ประเภทวัดความว่องไวและความจำเพาะเจาะจง

$$\text{Sensitivity} = \text{Recall} = \frac{TP}{TP + FN} \tag{4.55}$$

$$\text{Sensitivity} = \frac{TN}{FP + TN} \tag{4.56}$$

นอกจากนี้ยังมี Area Under the Curve (AUC) ซึ่งเป็นการใช้พื้นที่ใต้เส้นโค้งสำหรับการแบ่งกลุ่ม (Classifier) อีกด้วย ผู้อ่านที่ต้องการศึกษา Metrics เพิ่มเติมสามารถดูได้ที่ https://neptune.ai/blog/keras-metrics

4.8 ตัวปรับความเหมาะสม

ตัวปรับความเหมาะสมหรือปรับประสิทธิภาพการเรียนรู้ของโมเดล (Optimizer) เป็นฟังก์ชันทางคณิตศาสตร์ ซึ่งขึ้นอยู่กับพารามิเตอร์ที่เรียนรู้ได้ของโมเดล เช่น Weight และ Bias ซึ่ง Optimizer เป็นสิ่งที่จะช่วยให้โมเดลทราบวิธีการเปลี่ยน Weight และ Learning Rate ของ Neural Network เพื่อลดค่า Loss หรือ Error ที่เกิดขึ้นให้น้อยลงในแต่ละรอบของการฝึกสอนโมเดล

ตัวอย่างของ Optimizer ที่ได้รับความนิยมและมีประสิทธิภาพที่ยอดเยี่ยม

● Stochastic Gradient Descent (SGD) เป็นฟังก์ชันที่อัพเดทค่าพารามิเตอร์ในทุก ๆ ชุดข้อมูลที่ใช้ในการฝึกฝน SGD เป็นอัลกอริทึมที่ค่อนข้างไว โดยอัพเดทแค่ครั้งเดียวต่อการฝึกสอนโมเดล 1 รอบ นอกจากนี้ยังมีสิ่งที่เรียกว่าโมเมนตัม (Momentum) ซึ่งถูกพัฒนาขึ้นมาเพื่อเร่งความเร็วในการ Opitmization ของ SGD โดยจะเป็นตัวที่เข้ามาแก้ปัญหาความแปรปรวนที่เกิดขึ้นใน SGD ซึ่งทำให้เกิดความยากในการที่จะลู่เข้าจุดที่ต่ำที่สุดได้ โดยการให้ความสำคัญในการพุ่งไปยังทิศทางที่ใกล้จุดกลางมากที่สุดก่อนแล้วทำให้ทิศทางที่ไม่เกี่ยวข้องความสำคัญลดลง โดยสามารถอ่านรายละเอียดเพิ่มเติมได้ในหัวข้อที่ 4.2.2

● Mini-batch Stochastic Gradient Descent ถูกพัฒนาขึ้นเพื่อแก้ปัญหาของ Gradient Descent (GD) โดยการนำข้อดีของ GD แบบธรรมดาและ SGD มารวมกัน โดยสำหรับอัลกอริทึมนี้จะทำการอัพเดทค่าเป็น “ชุด” โดยภายในแต่ละชุดจะประกอบด้วยข้อมูลจำนวน n ข้อมูล เราจึงเรียกเทคนิคนี้ว่าเป็น Mini-batch หรือจำนวนชุดข้อมูลขนาดเล็กนั่นเอง โดยสามารถอ่านรายละเอียดเพิ่มเติมได้ในหัวข้อที่ 4.2.3

● Adagrad เป็นฟังก์ชันที่สามารถปรับค่าอัตราเร็วในการเรียนรู้ (Learning Rate) ให้เหมาะสมกับพารามิเตอร์ได้ โดยจะมีการอัพเดทจำนวนมากสำหรับค่าพารามิเตอร์ที่มีจำนวนน้อย และอัพเดทน้อยถ้าค่าพารามิเตอร์มีจำนวนมากและด้วยเหตุนี้ Optimizer ตัวนี้จึงเป็นที่นิยมสำหรับข้อมูลที่มีการกระจายตัว (Sparse Data)

● Adadelta เป็นฟังก์ชันที่พัฒนาต่อจาก AdaGrad โดยสามารถแก้ปัญหา Decaying Learning Rate ที่เกิดขึ้นใน AdaGrad ได้ โดยเคล็ดลับก็คือแทนที่จะเก็บสะสมการคำนวณทั้งหมดที่ผ่านมาของ Gradient ใน AdaDelta นั้นจะถูกจำกัดการสะสมค่าการคำนวณของ Gradient ได้เองเพื่อแก้ขนาดค่าของน้ำหนักที่จะเกิดขึ้น หมายความว่าแทนที่เราจะเก็บค่าน้ำหนักที่ได้รับการอัพเดทมาก่อนหน้านี้ที่ยังไม่เวิร์ค เราจะเปลี่ยนเป็นการหาผลรวมของ Gradients แทน ซึ่งจะทำแบบนี้ซ้ำไปเรื่อย ๆ เพื่อการแก้ปัญหา Decaying Learning Rate ของ Gradients ที่ผ่านมาทั้งหมด

● Adam ย่อมาจาก “Adaptive Moment Estimation” เป็นวิธีการสุ่มเกรเดียนต์ที่อิงจากการประมาณค่าแบบปรับตัว (Adaptive Estimation) ของช่วงเวลาอันดับที่หนึ่งและอันดับสอง ซึ่งความสามารถของ Adam ก็คือสามารถปรับอัตราเร็วของการเรียนรู้พารามิเตอร์ในแต่ละครั้งได้และยังสามารถแก้ปัญหาการลดลงที่เร็วเกินไป (Decaying) ของ Gradients ในแต่ละ Step ที่ผ่านมาได้เหมือนกับ AdaDelta อีกทั้งยังอธิบายการเกิด Decaying Average ของ Gradients ที่ผ่านมาได้อีกด้วย ซึ่งจะเหมือนกับโมเมนตัมนั่นเอง

Adam เป็นอัลกอริทึมเป็นที่นิยมมากที่สุดเพราะรวมข้อดีของแต่ละอัลกอริทึมที่อธิบายไว้ก่อนหน้านี้เข้าด้วยกัน และแก้ปัญหาหรือข้อบกพร่อมออกไป เช่น Decaying Learning Rate ของ Adagrad และยังมีความเร็วที่มากกว่า GD และลดปัญหาการแกว่งของพารามิเตอร์ได้อีกด้วย

นอกเหนือจากตัวปรับความเหมาะสมข้างต้นแล้ว ยังมีอัลกอริทึมอีกหลายแบบที่ได้รับความนิยม เช่น Conjugate Gradients, Momentum (ใช้ใน Stochastic Gradient Descent), Broyden-Fletcher-Goldfarb-Shanno (BFGS), Nesterov Momentum, วิธีของ Newton, และ RMSProp ซึ่งผู้อ่านสามารถศึกษาเพิ่มเติมได้จากหนังสือ Algorithms for Optimization[33] โดย Mykel J. Kochenderfer โดยสามารถอ่านและดาวน์โหลดได้ฟรีที่ https://algorithmsbook.com

4.9 สถาปัตยกรรมของโครงข่ายประสาท

สถาปัตยกรรม (Architecture) ของ Neural Network เปรียบเสมือนเป็นการจำลองหรือเลียนแบบการเชื่อมโยงเซลล์ประสาทเทียมเข้าด้วยกันเป็นโครงข่ายประสาทซึ่งสามารถเชื่อมโยงแบบใดก็ได้อย่างไม่มีขอบเขตจำกัด อย่างไรก็ตาม ในทางปฏิบัตินั้น เทคนิคการเรียนรู้ของ Neural Network มักจะถูกออกแบบมาให้ใช้งานได้กับสถาปัตยกรรม Neural Network ที่มีลักษณะเฉพาะเท่านั้น

4.9.1 โครงข่ายประสาทมาตรฐาน

Perceptron เพอร์เซ็ปตรอนเป็น Neural Network แบบที่ง่ายที่สุด โดยมีเพียงแค่หน่วยการเรียนรู้ที่รับข้อมูลอินพุตและคืนค่าเอาต์พุตออกมา ซึ่ง Perceptron คือองค์ประกอบพื้นฐานของ Neural Network ทั้งหมด

Multi-layer Perceptron Neural Network เป็นการนำ Perceptron หลาย ๆ อันมารวมกันได้เป็น Neural Network ที่มีหลายชั้น ถูกนำมาใช้สำหรับปัญหาที่มีความซับซ้อนได้ผลเป็นอย่างดี โดยมีกระบวนการฝึกฝนเป็นแบบมีผู้สอน (Supervised ML) และใช้ขั้นตอนการส่งค่าย้อนกลับ (Backpropagation) สำหรับการฝึกฝนกระบวนการส่งค่าย้อนกลับ

Residual Networks หรือ ResNet เป็น Neural Network ที่ถูกพัฒนาขึ้นเพื่อแก้ปัญหา Vanishing Gradient ซึ่งมักเจอได้บ่อยใน Deep Neural Network ที่มี Hidden Layer หลายชั้น โดยไอเดียของ ResNet คือการนำ Weights จากชั้นตื้น (Shallow Layer) มาใช้ในชั้นลึก (Deep Layer)

4.9.2 โครงข่ายประสาทแบบวนซ้ำ

โครงข่ายประสาทแบบวนซ้ำ (Recurrent Neural Network หรือ RNN)[34] เป็นสถาปัตกรรมที่ถูกออกแบบเพื่อเพิ่มความสามารถในการจดจำข้อมูลในอดีตที่ได้เรียนรู้ไปแล้ว ซึ่งในที่นี้คือข้อมูลของชั้นก่อนหน้า นั่นก็เพราะว่า Deep Neural Network แบบทั่วไปนั้นมักจะนำข้อมูลของชั้นในปัจจุบันมาใช้เท่านั้นและไม่ได้มีการนำข้อมูลที่เป็น Memory มาใช้ ดังนั้น RNN จึงเหมาะกับการฝึกสอนโมเดลเพื่อเรียนรู้ข้อมูลที่มีลักษณะเป็นแบบลำดับ (Sequential) และต่อเนื่องเป็นชุด ๆ (Series)

นอกจากนี้ RNN ยังได้ถูกนำไปพัฒนาต่อให้มีประสิทธิภาพมากขึ้น โดยมีอีก 2 สถาปัตยกรรมย่อยที่ได้รับความนิยม คือ

- Long Short-Term Memory (LSTM)[35]
- Echo State Network (ESN)[36]

4.9.3 โครงข่ายประสาทแบบคอนโวลูชัน

โครงข่ายประสาทแบบคอนโวลูชัน (Convolutional Neural Network หรือ CNN)[37] เป็นโครงข่ายประสาทเทียมที่ได้แรงบันดาลใจมาจากสิ่งมีชีวิต (Bio-inspired) โดยที่ CNN จะจำลองการมองเห็นของมนุษย์ที่มองพื้นที่เป็นที่ย่อย ๆ และนำพื้นที่ย่อย ๆ เหล่านั้นมาผนวกหรือผสานกันเพื่อดูว่าสิ่งที่มองเห็นอยู่นั้นเป็นอะไรกันแน่

ไอเดียหลักของ CNN คือการใช้ Layer ชนิดพิเศษที่เรียกว่า Convolution Layer (ถ้าแปลเป็นภาษาไทยก็จะได้ความหมายว่า ชั้นที่เกิดการม้วนขดกันหรือพับไปพับมา) ซึ่งทำหน้าที่สกัด (Extract) องค์ประกอบส่วนต่าง ๆ ของภาพออกมา เช่น เส้นขอบของวัตถุต่าง ๆ เพื่อให้โมเดลสามารถเรียนรู้ลักษณะของภาพได้อย่างมีประสิทธิภาพและแม่นยำ CNN จะใช้ Convolution Layer มาประกอบกับ Layer ชนิดอื่น เช่น Pooling Layer แล้วนำกลุ่ม Layer ดังกล่าวมาซ้อนต่อ ๆ กันโดยอาจเปลี่ยน Hyperparameter บางอย่าง เช่น ขนาดของ Filter Layer (ซึ่งเป็นส่วนหนึ่งของ Convolution Layer) และจำนวนช่อง (Channel) ของ Layer โดยวิธีการนำส่วนต่าง ๆ มาประกอบเข้าด้วยกันนี้สามารถทำได้หลายแบบ เช่น LeNet, GoogLeNet, AlexNet, VGG, ResNet, Network-in-network, SqueezeNet, Xception, MobileNets, Inception Network

4.10 การสร้างและฝึกสอนโมเดลด้วย TensorFlow

ในหัวข้อนี้จะเป็นการยกตัวอย่างประกอบโค้ดของการสร้างและฝึกสอนโมเดลในการทำนายค่าสภาพการละลายของโมเลกุล ซึ่งค่าสภาพการละลายหรือ Solubility เป็นความสามารถของสสารในการละลายในน้ำ ยิ่งสสารมีค่าการละลายสูงหมายความว่าสสารนั้นยิ่งละลายในน้ำได้ดี โดยเราจะใช้ Module Keras ซึ่งเป็นไลบรารี่ที่ถูกพัฒนาให้เป็น API หลักของ TensorFlow ในการสร้างโมเดล Neural Network ซึ่งตั้งแต่ TensorFlow ได้อัพเกรด Framework ครั้งใหญ่จากเวอร์ชัน 1 มาเป็นเวอร์ชัน 2 ก็ได้รวบ Keras เข้ามาเป็นส่วนหนึ่งของ TensorFlow เลย ซึ่งสามารถเรียกใช้งาน Keras ได้ง่าย ๆ ผ่าน `tensorflow.keras`[1]

1. ขั้นตอนแรกเราจะต้องทำการกำหนดรายละเอียดของ Neural Network

```
import numpy as np
import tensorflow as tf

# Our hidden layer
# We only need to define the output dimension - 32.
hidden_layer = tf.keras.layers.Dense(32, activation="tanh")
# Last layer - which we want to output one number
# the predicted solubility.
output_layer = tf.keras.layers.Dense(1)

# Now we put the layers into a sequential model
model = tf.keras.Sequential()
model.add(hidden_layer)
model.add(output_layer)
```

2. ทำการทดสอบโดยเรียกใช้โมเดลเพื่อแสดงข้อมูลของสสารสามตัวแรกในชุดข้อมูล

[1]ดูรายละเอียดการสร้างโมเดลแบบ Sequential ของ Keras ได้ที่ https://keras.io/guides/sequential_model

```
# Try out our model on first few datapoints
model(soldata[feature_names].values[:3])

# Output
<tf.Tensor: shape=(3, 1), dtype=float32, numpy=
array([[ 0.18162721],
       [-0.416314  ],
       [-0.32956678]], dtype=float32)>
```

3. คอมไพล์โมเดล

```
model.compile(optimizer="SGD", loss="mean_squared_error")
```

4. ฝึกสอนโมเดล

```
model.fit(train_data, epochs=50)
```

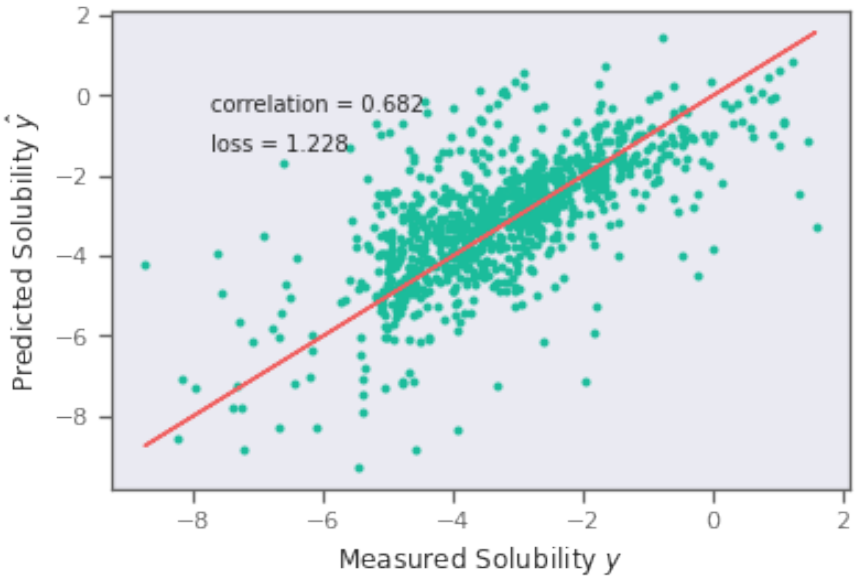
5. ทำนายค่าความสามารถในการละลายของโมเลกุล

```
yhat = np.squeeze(model.predict(test_data))
test_y = soldata["Solubility"].values[:test_N]
```

6. พล็อตกราฟเปรียบเทียบค่าสภาพการละลายที่ได้จากการทำนายและค่าอ้างอิง

```
plt.plot(test_y, yhat, ".")
plt.plot(test_y, test_y, "-")
plt.xlabel("Measured Solubility $y$")
plt.ylabel("Predicted Solubility $\hat{y}$")
plt.text(
    min(test_y) + 1,
    max(test_y) - 2,
    f"correlation = {np.corrcoef(test_y, yhat)[0,1]:.3f}",
)
plt.text(
    min(test_y) + 1,
    max(test_y) - 3,
    f"loss = {np.sqrt(np.mean((test_y - yhat)**2)):.3f}",
)
plt.show()
```

ได้กราฟดังต่อไปนี้

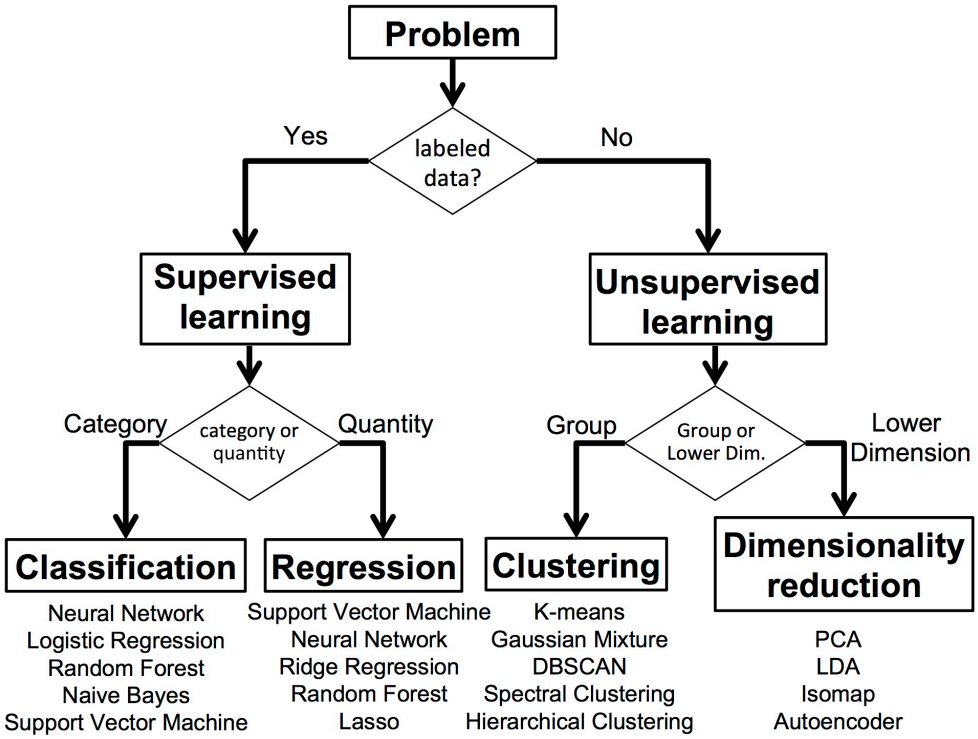


ภาพ 4.19 เปรียบเทียบค่าสภาพการละลายที่ได้จากการทำนายและค่าอ้างอิง

โดยภาพที่ 4.19 แสดงการเปรียบเทียบค่าสภาพการละลายที่ได้จากการทำนายด้วย Neural Network และค่าอ้างอิง เส้นตรงสีแดงที่ลากผ่านข้อมูลนั้นเป็นเส้นตรงที่มีความชันเท่ากับ 1

บทที่ 5

การเลือกและปรับแต่งโมเดล



ภาพ 5.1 การเลือกโมเดล ML (เครดิตภาพ: https://pythonnumericalmethods.berkeley.edu)

การทำให้โมเดลมีความสม่ำเสมอ (Regularization) เพื่อเพิ่มความถูกต้องและการเลือกโมเดล (Model Selection) เป็นสิ่งที่จำเป็นมากในขั้นตอนของการฝึกสอนโมเดล ในบทนี้เราจะมาดูรายละเอียดและแนวทางในการเลือกอัลกอริทึมสำหรับสร้างโมเดล ML รวมไปถึงเทคนิคการปรับแต่งโมเดลเพื่อให้มีประสิทธิภาพในการทำนายมากที่สุด ตัวอย่างของขั้นตอนการพิจารณาเลือกอัลกอริทึม ML นั้นแสดงตามภาพที่ 5.1 โดยเริ่มต้นจากปัญหาที่เราต้องการศึกษาก่อน แล้วก็พิจารณาว่าชุดข้อมูลของเรานั้นมี Label หรือคำตอบของแต่ละข้อมูลหรือไม่ ถ้าหากว่ามี Label เราก็สามารถใช้อัลกอริทึมแบบ Supervised ML ได้ แต่ถ้าหากไม่มี Label

เราก็ไม่มีทางเลือกอื่นนอกจากจะต้องใช้อัลกอริทึมแบบ Unsupervised ML เท่านั้น (สำหรับกรณีที่มี Label นั้นเราอาจจะใช้ Unsupervised ML ด้วยก็ได้ โดยแกล้งทำเป็นไม่สนใจ Label) เมื่อเราแบ่งประเภทของโจทย์ปัญหาได้แล้ว ขั้นตอนต่อมาคือการเลือกวิธีในการแก้ปัญหา ซึ่งขั้นตอนนี้ก็จะนำไปสู่การเลือกอัลกอริทึม ML แบบต่าง ๆ นั่นเอง โดยโจทย์ปัญหาหลัก ๆ ที่เรามักจะเจอนั้นมีด้วยกัน 4 แบบดังนี้

1. Classification (การแบ่งประเภท)
2. Regression (การถดถอย)
3. Clustering (การจัดกลุ่ม)
4. Dimensionality Reduction (การลดมิติของข้อมูล)

โดยโจทย์ปัญหาแต่ละแบบนั้นก็จะมีอัลกอริทึมที่เหมาะสมสำหรับการแก้ปัญหานั้น ๆ เช่น Ridge Regression ก็จะเหมาะสำหรับโจทย์ Regression

5.1 การเลือกโมเดล

อัลกอริทึมหรือโมเดล ML แต่ละอันนั้นก็มีข้อดีข้อเสียแตกต่างกันไป ผู้เขียนขอสรุปง่าย ๆ ดังนี้ (เน้นเฉพาะโมเดลที่ได้รับความนิยมในการใช้งาน)

5.1.1 Linear Regression

ข้อดี

- สามารถเขียนโค้ดได้ง่าย และฝึกสอนโมเดลได้อย่างมีประสิทธิภาพ
- ปัญหา Overfitting ของ Linear Regression สามารถแก้ได้ด้วยการทำ Regularization
- มีประสิทธิภาพมาก ๆ เมื่อชุดข้อมูลสามารถแยกได้เชิงเส้น (Linearly Separable)

ข้อด้อย

- ข้อมูลที่อยู่ในชุดข้อมูลที่ใช่สำหรับการฝึกสอนโมเดล Linear Regression นั้นควรจะต้องไม่ขึ้นต่อกัน แต่ทว่าในชีวิตจริงนั้นข้อมูลก็มักจะขึ้นต่อกันเสมอ
- สามารถเกิด Noise และ Overfitting ได้ง่าย
- การที่มี Outlier ในชุดข้อมูลนั้นจะส่งผลให้โมเดลมีประสิทธิภาพที่ต่ำลงมาก ๆ

5.1.2 Logistic Regression

ข้อดี

- โอกาสเกิด Overfitting น้อย แต่ว่าสามารถเกิด Overfitting ได้ในชุดข้อมูลที่มีจำนวนมิติสูง ๆ
- มีประสิทธิภาพมาก ๆ เมื่อชุดข้อมูลมี Features ที่สามารถแยกกันได้แบบเชิงเส้น
- สามารถเขียนโค้ดได้ง่าย และฝึกสอนโมเดลได้อย่างมีประสิทธิภาพ

ข้อด้อย

- ไม่ควรใช้อัลกอริทึมนี้สำหรับกรณีที่จำนวน Observation นั้นมีน้อยกว่าจำนวนของ Feature
- เหมาะสำหรับชุดข้อมูลที่มีความเป็นเชิงเส้น ซึ่งหาได้ยากในชีวิต (ปกติเรามักจะเจอชุดข้อมูลแบบที่ไม่เป็นเชิงเส้น)
- ใช้ทำนายได้แค่ฟังก์ชันที่ไม่ต่อเนื่อง (Discrete Function)

5.1.3 Support Vector Machine

ข้อดี

- เหมาะสำหรับข้อมูลที่มีจำนวนมิติเยอะ ๆ (High-dimensional Data)
- สามารถใช้กับชุดข้อมูลที่มีขนาดเล็กได้ (จำนวนข้อมูลไม่เยอะ)
- สามารถแก้ปัญหาแบบไม่เป็นเชิงเส้นได้ (Non-linear Problem)

ข้อด้อย

- ไม่ค่อยมีประสิทธิภาพเมื่อใช้กับชุดข้อมูลที่มีขนาดใหญ่
- ต้องเลือก Kernel ที่เหมาะสม ถ้าเลือก Kernel ไม่ดีก็จะได้โมเดลที่มีประสิทธิภาพต่ำ

5.1.4 Neural Network

ข้อดี

- มีคุณสมบัติที่ทำให้โมเดลสามารถทำงานต่อไปได้แม้จะเกิด Failure ขึ้น เรียกง่าย ๆ ว่าทนทานต่อความเสียหาย (Fault Tolerance)

- มีความสามารถในการเรียนรู้โมเดลที่เป็นแบบไม่เชิงเส้นและความสัมพันธ์ระหว่างตัวแปรที่มีความซับซ้อน
- สามารถ Generalize บนชุดข้อมูลที่ไม่เคยเห็นมาก่อนได้ (Unseen Data)

ข้อด้อย

- ใช้ระยะเวลาในการฝึกสอนโมเดลนาน
- ไม่การันตีว่าการฝึกสอนโมเดลจะลู่เข้า (Non-gauranteed Convergence)
- ตีความโมเดลได้ยาก เช่น เราไม่สามารถบอกความสัมพันธ์ระหว่างโหนดใน Hidden Layer ได้ ซึ่งเราเรียกว่าโมเดลแบบนี้ว่า Black Box
- ประสิทธิภาพในการฝึกสอนโมเดลขึ้นอยู่กับประสิทธิภาพของของเครื่องที่ใช้รันด้วย (Hardware)
- ไม่เหมาะสำหรับผู้เริ่มต้นศึกษา ML เพราะว่าต้องใช้ประสิทธิภาพและความสามารถในการนำไปจัดการปัญหาและปรับแก้โมเดลเพื่อทำให้เรียนรู้ได้ดียิ่งขึ้น

5.1.5 Printipal Component Analysis

ข้อดี

- สามารถลดความซับซ้อนของความสัมพันธ์ระหว่าง Feature ได้
- สามารถลดปัญหา Overfitting

ข้อด้อย

- องค์ประกอบหลัก (Principal Component) นั้นวิเคราะห์และตีความได้ยาก
- การใช้วิธีนี้ทำให้เราสูญเสียข้อมูล (ความสัมพันธ์ระหว่าง Feature) หรือ Information Loss
- จำเป็นจะต้องทำการ Standardize ชุดข้อมูลก่อน

5.1.6 K-Means Clustering

ข้อดี

- สามารถเขียนโค้ดได้ง่าย ไม่ซับซ้อน
- สามารถนำไปใช้กับชุดข้อมูลที่มีขนาดใหญ่มาก ๆ ได้
- การันตีว่าการฝึกสอนโมเดลนั้นลู่เข้าแน่นอน

- สามารถปรับให้เข้ากับชุดข้อมูลใหม่ได้ง่าย ๆ

ข้อด้อย

- ไม่เหมาะสำหรับชุดข้อมูลที่มี Outlier
- การเลือกค่า K สำหรับ Clustering นั้นค่อนข้างยุ่งยาก
- ประสิทธิภาพของโมเดลขึ้นอยู่กับพารามิเตอร์เริ่มต้น
- ความสามารถในการ Scale นั้นจะลดลงเมื่อจำนวนมิติเพิ่มขึ้น

5.1.7 K Nearest Neighbor

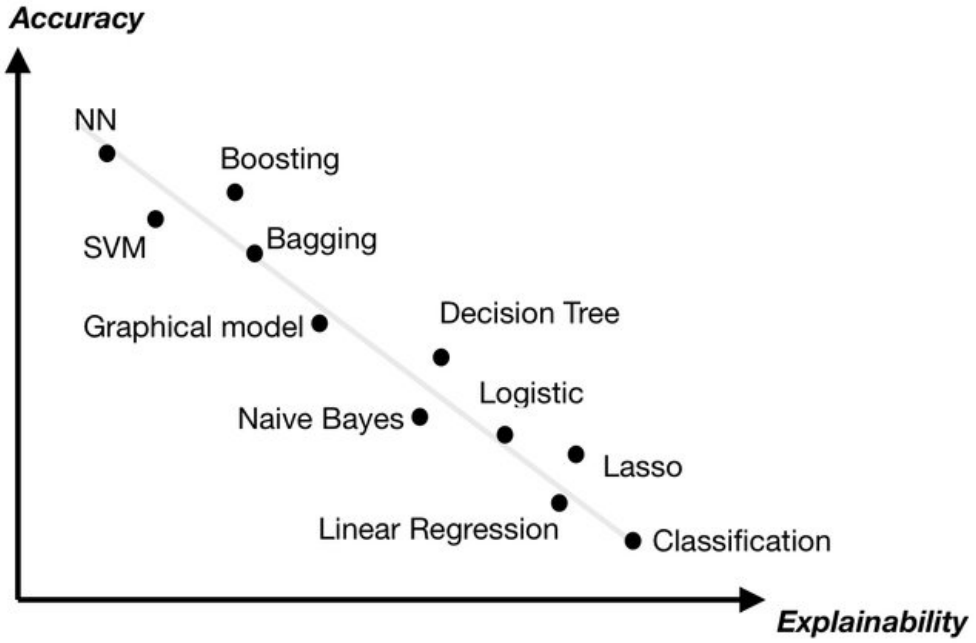
ข้อดี

- สามารถทำนายได้โดยไม่ต้องฝึกสอนโมเดล
- เป็นวิธีที่สิ้นเปลืองน้อยมาก โดยมี Time Complexity เท่ากับ $\mathcal{O}(n)$
- สามารถนำไปใช้ได้กับโจทย์ Regression และ Classification

ข้อด้อย

- ไม่เหมาะสำหรับชุดข้อมูลขนาดใหญ่
- ไม่เหมาะสำหรับชุดข้อมูลที่มี Noise เยอะมากเกินไป และข้อมูลไม่ครบ รวมไปถึง Outlier ด้วย
- จำเป็นต้องมีการทำ Feature Scaling
- การเลือกค่า K นั้นค่อนข้างยุ่งยาก

5.1.8 Machine Learning Trade-off



ภาพ 5.2 เปรียบเทียบอัลกอริทึม ML แบบต่าง ๆ ตามความสัมพันธ์ระหว่างประสิทธิภาพของโมเดลกับการตีความโมเดล

นอกจากนี้ยังมีสิ่งที่เราจะต้องให้ความสำคัญในการเลือกโมเดล ML ด้วยนั่นก็คือความถูกต้องในการทำนายหรือประสิทธิภาพของโมเดล (Accuracy) กับการที่เราสามารถตีความโมเดลนั้น ๆ ได้ (Interpretability หรือ Exaplainability) โดยจากภาพที่ 5.2 จะเห็นได้ว่าโมเดล Neural Network นั้นมีประสิทธิภาพในการทำนายที่สูงมาก (ทำนายได้ถูกต้อง) เมื่อเทียบกับโมเดล ML อื่น ๆ เช่น Linear Regression ซึ่งมีความสามารถที่น้อยกว่า แต่ถ้าหากเรามาดูที่ Explainability แล้วจะพบว่าจะตรงข้ามกับ Accuracy เลย ซึ่งโมเดล Linear Regression นั้นสามารถนำมาตีความและอธิบายถึงสิ่งกระบวนที่เกิดขึ้นในโมเดลได้ดีกว่า เช่น สามารถอธิบายความสัมพันธ์ระหว่างอินพุตและเอาต์พุตได้ดีกว่าโมเดล Neural Network ดังนั้นการที่เราจะต้องตัดสินใจเลือกใช้โมเดลสักโมเดลเพื่อมาใช้แก้ปัญหาโจทย์ของเรานั้นเราควรจะต้องพิจารณาทั้งสองปัจจัยนี้เป็นหลัก ซึ่งเราเรียกกระบวนการแบบนี้ว่าเป็นการชั่งน้ำหนักสำหรับการตัดสินใจหรือ Trade-off นั่นเอง

ถ้าหากถามว่าทำไมการตีความโมเดลได้นั้นถึงสำคัญ คำตอบก็คือเราต้องการที่จะสามารถอธิบายผลกระทบและความสำคัญของพารามิเตอร์ต่าง ๆ ของโมเดลต่อประสิทธิภาพของโมเดลโดยที่เราต้องเข้าใจและสามารถวิเคราะห์ข้อมูลได้อย่างตรงไปตรงมาด้วย โดยโมเดลที่ตีความได้ (Interpretable Model) นั้นไม่ว่าจะเป็นโมเดลอยู่ในรูปแบบของสมการ เช่น โมเดลจำพวก Regression ที่เราสามารถนำค่าสัมประสิทธิ์ของตัวแปรในสมการต่าง ๆ มาอธิบายผลกระทบที่ตัวแปรมีแก่ผลลัพธ์ที่ต้องการทำนาย หรือโมเดลจำพวกต้นไม้ตัดสินใจ (Decision Tree) ซึ่งหลังจากการฝึกฝนโมเดลแล้วนั้น เราสามารถศึกษากฎการตัดสินใจในขั้นต่าง ๆ เพื่อจำแนกผลลัพธ์ที่ต้องการได้

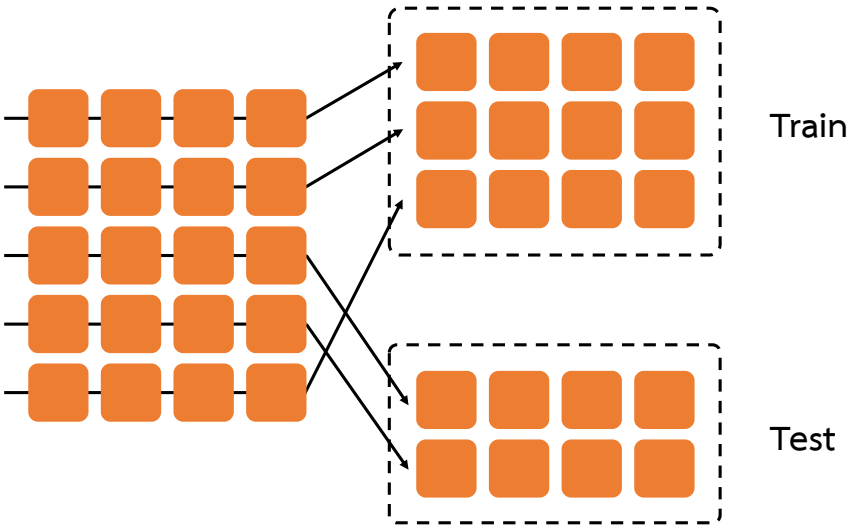
5.2 Cross Validation

วิธีการตรวจสอบโมเดลวิธีแรกนี้เป็นวิธีที่ได้รับความนิยมเป็นอย่างมากเพราะว่าสามารถทำได้ง่ายและให้ผลลัพธ์ที่น่าเชื่อถือ นั่นก็คือ “K-Fold Cross Validation” หรือเรียกสั้น ๆ ว่า Cross Validation วิธีนี้เริ่มด้วยการแบ่งข้อมูล k ให้มีขนาดของแต่ละส่วนเท่า ๆ กัน หลังจากนั้นทำเก็บข้อมูลหนึ่งส่วนไว้ใช้สำหรับเป็นตัวทดสอบโมเดลนั่นก็คือการทำ Validation แล้วทำวนไปเช่นนี้จนครบจำนวนที่แบ่งไว้ เช่น การทดสอบด้วยวิธี 5-fold Cross Validation ในรอบแรกเราจะทำการเทรนโมเดลด้วยชุดข้อมูลที่เกิดจากการวมส่วนที่ 2, 3, 4, และ 5 และทำการทดสอบด้วยข้อมูลส่วนที่ 1 และในรอบที่สองเราจะเปลี่ยนมาเทรนโมเดลด้วยข้อมูลของส่วนที่ 1, 3, 4, และ 5 แล้วนำโมเดลมาทดสอบด้วยข้อมูลส่วนที่ 2

จริง ๆ แล้วมีวิธีการทำ Cross Validation หลากหลายวิธีมาก โดยมีภาพประกอบโค้ดที่เขียนด้วยภาษา Python และใช้ไลบรารี่ Scikit-learn สำหรับ Cross Validation แต่ละแบบดังนี้

5.2.1 Train Test Split

ฟังก์ชัน `train_test_split` จะทำการแบ่ง Array หรือ Matrices ออกเป็น Train Set กับ Test Set แบบสุ่ม

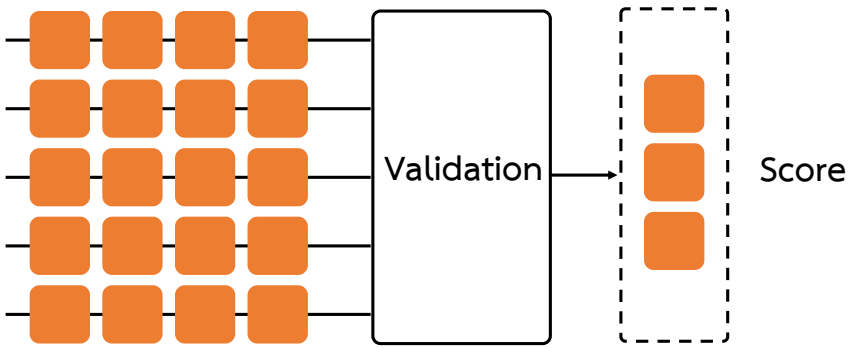


ภาพ 5.3 การทำ Cross Validation ด้วย `train_test_split`

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]])
y = np.array([0, 1, 2, 3, 4])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

5.2.2 Cross Val Score

ฟังก์ชัน `cross_val_score` ทำการคำนวณคะแนน (Score) โดยการทำ Cross Validation และแสดงค่า Score ของแต่ละส่วน

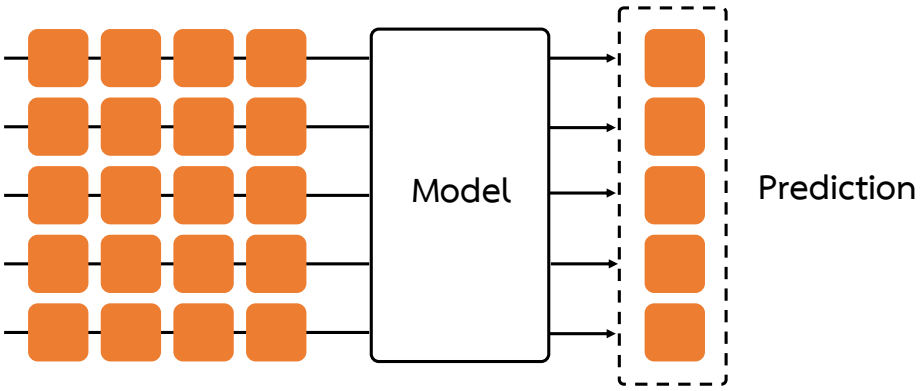


ภาพ 5.4 การทำ Cross Validation ด้วย `cross_val_score`

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris

iris = load.iris()
clf = svm.SVC(kernel="linear", C=1)
scores = cross_val_score(clf, iris.data, iris.target, cv=5)
```

5.2.3 Cross Val Predict

ฟังก์ชัน `cross_val_predict` ทำการทำนายสมาชิกหรือข้อมูลแต่ละตัวที่อยู่ใน Test Set

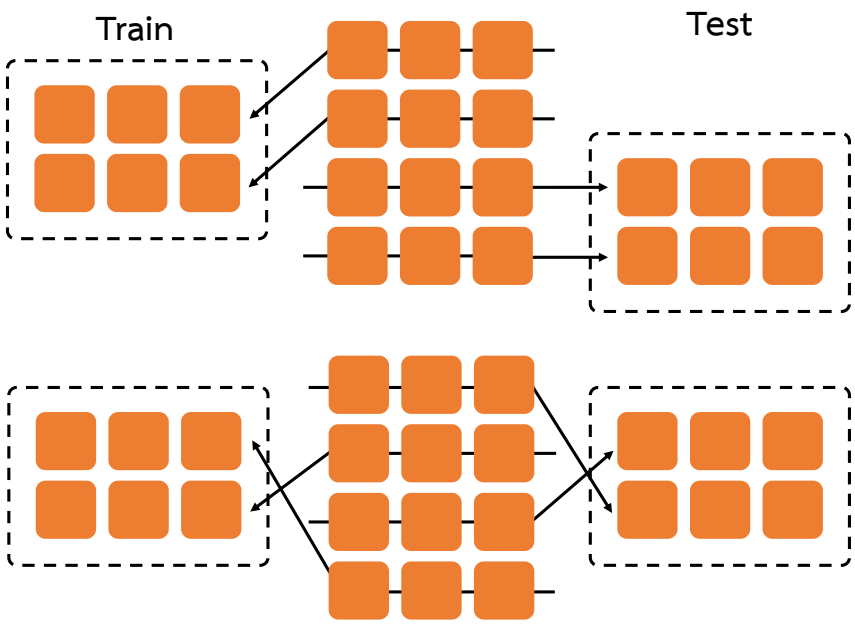


ภาพ 5.5 การทำ Cross Validation ด้วย `cross_val_predict`

```
from sklearn.model_selection import cross_val_predict
from sklearn.datasets import load_iris

iris = load.iris()
clf = svm.SVC(kernel="linear", C=1)
predicted = cross_val_predict(clf, iris.data, iris.target, cv=10)
```

5.2.4 K Fold

ฟังก์ชัน `KFold` จะทำการแบ่ง Dataset ออกเป็น K Fold (โดยที่ K คือจำนวนของการแบ่ง เช่น 3) โดยไม่มีการสลับข้อมูล โดยที่แต่ละ Fold จะถูกนำมาใช้เป็น Validation

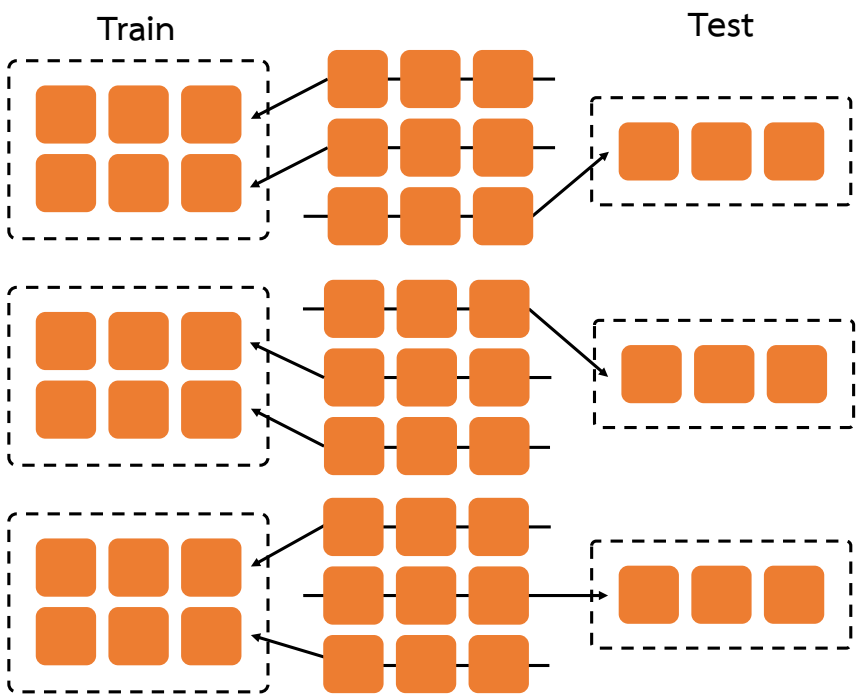


ภาพ 5.6 การทำ Cross Validation ด้วย `KFold`

```python
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
kf = KFold(n_splits=2)
kf.split(X)
kf.get_n_splits(X) # Output = 2

for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
```

5.2.5 Leave One Out

ฟังก์ชัน `LeaveOneOut` เป็นการนำข้อมูลแต่ละตัวมาใช้เป็น Test Set 1 ครั้งหรือเรียกว่า Singleton ซึ่งจริง ๆ แล้ว `LeaveOneOut()` นั้นจะเหมือนกับการใช้ `KFold(n_splits=n)` และ `LeavePOut(p=1)` โดยที่ `n` คือจำนวนของข้อมูลหรือ Sample

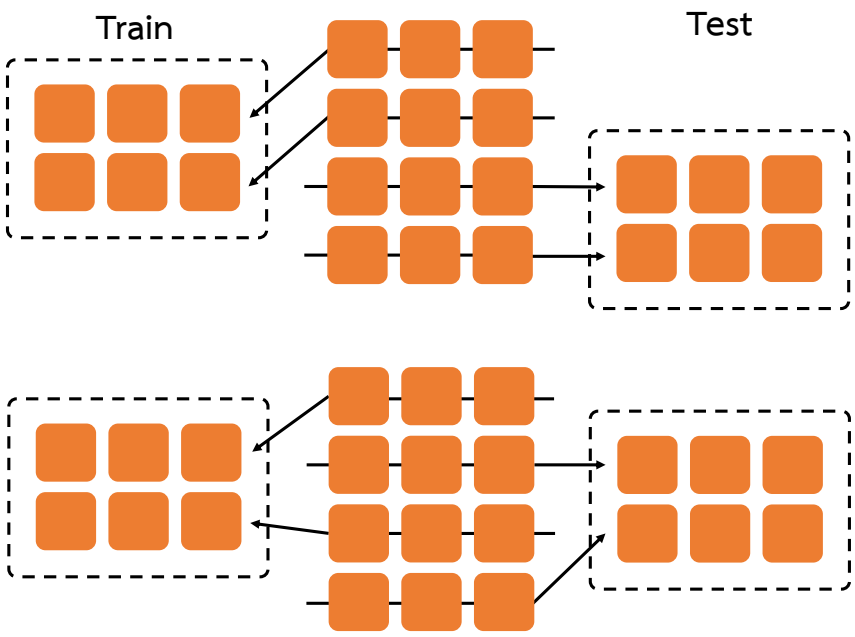


ภาพ 5.7 การทำ Cross Validation ด้วย `LeaveOneOut`

```python
import numpy as np
from sklearn.model_selection import LeaveOneOut

X = np.array([[1, 2], [3, 4]])
y = np.array([1, 2])
loo = LeaveOneOut()
loo.get_n_splits(X) # Output = 2

for i, (train_index, test_index) in enumerate(loo.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
```

5.2.6 Leave P Out

ฟังก์ชัน `LeavePOut` นั้นคล้ายกับ `LeaveOneOut()` มาก แต่จะมีความแตกต่างกันตรงที่วิธีนี้นั้นสามารถกำหนดจำนวนของข้อมูลหรือ Sample ที่เรานำไปไปใช้เป็น Test Set ได้



ภาพ 5.8 การทำ Cross Validation ด้วย `LeavePOut`

```python
import numpy as np
from sklearn.model_selection import LeavePOut

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 3, 4])
lpo = LeavePOut(2)
lpo.get_n_splits(X) # Output = 6

for i, (train_index, test_index) in enumerate(lpo.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")
```

5.3 การคัดเลือกลักษณะเฉพาะ

การคัดเลือกลักษณะเฉพาะ (Feature Selection) เป็นการหา Feature ที่เหมาะสมที่สุดสำหรับการใช้อธิบายข้อมูลของโมเลกุล โดยเราจะทำการเรียงลำดับความสำคัญของ Feature แล้วทำการคัดเลือกเฉพาะ Feature ที่คิดว่าสอดคล้องกับเอาต์พุตที่ต้องการทำนายและคัด Feature ที่มีความสำคัญน้อยออกไปเพื่อหลีก

Algorithm 5.1 อัลกอริทึม Forward Search สำหรับการทำ Feature Selection

Initialize $\mathcal{F} = \emptyset$.
repeat
  for $i = 1, \ldots, d$ **do**
    if $i \notin \mathcal{F}$ **then**
      $\mathcal{F}_i = \mathcal{F} \cup \{i\}$
      Use some version of cross validation to evaluate features $\mathcal{F}_i$.
      (i.e., train your learning algorithm using only the features in $\mathcal{F}_i$, and estimate its generalization error.)
    end if
  end for
  Set $\mathcal{F}$ to be the best feature subset found in the previous step.
until convergence
Select and output the best feature subset that was evaluated during the entire search procedure.

เลี่ยง Bias ที่อาจจะเกิดขึ้น อธิบายง่าย ๆ คือเป็นเทคนิคที่เรานำมาใช้เพื่อลดจำนวณของ Feature นั่นเอง

อัลกอริทึมของ Feature Selection แบบที่ง่ายที่สุดนั้นชื่อว่า Forward Search ซึ่งดูได้ตามอัลกอริทึมที่ 5.1 โดยเริ่มต้นนั้นกำหนดให้ $\mathcal{F}$ เป็นเซตของจำนวน Feature ทั้งหมดซึ่งยังเป็นเซตว่างอยู่ แล้วเราก็ทำการ Cross Validation ไปทีละ Feature โดยในลูปด้านในนั้นจะเพิ่ม Feature เข้าไปใน $\mathcal{F}$ ทีละอันจนกระทั่งครบทุก Feature $\mathcal{F} = \{1, \ldots, d\}$ ซึ่งจะเป็นการสิ้นสุดกระบวนการทำ Feature Search

นอกจากนี้ยังมีอัลกอริทึมที่ตรงข้ามกับ Forward Search เรียกว่า Backward Search โดยแทนที่เราจะกำหนด $\mathcal{F}$ ให้เป็นเซตว่างนั้นเราจะเริ่มด้วย $\mathcal{F}$ ที่เป็นเซตที่มี Feature อยู่ครบทั้งหมดแล้วทำการลบ Feature ออกทีละอันจนกระทั่ง $\mathcal{F}$ เป็นเซตว่าง

5.4 ปัญหา Bias-Variance

หนึ่งในปัญหาที่เราทุกคนจะต้องเจอในการสร้างโมเดลนั่นก็คือ Bias-Variance Problem ซึ่งนำไปสู่ปัญหาเรื่อง Overfitting ต่อไป เราลองมาดูรายละเอียดกันครับ กำหนดให้โมเดลของเราแทนด้วย $\hat{f}(\vec{x})$ และค่าอ้างอิงหรือคำตอบที่เราจะมาเทียบกับการทำนายเป็น y และความคลาดเคลื่อนที่เกิดขึ้นเป็น

$$E\left[\left(y - \hat{f}(\vec{x})\right)^2\right] \tag{5.1}$$

ซึ่งจริง ๆ แล้ว เป็นฟังก์ชันที่สมบูรณ์แบบมาก แต่ทว่าในความเป็นจริงแล้วในชุดข้อมูลของเรานั้นย่อมมี

Noise (ϵ) ซึ่งค่าความแตกต่างระหว่างโมเดลของเรากับคำตอบก็จะมีการปนเปื้อนหรือ Contaminate โดย ϵ ดังนี้

$$y = f(\vec{x}) + \epsilon \tag{5.2}$$

จึงทำให้ค่าความคลาดเคลื่อนที่เกิดขึ้นจริง ๆ นั้นมีสมการดังต่อไปนี้

$$E\left[\left(y - \hat{f}(\vec{x})\right)^2\right] = E\left[y^2\right] + E\left[\hat{f}(\vec{x})^2\right] - 2E\left[y\hat{f}(\vec{x})\right] \tag{5.3}$$

$$= E\left[\left(f(\vec{x}) - \epsilon\right)^2\right] + \hat{f}(\vec{x})^2 - 2E\left[\left(f(\vec{x}) - \epsilon\right)\right]\hat{f}(\vec{x}) \tag{5.4}$$

ซึ่งถ้าหากเราทำการพิสูจน์สมการด้านบนโดยพยายามจัดรูปให้อยู่ในเทอมที่มี Bias และ Variance จากชุดข้อมูล เราจะได้สมการดังต่อไปนี้

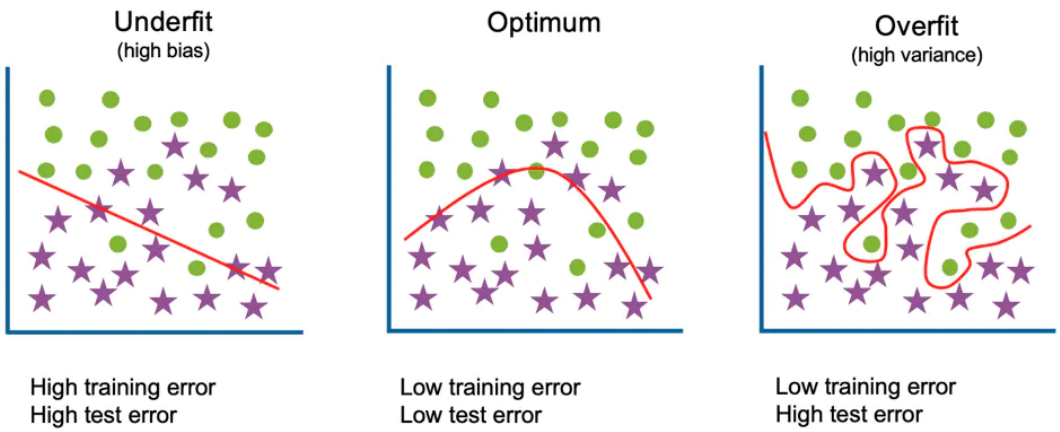
$$E\left[\left(y - \hat{f}(\vec{x})\right)^2\right] = \underbrace{E\left[f(\vec{x}) - \hat{f}\left(\vec{x}; \boldsymbol{D}\right)\right]^2}_{\text{Bias}} + \underbrace{E\left[\left(E\left[\hat{f}\left(\vec{x}; \boldsymbol{D}\right)\right] - \hat{f}\left(\vec{x}; \boldsymbol{D}\right)\right)^2\right]}_{\text{Variance}} + \sigma^2 \tag{5.5}$$

โดยพจน์แรกนั้นเป็น Bias, พจน์ที่สองเป็น Variance และพจน์ที่สามเป็นค่าความแปรปรวนที่คำนวณจาก Standard Deviation ของ Noise (ϵ) ประเด็นก็คือว่าเราสามารถควบคุม Bias กับ Variance ได้ แต่เราไม่สามารถควบคุม Noise ได้เพราะมันเป็นสิ่งที่ผูกติดมากับชุดข้อมูล ซึ่งการที่เรามี Bias และ Variance ที่ไม่สมดุลกันนั้นจะทำให้เกิดผลลัพธ์ที่ตามมาในระหว่างการฝึกสอนโมเดล นั่นคือ Overfitting และ Underfitting

5.5 การเพิ่มประสิทธิภาพการเรียนรู้และแก้ปัญหา Overfitting

Overfitting

โมเดลตอบสนองต่อ Noise ที่มากเกินไป ทำให้เกิดการเรียนรู้และจดจำ Noise และไม่สามารถที่จะเรียนรู้รายละเอียดจริง ๆ ของข้อมูลได้ ซึ่งส่งผลให้ทำนายข้อมูลไม่ได้หรือผิดพลาดมากกว่าที่คาดไว้หรือยอมรับได้ โดยกรณีนี้โมเดลจะมีค่าความแปรปรวนของข้อมูลสูง (High Variance)



ภาพ 5.9 โมเดลที่มีความ Overfitting และ Underfitting กับชุดข้อมูลมากเกินไป

Underfitting

โมเดลของเราไม่สามารถหาความสัมพันธ์ระหว่างอินพุต (x) กับเอาต์พุต (y) ได้เพราะว่ามีข้อมูลที่ใช้ในการเทรนน้อยเกินไปหรือดึงข้อมูลออกมาจาก Training Set ได้ไม่เพียงพอที่จะเรียนรู้ โดยในกรณีนี้โมเดลจะมีค่าความเอนเอียงสูง (High Bias)

Noisy

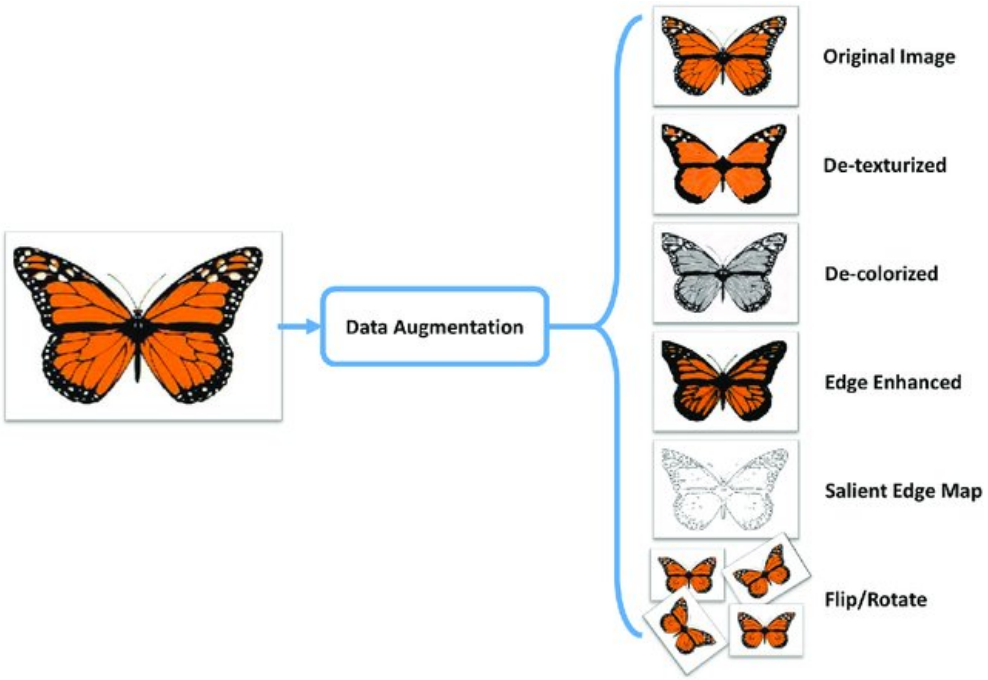
โมเดลไม่มี Overfitting และ Underfitting แต่ยังมีค่า Error ของการฝึกสอนที่ยังสูงอยู่มาก ซึ่งสาเหตุก็อาจจะมาจากการที่ชุดข้อมูลมี Noise มากเกินไปนั่นเอง ซึ่งเราแปล Noise ได้ตรงตัวเลยก็คือเป็นสิ่งที่รบกวนโมเดลของเรานั่นเอง

ภาพที่ 5.9 แสดงการเปรียบเทียบระหว่างกรณีของ Underfitting และ Overfitting ซึ่งเป็นหนึ่งในปัญหาหลักที่มักจะพบเจอได้ทั่วไปใน ML โดยเราสามารถสรุปความสัมพันธ์จากกรณีได้กล่าวได้ดังนี้

$$\begin{aligned} \text{Bias สูง} &\longleftrightarrow \text{Underfitting} \\ \text{Variance สูง} &\longleftrightarrow \text{Overfitting} \\ \sigma^2 \text{ มาก} &\longleftrightarrow \text{Noisy Data} \end{aligned}$$

วิธีการจัดการกับ Overfitting แบบที่ง่ายที่สุดคือการเพิ่มจำนวนข้อมูลในการฝึกสอนโมเดล นอกจากนี้ยังมีวิธีอื่น ๆ ที่เราสามารถใช้ในการจัดการกับปัญหาข้างต้นได้เช่นเดียวกัน มีดังต่อไปนี้

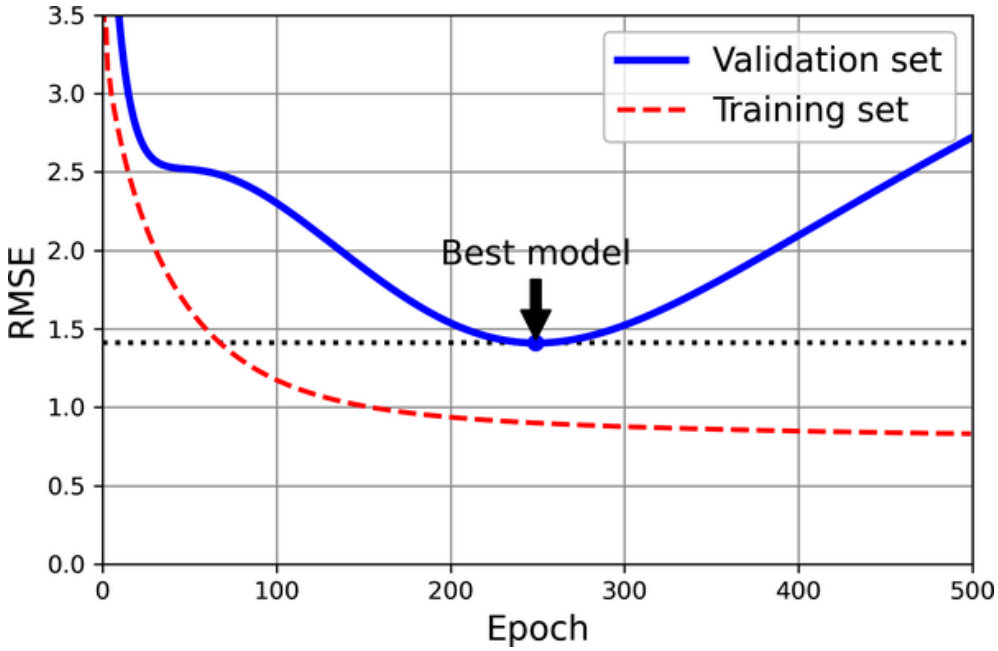
5.5.1 Data Augmentation



ภาพ 5.10 ตัวอย่างการทำ Data Augmentation สำหรับข้อมูลที่เป็นรูปภาพ เช่น การเปลี่ยนสี การเพิ่มความคมชัด การเพิ่ม Noise (เครดิตภาพ: *PLoS ONE* 12(8): e0183838)

วิธีการทำ Data Augmentation นั้นจะตรงข้ามกับการทำความสะอาดข้อมูล (Data Cleaning) นั่นก็คือจะเป็นวิธีที่เราจะใส่ Noise หรือสิ่งที่ไม่ได้เกี่ยวข้องกับข้อมูลโดยตรงเข้าไปในชุดการฝึกสอน รวมไปถึงการแก้ไขข้อมูลให้แตกต่างไปจากเดิม แต่ยังคงไว้ซึ่งลักษณะของข้อมูลนั้น ซึ่งการทำ Data Augmentation จะเป็นการช่วยไม่ให้เกิดการเรียนรู้ที่มันยึดติดกับชุดข้อมูลฝึกสอนมากเกินไป ในปัจจุบันวิธีการนี้ได้รับความนิยมเพราะสามารถทำได้ง่าย สะดวก และไม่มีความซับซ้อนในการทำ โดยมีความจำเป็นอย่างยิ่งกรณีที่ชุดข้อมูลมีขนาดเล็ก (จำนวนข้อมูลไม่เยอะ) แต่ต้องการนำมาใช้ในการฝึกสอนด้วยเทคนิค ML ที่ต้องการข้อมูลในปริมาณที่เยอะในการฝึกสอน เช่น Deep Learning[38]

5.5.2 Early Stopping



ภาพ 5.11 การทำ Regularization ด้วยวิธี Early Stopping สำหรับการฝึกสอนโมเดล High-degree Polynomial Regression โดยใช้ Batch Gradient Descent และใช้ RMSE ในการวัดค่าความคลาดเคลื่อน (เครดิตภาพ: https://www.oreilly.com)

วิธี Early Stopping มีความหมายวิธีการทำงานตามชื่อเลยนั่นก็คือหยุดให้เร็วขึ้น เป็นวิธีการที่เราจะกำหนด (บังคับ) ให้การฝึกสอนหรือ Training นั้นหยุดก่อนที่โมเดลของเราจะเริ่มเรียนรู้ Noise ที่อยู่ภายในชุดข้อมูล แทนที่จะเรียนรู้เฉพาะชุดข้อมูลอย่างเดียว ซึ่งวิธีการนี้จะเป็นการป้องกันการเปิด Bias แบบตรงไปตรงมา อย่างไรก็ตามเราควรจะต้องระมัดระวังในการใช้เทคนิค Early Stopping เพราะว่าถ้าเราบังคับให้โมเดลหยุดเรียนรู้เร็วเกินไป ปัญหาที่อาจจะเกิดขึ้นแทนการ Overfitting นั่นก็คือการ Underfitting ของโมเดล ซึ่งการเลือกจุดที่จะให้โมเดลนั้นหยุดการเรียนรู้ก็ถือว่ามีความเป็น Art อย่างหนึ่ง ซึ่งจุดที่เราเลือกต้องมีความเหมาะสมระหว่าง Overfitting และ Underfitting

โค้ดของการทำ Early Stopping โดยใช้ไลบรารี่ Scikit-Learn

```
from copy import deepcopy
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

# Split the quadratic dataset
X_train, y_train, X_valid, y_valid = [...]

preprocessing = make_pipeline(
```

```
    PolynomialFeatures(degree=90, include_bias=False),
    StandardScaler()
    )
X_train_prep = preprocessing.fit_transform(X_train)
X_valid_prep = preprocessing.transform(X_valid)

sgd_reg = SGDRegressor(penalty=None, eta0=0.002, random_state=42)
n_epochs = 500
best_valid_rmse = float('inf')

# Training with applying early stopping
for epoch in range(n_epochs):
    sgd_reg.partial_fit(X_train_prep, y_train)
    y_valid_predict = sgd_reg.predict(X_valid_prep)
    val_error = mean_squared_error(
        y_valid,
        y_valid_predict,
        squared=False
        )
    if val_error < best_valid_rmse:
        best_valid_rmse = val_error
        best_model = deepcopy(sgd_reg)
```

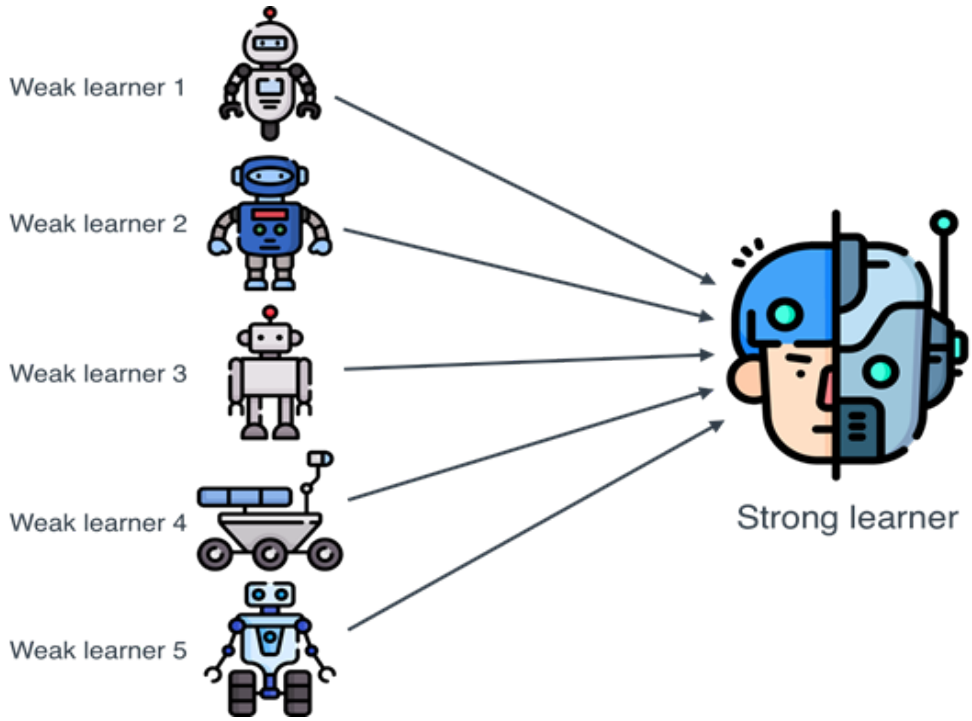
โค้ดของการสร้าง Callback ของ Early Stopping โดยใช้ไลบรารี่ TensorFlow

```
import tensorflow as tf

callback = tf.keras.callbacks.EarlyStopping(
    monitor='loss',
    patience=3
    )

model = tf.keras.models.Sequential(
    [tf.keras.layers.Dense(10)]
    )
model.compile(tf.keras.optimizers.SGD(), loss='mse')

history = model.fit(
    np.arange(100).reshape(5, 20),
    np.zeros(5), epochs=10, batch_size=1,
    callbacks=[callback], verbose=0
    )

```

```
19 len(history.history['loss'])  # Number of epochs
20 # Output
21 4
```

โดย Callback จะทำการหยุดการฝึกสอน (Training) เมื่อค่า Loss ไม่มีการลดลงภายใน 3 Epochs ที่ต่อเนื่องกัน

5.5.3 Ensemble Method



ภาพ 5.12 การทำงานร่วมกันของโมเดลหลาย ๆ โมเดลโดยใช้วิธี Ensemble (เครดิตภาพ: https://www.manning.com)

เทคนิคนี้เป็นการนำโมเดลหลาย ๆ โมเดลมารวมกันเพื่อที่จะทำให้ผลลัพธ์ของการทำนายคำตอบมีค่าที่ดีที่สุด โดยโมเดล ML ที่เราจะมานำผสมกันนั้นจะเป็นอะไรก็ได้ เช่น Linear Regression, Logistic Regression, Gaussian Process Regression ผู้อ่านสามารถดูภาพที่ 5.12 ประกอบได้ โดยจะเห็นว่าเรามีโมเดลที่มีประสิทธิภาพไม่ค่อยดีนักหลาย ๆ โมเดล เราสามารถนำโมเดลเหล่านี้มารวมกันเพื่อให้ได้โมเดลที่มีประสิทธิมากขึ้นได้

โดยเทคนิคย่อยของEnsemble Method ที่นิยมใช้กันนั้นมีอยู่ด้วยกัน 3 วิธี ดังนี้

- **Bagging** เราจะทำการสร้างข้อมูลประเภทเดียวกันแบบหลาย ๆ ชุด แล้วทำการทดสอบกับข้อมูลเพียงแค่บางส่วน (Subset) ของชุดข้อมูล จากนั้นนำผลการทำนายของโมเดลต่าง ๆ มารวมกัน ตัวอย่างขอ

งอัลกอริทึมที่ใช้ในการเรียนรู้สำหรับเทคนิค Bagging นี้ เช่น Decision Tree, Random Forest และ Extra Tree

- **Boosting** จะทำคล้ายกับ Bagging เลยก็คือเริ่มต้นด้วยการสร้างข้อมูลประเภทเดียวกันแบบหลาย ๆ ชุด แล้วทำการทดสอบกับข้อมูลชุดเดียวกันโดยทำการทดสอบแบบวนซ้ำ (Iteration) แล้วปรับค่าน้ำหนักเพื่อทำให้ผลการทำนายของโมเดลนั้นดีขึ้นเรื่อย ๆ ซึ่งวิธีนี้ค่อนข้างเป็นที่นิยมเพราะมีความยืดหยุ่นและใช้ได้กับทุกอัลกอริทึม นอกจากนี้ยังสามารถปรับลดค่าความคลาดเคลื่อนของ Bias ของโมเดลได้ดีอีกด้วย ตัวอย่างของอัลกอริทึมที่ใช้ในการเรียนรู้สำหรับเทคนิค Boosting นี้ เช่น AdaBoost และ Stochastic Gradient Boosting

- **Voting** เราจะเริ่มด้วยการสร้างโมเดลที่แตกต่างกันหลาย ๆ โมเดล เช่น Decision Tree, Support Vector Machine, K-Nearest Neighbors จากนั้นทำการฝึกสอนโมเดลด้วยชุดข้อมูลชุดเดียวกันเพื่อดูผลการทำนายที่ดีที่สุดของแต่ละโมเดล แล้วใช้การโหวตผลที่เหมือนกันหรือคล้ายกันเพื่อเป็นคำตอบสุดท้าย

5.5.4 Dropout

วิธีการ Dropout เป็นเทคนิคพิเศษที่ถูกคิดค้นขึ้นมาเพื่อแก้ปัญหา Overfitting ใน Deep Learning โดยเฉพาะ ซึ่งไอเดียของเทคนิคนี้ก็คือเราจะทำการตัด (Drop out หรือเอาออกไป) หน่วยการเรียนรู้ (Learning Unit หรือ Neuron) ใน Neural Network ออกไป ซึ่งจะเป็นการช่วยให้โมเดลของเราลด Bias ที่เกิดจากการเรียนรู้ของข้อมูลที่มากเกินไป โดยจำนวนของ Neuron ที่จะตัดออกไปนั้นส่วนใหญ่แล้วจะคิดเป็นเปอร์เซนต์ของ Neuron ทั้งหมด เช่น ตัดออกไป 5 เปอร์เซนต์

โค้ดของการทำ Dropout โดยใช้ไลบรารี่ TensorFlow

```
>>> tf.random.set_seed(0)
>>> layer = tf.keras.layers.Dropout(.2, input_shape=(2,))
>>> data = np.arange(10).reshape(5, 2).astype(np.float32)
>>> print(data)
[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
>>> outputs = layer(data, training=True)
>>> print(outputs)
tf.Tensor(
[[ 0.    1.25]
 [ 2.5   3.75]
 [ 5.    6.25]
 [ 7.5   8.75]
```

```
17  [10.    0.  ]], shape=(5, 2), dtype=float32)
```

5.5.5 L1 Regularization

ตามที่เราได้ศึกษาเรื่อง L1 กันไปแล้วในบทที่ 3 เราสามารถทำการปรับปรุง Loss Function ของเราได้ด้วยการเพิ่มพารามิเตอร์แบบพิเศษเข้าไป นั่นก็คือการใส่การลงโทษหรือ Penalty ให้กับการเรียนรู้ของโมเดล โดยการปรับพารามิเตอร์ λ (ในบทที่ 3 จะใช้ตัวแปร α ซึ่งมีความหมายเหมือนกัน) ให้เพิ่มขึ้นนั้นจะเป็นการลด Variance แต่ในขณะเดียวกันก็จะเป็นการเพิ่ม Bias โดยใน Linear Regression นั้นเราจะเรียก Regularization แบบ L1 ว่า LASSO

$$L = \frac{1}{N}\sum_{i}^{N}\Big[y_i - \hat{f}(\vec{x}_i, \vec{w}, b)\Big]^2 + \lambda\sum_{k}|w_k| \tag{5.6}$$

5.5.6 L2 Regularization

สำหรับ Regularization แบบ L2 นั้นก็จะมีความคล้ายกับ L1 มาก ซึ่งวิธีนี้ในการทำ Linear Regression จะมีชื่อเรียกว่า Ridge Regression

$$L = \frac{1}{N}\sum_{i}^{N}\Big[y_i - \hat{f}(\vec{x}_i, \vec{w}, b)\Big]^2 + \lambda\sum_{k}w_k^2 \tag{5.7}$$

สำหรับการเลือก Regularization นั้น ผู้เขียนขอยกประโยคของศาสตราจารย์ Frank Harrell ที่ได้แนะนำการเลือก L1 และ L2 ไว้ดังนี้[1]

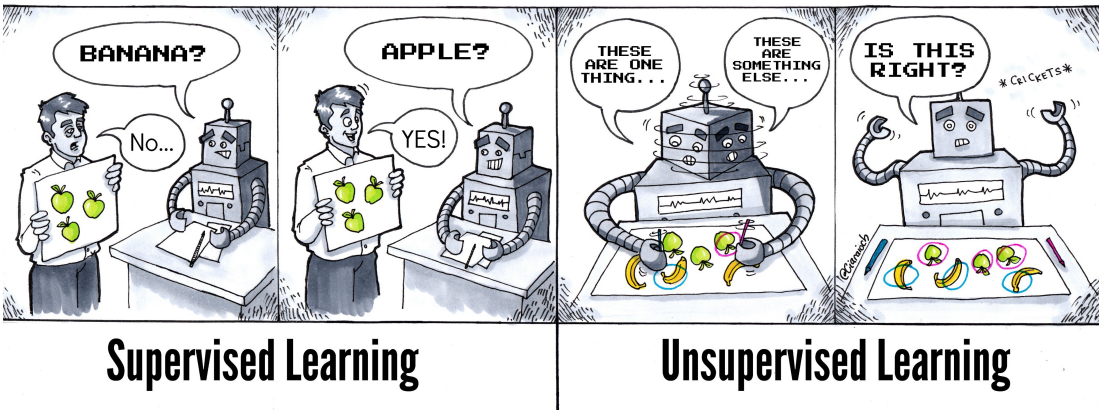
> "Generally speaking if you want optimum prediction use L2. If you want parsimony at some sacrifice of predictive discrimination use L1. But note that the parsimony can be illusory, e.g., repeating the *lasso* process using the bootstrap will often reveal significant instability in the list of features 'selected' especially when predictors are correlated with each other."

ซึ่งตีความได้คร่าว ๆ ว่าถ้าหากต้องการการทำนายที่เหมาะสมที่สุดให้ใช้ L2 หรือ Ridge Regression แต่ถ้าหากต้องการทำให้การจำแนกเชิงพยากรณ์ (Predictive Discrimination) มีความสม่ำเสมอกันสำหรับทุก ๆ Feature ให้ใช้ L1 หรือ Lasso Regression แต่ควรเข้าใจไว้ด้วยว่าการใช้ L1 สามารถทำให้เกิดปัญหาได้ เช่น การทำ Lasso Regression โดยใช้เทคนิค Bootstrap (การ Sample ตัวอย่างจากชุดข้อมูล)

[1] อ้างอิง https://stats.stackexchange.com/a/184022/283188

บทที่ 6

การเรียนรู้แบบไม่มีผู้สอน



ภาพ 6.1 การเปรียบเทียบระหว่างการเรียนรู้แบบมีผู้สอน (Supervised Learning) และการเรียนรู้แบบไม่มีผู้สอน (Unsupervised Learning) ของเครื่องจักร (เครดิตภาพ: https://twitter.com/Ciaraioch)

การเรียนรู้แบบไม่มีผู้สอน (Unsupervised Learning) เป็นเทคนิคที่อาจจะเรียกว่าได้ตรงข้ามกับการเรียนรู้แบบมีผู้สอน (Supervised Learning) ก็ได้ เพราะว่าเทคนิคประเภทนี้จะเป็นการฝึกสอนโมเดลแบบไม่มีการบอกคำตอบหรือเอาต์พุตให้โมเดลได้รับรู้ ดังนั้นสิ่งที่โมเดลมีอย่างเดียวก็คืออินพุต ซึ่งสิ่งเดียวที่โมเดลจะทำได้นั่นก็คือการเรียนรู้หาความสัมพันธ์ (Relation) ระหว่างข้อมูลแต่ละตัวภายในชุดข้อมูลที่เราได้ป้อนเข้าไป การเรียนรู้ประเภทนี้จะพิจารณาข้อมูลเป็นเซตของตัวแปรสุ่ม แล้วจึงสร้างโมเดลความหนาแน่นร่วมของชุดข้อมูล การเรียนรู้แบบไม่มีผู้สอนสามารถนำไปใช้ร่วมกับการทฤษฎีของเบย์ (Bayes' theorem) เพื่อหาความน่าจะเป็นแบบมีเงื่อนไขของตัวแปรสุ่มโดยกำหนดตัวแปรที่เกี่ยวข้องให้ นอกจากนี้ยังสามารถนำไปใช้ในการบีบอัดข้อมูล (Data Compression) ซึ่งขั้นตอนวิธีการบีบอัดข้อมูลจะขึ้นอยู่กับการแจกแจงความน่าจะเป็นของข้อมูล

จริง ๆ แล้วการเรียนรู้แบบไม่มีผู้สอนนั้นมีเทคนิคย่อยต่าง ๆ มากมาย เพื่อไม่ให้สับสน ในบทนี้ผู้เขียน

จะขอจัดกลุ่มอัลกอริทึม Unsupervised ML ออกเป็น 3 กลุ่ม ดังนี้

- การวิเคราะห์การจัดกลุ่ม Clustering Analysis
- การลดจำนวนมิติของข้อมูลแบบเชิงเส้น (Linear Dimensionality Reduction)
- การลดจำนวนมิติของข้อมูลแบบไม่เชิงเส้น (Nonlinear Dimensionality Reduction)

6.1 วิธีการจัดกลุ่ม

เทคนิคการวิเคราะห์การจัดกลุ่ม (Clustering Analysis) หรือจะเรียกสั้น ๆ ว่า Clustering ก็ได้ มีเทคนิคย่อยอีกมากมายที่ถูกนำมาใช้อย่างแพร่หลายในการวิเคราะห์ข้อมูลทางเคมี นั่นก็เพราะว่าข้อมูลทางเคมีนั้นมีปริมาณที่เยอะมากและข้อมูลส่วนใหญ่นั้นก็ไม่มี Label ที่แน่นอน ดังนั้นการที่เราจะศึกษาทำความเข้าใจถึงความสัมพันธ์ระหว่างข้อมูลได้นั้นเราจึงจำเป็นที่จะต้องนำเทคนิคการจัดกลุ่มในลักษณะนี้เข้ามาช่วย

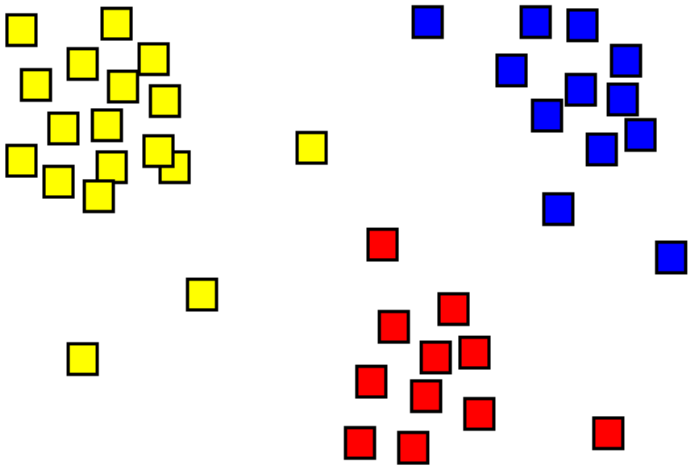
นอกจากนี้ สิ่งที่หลายคนมักจะสับสนและเข้าใจผิดก็คือ Clustering กับ Classification นั้นเป็นสิ่งเดียวกัน ซึ่งจริง ๆ แล้วไม่ใช่ โดยทั้งสองวิธีนี้มีความแตกต่างกันดังนี้

Clustering ชุดข้อมูลไม่มีการถูกแบ่งกลุ่มมาก่อน (Unlabelled) มีเพียงแค่ข้อมูลมาให้ โดย Clustering ยังสามารถแบ่งออกได้เป็นสองแบบ ดังนี้

- Hard Clustering เป็นการกำหนดให้แต่ละชุดข้อมูลแบ่งออกเป็นกลุ่มที่แยกออกจากกันโดยสิ้นเชิง
- Soft Clustering เป็นการที่ข้อมูลมีโอกาสที่จะอยู่ในหลาย ๆ กลุ่มได้ ขึ้นอยู่กับความน่าจะเป็นของตัวข้อมูล

Classification ชุดข้อมูลนั้นมีการแบ่งกลุ่มมาแล้ว (Labelled) โดยรู้จำนวนของกลุ่มและรู้ว่ามีการแบ่งกลุ่มอย่างไร

เทคนิค Clustering นั้นเป็นการจัดกลุ่มข้อมูลที่ไม่เคยมีการจัดกลุ่มมาก่อน กล่าวคือถ้าหากเรามีชุดข้อมูลที่ไม่เคยถูกวิเคราะห์ก่อนซึ่งข้อมูลเหล่านั้นก็ผสมกันแบบมั่ว ๆ หรือไม่มีรูปแบบ เราสามารถแบ่งกลุ่มข้อมูลได้โดยพิจารณาจากลักษณะที่คล้ายกันของข้อมูล โดยจะนำข้อมูลที่มีลักษณะคล้ายกันมาอยู่กลุ่มเดียวกัน ส่วนข้อมูลที่มีลักษณะต่างออกไปก็ให้ไปอยู่อีกกลุ่มหนึ่ง การนำเทคนิคนี้ไปใช้คือจะไม่ใช่การหาผลลัพธ์ที่ต้องการวัดค่าความแม่นยำ แต่จะเป็นการหาความสัมพันธ์ของข้อมูลอีกรูปแบบหนึ่ง เช่น การจัดกลุ่มข้อมูลของสารประกอบเคมีอินทรีย์โดยใช้ความว่องไวในการเข้าทำปฏิกิริยา การจัดกลุ่มหมู่ฟังก์ชันจากประเภทของการเข้าทำปฏิกิริยา



ภาพ 6.2 ตัวอย่างของชุดข้อมูลที่ถูกจัดกลุ่มโดยแบ่งตามสี

6.1.1 การจัดกลุ่มแบบโครงสร้างลำดับชั้น

การจัดกลุ่มแบบโครงสร้างลำดับชั้นจะใช้ลักษณะของต้นไม้ในการแทนความเชื่อมโยงของข้อมูล เป็นเทคนิคที่เหมาะสำหรับข้อมูลที่มีลำดับชั้น เช่น อนุกรมวิธาน (Taxonomy) การแบ่งกลุ่มลักษณะนี้มี 2 ประเภท คือ ล่างขึ้นบน (Agglomerative) และ บนลงล่าง (Divisive) ดังนี้

- Agglomerative ในขั้นตอนแรกข้อมูลแต่ละตัวในชุดข้อมูลนั้นนับเป็นหนึ่งกลุ่ม หลังจากนั้นทำการคำนวณหาค่าความใกล้ชิดของกลุ่มที่อยู่ใกล้กัน ซึ่งกลุ่มที่อยู่ใกล้กันก็จะถูกรวบกันเป็นหนึ่งกลุ่ม โดยวนทำเช่นนี้ไปเรื่อย ๆ จนกว่าจะได้กลุ่มข้อมูลเดียวในที่สุด

- Divisive เทคนิคนี้จะดำเนินการตรงกันข้ามกับ Agglomerative ในขั้นตอนแรกเริ่มจากกลุ่มใหญ่กลุ่มเดียว และแยกกลุ่มที่ไม่เหมือนกันออกไปเรื่อย ๆ จนกระทั่งได้จำนวนกลุ่มเท่ากับจำนวนข้อมูลหรือจนกว่าจะแยกต่อไม่ได้

นอกจากนี้ยังมีเทคนิคการจัดกลุ่มข้อมูลแบบอื่นอีกที่ไม่ได้กล่าวถึง เช่น Centroid-based Clustering ใช้จุดกึ่งกลางของกลุ่ม, Density-based Clustering ใช้ความหนาแน่นหรือความแออัดของข้อมูล และ Distribution-based Clustering ใช้รูปแบบการแจกแจงของข้อมูล

6.1.2 วิธีรวบกลุ่มจับคู่แบบถ่วงน้ำหนักโดยใช้ค่าเฉลี่ยเลขคณิต

วิธีรวบกลุ่มจับคู่แบบถ่วงน้ำหนักโดยใช้ค่าเฉลี่ยเลขคณิต (Weighted Pair Group Method with Arithmetic Mean หรือ WPGMA) เป็นเทคนิคการจัดกลุ่ม (Clustering) แบบหนึ่งของวิธีโครงสร้างลำดับชั้น (Hierarchical Method) ซึ่งพัฒนาขึ้นมาโดยใช้ Pairwise Similarity Matrix[39] โดยนักชีววิทยาเชิงสถิติ Robert Reuven Sokal และนักกีฏวิทยา Robert Reuven Sokal

WPGMA เริ่มต้นด้วยการสร้างเดนโดรแกรม (Dendrogram)[1] ซึ่งจะแสดงโครงสร้างของ Pairwise Similarity Matrix โดยในแต่ละสเต็ป กลุ่ม i และกลุ่ม j จะถูกจับรวมกันเป็นกลุ่มที่อยู่ในระดับที่สูงขึ้น หลังจากนั้นเราจะทำการคำนวณระยะห่างระหว่างกลุ่มที่เราสนใจกับกลุ่ม k ซึ่งจะใช้ระยะห่างแบบค่าเฉลี่ยเลขคณิต (Airthmetic Mean) ระหว่างสมาชิกภายในกลุ่ม k กับกลุ่ม i และระยะห่างระหว่างสมาชิกกลุ่ม k กับกลุ่ม j ดังนี้

$$d_{(i\cup j),k} = \frac{d_{i,k} + d_{j,k}}{2} \tag{6.1}$$

6.2 การแบ่งกลุ่มข้อมูลแบบเคมีน

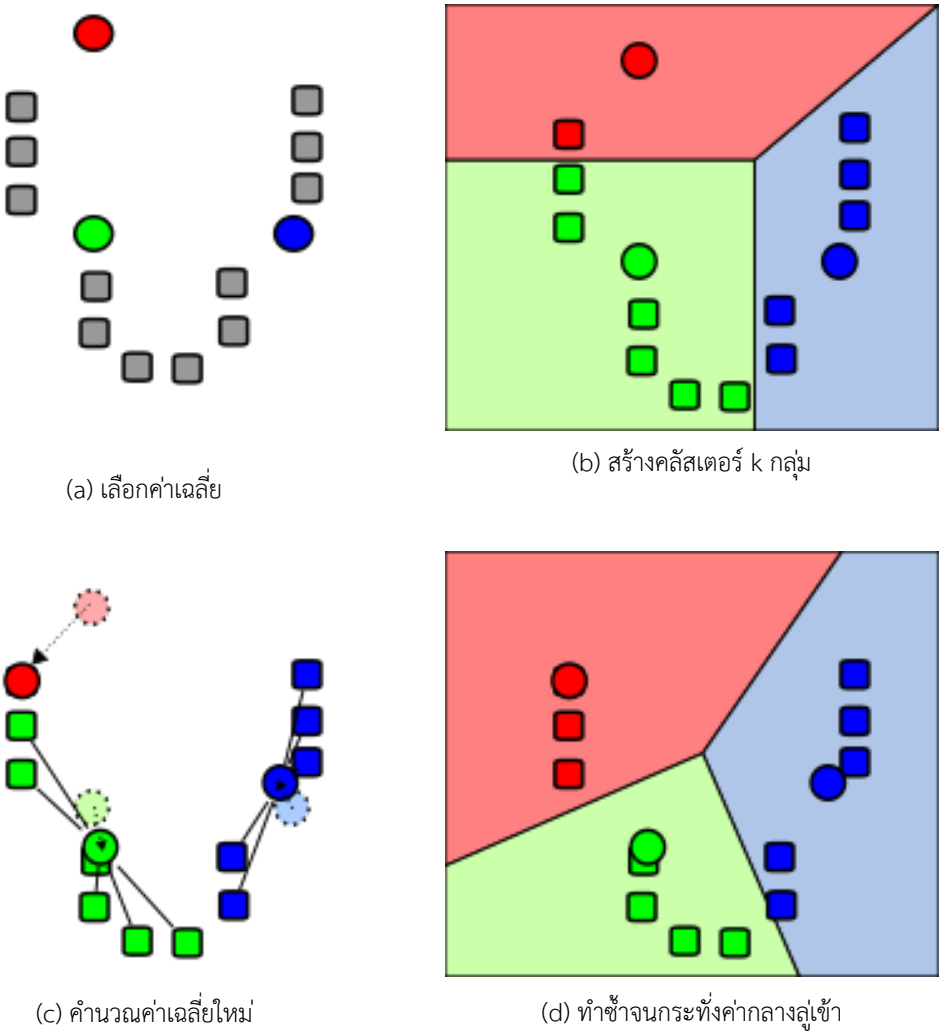
การแบ่งกลุ่มข้อมูลแบบเคมีน (K-means Clustering) เป็นหนึ่งในวิธีการแบ่งเวกเตอร์ซึ่งจะเป็นการแบ่งชุดข้อมูลที่มี n ข้อมูลออกเป็น k กลุ่ม ซึ่งเทคนิคนี้ได้ถูกใช้มาอย่างยาวนานตั้งแต่ในช่วงยุคเริ่มต้นของการทำ Data Mining[40]

ภาพที่ 6.3 แสดงอัลกอริทึมของ K-means Clustering ที่มี 4 ขั้นตอน โดยขั้นตอนที่หนึ่งจะเป็นเลือกค่าเฉลี่ยเริ่มต้น k (ในกรณีนี้ k=3) แบบสุ่มจากโดเมนข้อมูล ขั้นตอนที่สองเป็นการสร้างคลัสเตอร์ k กลุ่มรอบ ๆ ค่าเฉลี่ยที่ได้กำหนดไว้โดยการเชื่อมโยงทุกข้อมูลการสังเกตด้วยค่าเฉลี่ยที่ใกล้ที่สุด (สังเกตระยะห่างระหว่างจุดวงกลมในแต่ละคลัสเตอร์ไปจนถึงเส้นแบ่ง) เส้นแบ่งในที่นี้แสดงให้เห็นแผนภาพของโวโรนอย (Voronoi Diagram) ที่สร้างขึ้นจากค่าเฉลี่ย ขั้นตอนที่สามคือคำนวณจุดศูนย์กลางหรือจุดเซนทรอยด์ (Centroid) ของโวโรนอยของแต่ละคลัสเตอร์และทำการกำหนดเป็นค่าเฉลี่ยค่าใหม่ ขั้นตอนที่สี่จะเป็นการทำสามขั้นตอนแรกซ้ำ ซึ่งจะทำซ้ำไปเรื่อย ๆ จนกว่าค่ากลางของแต่ละคลัสเตอร์จะไม่เปลี่ยนแปลง โดยเมื่อค่ากลางลู่เข้าแล้ว เราจะได้ฟังก์ชันที่มีเส้นแบ่งสำหรับการนำไปจัดคลัสเตอร์ของข้อมูล Test Set ต่อไป

ลำดับต่อไปคือวิธีการลดขนาดมิติของข้อมูลแบบเชิงเส้นซึ่งผู้เขียนขออธิบาย 2 วิธี คือ

- Principal Component Analysis (PCA)
- Multidimensional Scaling (MDS)

[1] เดนโดรแกรมคือแผนผังต้นไม้ ถูกใช้อย่างแพร่หลายในการวิเคราะห์แบบกลุ่ม ชีววิทยาเชิงคำนวณหรือชีวสารสนเทศศาสตร์



(a) เลือกค่าเฉลี่ย

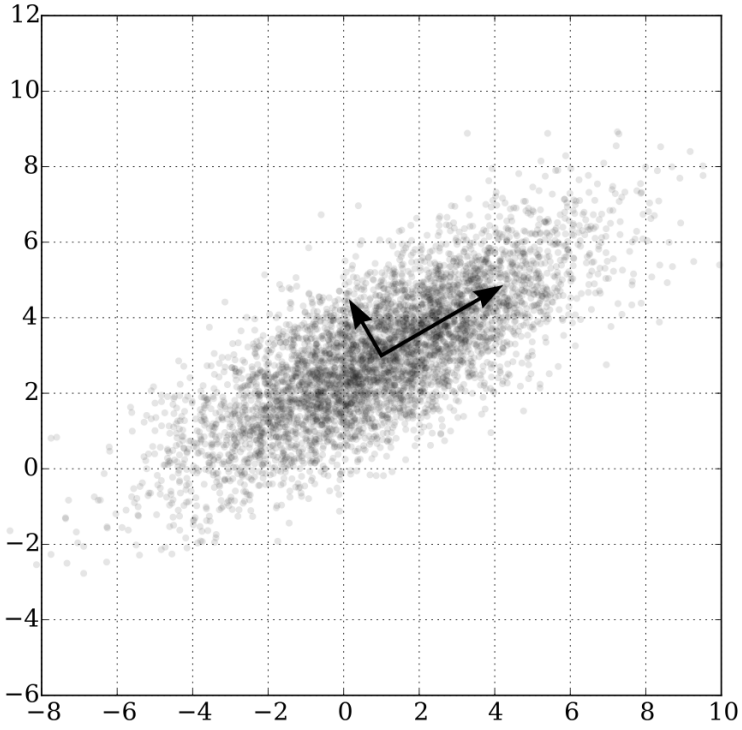
(b) สร้างคลัสเตอร์ k กลุ่ม

(c) คำนวณค่าเฉลี่ยใหม่

(d) ทำซ้ำจนกระทั่งค่ากลางลู่เข้า

ภาพ 6.3 ขั้นตอนการทำ K-means Clustering (เครดิตภาพ: https://en.wikipedia.org/wiki/K-means_clustering)

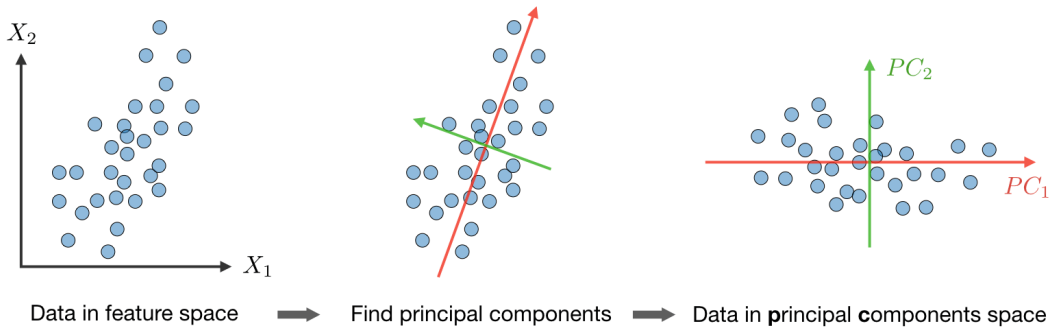
6.3 การวิเคราะห์องค์ประกอบหลัก



ภาพ 6.4 Principal Component Analysis ของข้อมูลที่มีการกระจายตัวแบบเกาส์เซียนหลายตัวแปร (Multivariate Gaussian Distribution) เวกเตอร์ที่แสดงนั้นเป็น Eigenvector ของ Covariance Matrix ที่มีการปรับขนาด (Scaled) โดยใช้ค่ายกกำลังสองของค่าไอเกน (Eigenvalue) และมีการปรับตำแหน่งโดยใช้ค่าเฉลี่ย

การวิเคราะห์องค์ประกอบหลัก (Principal Component Analysis หรือ PCA) เป็นวิธีทางสถิติสำหรับการจัดกลุ่มตัวแปร ถูกนำมาใช้เพื่อรับมือกับข้อมูลที่มีจำนวนหลายมิติหรือมีหลายตัวแปร วิธี PCA ถูกพัฒนาโดยนักคณิตศาสตร์และชีววิทยาเชิงสถิติชาวอังกฤษ Karl Pearson โดยเทคนิค PCA สามารถหาความสัมพันธ์ของตัวแปรเหล่านั้นโดยทำการลดขนาดของมิติโดยสร้างชุดข้อมูลใหม่ที่อาศัยแกนอ้างอิงจากชุดข้อมูลเดิม ซึ่งจำนวนมิติที่ถูกลดลงนั้นก็มีจำนวนมิติเพียง 2 หรือ 3 มิติเท่านั้น ซึ่งทำให้ง่ายต่อการตีความและวิเคราะห์ข้อมูล เช่น การจัดกลุ่มชุดข้อมูลโดยจำแนกตาม Feature ซึ่ง Feature แต่ละคู่จะมีคุณสมบัติ Orthogonality หรือตั้งฉากกันนั่นเอง ทำให้เราสามารถแสดงผลลัพธ์ของ PCA ออกมาได้ในปริภูมิทั่วไป

การทำ PCA ประกอบไปด้วยขั้นตอนดังต่อไปนี้



ภาพ 6.5 ขั้นตอนการทำ Principal Component Analysis ซึ่งเป็นการทำให้ค่าความแปรปรวน (Variance) ของข้อมูลนั้นมีค่ามากที่สุดในปริภูมิที่มี k มิติ

1. Normalize ข้อมูลทั้งหมดเพื่อให้มีค่าเฉลี่ยเลขคณิต (Mean) หรือ μ เท่ากับ 0 และมีส่วนเบี่ยงเบนมาตรฐาน (Standard Deviation) หรือ σ เท่ากับ 1 ซึ่งสามารถทำได้โดยการนำข้อมูลไปลบออกด้วย μ แล้วหารด้วย σ

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j} \tag{6.2}$$

โดยที่ μ และ σ สามารถหาได้ดังนี้

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n}(x_i) \tag{6.3}$$

และ

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{6.4}$$

นอกจากนี้เรายังสามารถหา Covariance ระหว่าง 2 ตัวแปรได้ดังนี้

$$\sigma(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y) \tag{6.5}$$

2. คำนวณ Covariance Matrix Σ ซึ่งมีความสมมาตรกับค่าไอเกนจริง (Real Eigenvalue) โดยมีนิยามดังต่อไปนี้

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})^T \tag{6.6}$$

สำหรับกรณีแบบที่ง่ายที่สุดเราสามารถพิจารณา Covariance Matrix ของ 2 มิติ ดังนี้

$$C = \begin{pmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{pmatrix} \tag{6.7}$$

3. คำนวณค่าไอเกน (Eigenvalue) และเวกเตอร์ไอเกน (Eigenvector) ของ Covariance Matrix หลังจากนั้นเราจะทำการตัดสินใจเลือกว่าเราจะเก็บแกนไหนไว้บ้าง ซึ่งเราจะมาทำการเลือก Eigenvector นั่นเอง โดยเราจะทำการสร้าง Feature Vectors ซึ่งเป็นเมทริกซ์โดยมีคอลัมน์เป็น Eigenvector ที่เราจะเลือกว่าจะเก็บไว้โดยมาตรวจสอบค่าความสำคัญ (Significance) ของแต่ละ Eigenvector โดยดูที่ค่า Eigenvalue ถ้าหากว่ามี Eigenvalue น้อยก็หมายความว่ามี Significance ที่น้อย ซึ่งหลังจากที่เราได้ Feature Vectors แล้วเราจะนำไปใช้ในขั้นตอนต่อไป
4. ขั้นตอนสุดท้ายคือแปลงตำแหน่งของข้อมูลโดยการเปลี่ยนแกน กล่าวคือขั้นตอนก่อนหน้านี้นั้นเราไม่ได้แก้ไขข้อมูลเลย (ยกเว้นแค่การทำ Standardization) โดยเราทำแค่การเลือกองค์ประกอบหลัก (Principal Components) แล้วก็สร้าง Feature Vector เท่านั้นเอง ดังนั้นข้อมูลจะยังคงอ้างอิงกับแกนเดิมอยู่ โดยในขั้นตอนนี้เราจะทำการ Projection ข้อมูลลงบนแกนใหม่ซึ่งเป็น Principal Components ที่ได้จาก Feature Vectors ซึ่งทำได้โดยการนำเมทริกซ์สลับเปลี่ยนของข้อมูลเดิมคูณกับเมทริกซ์สลับเปลี่ยน (Transpose Matrix) ของ Feature Vector

6.4 การสเกลแบบหลายมิติ

การสเกลแบบหลายมิติ (Multidimensional Scaling หรือ MDS)[41,42]

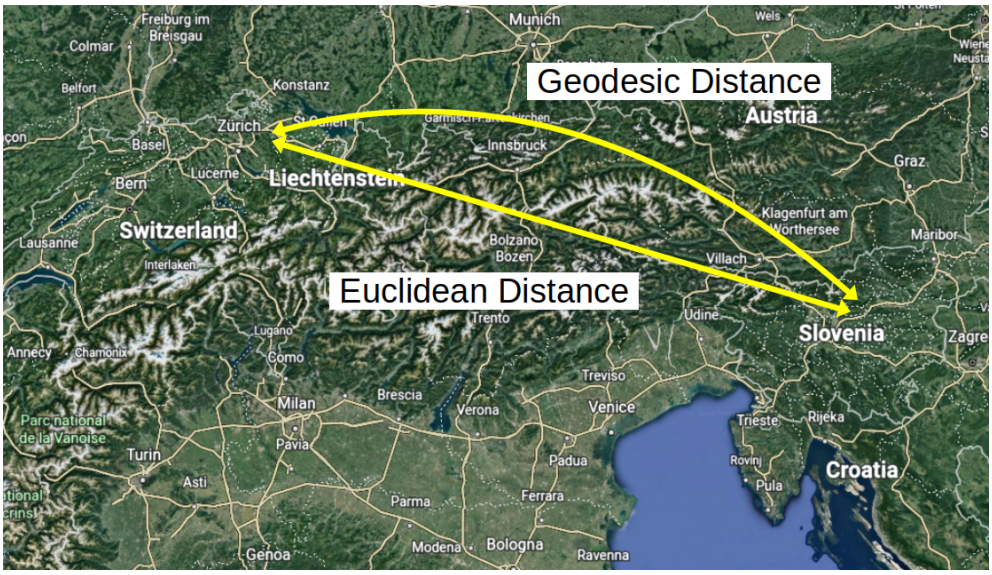
สำหรับวิธีการลดขนาดมิติของข้อมูลแบบเชิงเส้นนั้นไม่ซับซ้อนและมีจำนวนวิธีที่ไม่มาก แต่ในกรณีของวิธีการลดขนาดมิติแบบไม่เชิงเส้นนั้นจะซับซ้อนมากกว่า โดยมีอัลกอริทึมที่ถูกพัฒนาแตกต่างกันไปตามหัวข้อย่อยต่อไปนี้[43]

- Isometric Feature Mapping (Isomap)
- Kernel PCA
- Diffusion Map
- Sketch-Map (ไม่ได้ลงรายละเอียด)

6.5 การเชื่อมโยงลักษณะเฉพาะแบบไอโซเมตริก

การเชื่อมโยงลักษณะเฉพาะแบบไอโซเมตริก (Isometric Feature Mapping หรือ Isomap)[44] ถูกพัฒนาขึ้นมาเพื่อแก้ปัญหาที่เรามักจะพบเมื่อเราใช้วิธีการลดจำนวนมิติของข้อมูลแบบเส้นตรง เช่น วิธี PCA

ซึ่งไม่สามารถหาตำแหน่งของข้อมูลที่ถูกต้องได้ ไอเดียของ Isomap ก็คือการสร้าง Representation ที่อยู่ในมิติต่ำ ๆ ที่สามารถรักษาหรือคงไว้ซึ่งระยะห่างแบบจีโอดีซิค (Geodesic Distance) ระหว่างจุดข้อมูลซึ่งเป็นระยะทางที่สั้นที่สุดบนทรงรี แทนที่จะใช้ระยะห่างแบบคาร์ทีเซียน (Cartesian Distance) ตัวอย่างแสดงตามภาพที่ 6.6 โดยโลกของเราคือ Manifold และประเทศสวิตเซอร์แลนด์กับสโลวีเนียคือจุดของข้อมูล

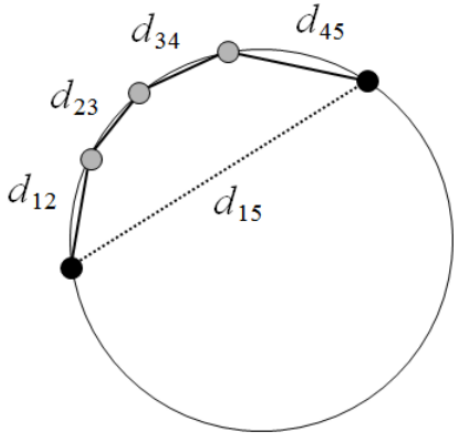


ภาพ 6.6 แสดงระยะห่างแบบ Euclidean และ Geodesic ระหว่างประเทศสวิตเซอร์แลนด์และสโลวีเนีย

การทำ Isomap ประกอบไปด้วย 3 ขั้นตอนดังนี้

1. สร้างกราฟโดยใช้ข้อมูลที่เป็นแบบ Local Connectivities ซึ่งในกราฟอันนี้จุดข้อมูลแต่ละจุดจะถูกเชื่อมโยงไปยังจุดที่ใกล้ที่สุด (*k*th Nearest Neighbor) ซึ่งถูกคำนวณด้วยระยะห่างของคู่ของข้อมูล (Pairwise Distance)

2. คำนวณหาระยะห่างที่สั้นที่สุดของข้อมูลทุกคู่ในกราฟโดยสามารถใช้การประมาณค่าระยะห่างแบบย่อย ๆ ได้ ตามภาพที่ 6.7



ภาพ 6.7 การประมาณค่าระยะห่างแบบ Geodesic

3. นำเมทริกซ์ R_{ij} มาทำ Multidimensional Scaling ซึ่งเมทริกซ์นี้จะเก็บข้อมูล Geodesic Distance ของข้อมูลแต่ละคู่ไว้

6.6 การวิเคราะห์องค์ประกอบหลักแบบเคอร์เนล

วิธีต่อมาที่ได้รับความนิยมเช่นเดียวกันคือการวิเคราะห์องค์ประกอบหลักแบบเคอร์เนล (Kernel PCA)[45] ซึ่งเป็นวิธีที่ถูกใช้สำหรับการหา Low-dimensional Representation เช่นเดียวกันแต่ว่ามีประสิทธิภาพมากเมื่อนำมาใช้กับข้อมูลที่เป็นถูกมองเป็น Manifold แบบที่เป็นเส้นโค้ง (Curved) หรือบิดเบี้ยวไป (Twisted) โดยใน Kernel PCA นั้นจริง ๆ แล้วไม่ได้ทำการแปลงข้อมูล (Transformation) โดยตรงแต่ว่าทำอ้อม ๆ ผ่านฟังก์ชันเคอร์เนล $K(x, x')$ ในกรณีที่เราใช้ฟังก์ชันเคอร์เนลเป็นแบบเชิงเส้น (Linear Kernel) เช่น

$$K(x_i, x_j) = x_i^T x_j \tag{6.8}$$

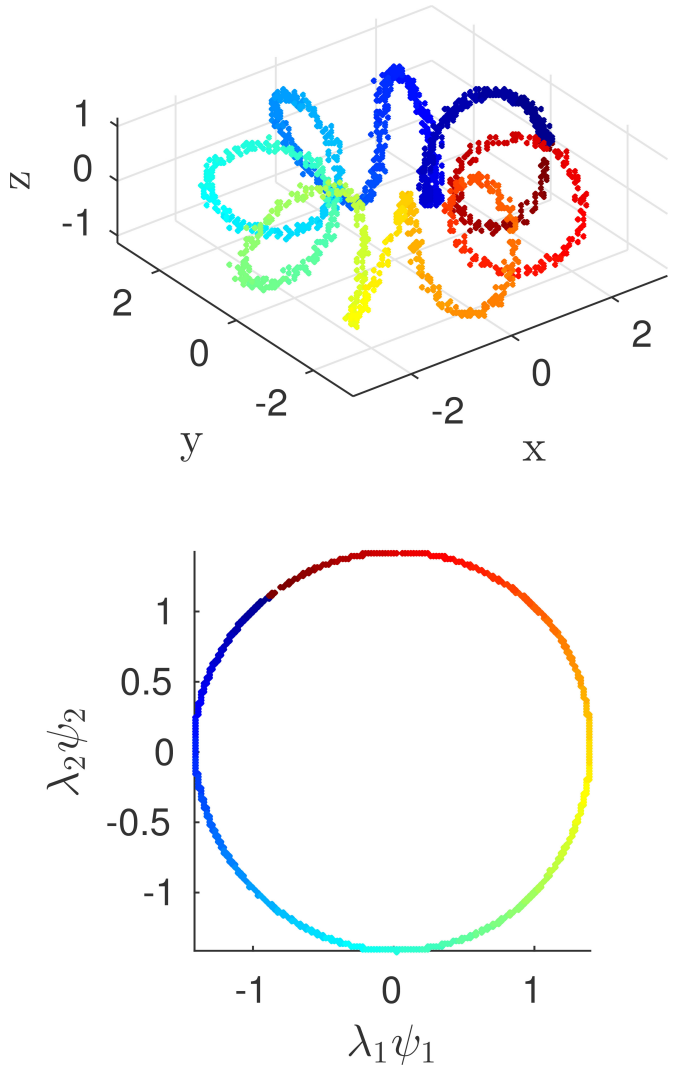
จะทำให้ Kernel PCA กลายเป็น PCA แบบมาตรฐานทั่วไป ดังนั้นการที่เราเปลี่ยนไปใช้ฟังก์ชันเคอร์เนลแบบพหุนาม (Polynomial Kernel) ดังนี้

$$K(x_i, x_j) = (x_i^T x_j)^p \tag{6.9}$$

จะสามารถเพิ่มขนาดของ Feature Space ได้ตามขนาดของค่าดีกรี p

6.7 วิธีแผนที่แบบแพร่กระจาย

วิธีแผนที่แบบแพร่กระจาย (Diffusion Map)[46,47] เป็นอีกเทคนิคหนึ่งที่มีความคล้ายกับ Kernel PCA เพราะว่าเป็นวิธีที่สามารถใช้ในการหาตัวแปรแบบไม่เป็นเส้นตรง (Nonlinear Variables) ที่สามารถอธิบายระบบของเรา (ชุดข้อมูล) ได้ในปริภูมิที่มีจำนวนมิติต่ำ ๆ โดยเราสามารถประยุกต์ใช้ Diffusion Map ในการวิเคราะห์ Trajectory[1] ที่ได้จากการคำนวณ Molecular Dynamics ได้



ภาพ 6.8 ภาพบน: ข้อมูลที่มีการกระจายอย่างไม่สม่ำเสมอบนเกลียวแบบวงแหวน (Toroidal Helix) ภาพล่าง: พิกัดของ Diffusion Map 2 อันดับแรกซึ่งมีการทำ Normalization โดยใช้เทคนิค Laplace-Beltrami Normalization ด้วย (เครดิตภาพ: https://en.wikipedia.org/wiki/Diffusion_map)

[1]Trajectory (แปลเป็นไทยว่า วิธี) คือการเก็บข้อมูล Configuration ของโมเลกุลที่เปลี่ยนแปลงไปตามเวลาที่ใช้ในการจำลองด้วยคอมพิวเตอร์

ในการคำนวณ Diffusion Map นั้นเราจะเริ่มต้นด้วยการคำนวณ Gaussian Kernel

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) \tag{6.10}$$

ซึ่งเป็นฟังก์ชันเคอร์เนลอันเดียวกันกับสมการที่ (11.2) แต่ว่าสิ่งที่ทำให้ Diffusion Map นั้นแตกต่างไปจากวิธีเคอร์เนลทั่วไปก็คือในเทคนิคจะมีการประมาณค่าฟังก์ชันไอเกน (Eigenfunction) ที่ชื่อว่าโอเปอเรเตอร์ฟอกเกอร์-พลังค์ (Fokker-Planck Operator) สำหรับการทำ Diffusion Process[48] โดยเราจะต้องมีการปรับฟังก์ชันเคอร์เนลโดยการทำให้เป็นปกติ (Normalization)[49] ตามสมการต่อไปนี้

$$\tilde{K}_{ij} = \frac{K_{ij}}{\sqrt{\sum_k K_{jk} \sum_s K_{js}}} \tag{6.11}$$

แล้วตามด้วยการคำนวณเมทริกซ์การเปลี่ยนแปลง (Transition Matrix) ของ Diffusion Map ดังนี้

$$P_{ij} = \frac{\tilde{K}_{ij}}{\sum_j \tilde{K}_{ij}} \tag{6.12}$$

ซึ่งผลลัพธ์ที่ได้นั้น P_{ij} คือความน่าจะเป็นของการเปลี่ยนแปลง (Transition Probability) จากข้อมูลที่ i ไปยังข้อมูลที่ j ซึ่งผลรวมของ Proability ในการเปลี่ยนแปลงไปยังข้อมูลหนึ่ง ๆ นั้น $(\sum_j P_{ij})$ จะต้องมีค่าเท่ากับ 1

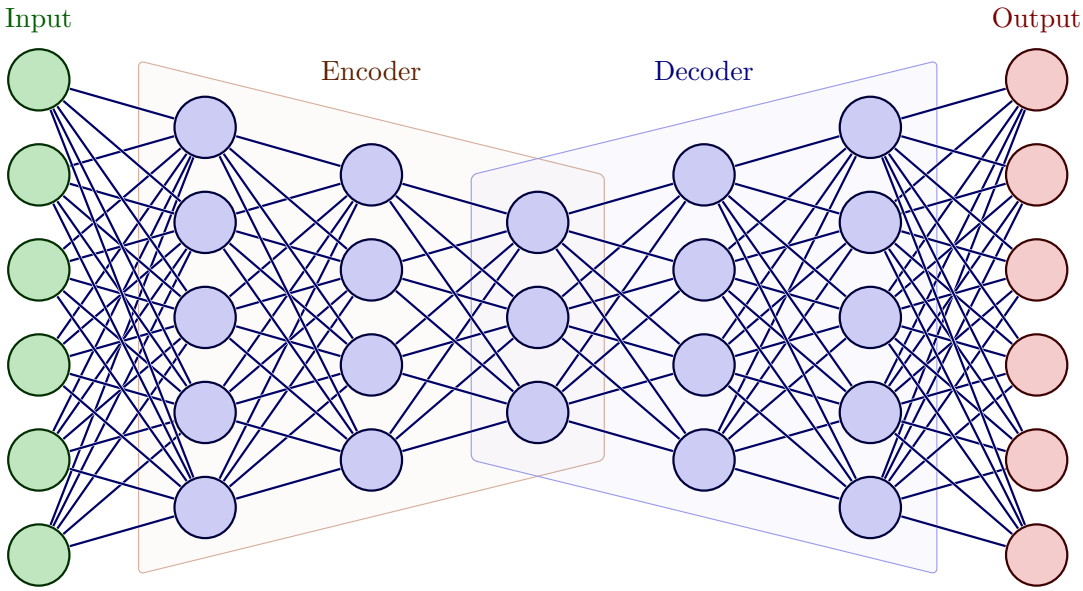
เมื่อเราได้ P_{ij} ขั้นตอนต่อไปที่เราสามารถคำนวณได้ก็คือ Stochastic Matrix ซึ่งมีคุณสมบัติหลาย ๆ อย่างที่น่าสนใจเช่นเดียวกันแต่ว่าผู้เขียนไม่ขอลงรายละเอียด

6.8 การเข้ารหัสแบบอัตโนมัติ

ตัวเข้ารหัสแบบอัตโนมัติ (Autoencoder หรือ AE)[50] เป็นอัลกอริทึม Unsupervised Learning แบบหนึ่งที่ใช้โมเดล ANN โดยมีรูปแบบของโครงข่าย (Network) ที่เฉพาะตัวนั่นก็คือจะทำการลดมิติของข้อมูลโดยทำการบีบอัดข้อมูลหรือเข้ารหัส (Encoding) หรือ E และทำการถอดรหัส (Decoding) ออกมาเป็นข้อมูลเดิม[51] ผ่านตัวถอดรหัส (Decoder) หรือ D

$$y = E(x) \tag{6.13}$$

$$\tilde{x} = D(y) \tag{6.14}$$



ภาพ 6.9 โครงข่ายประสาทของ Autoencoder ที่มีลักษณะความเป็นสมมาตร

โดยที่ x, y และ $\tilde{x}$ คือข้อมูลอินพุต ข้อมูลเอาต์พุต และข้อมูลอินพุตที่ถูกถอดรหัสออกมา ตามลำดับ

ตัวโมเดล AE มีความพิเศษคือจะมีลักษณะของความสมมาตรและมีความแตกต่างจาก PCA นั่นก็คือสามารถบีบอัดหรือลดจำนวนมิติของข้อมูลแบบไม่เป็นเส้นตรงได้ (Nonlinear) ได้ด้วยการใช้ฟังก์ชันกระตุ้นแบบ Nonlinear

ตัวอย่างการเขียนโค้ดของ Autoencoder ด้วย TensorFlow มีดังต่อไปนี้

```
import tensorflow as tf

from tensorflow.keras import layers, losses
from tensorflow.keras.models import Model

latent_dim = 64

class Autoencoder(Model):
    def __init__(self, latent_dim):
        super(Autoencoder, self).__init__()
        self.latent_dim = latent_dim
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(latent_dim, activation='relu'),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Dense(784, activation='sigmoid'),
            layers.Reshape((28, 28))
```

```
        ])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

autoencoder = Autoencoder(latent_dim)
autoencoder.compile(optimizer='adam',
                    loss=losses.MeanSquaredError())
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

จริง ๆ แล้ว Autoencoder แบบทั่วไปนั้นก็คือ Neural Network แบบง่ายที่มีจำนวนและขนาดของ Hidden Layer ที่สมมาตรกัน

Science never solves a problem without creating ten more.

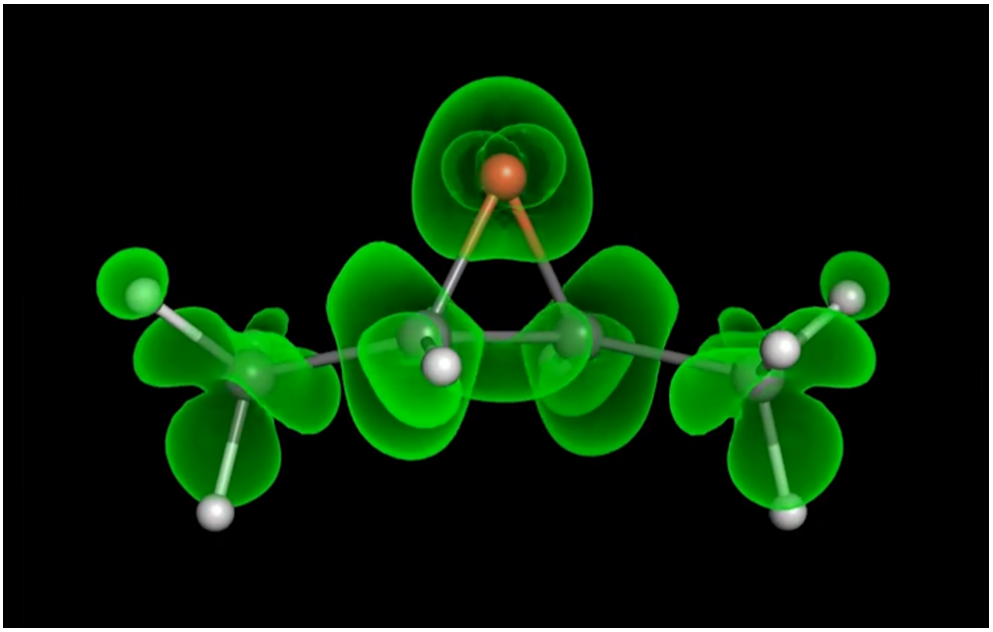
- George Bernard Shaw (1856 - 1950)

ส่วนที่ 2

การทำนายคุณสมบัติเชิงเคมีควอนตัม

บทที่ 7

วิธีคำนวณทางโครงสร้างเชิงอิเล็กทรอนิกส์



ภาพ 7.1 ความหนาแน่นของอิเล็กตรอนของโมเลกุล 2,3-(S,S)-dimethyloxirane ซึ่งคำนวณด้วยวิธี Real-Time Density Functional Theory

ส่วนที่สองของหนังสือเล่มนี้จะเกี่ยวข้องกับเคมีควอนตัมเป็นหลัก เคมีควอนตัมเป็นพื้นฐานสำคัญของการพัฒนาเทคนิคสำหรับการวิเคราะห์คุณสมบัติของโมเลกุลโดยนักเคมีนั้นส่วนใหญ่แล้วจะใช้เทคนิคทางสเปกโทรสโกปี เช่น Infrared (IR) Spectroscopy, Nuclear Magnetic Resonance (NMR) Spectroscopy, และ Scanning Probe Microscopy ซึ่งเทคนิคเหล่านี้ล้วนเกี่ยวข้องกับการคำนวณหาพลังงานในระดับโมเลกุล นอกจากนี้แล้วเคมีควอนตัมยังเกี่ยวข้องกับการศึกษาสถานะพื้น (Ground State) และสถานะกระตุ้น (Excited State) ของอะตอมแต่ละตัว รวมไปถึงการศึกษากลไกการเกิดปฏิกิริยาเคมีและสถานะทรานซิชั่น (Transition State) ที่เป็นสถานะที่เกิดขึ้นในการเปลี่ยนแปลงโครงสร้างของโมเลกุล การที่เราเข้าใจองค์

ความรู้ขั้นพื้นฐานในระดับอะตอมและโมเลกุลนั้นทำให้เราประยุกต์ใช้และนำไปสู่การศึกษาคุณสมบัติของโมเลกุลในระดับที่ใหญ่ขึ้นได้ เช่น เทอร์โมไดนามิกส์ (Thermodynamics) และจลนศาสตร์เชิงเคมี (Chemical Kinetics) ซึ่งนำไปสู่การพัฒนาแบบจำลองทางคณิตศาสตร์ต่าง ๆ ซึ่งสามารถใช้คอมพิวเตอร์ในการศึกษาระบบที่เราสนใจได้ก่อนที่จะไปศึกษาจริงในห้องทดลอง

บทนี้ซึ่งเป็นบทแรกของส่วนที่สองนั้นผู้อ่านจะได้ศึกษาโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ของโมเลกุล ซึ่งเป็นการศึกษาว่าอิเล็กตรอนที่อยู่ภายในอะตอมและโมเลกุลนั้นมีพฤติกรรมอย่างไรทั้งในสถาวะพื้นและสถาวะกระตุ้น โดยเราจะมาดูรายละเอียดของทฤษฎีควอนตัมในมุมมองของนักเคมีทฤษฎี รวมไปถึงการพัฒนาระเบียบวิธีการคำนวณเพื่อใช้ในการอธิบายอันตรกิริยาระหว่างอิเล็กตรอนและศึกษาคุณสมบัติของอะตอมและโมเลกุลต่อไป

7.1 ฟังก์ชันคลื่น

7.1.1 ประวัติศาสตร์และความสำคัญของสมการชโรดิงเงอร์

โมเลกุลเป็นหน่วยพื้นฐานของสิ่งต่าง ๆ รอบตัวเรา โมเลกุลก็คือกลุ่มของอะตอมหลาย ๆ อะตอมมารวมกันและในอะตอมนั้นเราสนใจพฤติกรรมของอิเล็กตรอนเป็นพิเศษ ในวิชากลศาสตร์ควอนตัมนั้นเราจะอธิบายพฤติกรรมของโมเลกุลโดยมุ่งเน้นไปที่อิเล็กตรอนซึ่งสามารถที่จะถูกอธิบายได้ด้วยฟังก์ชันทางคณิตศาสตร์ที่เรียกว่า *"ฟังก์ชันคลื่น (Wavefunction)"* ซึ่งถูกพัฒนาขึ้นมาเพื่อเป็นแนวคิดสำหรับการอธิบายอิเล็กตรอนและระบบที่ประกอบไปด้วยอิเล็กตรอนหลายตัว โดยหนึ่งในสมการที่โด่งดังที่สุดสมการหนึ่งของวงการวิทยาศาสตร์นั่นคือสมการชโรดิงเงอร์ (Schrödinger Equation)[52] ซึ่งนำเสนอโดยศาสตราจารย์ Erwin R. J. A. Schrödinger (นักฟิสิกส์เชื้อสายออสเตรีย-ไอริช ซึ่งในขณะนั้นดำรงตำแหน่งอยู่ที่ University of Zurich) สำหรับการอธิบาย Wavefunction โดย Schrödinger ได้ตีพิมพ์บทความงานวิจัยในวารสาร Annalen der Physik[1] ในปี ค.ศ. 1926 ที่ต่อเนื่องกันเป็นจำนวน 4 บทความในซีรีย์ที่ชื่อว่า *Quantisierung als Eigenwertproblem* โดยบทความฉบับแรกนั้นเป็นการนำเสนอสมการ **Time-independent Schrödinger Equation**[53] และในเวลาต่อมา Schrödinger ก็สามารถพิสูจน์หารูปแบบของสมการ **Time-dependent Schrödinger Equation** และตีพิมพ์ในบทความฉบับที่ 4 ได้สำเร็จ[54] โดยสมการชโรดิงเงอร์ถูกนำมาใช้ในการศึกษาระบบทางกลศาสตร์ควอนตัมซึ่งการแก้สมการชโรดิงเงอร์ได้นั้นจะทำให้ได้มาซึ่งผลเฉลยของสมการคณิตศาสตร์ที่อธิบาย Wavefunction ได้นั่นเอง

จากผลงานดังกล่าวทำให้ศาสตราจารย์ Erwin Schrödinger ได้รับรางวัลโนเบลสาขาฟิสิกส์ ค.ศ. 1933 ร่วมกับศาสตราจารย์ Paul A. M. Dirac (ศาสตราจารย์ที่ University of Cambridge) ซึ่งเป็นหนึ่งในผู้บุกเบิกกลศาสตร์ควอนตัมและการพัฒนาสมการดิแรก (Dirac Equation) ซึ่งถูกนำมาใช้อธิบายพฤติกรรมของแฟร์มิออน (Fermions)[2]

[1] ปัจจุบันนี้วารสาร Annalen der Physik ยังตีพิมพ์บทความวิชาการอย่างต่อเนื่อง

[2] อ้างอิง https://www.nobelprize.org/prizes/physics/1933/summary

Da für die Aufstellung der Variationsgleichungen die Koordinatenwahl belanglos ist, wählen wir rechtwinkelige kartesische. Dann lautet (1') in unserem Fall (e, m sind Ladung und Masse des Elektrons):

$$(1'') \qquad \left(\frac{\partial\psi}{\partial x}\right)^2 + \left(\frac{\partial\psi}{\partial y}\right)^2 + \left(\frac{\partial\psi}{\partial z}\right)^2 - \frac{2m}{K^2}\left(E + \frac{e^2}{r}\right)\psi^2 = 0 \ .$$

$$r = \sqrt{x^2 + y^2 + z^2} \ .$$

Und unser Variationsproblem lautet

$$(3) \qquad \begin{cases} \delta J = \delta \iiint dx\,dy\,dz \left[\left(\frac{\partial\psi}{\partial x}\right)^2 + \left(\frac{\partial\psi}{\partial y}\right)^2 + \left(\frac{\partial\psi}{\partial z}\right)^2 - \right. \\ \qquad \left. - \frac{2m}{K^2}\left(E + \frac{e^2}{r}\right)\psi^2\right] = 0 \ , \end{cases}$$

das Integral erstreckt über den ganzen Raum. Man findet daraus in gewohnter Weise

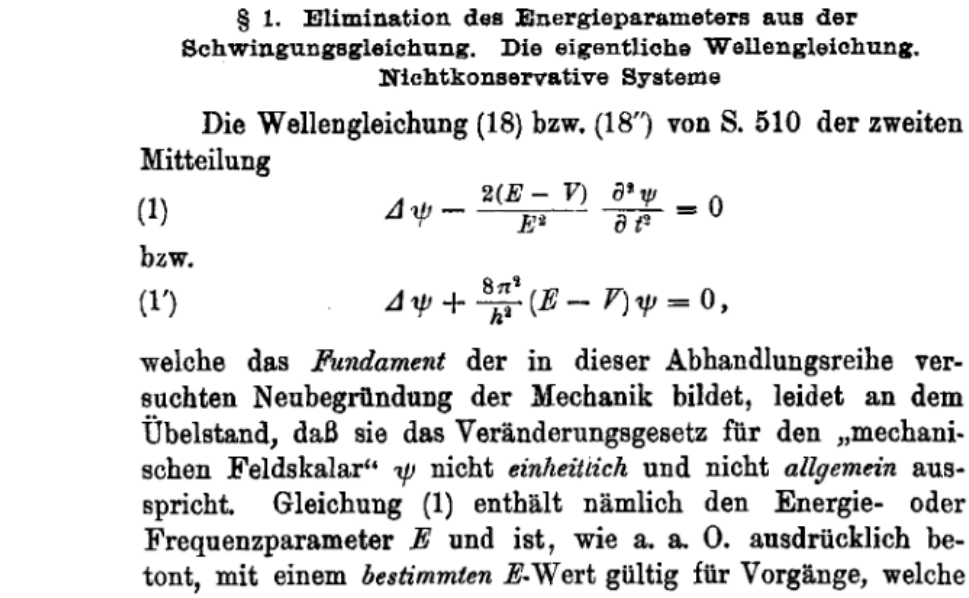
$$(4) \qquad \begin{cases} \frac{1}{2}\delta J = \int df\,\delta\psi\,\frac{\partial\psi}{\partial n} - \iiint dx\,dy\,dz\,\delta\psi\left[\Delta\psi + \right. \\ \qquad \left. + \frac{2m}{K^2}\left(E + \frac{e^2}{r}\right)\psi\right] = 0 \ . \end{cases}$$

Es muß also erstens

$$(5) \qquad \Delta\psi + \frac{2m}{K^2}\left(E + \frac{e^2}{r}\right)\psi = 0$$

1) Es entgeht mir nicht, daß diese Formulierung nicht ganz eindeutig ist.

ภาพ 7.2 ส่วนหนึ่งของบทความฉบับแรกที่ตีพิมพ์โดย Erwin Schrödinger ในเดือนมกราคมของปี ค.ศ. 1926 โดยเสนอสมการ Time-independent Schrödinger Equation (สมการที่ (1") และ (5))



§ 1. Elimination des Energieparameters aus der Schwingungsgleichung. Die eigentliche Wellengleichung. Nichtkonservative Systeme

Die Wellengleichung (18) bzw. (18″) von S. 510 der zweiten Mitteilung

$$\Delta\psi - \frac{2(E-V)}{E^2}\frac{\partial^2\psi}{\partial t^2} = 0 \tag{1}$$

bzw.

$$\Delta\psi + \frac{8\pi^2}{h^2}(E-V)\psi = 0, \tag{1'}$$

welche das *Fundament* der in dieser Abhandlungsreihe versuchten Neubegründung der Mechanik bildet, leidet an dem Übelstand, daß sie das Veränderungsgesetz für den „mechanischen Feldskalar" ψ nicht *einheitlich* und nicht *allgemein* ausspricht. Gleichung (1) enthält nämlich den Energie- oder Frequenzparameter E und ist, wie a. a. O. ausdrücklich betont, mit einem *bestimmten* E-Wert gültig für Vorgänge, welche

ภาพ 7.3 ส่วนหนึ่งของบทความฉบับที่ 4 ที่ตีพิมพ์โดย Erwin Schrödinger โดยเสนอสมการ Time-dependent Schrödinger Equation

สมการชโรดิงเงอร์สามารถแบ่งออกได้เป็นสองแบบคือแบบที่ไม่ขึ้นกับเวลาและแบบที่ขึ้นกับเวลา ดังนี้

- **1. Time-independent Schrödinger Equation**

$$\hat{H}\Psi = E\Psi \tag{7.1}$$

- **2. Time-dependent Schrödinger Equation**

$$i\hbar\frac{d}{dt}\Psi(t) = \hat{H}\Psi(t) \tag{7.2}$$

โดย Wavefunction $\Psi(t)$ ที่เป็นฟังก์ชันไอเกน (Eigenfunction) นั้นจะบรรจุข้อมูลเชิงอิเล็กทรอนิกส์ทุกอย่างเกี่ยวกับระบบของเราเอาไว้[55,56,57] ซึ่งระบบในที่นี้ก็คือโมเลกุล โดยสมการข้างต้นเป็นการคำนวณหาพลังงานของระบบโดยใช้ Hamiltonian Operator $(\hat{H})$ ซึ่งเป็น Operator ที่สอดคล้องกับพลังงาน ซึ่งจริง ๆ แล้วค่าไอเกน (Eigenvalue) ของสมการข้างต้น (สมการที่ (7.2) และ (7.1)) จะเป็นคุณสมบัติของโมเลกุลอะไรก็ได้ ตราบใดที่เราใช้ Operator ที่สอดคล้องกับคุณสมบัตินั้น ๆ

7.1.2 คุณสมบัติของฟังก์ชันคลื่น

ฟังก์ชันคลื่นเชิงอิเล็กทรอนิกส์ที่ได้มาจากผลเฉลยที่ถูกต้องหรือได้มาจากการประมาณค่านั้นจะต้องมีคุณสมบัติต่อไปนี้

- ฟังก์ชันมีค่าที่แน่นอนและมีขอบเขต (Be Finite)
- ฟังก์ชันมีความต่อเนื่องและหาค่าได้ตลอดทั้งโดเมน (Be Continuous)
- มีผลเฉลยเพียงแค่ค่าเดียวเท่านั้นสำหรับโดเมนหนึ่งค่า (Single-valued) กล่าวคือโดเมนหรืออินพุต x จะต้องให้เรนจ์หรือเอาต์พุต y แค่หนึ่งค่าเท่านั้น
- เป็นฟังก์ชันที่มีคุณสมบัติในการมองอิเล็กตรอนทุก ๆ ตัวเหมือนกัน (Indistinguishability of Electron)
- ค่ายกกำลังสองของฟังก์ชันเป็นการกระจายตัวของความน่าจะเป็น
- ต้องมีความปฏิสมมาตร (Antisymmetry) กล่าวคืออิเล็กตรอนนั้นคือเฟอร์มิออน (Fermion)[58] ดังนั้นฟังก์ชันคลื่นจะต้องเปลี่ยนมีการเปลี่ยนเครื่องหมายเมื่ออิเล็กตรอนสองตัวใด ๆ มีการแลกเปลี่ยนพิกัดเชิงพื้นที่หรือพิกัดเชิงสปินกัน

ถ้าฟังก์ชันนั้นไม่มีคุณสมบัติข้างต้นนี้จะถือว่าไม่มีความเหมาะสมในการนำมาใช้งานและจะให้ผลการคำนวณที่ผิดพลาด

7.2 แฮมิลโทเนียน

Hamiltonian เป็นสิ่งที่สำคัญมากในเคมีควอนตัมเพราะเปรียบเสมือนเป็นกุญแจที่สามารถไขรหัสหาคำตอบหรือความลับจาก Wavefunction ได้ โดย Hamiltonian Operator ที่เรานำมาใช้งานนั้นจริง ๆ แล้วก็คือ Operator สำหรับการหาพลังงานรวมนั่นเอง โดยเป็นผลรวมของ Operator พลังงานจลน์และพลังงานศักย์

$$\hat{H} = \hat{T} + \hat{V} \tag{7.3}$$

โดยที่พลังงานจลน์นั้นสามารถเขียนให้อยู่ในรูปของ Momentum Operator ได้โดยพิสูจน์จากพลังงานจลน์ในกรณีแบบดั้งเดิม ดังนี้

$$T = \frac{1}{2}mv_x^2 \tag{7.4}$$

$$= \frac{(mv_x)^2}{2m} \tag{7.5}$$

ทำการจัดรูปใหม่แล้วทำการแทนเทอม mv_x ด้วย Momentum Opeator ในทางกลศาสตร์ควอนตัม $(-ih\frac{d}{dx})$ จะได้ Operator ใหม่ดังนี้

$$\hat{T} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} \tag{7.6}$$

สำหรับพลังงานศักย์นั้นตรงไปตรงมา นั่นคือเราสามารถเขียนพลังงานศักย์ในทางควอนตัมได้แบบเดียวกับกรณีกลศาสตร์ดั้งเดิมได้เลย ดังนี้

$$\hat{V} = V(x) \tag{7.7}$$

เมื่อเรานำ Operator ของทั้งสองพลังงาน (สมการที่ (7.6) และสมการที่ (7.7)) มารวมกันเราจะได้ Hamiltonian Operator ดังนี้

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x) \tag{7.8}$$

ลำดับต่อมาคือเราจะมาทำการพิจารณาพลังงานศักย์กันก่อนเพราะว่าไม่ซับซ้อนเหมือนกับกรณีของพลังงานจลน์ โดยพลังงานศักย์ที่เราจะพิจารณาก็คือพลังงานงานศักย์คูลอมบ์ (Coulomb Potential Energy หรือ Operator นั่นเอง) โดยมีสมการดังต่อไปนี้

$$E_{q_1q_2} = q_1\frac{q_2}{4\pi\epsilon_0|\boldsymbol{R}|} \tag{7.9}$$

โดยเมื่อเราพิจารณาระบบง่าย ๆ เช่น อะตอมไฮโดรเจนซึ่งมี 1 อิเล็กตรอนและ 1 นิวเคลียส แล้วกำหนดจุดกำเนิด (Origin Point) ซึ่งมีระยะห่างจากอิเล็กตรอนเท่ากับ r หน่วยและมีระยะห่างจากนิวเคลียสเท่ากับ $\boldsymbol{R}$ หน่วย จะได้ว่าระยะห่างระหว่างอิเล็กตรอนและนิวเคลียสคือ $\boldsymbol{r} - \boldsymbol{R}$ หน่วย ดังนั้นเราสามารถเขียน Hamiltonian Operator ได้ดังนี้

$$\hat{H} = \underbrace{-\frac{\hbar^2}{2M}\left(\frac{\partial^2}{\partial X^2}+\frac{\partial^2}{\partial Y^2}+\frac{\partial^2}{\partial Z^2}\right)}_{\text{Nuclear Kinetic Energy}} \underbrace{-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)}_{\text{Electronic Kinetic Energy}} \underbrace{-\frac{1}{4\pi\epsilon_0}\frac{e^2}{|\boldsymbol{r}-\boldsymbol{R}|}}_{\text{Electron-Nucleus Attraction}} \tag{7.10}$$

โดยเราสามารถใช้สัญลักษณ์ ∇^2 หรือ Laplace Operator (∇ อ่านว่า Nabla) ซึ่งเป็นอนุพันธ์อันดับที่สองของพลังงานจลน์ของนิวเคลียส (เทอมแรก) และของพลังงานจลน์ของอิเล็กตรอน (เทอมที่สอง) ของสมการที่ (7.10) โดยสามารถเขียนสมการใหม่ได้ดังนี้

$$\hat{H} = -\frac{\hbar^2}{2M}\nabla^2_{\boldsymbol{R}} - \frac{\hbar^2}{2m}\nabla^2_{\boldsymbol{r}} - \frac{1}{4\pi\epsilon_0}\frac{e^2}{|\boldsymbol{r}-\boldsymbol{R}|} \tag{7.11}$$

ถึงแม้ว่าสมการที่ (7.11) มีความเรียบง่ายแล้วแต่ว่าในเคมีควอนตัมนั้นเราจะไม่ได้ใช้สมการของ Operator ที่อยู่ในหน่วย SI (SI Units) โดยนักเคมีทฤษฎีนั้นจะใช้หน่วยอะตอม (Atomic Units หรือย่อได้เป็น a.u. หรือบางครั้งก็เขียนแค่ au)[1] ซึ่งเมื่อเราเขียนสมการในรูปของ Atomic Units แล้วจะได้สมการที่เรียบง่ายกว่าเดิม ดังนี้

$$\hat{H} = -\frac{1}{2M}\nabla^2_{\boldsymbol{R}} - \frac{1}{2}\nabla^2_{\boldsymbol{r}} - \frac{1}{|\boldsymbol{r}-\boldsymbol{R}|} \tag{7.12}$$

โดยจะสังเกตได้ว่าตัวแปรที่เกี่ยวข้องกับอิเล็กตรอนนั้นจะถูกลดรูปไป ปริมาณที่กำหนดให้มีหน่วยเป็น Atomic Units ได้มีดังนี้

ตาราง 7.1 เปรียบเทียบปริมาณทางเคมีควอนตัมในหน่วย Atomic Units และ SI Units

ปริมาณ	Atomic Units	ค่าตาม SI Units
พลังงาน	$\hbar^2/m_e a_0$ (Hartree)	$4.36 \times 10^{-18} J$
ประจุ	e	$1.60 \times 10^{-19} C$
ความยาว	a_0	$5.29 \times 10^{-11} m$
มวล	m_e	$9.11 \times 10^{-31} kg$

[1] อ่านรายละเอียดเกี่ยวกับ Atomic Units ได้ที่ https://en.wikipedia.org/wiki/Hartree_atomic_units

สำหรับกรณีของระบบที่มีอิเล็กตรอนมากกว่าหนึ่งตัว เช่น อะตอมฮีเลียมที่มี 2 อิเล็กตรอน เราสามารถกระจายเทอมของ Hamiltonian ได้ดังนี้

$$\hat{H} = -\frac{1}{2M}\nabla^2_{\boldsymbol{R}} - \frac{1}{2}\nabla^2_{\boldsymbol{r_1}} - \frac{1}{2}\nabla^2_{\boldsymbol{r_2}} - \frac{2}{|\boldsymbol{r_1} - \boldsymbol{R}|} - \frac{2}{|\boldsymbol{r_2} - \boldsymbol{R}|} + \frac{1}{|\boldsymbol{r_1} - \boldsymbol{r_1}|} \quad (7.13)$$

โดยทั้ง 6 เทอมคือพลังงานจลน์ของนิวเคลียส, พลังงานจลน์ของอิเล็กตรอนตัวที่ 1, พลังงานจลน์ของอิเล็กตรอนตัวที่ 2, แรงดึงดูดระหว่างอิเล็กตรอนตัวที่ 1 และนิวเคลียส, แรงดึงดูดระหว่างอิเล็กตรอนตัวที่ 2 และนิวเคลียส, และแรงผลักระหว่างอิเล็กตรอน ตามลำดับ

นอกจากเราสามารถใช้การประมาณของบอร์น-ออปเพนไฮเมอร์ (Born-Oppenheimer (BO) Approximation) ซึ่งเป็นเทคนิคที่นำมาใช้เพื่อการประมาณว่า Wavefunction ของโมเลกุลนั้นขึ้นอยู่กับตำแหน่งของอิเล็กตรอนเพียงอย่างเดียวและไม่ขึ้นกับตำแหน่งของนิวเคลียสเนื่องจากว่ามวลของนิวเคลียสนั้นเยอะกว่ามวลของอิเล็กตรอนมาก ซึ่งถ้าหากใช้ BO Approximation กับ Hamiltonian ของอะตอมฮีเลียมนั้น เทอมแรกของสมการที่ (7.13) จะไม่ถูกนำมาพิจารณาในการคำนวณพลังงานของระบบ

7.3 การแก้สมการฟังก์ชันคลื่นเพื่อคำนวณพลังงาน

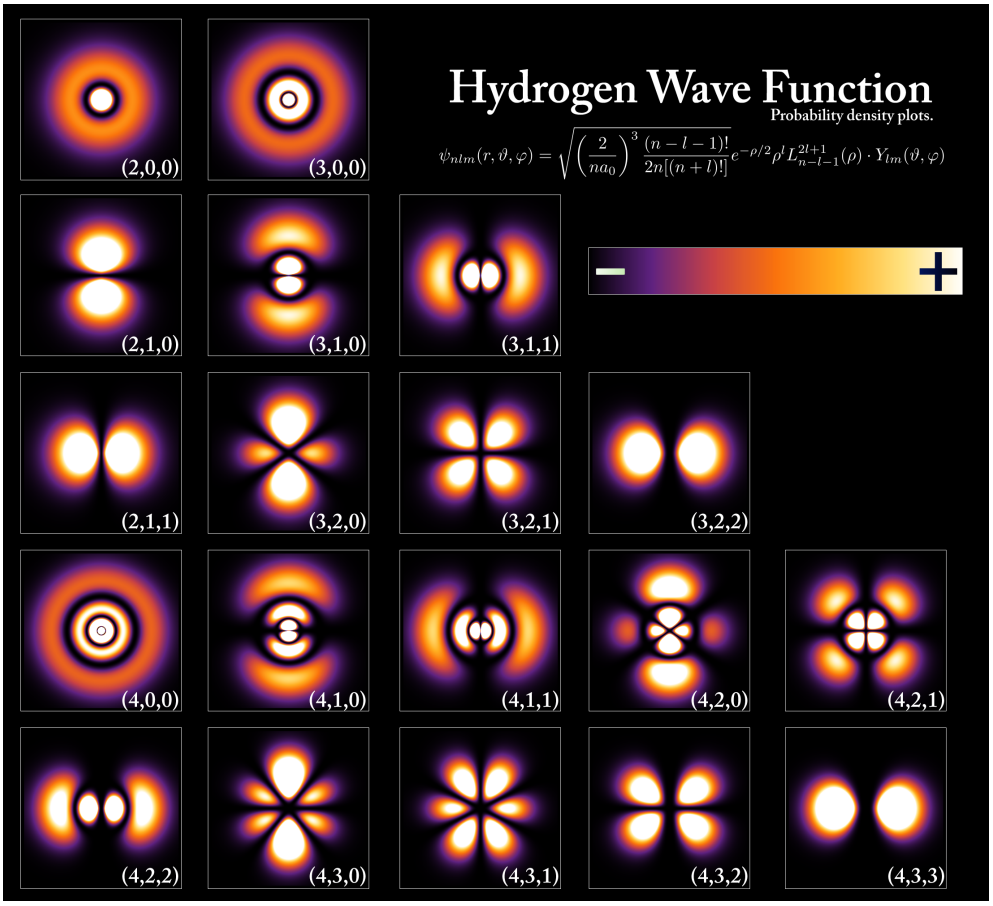
หนึ่งในเป้าหมายสำคัญของกลศาสตร์ควอนตัมเชิงโมเลกุล (Molecular Quantum Mechanics) ก็คือการหาวิธีแก้สมการ Time-independent Schrödinger Equation เพื่อให้ได้มาซึ่งคำตอบหรือผลเฉลยที่แม่นยำมากที่สุด ซึ่งจะช่วยให้นักเคมีคำนวณสามารถคำนวณคุณสมบัติโครงสร้างเชิงอิเล็กทรอนิกส์ (Electronic Structure) ของโมเลกุล โดยหัวข้อแรกของบทนี้ที่เราจะมาดูกันแบบละเอียดก็คือการใช้เทคนิคควอนตัมเชิงคำนวณและอาศัยการประมาณค่าในการแก้สมการดังกล่าว โดยทั่วไปนั้นจะมีวิธีการหลัก ๆ 2 วิธีที่สามารถช่วยให้เราหาคำตอบของสมการชโรดิงเงอร์ ได้นั่นคือ ***ab initio* method** ซึ่งเป็นวิธีที่ความแม่นยำของผลลัพธ์ที่ได้จากการแก้สมการนั้นจะขึ้นอยู่กับโมเดลที่เรานำมาใช้ในการอธิบาย Wavefunction ของระบบของเรา (โมเลกุลจะถูกมองเป็น Many-body System) วิธี *ab initio* ที่ได้มีการพัฒนากันมาตั้งแต่อดีตจนถึงปัจจุบันนั้นมีหลายวิธีมาก[59,60,57] วิธีที่ได้รับความนิยมมีดังไปนี้

วิธี Hartree-Fock

- Hartree-Fock (HF)
- Restricted open-shell Hartree-Fock (ROHF)
- Unrestricted Hartree-Fock (UHF)

วิธี Post-Hartree-Fock

- Møller-Plesset Perturbation Theory (MPn)



ภาพ 7.4 แบบจำลองของออร์บิทัลเชิงอะตอม (Atomic Orbitals) ของอิเล็กตรอนของอะตอมไฮโดรเจนที่ระดับพลังงานที่แตกต่างกัน โดยความเข้มของสีที่ไฮไลท์บ่งบอกถึงโอกาสที่จะพบอิเล็กตรอน ณ ตำแหน่งนั้น (เครดิตภาพ: https://en.wikipedia.org/wiki/Atomic_orbital)

- Configuration Interaction (CI)
- Coupled Cluster (CC)
- Quadratic Configuration Interaction (QCI)
- Quantum Chemistry Composite Methods

วิธี Multi-Reference

- Multi-Configurational Self-Consistent Field (MCSCF) รวมถึงวิธี CASSCF and RASSCF
- Multi-Reference Configuration Interaction (MRCI)
- n-electron Valence State Perturbation Theory (NEVPT)
- Complete Active Space Perturbation Theory (CASPTn)
- State Universal Multi-Reference Coupled-cluster Theory (SUMR-CC)

นอกจากนี้เป็นที่ทราบกันดีว่าสำหรับโมเลกุลที่มีขนาดใหญ่นั้นการคำนวณด้วยวิธี *ab initio* มีความสิ้นเปลืองสูงมาก (Computationally Expensive)[61] ดังนั้นจึงเป็นที่มาของการพัฒนาวิธีการที่สองนั่นคือ **Semiempirical method**[62,63,64] ซึ่งจะใช้แนวคิดในการตีความ Hamiltonian ในรูปแบบที่ง่ายกว่าซึ่งอ้างอิงด้วยออร์บิทัลเชิงโมเลกุล (Molecular Orbital หรือ MO) และอาศัยค่าพารามิเตอร์ที่ได้จากการทดลองเพื่อเพิ่มความแม่นยำ อย่างไรก็ตาม วิธี Density Functional Theory (DFT) ก็ถูกพัฒนาขึ้นมาเพื่อแก้ปัญหาที่เราจะต้องมาแก้หรือประมาณค่า Wavefunction ตรง ๆ ซึ่งทำได้ยากโดยเฉพาะกรณีที่ระบบมีหลายอิเล็กตรอน ดังนั้นในปัจจุบันการคำนวณเชิงควอนตัมส่วนใหญ่จึงเป็นการใช้ DFT เพราะว่ามีความสิ้นเปลืองของการคำนวณที่ต่ำมากเมื่อเทียบกับสองวิธีข้างต้นที่ได้กล่าวไปนั่นเอง

ตัวอย่างของความสำเร็จในการแก้สมการ Wavefunction ก็คือผลลัพธ์ที่แน่นอนของคุณสมบัติของระบบ กรณีที่เราสามารถหาผลเฉลยได้แน่นอนก็คือระบบที่มีอิเล็กตรอน 1 ตัว ตามแสดงในภาพที่ 7.4 ซึ่งเป็นแบบจำลองของออร์บิทัลเชิงอะตอม (Atomic Orbital หรือ AO) ของอิเล็กตรอนของอะตอมไฮโดรเจน ผู้อ่านสามารถศึกษาการเขียนโค้ดสำหรับพล็อตออร์บิทัลของอะตอทไฮโดรเจนได้ที่ภาพผนวกหัวข้อที่ E

7.3.1 วิธี Self-Consistent Field

ในหัวข้อนี้เราจะมาพูดถึงการแก้สมการชโรดิงเงอร์โดยใช้วิธีที่ชื่อว่า Self-Consistent Field (SCF) ซึ่งเป็นการประมาณค่า Hamiltonian แบบวนซ้ำ (เป็นที่มาของคำว่า *Self-Consistent* ซึ่งมีความหมายประมาณว่าเป็นดำเนินการเปรียบเทียบพารามิเตอร์ใหม่กับพารามิเตอร์เดิมโดยที่ยังคงใช้โมเดลอันเดียวกัน) เริ่มต้นเราจะต้องมาดูกันก่อนว่าการมอง Wavefunction ของระบบหลายอิเล็กตรอนสำหรับวิธี SCF นั้นจะมีการตัดสิ่งที่ซับซ้อนออกไปนั่นก็คืออันตรกิริยาแรงผลักระหว่างอิเล็กตรอน (Electron-electron Repulsion) โดย Wavefunction สามารถถูกอธิบายได้ด้วยสมการชโรดิงเงอร์ที่ไม่ขึ้นกับเวลา ดังต่อไปนี้[56]

$$H^{\circ}\Psi^{\circ} = E^{\circ}\Psi^{\circ} \tag{7.14}$$

โดยกำหนดให้ $H^{\circ} = \sum_{i=1}^{N} h_i$ เมื่อ h คือ Hamiltonian สำหรับอิเล็กตรอนตัวที่ i ในระบบที่มี

อิเล็กตรอน N ตัว นั่นคือสมการสำหรับระบบที่มีอิเล็กตรอน N ตัวนั้น จะสามารถถูกแยกออกมาได้เป็นสมการของระบบหนึ่งอิเล็กตรอนได้ N สมการและ Wavefunction ของอิเล็กตรอนหนึ่งตัวนั้นจริง ๆ แล้วก็คือออร์บิทัล (Orbital) เราจึงสามารถเขียนสมการของอิเล็กตรอนหนึ่งตัวโดยอ้างอิงจากสมการที่ (7.14) ได้เป็นสมการที่จำเพาะเจาะจงมากขึ้น ดังนี้

$$h_i \Psi^\circ(i) = E_m^\circ \Psi^\circ(i) \tag{7.15}$$

โดยที่ E_m° คือพลังงานของอิเล็กตรอนหนึ่งตัวใน MO ซึ่งเขียนแทนด้วย m นั่นเอง สำหรับระบบที่อิเล็กตรอนไม่ขึ้นต่อกัน

ด้วยเหตุนี้ Wavefunction รวมของระบบ (Ψ°) จึงสามารถเขียนให้อยู่ในรูปของ Wavefunction ของอิเล็กตรอนหนึ่งตัวได้ดังนี้

$$\Psi^\circ = \psi_a^\circ(1)\psi_b^\circ(1)\ldots\psi_z^\circ(N) \tag{7.16}$$

ซึ่ง Wavefunction ด้านบนนี้จะขึ้นอยู่กับพิกัดของอิเล็กตรอนทุกตัวและขึ้นกับตำแหน่งของนิวเคลียสหรืออะตอมด้วย[1]

สำหรับกระบวนการหรือขั้นตอนที่เราจะนำมาใช้ในการแก้สมการของระบบอิเล็กตรอนหลายตัวนั้น เราจะพิจารณาสมการรูทฮาน (Roothaan Equation) เป็นหลัก ซึ่งเป็นวิธีหนึ่งในการแก้สมการ Hartree-Fock (HF) ซึ่งมีการกำหนดตัวดำเนินการใหม่ขึ้นมาใช้แทน Hamiltonian นั่นก็คือ Fock Operator โดยที่ Fock Operator (f_1) ถูกนิยามในเทอมของ Coulomb Operator และ Exchange Operator ขึ้นมา นั่นก็คือ Fock Operator ซึ่งเขียนสมการสำหรับอิเล็กตรอน 1 ตัวได้เป็น

$$f_1 \psi_m(1) = \varepsilon_n \psi_m(1) \tag{7.17}$$

7.3.2 สมการ Roothaan

สำหรับการแก้สมการ HF ตรง ๆ โดยใช้ SCF นั้นสามารถทำได้ตรง ๆ ด้วยวิธีการเชิงตัวเลข (Numerical Method) แต่ว่าผลเฉลยที่ได้มานั้นมีความซับซ้อนมาก โดยในเวลาต่อมานักฟิสิกส์และนักเคมีชาวดัตช์ที่ชื่อว่า Clemens C.J. Roothaan จึงได้เสนอวิธีการใหม่สำหรับการอธิบาย MO โดยเรียกวิธีนั้นว่าผลรวมเชิงเส้น (Linear Combination of Atomic Orbitals หรือ LCAO)[58] เรามาดูรายละเอียดของ LCAO กันครับ

เริ่มต้นเราจะนิยามฟังก์ชันพื้นฐานหรือฟังก์ชันพื้นฐาน (Basis Function) สำหรับระบบที่มีอิเล็กตรอน N ตัวขึ้นมาก่อน ซึ่งเขียนแทนด้วย χ_o ซึ่งไอเดียตอนนี้ก็คือเราจะมองว่า Basis Function แบบที่ง่ายที่สุดที่

[1] ตอนนี้เราจะยังไม่พิจารณาสปินของอิเล็กตรอนที่จะต้องสอดคล้องและไม่ขัดกับหลักกีดกันของเพาลี (Pauli Exclusion) ซึ่งจะมีการรวม Spin-orbital สำหรับ Molecular Orbital m (φ_m) เข้าไปด้วย

เราสามารถนำมาใช้ได้นั้นก็คือ AO โดยเราสามารถเขียนฟังก์ชันคลื่นเชิงพื้นที่ (Spatial Wavefunction) ซึ่งเป็น Wavefunction ที่ขึ้นกับตำแหน่งของ AO ให้อยู่ในผลรวมเชิงเส้นของการคูณระหว่างสัมประสิทธิ์เชิงเส้นที่เรายังไม่ทราบค่า (c_{om}) กับ Basis Function χ_o ได้ดังนี้

$$\psi_m = \sum_{o=1}^{N_o} c_{om}\chi_o \tag{7.18}$$

เมื่อเราแทนสมการ (7.18) เข้าไปในสมการ (7.17) เราจะได้

$$f_1 \sum_{o=1}^{N_o} \chi_o(1) = \varepsilon \sum_{o=1}^{N_o} c_{om}\chi_o(1) \tag{7.19}$$

แล้วทำการคูณสมการ (7.19) ทั้งสองข้างด้วย $\chi_o^*(1)$ และทำการอินทิเกรตทั่วทั้ง Space ซึ่งจะทำให้เราได้ความสัมพันธ์ต่อไปนี้

$$\sum_{o=1}^{N_o} c_{om} \int \chi_o^*(1) f_1 \chi_o(1) d\tau_1 = \varepsilon_m \sum_{o=1}^{N_o} c_{om} \int \chi_o^*(1)\chi_o(1) d\tau_1 \tag{7.20}$$

จากสมการข้างต้นเราจะพบว่าจะมีผลคูณของ Basis Function ทั้งสองฝั่ง โดยทางฝั่งซ้ายนั้นเราสามารถนิยามเมทริกซ์ฟ็อกก์หรือ Fock Matrix (F) ได้

$$F_{o'o} = \int \chi_{o'}^*(1) f_1 \chi_o(1) d\tau_1 \tag{7.21}$$

และทางฝั่งขวา เรานิยามสิ่งที่เรียกว่าเมทริกซ์ซ้อนทับหรือ Overlap Matrix (S) ซึ่งเป็นเมทริกซ์ที่อธิบายถึงการซ้อนทับกันระหว่างสถานะ 2 สถานะ

$$S_{o'o} = \int \chi_{o'}^*(1)\chi_o(1) d\tau_1 \tag{7.22}$$

ซึ่งเราสามารถเขียนสมการ (7.20) ให้อยู่ในรูปของสมการที่เรียกว่า Roothaan Equation ได้กระชับ ๆ ดังนี้

$$Fc = \varepsilon Sc \tag{7.23}$$

โดยที่ c คือเมทริกซ์ขนาด $N_o \times N_o$ ซึ่งประกอบไปด้วยสมาชิกของ Coefficient c_{om} และ ε คือเมทริกซ์ที่มีขนาด $N_o \times N_o$ เช่นเดียวกันซึ่งเป็นเมทริกซ์แบบ Diagonal Matrix (สมาชิกที่ไม่ใช่แนวทแยงมีค่าเป็น

0 ทั้งหมด) ซึ่งก็คือพลังงานของ Orbital นั่นเอง ซึ่งตรงจุดนี้เราต้องไม่ลืมว่า Fock Operator (f_1) นั้นถูกกำหนดให้อยู่ในรูปของ Integral บน MO และขึ้นอยู่กับค่าของ Coefficient c_{om} ด้วย

สำหรับการแก้สมการ (7.23) นั้นสามารถทำได้ผ่าน Determinant ดังนี้

$$det|F - \varepsilon S| = 0 \tag{7.24}$$

เนื่องจากว่าสมการที่ (7.24) นั้นไม่สามารถถูกแก้ได้แบบตรงไปตรงมาเพราะว่าสมาชิกของเมทริกซ์ $F_{o'o}$ นั้นเกี่ยวเนื่องโดยตรงกับ Integral ของ Coulomb Operator และ Exchange Operator ซึ่งขึ้นอยู่กับ Spatial Wavefunction ดังนั้นจึงทำให้การแก้สมการที่ (7.24) นั้นเป็นปัญหาแบบงูกินหาง อย่างไรก็ตามเราสามารถใช้กระบวนการวนซ้ำ (Iterative Method) ในการแก้สมการนี้ (เรียกว่าเป็นการประมาณค่าก็ได้) จนกว่าคำตอบหรือผลลัพธ์ที่เราต้องการ (พลังงาน) จากสมการนั้นลู่เข้า

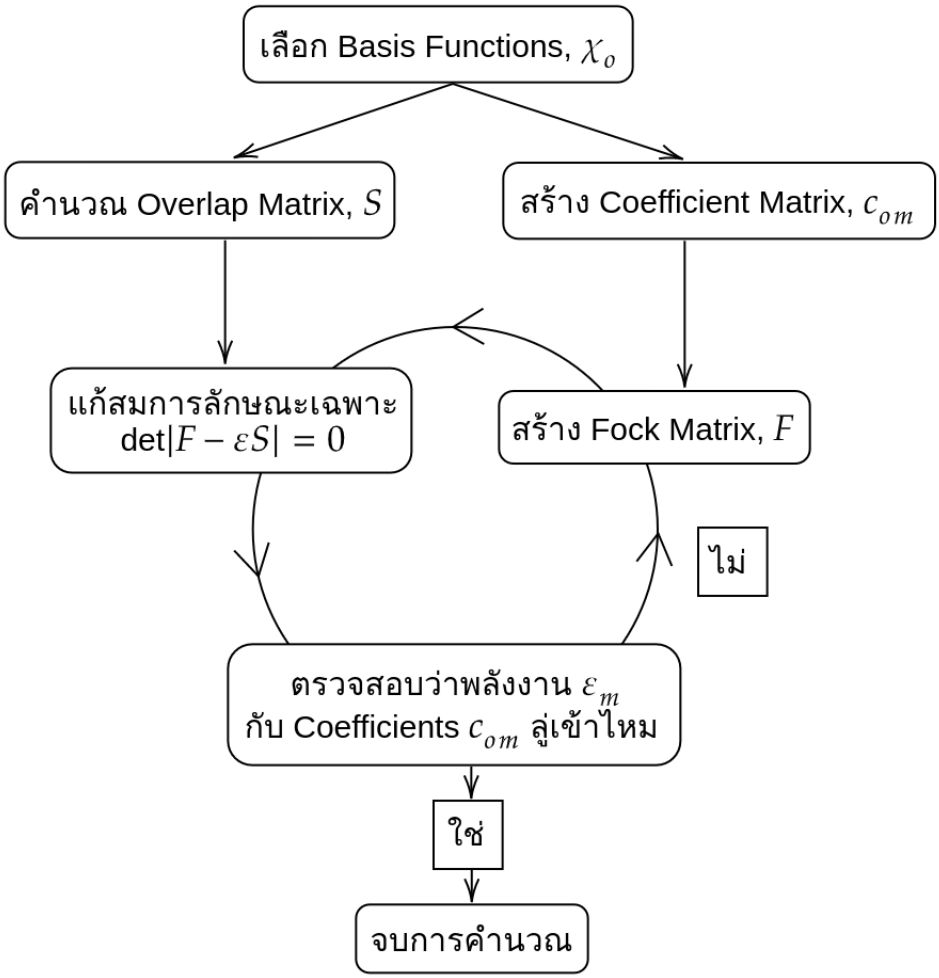
7.3.3 การแก้สมการ Roothaan ด้วย Self-Consistent Field

ภาพที่ 7.5 แสดงแผนผังอัลกอริทึมของวิธี SCF โดยเริ่มจากการเลือก Atomic Basis Function ซึ่งถือว่าเป็นองค์ประกอบหลักของการนำไปสร้าง (Formulate) S โดยใช้สมการ (7.22) กับ c_{om} ซึ่งเราจะใช้วิธีการสร้างค่าเริ่มต้นด้วยวิธี Guess ซึ่งมีด้วยกันหลายวิธี เช่น

1. **Hückel guess** : ใช้ Hückel Orbital[57]
2. **Superposition of Atomic Densities (SAD)** : ใช้ผลรวมของ Atomic Density ในการสร้าง Density Matrix
3. **Generalized Wolfsberg-Helmholtz (GWH)** : เป็นวิธีการที่อาศัย Hückel Theory โดยการใช้ Overlap Matrix และ Core Hamiltonian[65]
4. **CORE** : ทำการทำ Core Hamiltonian ให้เกิดเมทริกซ์รูปทแยง (Diagonalization)
5. **Harris** : ใช้ Harris Functional ซึ่งเป็น Non-Self-Consistent Approximation สำหรับ Kohn-Sham Orbital[66]

ซึ่งโปรแกรมเคมีเชิงคำนวณต่างก็มีการเลือกใช้ Guess Method สำหรับการเดา Coefficient หรือ Wavefunction เริ่มต้นในการแก้ SCF แตกต่างกันไป โปรแกรม Gaussian ใช้วิธี Harris สำหรับการคำนวณ HF และ DFT และใช้ Hückel หรือ CORE สำหรับ Semiempirical Methods, โปรแกรม Q-Chem และ Psi4 ใช้วิธี SAD กับ GWH เป็นวิธีเริ่มต้นโดยอัตโนมัติ เป็นต้น

หลังจากสร้าง Coefficient Matrix ขั้นตอนต่อไปคือการสร้าง Fock Matrix (F) โดยใช้สมการ (7.21) หลังจากนั้นเราจะทำการแก้สมการลักษณะเฉพาะ (Secular Equation) สมการที่ (7.24) เพื่อหา Energy Matrix แล้วก็ทำการวนซ้ำขั้นตอนการสร้าง S กับ F ไปปรับหาค่าพลังงานไปเรื่อย ๆ จนกว่าค่าความคลาด



ภาพ 7.5 แผนผังขั้นตอนของการประมาณค่าหาพลังงานของออร์บิทัลด้วยวิธี SCF

เคลื่อนหรือ Error จะมีค่าน้อยกว่าค่าที่กำหนดไว้ (Threshold) แล้วจึงสิ้นสุดกระบวนการ SCF เมื่อค่าพลังงานนั้นลู่เข้า

7.3.4 การคำนวณอนุพันธ์ของพลังงานและเมทริกซ์เฮสเซียน

หลังจากที่เราสามารถหาพลังงานเชิงอิเล็กทรอนิกส์ (Electronic Energy) ได้แล้ว ลำดับถัดไปที่เราสามารถคำนวณได้ก็คือคุณสมบัติต่าง ๆ ของโมเลกุล สิ่งแรกที่เราทำได้และถือว่าสำคัญมาก ๆ ในงานวิจัยทางด้านเคมีควอนตัมก็คือการหาโครงสร้างที่เหมาะสมหรือเสถียรที่สุดของโมเลกุลโดยใช้หลักเกณฑ์พลังงานรวมที่ต่ำที่สุด ซึ่งการที่เราทราบโครงสร้างที่เหมาะสมที่สุดนั้นมีประโยชน์อย่างมากเพราะเราสามารถนำผลการคำนวณไปเทียบกับผลจากการทดลองด้วยเทคนิค X-ray Crystallography, Electron Diffractiom, หรือ Microwave Spectroscopy เป็นต้น โดยการหาโครงสร้างที่สภาวะเหมาะสมหรือสมดุล (Equilibrium Structure) นั้นสามารถทำได้โดยหาอนุพันธ์ของพลังงานศักย์ของโมเลกุลเทียบกับพิกัดนิวเคลียร์ ซึ่งวิธีการที่เราสามารถนำมาหาอนุพันธ์เพื่อให้ได้ผลลัพเชิงวิเคราะห์ (Analytical Method) เรียกว่า Gradient Method ซึ่งเร็วและให้ผลลัพธ์ที่แม่นยำกว่าระเบียบวิธีเชิงตัวเลข (Numerical Method)

สำหรับอนุพันธ์ของพลังงานนั้นเราจะเริ่มต้นพิจารณากรณีแบบง่ายก่อนนั่นก็คือโมเลกุลที่มีอะตอมสองอะตอม โดยเราจะเขียนพลังงานศักย์ของโมเลกุลเป็น E ซึ่งจะมีเทอมที่เป็นแรงผลักระหว่างนิวเคลียสของทั้งสองอะตอมด้วย ซึ่งแรงผลักนี้จะขึ้นกับระยะห่างระหว่างนิวเคลียส (Internuclear Distance) หรือ R นอกจากนี้เรายังทราบอีกด้วยว่าสำหรับโครงสร้างที่อยู่ในสมดุลนั้น แรง (Force) ที่กระทำต่อนิวเคลียสโดยอิเล็กตรอนนั้นจะเท่ากับศูนย์ ซึ่งแรงดังกล่าวเป็นแรงย่อยมีนิยามคืออนุพันธ์อันดับที่หนึ่งของพลังงานศักย์เทียบกับพิกัดของนิวเคลียสที่ i

$$f_i = -\frac{\partial E}{\partial q_i} \tag{7.25}$$

$$= 0 \tag{7.26}$$

โดยการคำนวณหาอนุพันธ์ข้างต้นด้วยวิธีการวิเคราะห์หรือ Analytical Method นั้นเราจะต้องทำการคำนวณหาอนุพันธ์ของอินทิกรัลของอิเล็กตรอนหนึ่งตัวและอิเล็กตรอนสองตัว (One-electron กับ Two-electron Integrals) เทียบกับพิกัดนิวเคลียร์ นั่นคือเราจะต้องทำการหาอนุพันธ์ของ Basis Function นั่นเอง[1] ซึ่งเราสามารถทำได้ผ่านการใช้กฎลูกโซ่ (Chain Rule) โดยทำการหาอนุพันธ์ของพลังงานศักย์เทียบกับ Expansion Coefficient

ลำดับถัดมาคือการหาเมทริกซ์เฮสเซียน (Hessian Matrix) ซึ่งสามารถทำได้โดยการหาอนุพันธ์ย่อยอันดับที่สองของพลังงานศักย์เทียบกับนิวเคลียสของอะตอมตัวที่ i และ j $\left(\frac{\partial^2 E}{\partial q_i \partial q_j}\right)$ ซึ่งช่วยให้เราสามารถ

[1] Basis Function ก็คือ Basis ที่เกิดขึ้นมาจาก Atomic Orbtials ที่ถูก centered หรือมีตำแหน่งอยู่ที่จุดอ้างอิงของนิวเคลียสของอะตอมในโมเลกุล

ระบุได้ว่าค่าพลังงานที่คำนวณออกมาได้นั้นสอดคล้องกับจุดต่ำสุดหรือสูงสุดบนพื้นผิวพลังงานศักย์ (Potential Energy Surface หรือ PES) โดยจะสอดคล้องกับอนุพันธ์อันดับที่สองที่ได้ค่าออกมาเป็นบวก (สำหรับ Minimum Point) และลบ (สำหรับ Maximum Point) ตามลำดับ

7.3.5 จากอนุพันธ์ของพลังงานสู่คุณสมบัติเชิงโมเลกุล

คุณสมบัติเชิงโมเลกุลที่เกี่ยวข้องโครงสร้างเชิงอิเล็กทรอนิกส์นั้นเป็นสิ่งที่สำคัญและจำเป็นมากในการคำนวณทางด้านเคมีควอนตัม เพราะว่าคุณสมบัติหรือปริมาณเหล่านี้เป็นสิ่งที่เรานำไปใช้ในการศึกษาโมเลกุลและปฏิกิริยาเคมี แล้วเราสามารถนำผลการคำนวณไปเปรียบเทียบกับค่าที่วัดได้จากการทดลองเพื่อตรวจสอบและยืนยันความถูกต้องของทฤษฎีที่ใช้ในการคำนวณคุณสมบัตินั้น ๆ ด้วย ตามที่ได้อธิบายไปก่อนหน้านี้ว่าอนุพันธ์ของพลังงานนั้นเปรียบเสมือนเป็นกุญแจที่สามารถนำไปไขกล่องที่เก็บซ่อนความลับของโมเลกุลได้ โดยเราสามารถแบ่งความสำคัญของคุณบัติเชิงโมเลกุลออกได้เป็น 3 ประเภท ดังนี้

1. ความแตกต่างของพลังงาน (Energy Differences) เช่น พลังงานของปฏิกิริยา (Reaction Energies), พลังงานในการทำให้กลายเป็นอะตอม (Atomization Energies), พลังงานที่ใช้ในการสลายโมเลกุล (Dissociation Energies), พลังงานที่แตกต่างกันระหว่างคอนฟอร์เมอร์หรือไอโซเมอร์

2. คุณสมบัติเชิงโมเลกุลสำหรับสถานะเชิงอิเล็กทรอนิกส์ เช่น โครงสร้าง ณ สภาวะสมดุล (Equilibrium Structure), ไดโพลโมเมนต์ (Dipole Moment), ความสามารถในการมีสภาพขั้ว (Polarizability), ความถี่เชิงการสั่น (Vibrational Frequencies), ความสามารถในการมีสภาพแม่เหล็ก (Magnetazibility), NMR Chemical Shifts

3. คุณสมบัติที่บ่งบอกการทรานซิชั่นระหว่างสถานะเชิงอิเล็กทรอนิกส์ที่แตกต่างกันได้ เช่น พลังงานกระตุ้นเชิงอิเล็กทรอนิกส์ (Electronic Excitation Energies), ความเข้มของการทรานซิชั่นของโฟตอน 1 ตัวและ 2 ตัว (One- and two-photon Transition Strengths), ระยะเวลาชีวิตในการแผ่รังสี (Radiative Life Times), พลังงานศักย์ในการทำให้เกิดไอออน (Ionization Potentials)

โดยในหัวข้อนี้เราจะสนใจคุณสมบัติเชิงโมเลกุลประเภทที่ 2 ซึ่งเกี่ยวกับสถานะเชิงอิเล็กทรอนิกส์เป็นพิเศษ โดยต้องเกริ่นก่อนว่าคุณสมบัติเชิงโมเลกุลนั้นเกิดขึ้นจากการที่โมเลกุลมีการตอบสนอง (Response) ต่อสนาม (Field) ที่กระทำต่อโมเลกุลซึ่งมองได้ในรูปของอนุพันธ์อันดับต่าง ๆ ของพลังงาน เช่น อนุพันธ์สามอันดับแรก ดังนี้

- อนุพันธ์อันดับหนึ่ง: แรง (Force), ความเครียด (Stress), Dipole Moment เป็นต้น
- อนุพันธ์อันดับสอง: Dielectric Susceptibility, Polarizability, Born Effective Charges เป็นต้น
- อนุพันธ์อันดับสาม: Nonlinear Dielectric Susceptibility, (First-order) Hyperpolarizability เป็นต้น

โดยเราสามารถเขียนพลังงานที่อยู่ภายใต้สนามภายนอก (External Field) ในรูปของฟังก์ชันการกระจายของเทเลอร์ (Taylor Expansion) รอบ ๆ ตำแหน่งที่ไม่มีสนาม (Field-free) ได้ดังนี้

$$E(\epsilon) = E(\epsilon = 0) + \underbrace{\left.\frac{\mathrm{d}E}{\mathrm{d}\epsilon}\right|_{\epsilon=0}\epsilon}_{\text{First Response}} + \underbrace{\frac{1}{2}\left.\frac{\mathrm{d}^2E}{\mathrm{d}\epsilon^2}\right|_{\epsilon=0}\epsilon^2}_{\text{Second Response}} + \ldots \tag{7.27}$$

สำหรับเทอมที่สองที่เป็นอนุพันธ์อันดับสองของพลังงานนั้นคือ Gradient ซึ่งเป็นฟังก์ชันแบบเส้นตรงสำหรับ (Linear) เราจึงเรียกคุณสมบัติที่ได้จากเทอมนี้ว่า Linear Response Properties ส่วนเทอมอื่น ๆ เช่น เทอมที่สามนั้นเป็นอนุพันธ์อันดับสองซึ่งจะเกี่ยวข้องกับฟังก์ชัน Quadratic นอกจากนี้เรายังสามารถสรุปได้ว่า

Dipole Moment (μ)	$=$	$-\left.\frac{\mathrm{d}E}{\mathrm{d}\epsilon}\right\|_{\epsilon=0}$
Polarizability (α)	$=$	$-\left.\frac{\mathrm{d}^2E}{\mathrm{d}\epsilon^2}\right\|_{\epsilon=0}$
First Hyperpolarizability (β)	$=$	$-\left.\frac{\mathrm{d}^3E}{\mathrm{d}\epsilon^3}\right\|_{\epsilon=0}$

โดยเราสามารถคำนวณคุณสมบัติเชิงโมเลกุลต่าง ๆ เหล่านี้ด้วยวิธีการคำนวณทั่วไป เช่น วิธี HF หรือ DFT เพื่อนำไปใช้เป็นลักษณะเฉพาะ (Feature) สำหรับการฝึกสอนโมเดล ML หรือนำมาใช้เป็นเอาต์พุตสำหรับการทำนายก็ได้

7.4 ทฤษฎีฟังก์ชันนอลความหนาแน่น

ตามที่ผู้เขียนได้พูดถึง *"ทฤษฎีฟังก์ชันนอลความหนาแน่น"* หรือ *"Density Functional Theory (DFT)"* ในบทก่อนหน้านี้แล้วว่าเป็นทฤษฎีที่มีความสำคัญมากในวงการวิทยาศาสตร์นั่นก็เพราะว่าทฤษฎี DFT ได้พลิกโฉมงานวิจัยที่เกี่ยวข้องกับการศึกษาโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลไปอย่างสิ้นเชิง แทนที่จะมองโมเลกุลเป็นระบบที่อิเล็กตรอนมีอันตรกิริยากันตรง ๆ แล้วใช้ Wavefunction ในการอธิบายระบบนั้น วิธี DFT จะมองโมเลกุลว่าเป็นกลุ่มก้อนของอิเล็กตรอนและใช้ความหนาแน่น (Density) ในการอธิบายแทน จึงทำให้เราไม่จำเป็นที่จะต้องแก้สมการเพื่อหาผลเฉลยแบบแม่นตรง (Exact Solution) ของ Wavefunction (ซึ่งเราไม่สามารถคำนวณหาบางเทอมของ Wavefunction ได้สำหรับกรณีที่ระบบมีอิเล็กตรอนมากกว่าหนึ่งตัว)

7.4.1 ฟังก์ชันและฟังก์ชันนอล

ก่อนที่ผู้อ่านจะได้ศึกษาในหัวข้อต่อไปซึ่งจะลงรายละเอียดมากกว่านี้ ผู้เขียนขอเริ่มด้วยการอธิบายความหมายและการใช้งานของสิ่งที่เรียกว่าฟังก์ชันและฟังก์ชันนอลก่อนครับ เพราะว่าการที่เราเข้าใจความหมายและความแตกต่างของคำศัพท์สองคำนี้จะเป็นพื้นฐานสำคัญในการเข้าใจทฤษฎี DFT ที่ว่าด้วยเรื่องของความหนาแน่นของอิเล็กตรอนที่ผู้อ่านจะได้ศึกษาในหัวข้อที่ 7.4.2 โดยฟังก์ชันกับฟังก์ชันนอลนั้นต่างกันอินพุต ดังนี้

- **ฟังก์ชัน (Function)** รับอินพุตที่เป็นตัวเลขและให้เอาต์พุตที่เป็นตัวเลขเช่นเดียวกัน โดยสามารถเขียนการ Mapping ได้เป็น $x_0 \mapsto f(x_0)$ โดยที่ x_0 คืออาร์กิวเมนต์หรืออินพุตของฟังก์ชัน f

  ตัวอย่างของฟังก์ชัน เช่น

$$f(x) = x^2$$
$$g(x,y) = \cos(x) + e^{-3\sqrt{x^2+y^2}}$$

- **ฟังก์ชันนอล (Functional)** เป็นฟังชันก์ชนิดหนึ่งซึ่งรับอินพุตที่เป็นฟังก์ชันและให้เอาต์พุตที่เป็นตัวเลข ซึ่งสรุปได้ง่าย ๆ ว่า “ฟังก์ชันนอลนั้นก็คือฟังก์ชันของฟังก์ชัน” โดยสามารถเขียนการ Mapping ได้เป็น $f \mapsto f(x_0)$ โดยที่ x_0 คือพารามิเตอร์

  ตัวอย่างของฟังก์ชันนอล เช่น

$$F[f] = \int_{-\infty}^{\infty} f^3(x)\,dx$$
$$H[g] = \int_{2}^{3}\int_{-10}^{4} \left(\frac{\partial^2 g(x,y)}{\partial x^2} - 2.3g(x,y) \right) dxdy$$

เมื่อทราบความแตกต่างแล้วผู้อ่านก็น่าจะพอเดาออกแล้วว่าคำว่า “Functional” ในชื่อของทฤษฎี Density Functional Theory นั้นบ่งบอกว่าเป็นทฤษฎีที่อธิบายโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลด้วยฟังก์ชันนอลที่สามารถอธิบายความหนาแน่นของอิเล็กตรอนได้

7.4.2 จากฟังก์ชันคลื่นสู่ความหนาแน่นเชิงอิเล็กทรอนิกส์

ในการพิจารณาฟังก์ชันคลื่นของอิเล็กตรอนนั้นเราไม่สามารถพิจารณาแค่พิกัดหรือตำแหน่งของอิเล็กตรอนเชิงพื้นที่ (Spatial Coordinates) หรือ (x, y, z) แต่ยังต้องพิจารณาตำแหน่งของสปิน (Spin Coordinates) หรือ ω ด้วย ดังนั้นจำนวนตัวแปรที่ส่งผลต่อ Wavefunction จึงมีทั้งหมด 4 ตัวแปรต่อหนึ่งอิเล็กตรอน ถ้าหากระบบของเรามี N อิเล็กตรอน จำนวนตัวแปรของ Wavefunction ก็จะเท่ากับ $4N_{\text{electrons}}$

เพื่อทำให้ชีวิตง่ายขึ้น แนวคิดในการใช้ความหนาแน่นสำหรับศึกษาโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลแทนที่จะใช้ Wavefunction โดยตรงนั้นจึงได้รับความสนใจและถูกพัฒนาเรื่อยมาจนถึงปัจจุบัน ข้อดีของการอธิบายระบบ (โมเลกุล) ด้วยความหนาแน่นแทนที่จะใช้ Wavefunction นั้นช่วยทำให้ลดความสิ้นเปลืองในการคำนวณไปได้เยอะมากเพราะความหนาแน่นนั้นสามารถถูกเขียนด้วยฟังก์ชันที่ขึ้นอยู่กับตัวแปรเพียงแค่ 3 ตัวเท่านั้น (สำหรับกรณีที่ไม่พิจารณาสปินของอิเล็กตรอน) กล่าวคือสำหรับ Wavefunction ที่เขียนด้วย Schrödinger Equation นั้นจะเป็นฟังก์ชันที่มีจำนวนมิติคือ $3N_{\text{electron}}$ แต่สำหรับความหนาแน่นนั้นเราจะได้สมการที่มีจำนวนมิติคือ 3 มิติด้วยกันทั้งหมดจำนวน N สมการ (ตามจำนวนอิเล็กตรอน) ซึ่งจะเห็นได้ว่าความซับซ้อนในการคำนวณจะต่างกันอย่างมาก โดยสรุปเป็นความสัมพันธ์ได้ดังนี้

$3N$-dimensional Schrödinger Equation
$\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3, \ldots, \boldsymbol{r}_N)$
$\downarrow$
N 3-dimensional Single Particle Equation
$n(\boldsymbol{r})$

โดยที่ Single Particle Equation ในที่นี้คือสมการที่ใช้ในการอธิบายอนุภาค 1 ตัวซึ่งก็คืออิเล็กตรอนนั่นเอง

จริง ๆ แล้วความหนาแน่นเชิงอิเล็กทรอนิกส์ก็คือความหนาแน่นของอิเล็กตรอน (Electron Density) หรือ $n(\boldsymbol{r})$ ซึ่งเป็นหัวใจสำคัญของทฤษฎี DFT เลยก็ว่าได้ โดยความน่าจะเป็นของโอกาสที่จะพบอิเล็กตรอนตัวที่ 1 ของระบบหรือโมเลกุลที่มี N อิเล็กตรอนนั้นสามารถคำนวณได้จากการใช้ปริพันธ์ (Integral) ตามสมการต่อไปนี้

$$P(\mathbf{1}) = \left[\int d^3\boldsymbol{r}_2 \cdots \int d^3\boldsymbol{r}_N \, \psi^*(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N)\psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N) \right] d^3\boldsymbol{r}_1 \qquad (7.28)$$

เนื่องจากว่าอิเล็กตรอนทุก ๆ ตัวนั้นมีคุณสมบัติเหมือนกันหมดทุกประการ (Indistinguishable) ดังนั้นความหนาแน่นของความน่าจะเป็น (Probability Density) ของอิเล็กตรอนแต่ละตัวก็จะเท่ากันด้วย หมายความว่าความน่าจะเป็นของความหนาแน่นของอิเล็กตรอนตัวที่ 1 ก็เท่ากับของตัวที่ 2, ตัวที่ 3, ไปจนถึงตัวที่ N ดังนั้นในการคำนวณหาความหนาแน่นของความน่าจะเป็น (Probability Density) ของอิเล็กตรอนทั้งหมดนั้นจึงสามารถทำได้โดยการรวมแบบเชิงเส้น นั่นก็คือเราคูณความน่าจะเป็นของโอกาสที่จะพบอิเล็กตรอน 1 ตัวด้วย N นั่นเอง โดยเราจะได้สมการดังนี้

$$n(\boldsymbol{r}) = N \underbrace{\int d^3\boldsymbol{r}_2 \cdots \int d^3\boldsymbol{r}_N \, \psi^*(\boldsymbol{r}, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N)\psi(\boldsymbol{r}, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N)}_{\text{ความน่าจะเป็นที่จะพบอิเล็กตรอน 1 ตัว}} \qquad (7.29)$$

โดยที่ ψ คือฟังก์ชันคลื่นที่ผ่านการถูกทำให้เป็นปกติ (Normalized Wavefunction) มาแล้ว ซึ่งความหมาย

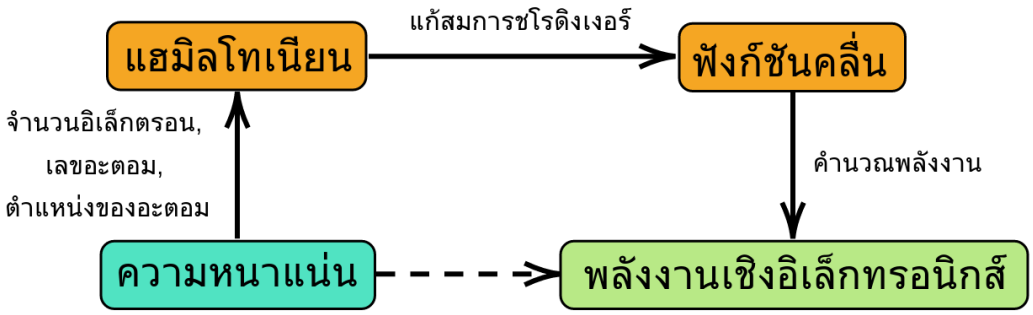
ของการทำให้เป็นปกติ (Normalization) ก็คือการหารูปแบบ (Form) ของ Wavefunction ที่สอดคล้องกับเงื่อนไขดังต่อไปนี้

$$\int_{-\infty}^{\infty} \psi^* \psi dx = 1 \tag{7.30}$$

อย่างไรก็ตาม ไม่ใช่ทุก Wavefunction ที่สามารถทำ Normalization ได้ ตัวอย่างของฟังก์ชันที่เป็นข้อยกเว้น เช่น Planewave Wavefunction ซึ่งเป็นฟังก์ชันที่ขึ้นกับพิกัดและเวลา ดังนี้ $\psi(x,t) = \psi_0 \mathrm{e}^{\mathrm{i}(kx-\omega t)}$ ไม่เป็น Square-integrable Function

7.4.3 จากความหนาแน่นเชิงอิเล็กทรอนิกส์สู่พลังงานของระบบ

เมื่อเราเข้าใจนิยามและไอเดียของความหนาแน่นเชิงอิเล็กทรอนิกส์หรือความหนาแน่นของอิเล็กตรอนแล้ว ลำดับต่อไปก็คือเราจะคำนวณพลังงานของระบบ (โมเลกุล) โดยใช้ความหนาแน่นได้อย่างไร ซึ่งตามทฤษฎีนั้นเราสามารถคำนวณพลังงานได้แบบอ้อม ๆ ผ่าน Wavefunction



ภาพ 7.6 ความเชื่อมโยงแบบตรงและแบบอ้อมระหว่างความหนาแน่นของอิเล็กตรอนและพลังงานเชิงอิเล็กทรอนิกส์ของระบบ

จากไดอะแกรมที่แสดงในภาพที่ 7.6 นั้นสามารถตีความได้ว่าเราสามารถคำนวณพลังงานของระบบโดยผ่าน Hamiltonian และ Wavefunction ได้ซึ่งก็จะมีความซับซ้อนในเชิงคำนวณ ดังนั้นคำถามสำคัญที่ตามมาก็คือ “จะเป็นไปได้ไหมที่เราจะคำนวณพลังงานจากความหนาแน่นของอิเล็กตรอนตรง ๆ” ซึ่งคำตอบก็คือจริง ๆ แล้วไม่สามารถหาได้ตรง ๆ แต่เรามีทริคที่สามารถทำได้ดังต่อไปนี้

เริ่มจากการกำหนดให้พลังงานเชิงอิเล็กทรอนิกส์ได้จากการคำนวณค่า Expectation Value (ค่าเฉลี่ย) ของ Hamiltonian Operator

$$E_{\text{el}} = \int \cdots \int \Psi^* \hat{H}_{\text{el}} \Psi \, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_{N_{\text{el}}} \tag{7.31}$$

โดยที่ Hamiltonian Operator $(\hat{H}_{\text{el}})$ สำหรับอิเล็กตรอนมีสมการดังต่อไปนี้

$$\hat{H}_{\text{el}} = \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \underbrace{\sum_{i=1}^{N_{\text{el}}} \sum_{A=1}^{N_{\text{nu}}} \frac{-Z_A}{|\boldsymbol{r}_i - \boldsymbol{R}_A|}}_{\text{Nuclear Attraction Energy}} \tag{7.32}$$

ซึ่งเทอมที่ 3 ของสมการที่ (7.32) นั้นคือพลังงานดึงดูดระหว่างอิเล็กตรอนกับนิวเคลียสซึ่งมีชื่อเรียกอีกชื่อว่า “ศักย์ภายนอก” (External Potential) โดยเป็นคำศัพท์ที่ใช้ในทฤษฎี DFT ซึ่งคำว่า External นี้มาจากการที่เราใช้การประมาณของ Born-Oppenheimer ซึ่งเป็นการกำหนดให้นิวเคลียสนั้นเป็นวัตถุที่ถูกตรึงอยู่กับที่ (Fixed) และทำให้เกิดพลังงานศักย์คูลอมป์ (Coulomb Potential) ต่ออิเล็กตรอน ดังนั้นจากสมการที่ (7.32) เราจึงเขียนใหม่ได้เป็น

$$\begin{aligned}
\hat{H}_{\text{el}} &= \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \sum_{i=1}^{N_{\text{el}}} \underbrace{\left(\sum_{A=1}^{N_{\text{nu}}} \frac{-Z_A}{|\boldsymbol{r}_i - \boldsymbol{R}_A|} \right)}_{\text{Nuclear Attraction Energy}} \\
&= \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \sum_{i=1}^{N_{\text{el}}} V_{\text{ext}}(\boldsymbol{r}_i)
\end{aligned} \tag{7.33}$$

โดยที่มี External Potential $V_{\text{ext}}(\boldsymbol{r}_i)$ กระทำต่ออิเล็กตรอนทุกตัวในโมเลกุล

ลำดับต่อไปก็คือเราลองมาทำการกระจาย Expectation Value ของพลังงานเชิงอิเล็กทรอนิกส์โดยการแทนสมการที่ (7.33) เข้าไปในสมการที่ (7.31) ซึ่งเราจะได้พลังงานที่ประกอบไปด้วย 3 เทอม ดังนี้

$$\begin{aligned}
E_{\text{el}} = &\int \cdots \int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 \right) \Psi \, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_{N_{\text{el}}} \\
&+ \int \cdots \int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \right) \Psi \, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_{N_{\text{el}}} \\
&+ \underbrace{\int \cdots \int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} V_{\text{ext}}(\boldsymbol{r}_i) \right) \Psi \, d\boldsymbol{x}_1 \ldots d\boldsymbol{x}_{N_{\text{el}}}}_{\int V_{\text{ext}}(\boldsymbol{r}) n(\boldsymbol{r}) \, d\boldsymbol{r}}
\end{aligned} \tag{7.34}$$

โดยสมการที่ (7.34) มีเพียงแค่เทอมที่ 3 เท่านั้นที่สามารถเขียนให้อยู่ในรูปของฟังก์ชันนอลของความหนาแน่นได้ (Explicit Functional of Density)

7.4.4 ความสัมพันธ์ระหว่างความหนาแน่นของอิเล็กตรอนและศักย์ภายนอก

PHYSICAL REVIEW VOLUME 136, NUMBER 3B 9 NOVEMEBR 1964

Inhomogeneous Electron Gas*

P. HOHENBERG†
École Normale Superieure, Paris, France
AND
W. KOHN‡
École Normale Superieure, Paris, France and Faculté des Sciences, Orsay, France
and
University of California at San Diego, La Jolla, California

(Received 18 June 1964)

This paper deals with the ground state of an interacting electron gas in an external potential $v(\mathbf{r})$. It is proved that there exists a universal functional of the density, $F[n(\mathbf{r})]$, independent of $v(\mathbf{r})$, such that the expression $E\equiv\int v(\mathbf{r})n(\mathbf{r})d\mathbf{r}+F[n(\mathbf{r})]$ has as its minimum value the correct ground-state energy associated with $v(\mathbf{r})$. The functional $F[n(\mathbf{r})]$ is then discussed for two situations: (1) $n(\mathbf{r})=n_0+\tilde{n}(\mathbf{r})$, $\tilde{n}/n_0<<1$, and (2) $n(\mathbf{r})=\varphi(\mathbf{r}/r_0)$ with φ arbitrary and $r_0\to\infty$. In both cases F can be expressed entirely in terms of the correlation energy and linear and higher order electronic polarizabilities of a uniform electron gas. This approach also sheds some light on generalized Thomas-Fermi methods and their limitations. Some new extensions of these methods are presented.

ภาพ 7.7 บทคัดย่อของบทความงานวิจัยที่นำเสนอทฤษฎีบท Hohenberg-Kohn ในปี ค.ศ. 1964

สำหรับระบบที่มีจำนวนอิเล็กตรอน N ตัวนั้น ศาสตราจารย์ Pierre Hohenberg (New York University) และศาสตราจารย์ Walter Kohn (University of California at Santa Barbara) ได้เสนอทฤษฎีบทที่เป็นรากฐานสำคัญของทฤษฎี DFT ในปี ค.ศ. 1964 นั่นก็คือทฤษฎีบทโฮเฮนเบิร์ค-โคห์น (Hohenberg-Kohn Theorem)[67] ซึ่งเป็นทฤษฎีที่ว่าด้วยการพิสูจน์ความสัมพันธ์ระหว่างความหนาแน่นและศักย์ภายนอกว่าเป็นแบบหนึ่งต่อหนึ่ง (One-to-one) โดยใช้หลักการแปรค่า (Variational Principle) โดยบทความงานวิจัยฉบับนี้ถือว่ามีความสำคัญอย่างมากต่อวงการวิทยาศาสตร์โดยเฉพาะสาขาฟิสิกส์และเคมีเชิงโมเลกุล

$$
\begin{array}{ccc}
n^{(1)}(\boldsymbol{r}) & \underset{\text{H-K}}{\rightleftarrows} & V_{\text{ext}^{(1)}}(\boldsymbol{r}) \\
n^{(2)}(\boldsymbol{r}) & \underset{\text{H-K}}{\rightleftarrows} & V_{\text{ext}^{(2)}}(\boldsymbol{r}) \\
n^{(3)}(\boldsymbol{r}) & \underset{\text{H-K}}{\rightleftarrows} & V_{\text{ext}^{(3)}}(\boldsymbol{r}) \\
\cdots & & \cdots
\end{array}
$$

นอกจากนี้แล้ว Hohenberh และ Kohn ยังได้นำเสนอพลังงานเชิงอิเล็กทรอนิกส์ที่เขียนให้อยู่ในรูปของฟังก์ชันทั่วไป ดังนี้

$$E_{\text{el}} = \underbrace{\int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 \right) \Psi \, d\boldsymbol{X} + \int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \right) \Psi \, d\boldsymbol{X}}_{F_{\text{H-K}}[n]}$$
$$+ \underbrace{\int \Psi^* \left(\sum_{i=1}^{N_{\text{el}}} V_{\text{ext}}(\boldsymbol{r}_i) \right) \Psi \, d\boldsymbol{X}}_{\int V_{\text{ext}}(\boldsymbol{r}) n(\boldsymbol{r}) \, d\boldsymbol{r}} \tag{7.35}$$
$$= E_{\text{el}}[n] \tag{7.36}$$

โดยผลรวมของสองเทอมแรกนั้นคือ “ฟังก์ชันนอลสากล (Universal Functional)” หรือ $F_{\text{H-K}}[n]$ ซึ่งไม่ขึ้นกับศักย์ภายนอก อย่างไรก็ตาม ปัญหาก็คือเราไม่ทราบหน้าตาหรือผลเฉลยแบบแม่นตรงของฟังก์ชันนอลสากล แต่ว่าเรายังคงต้องการฟังก์ชันนอลนี้สำหรับการคำนวณพลังงานซึ่งสิ่งที่เราทำได้ก็คือการหาฟังก์ชันนอลสากลโดยใช้วิธีการประมาณนั่นเอง

7.4.5 ฟังก์ชันนอลสากลและทฤษฎีฟังก์ชันนอลความหนาแน่นแบบไร้ออร์บิทัล

สำหรับพลังงานเชิงอิเล็กทรอนิกส์ที่สามารถเขียนได้จากองค์ประกอบ 3 ส่วนคือ

$$E_{\text{el}}[n] = E_{\text{kin}}[n] + E_{\text{pot}}[n] + E_{\text{ext}}[n] \tag{7.37}$$

เราสามารถเขียนเทอมที่ 2 ของฟังก์ชันในสมการที่ (7.37) ให้อยู่ในรูปของผลรวมของพลังงานศักย์คูลอมบ์ (Coulomb Energy) หรือ $E_{\text{Col}}[n]$ และพลังงานของอินตรกิริยาระหว่างอิเล็กตรอนซึ่งก็คือพลังงานแลกเปลี่ยน (Exchange Energy) หรือ $E_{\text{X}}[n]$ และพลังงานสหสัมพันธ์ (Correlation Energy) หรือ $E_{\text{C}}[n]$ ได้ดังนี้

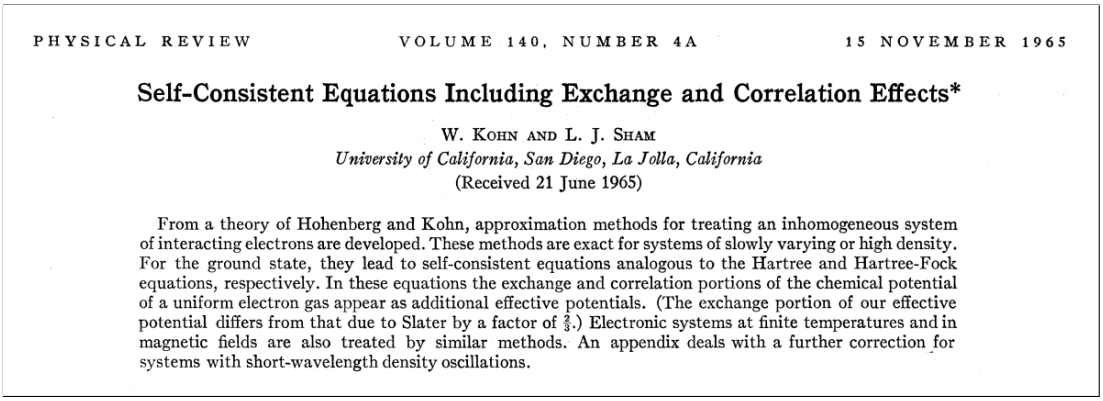
$$E_{\text{el}}[n] = E_{\text{kin}}[n] + \underbrace{E_{\text{Col}}[n] + E_{\text{X}}[n] + E_{\text{C}}[n]}_{E_{\text{pot}}[n]} + E_{\text{ext}}[n] \tag{7.38}$$

ซึ่งเทอมที่ 2 ที่เป็นพลังงานศักย์คูลอมบ์กับเทอมที่ 5 ที่เป็นศักย์ภายนอกนั้นเรารู้สมการของผลเฉลยแบบแม่นตรง ดังนี้

$$E_{\text{el}}[n] = \overbrace{E_{\text{kin}}[n]}^{\text{ไม่รู้}} + \overbrace{\frac{1}{2}\int\int \frac{n(\boldsymbol{r})n(\boldsymbol{r'})}{|\boldsymbol{r}-\boldsymbol{r'}|}\,d\boldsymbol{r}d\boldsymbol{r'}}^{\text{รู้}} + \overbrace{E_{\text{X}}[n]}^{\text{ไม่รู้}} + \underbrace{E_{\text{C}}[n]}_{\text{ไม่รู้}} + \underbrace{\int V_{\text{ext}}(\boldsymbol{r})n(\boldsymbol{r})\,d\boldsymbol{r}}_{\text{รู้}} \quad (7.39)$$

ส่วนเทอมที่ 1 (พลังงานจลน์), เทอมที่ 3 (พลังงานแลกเปลี่ยน), และเทอมที่ 4 (พลังงานสหสัมพันธ์) นั้นเราไม่รู้สมการที่แน่นอนซึ่งเป็นสิ่งที่ต้องประมาณค่าเอง และการประมาณค่าเพื่อหาฟังก์ชันของพลังงานทั้ง 3 เทอมนี้ที่แม่นยำที่สุดเท่าที่จะเป็นไปได้ก็เป็นหนึ่งในงานวิจัยที่ได้รับความสนใจจนถึงปัจจุบัน[68] เรียกได้ว่าตั้งแต่อดีตจนปัจจุบันได้มี Exchange-Correlation Functional ที่ถูกพัฒนาขึ้นมาและถูกทดสอบหลายร้อย Functional[69,70,71,72,73] สำหรับการศึกษาคุณสมบัติประเภทต่าง ๆ ของอะตอมและโมเลกุล[74,75,76,77,78,79,80]

วิธีข้างต้นที่คำนวณพลังงานเชิงอิเล็กทรอนิกส์โดยผ่านฟังก์ชันนอลสากล (Universal Functional) นั้นจะเรียกว่าฟังก์ชันนอลความหนาแน่นแบบบริสุทธิ์ (Pure DFT) ก็ได้เพราะว่าไม่มีการพิจารณาออร์บิทัล (Orbital-free) ซึ่งเป็นการคำนวณพลังงานของระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน (Interacting Electrons) ด้วยฟังก์ชันนอลของความหนาแน่น[81] ข้อดีของวิธี Orbital-free DFT คือมีความเรียบง่ายและไม่ซับซ้อนมากนัก (Simplicity) แต่ข้อด้อยก็คือมีความแม่นยำในการคำนวณที่ต่ำมากนั่นก็เพราะว่าการประมาณค่าของเทอมพลังงานจลน์ (เทอมแรกของสมการที่ (7.39)) นั้นทำได้ยากมากและขาดความแม่นยำในการประมาณ (เพราะว่าเทอม $\frac{1}{\boldsymbol{r}_i-\boldsymbol{r}_j}$ นั้นไม่สามารถถูกแยกแบ่งออกเป็นผลรวมของ $\boldsymbol{r}_i$ และ $\boldsymbol{r}_j$ ได้) เมื่อเราไม่สามารถประมาณค่าพลังงานจลน์ได้อย่างแม่นยำจึงทำให้พลังงานเชิงอิเล็กทรอนิกส์ที่คำนวณออกมานั้นมีความแม่นยำต่ำตามไปด้วย



PHYSICAL REVIEW VOLUME 140, NUMBER 4A 15 NOVEMBER 1965

Self-Consistent Equations Including Exchange and Correlation Effects*

W. KOHN AND L. J. SHAM
University of California, San Diego, La Jolla, California
(Received 21 June 1965)

From a theory of Hohenberg and Kohn, approximation methods for treating an inhomogeneous system of interacting electrons are developed. These methods are exact for systems of slowly varying or high density. For the ground state, they lead to self-consistent equations analogous to the Hartree and Hartree-Fock equations, respectively. In these equations the exchange and correlation portions of the chemical potential of a uniform electron gas appear as additional effective potentials. (The exchange portion of our effective potential differs from that due to Slater by a factor of $\frac{2}{3}$.) Electronic systems at finite temperatures and in magnetic fields are also treated by similar methods. An appendix deals with a further correction for systems with short-wavelength density oscillations.

ภาพ 7.8 บทคัดย่อของบทความงานวิจัยที่นำเสนอทฤษฎีบท Kohn-Sham ในปี ค.ศ. 1965

สำหรับการแก้ปัญหาดังกล่าวนั้น ในปี ค.ศ. 1965 ศาสตราจารย์ Walter Kohn และศาสตราจารย์ Lu Jeu Sham (University of California, San Diego) ก็ได้นำเสนอบทความงานวิจัย (หนึ่งปีหลังจากนำเสนอ

ทฤษฎี Pure (Orbital-free) DFT) โดยได้เสนอการคำนวณฟังก์ชันของพลังงานโดยใช้ระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกัน (Non-interacting Electrons) แทนการแก้ผ่านระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน[82] ซึ่งมีข้อดีคือทำให้ DFT มีความแม่นยำมากขึ้นเพราะว่าพลังงานจลน์ของระบบที่อิเล็กตรอนแต่ละตัวไม่ขึ้นหรือมีความสัมพันธ์กับอิเล็กตรอนตัวอื่น ๆ นั้นมีสมการที่เรารู้หน้าตาแน่นอน จึงไม่มีความจำเป็นที่จะต้องประมาณค่าฟังก์ชันนอลของพลังงานจลน์ในรูปของความหนาแน่นอีกต่อไป โดยในเวลาต่อมาทฤษฎีนี้คือ Kohn-Sham DFT นั่นเอง

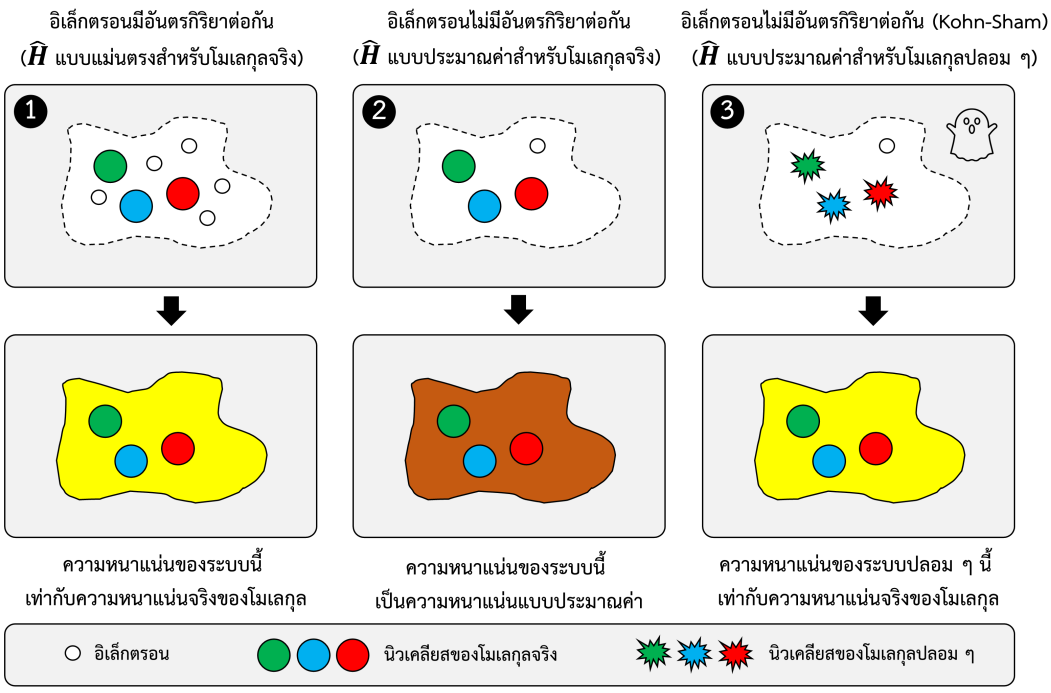
7.4.6 จาก Hohenberg-Kohn สู่ Kohn-Sham

ในหัวข้อนี้เราจะมารู้จักกับความแตกต่างระหว่างระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน (Interacting Electrons) และไม่มีอันตรกิริยาต่อกัน (Non-interacting Electrons) กันให้มากขึ้น เพราะว่าเป็นระบบที่ถูกนำมาใช้ในการพิจารณาความหนาแน่นของโมเลกุล

ตามที่ Kohn กับ Sham ได้เสนอการแก้ปัญหาของ Pure DFT โดยการเปลี่ยนมาพิจารณาระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันแทนนั้น จริง ๆ แล้ว Wavefunction และความหนาแน่นของทั้งสองระบบนั้นแตกต่างกันอย่างสิ้นเชิง แต่ว่าทริคของวิธี Kohn-Sham นั้นคือทำการจำลองหรือสร้างระบบอิเล็กตรอนที่ไม่มีอันตรกิริยาต่อกันแบบปลอม ๆ ขึ้นมา (Fictitious Non-interacting Electron System) หรืออาจจะเรียกว่าระบบอิเล็กตรอนแบบเสริมก็ได้ (Auxiliary Non-interacting Electron System)[83] โดยบังคับให้ความหนาแน่นของระบบนี้มีค่าเท่ากันกับความหนาแน่นของระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน ดังนั้นความท้าทายจึงเปลี่ยนจากการหาฟังก์ชันนอลสากลสำหรับ Pure DFT มาเป็นการหาระบบที่อิเล็กตรอนไม่มีอันตรกิริยากันแบบปลอม ๆ ที่มีความหนาแน่นเท่ากัน ซึ่งการใช้ทริคนี้ทำให้เราไม่ต้องมาประมาณค่าพลังงานจลน์และทำให้การคำนวณ DFT นั้นมีความแม่นยำมากขึ้นเพราะว่าเรามีผลเฉลยที่แน่นอนของพลังงานตามที่ได้อธิบายไว้ก่อนหน้านี้ในย่อหน้าสุดท้ายของหัวข้อที่ 7.4.5

เพื่อให้ผู้อ่านเข้าใจได้ง่ายขึ้นว่าทำไมระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันของ Kohn-Sham นั้นถึงมีความหนาแน่นเท่ากันกับระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน ให้ผู้อ่านเริ่มด้วยการศึกษาภาพที่ 7.9 ซึ่งเป็นการเปรียบเทียบระหว่างโมเดลของอิเล็กตรอนที่แตกต่างกัน

- ระบบที่ 1 คือระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน ซึ่งเราสามารถหาผลเฉลยแบบแม่นตรงของ Wavefunction ของระบบนี้ได้ และความหนาแน่นของโมเดลนี้จะเท่ากับความหนาแน่นของโมเลกุลจริงด้วย
- ระบบที่ 2 ระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกัน หมายความว่าอิเล็กตรอนแต่ละตัวจะมี Hamiltonian Operator เป็นของตัวเอง ซึ่งอิเล็กตรอนแต่ละตัวจะวิ่งอยู่ภายในสนามของศักย์เฉลี่ย (Average Potential) ที่เกิดจากอิเล็กตรอนตัวอื่นในระบบ สำหรับระบบนี้เราจะทำการรวม Hamiltonian ของอิเล็กตรอนแต่ละตัวเข้าด้วยกันเพื่อประมาณค่า Wavefunction สำหรับอิเล็กตรอนทุกตัว
- ระบบที่ 3 จะคล้ายกับระบบที่ 2 แต่จะมีความแตกต่างกันที่ศักย์เฉลี่ยที่กระทำต่ออิเล็กตรอน กล่าวคือในระบบนี้ (เรียกว่าระบบอิเล็กตรอน ของ Kohn-Sham ก็ได้) ศักย์เฉลี่ยที่เกิดขึ้นจะมาจากระบบของอิเล็กตรอนแบบปลอม ๆ (Fictitious System of Electrons) โดยเราจะทำการรวม Wavefunction ของอิเล็กตรอน (Molecular Orbitals) เข้าด้วยกันเพื่อประมาณค่า Wavefunction สำหรับ



ภาพ 7.9 การเปรียบเทียบแบบจำลองของ (1) ระบบที่อิเล็กตรอนมีอันตรกิริยาต่อกัน, (2) ระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกัน, และ (3) ระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันของโมเลกุลปลอมตามทฤษฎี Kohn-Sham

อิเล็กตรอนทุกตัว ซึ่งความหนาแน่นของระบบ Kohn-Sham นี้จะมีค่าเท่ากับความหนาแน่นของระบบที่เป็นโมเลกุลจริง ๆ นั่นคือเท่ากับความหนาแน่นของระบบที่ 1 ด้วย

7.4.7 แฮมิลโทเนียนสำหรับอิเล็กตรอนที่ไม่มีอันตรกิริยาต่อกัน

การที่เราเปลี่ยนมาพิจารณาระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันแทนนั้น เราสามารถเปลี่ยนเทอมที่เป็นพลังงานที่เกิดจากการผลักกันของอิเล็กตรอน (Direct Interaction) ให้เป็นโอเปอเรเตอร์สำหรับอิเล็กตรอน 1 ตัวได้ (เพราะว่าอิเล็กตรอนทุกตัวเป็นอิสระและไม่ขึ้นต่อกันอีกต่อไปแล้ว) ซึ่งโอเปอเรเตอร์ที่ว่านั้นคือพลังงานศักย์ที่อธิบายค่าเฉลี่ยของผลที่เกิดจากอันตรกิริยาระหว่างอิเล็กตรอน (Average Effect) อธิบายง่าย ๆ คือเราใช้เทอมนี้เป็นตัวแทนของอันตรกิริยาระหว่างอิเล็กตรอนในระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกัน (เป็นเทอมที่เป็นส่วนเติมเต็ม) โดยเราสามารถพิสูจน์แฮมิลโทเนียน (Hamiltonian) ของ Effective Interaction สำหรับอิเล็กตรอนที่ไม่มีอันตรกิริยาได้จาก Hamiltonian แบบดั้งเดิม ดังต่อไปนี้

$$\hat{H}_{\text{el}} = \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} \sum_{j=i+1}^{N_{\text{el}}} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \sum_{i=1}^{N_{\text{el}}} V_{\text{ext}}(\boldsymbol{r}_i) \tag{7.40}$$

สมการที่ (7.40) นี้จะเปลี่ยนเป็นสมการของ Hamiltonian สำหรับ Effective Interaction ได้ดังนี้

$$\begin{aligned}
\hat{H}_{\text{eff}} &= \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} V_{\text{aver}}(\boldsymbol{r}_i) + \sum_{i=1}^{N_{\text{el}}} V_{\text{ext}}(\boldsymbol{r}_i) \\
&= \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} \{V_{\text{aver}}(\boldsymbol{r}_i) + V_{\text{ext}}(\boldsymbol{r}_i)\} \\
&= \sum_{i=1}^{N_{\text{el}}} -\frac{1}{2}\nabla_i^2 + \sum_{i=1}^{N_{\text{el}}} V_{\text{eff}}(\boldsymbol{r}_i) \\
&= \sum_{i=1}^{N_{\text{el}}} \{-\frac{1}{2}\nabla_i^2 + V_{\text{eff}}(\boldsymbol{r}_i)\} \\
&= \sum_{i=1}^{N_{\text{el}}} \hat{h}(\boldsymbol{r}_i)
\end{aligned} \tag{7.41}$$

จากการพิสูจน์ข้างต้นจะได้ว่าสุดท้ายแล้ว Hamiltonian ทั้งหมดของระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันนั้นคือผลรวมของ Hamiltonian ของอิเล็กตรอนหนึ่งตัว (1 Hamiltonian ต่ออิเล็กตรอน 1 ตัว) กล่าวคือจากการพิสูจน์ของสมการที่ (7.41) เราจะได้ความสัมพันธ์ที่สั้นและกระชับ ดังนี้

$$\hat{H}_{\text{eff}} = \sum_{i=1}^{N_{\text{el}}} \hat{h}(\boldsymbol{r}_i) \tag{7.42}$$

นอกจากนี้เราสามารถเขียนและแก้ Schrödinger Equation สำหรับ Hamiltonian ของอิเล็กตรอนแต่ละตัวแยกกันได้ดังนี้

$$\hat{h}(\boldsymbol{r}_i) \underbrace{\psi_a(\boldsymbol{r}_i)}_{\text{MOs}} = \underbrace{\epsilon_a}_{\text{Energy}} \psi_a(\boldsymbol{r}_i) \tag{7.43}$$

ซึ่งจากตรงนี้เราสามารถคำนวณหาออร์บิทัลเชิงโมเลกุล (Molecular Orbitals หรือ MOs) และพลังงานที่สอดคล้องกันได้

7.4.8 ออร์บิทัลเชิงโมเลกุลและผลคูณฮาร์ทรี

เนื่องจากว่า Kohn-Sham DFT นั้นอ้างอิงอยู่กับออร์บิทัลเชิงโมเลกุล (Molecular Orbital หรือ MO) เราจึงควรที่จะเข้าใจเกี่ยวกับ MO ด้วย ซึ่ง MO นั้นจริง ๆ แล้วคือฟังก์ชันคลื่นของอิเล็กตรอนหนึ่งตัว (One-electron Wavefunction) โดย MO ที่ประกอบไปด้วยพิกัดเชิงพื้นที่ (Spatial Coordinates) และพิกัดเชิงสปิน (Spin Coordinates) นั้นจะมีชื่อเรียกว่าออร์บิทัลเชิงสปิน (Spin Orbital) ซึ่งเป็นผลคูณของทั้ง Spatial Function และ Spin Function ตามสมการดังต่อไปนี้

สปินขึ้น (Up Spin):

$$\underbrace{\chi^{\uparrow}(\boldsymbol{x})}_{\text{Spin Orbital}} = \underbrace{\psi(\boldsymbol{r})}_{\text{Spatial}} \underbrace{\alpha(\omega)}_{\text{Spin}} \tag{7.44}$$

สปินลง (Down Spin)

$$\underbrace{\chi^{\downarrow}(\boldsymbol{x})}_{\text{Spin Orbital}} = \underbrace{\psi(\boldsymbol{r})}_{\text{Spatial}} \underbrace{\beta(\omega)}_{\text{Spin}} \tag{7.45}$$

ถึงแม้ว่า Hamilnotian สำหรับอิเล็กตรอนหนึ่งตัว (สมการที่ (7.43)) เป็นฟังก์ชันที่ขึ้นอยู่กับเพียงแค่ Spatial Coordinates เท่านั้น เราก็สามารถใช้ Spin Orbitals ซึ่งก็เป็น Eigenfunction ได้เช่นกัน ดังนั้นจากสมการที่ (7.43) เราจะได้ Schrödinger Equation ที่ใช้ Spin Orbitals สำหรับอิเล็กตรอนตัวที่ i ดังนี้

$$\hat{h}(\boldsymbol{r}_i)\underbrace{\chi_a(\boldsymbol{r}_i)}_{\text{MOs}} = \underbrace{\epsilon_a}_{\text{Energy}}\chi_a(\boldsymbol{r}_i) \tag{7.46}$$

คราวนี้มาดูตัวอย่างง่าย ๆ นั่นคือระบบที่มีอิเล็กตรอนสองตัวที่ไม่มีอัตรกิริยาต่อกัน (Two Non-interacting Electrons) โดยเราสามารถเขียน Wavefunction สำหรับทั้งสองอิเล็กตรอนได้ดังนี้

$$\hat{h}(\boldsymbol{r}_1)\underbrace{\chi_a(\boldsymbol{r}_1)}_{\text{MOs}} = \underbrace{\epsilon_a}_{\text{Energy}}\chi_a(\boldsymbol{r}_1) \tag{7.47}$$

$$\hat{h}(\boldsymbol{r}_2)\underbrace{\chi_a(\boldsymbol{r}_2)}_{\text{MOs}} = \underbrace{\epsilon_a}_{\text{Energy}}\chi_a(\boldsymbol{r}_2) \tag{7.48}$$

ถ้าเป็นกรณีหลายวัตถุ (Many-body) แบบที่มีอิเล็กตรอน N_{el} ตัวในระบบที่อิเล็กตรอนไม่มีอัตรกิริยาต่อกัน เราสามารถเขียนผลเฉลยแบบแม่นตรง (Exact Solution) ของ Schrödinger Equation ได้ดังนี้

$$\left[\sum_{i=1}^{N_{\text{el}}}\hat{h}(\boldsymbol{r}_i)\right]\chi_a(\boldsymbol{x}_1)\chi_b(\boldsymbol{x}_2)\cdots\chi_z(\boldsymbol{x}_{N_{el}}) = (\epsilon_a+\epsilon_b\ldots\epsilon_z)\chi_a(\boldsymbol{x}_1)\chi_b(\boldsymbol{x}_2)\cdots\chi_z(\boldsymbol{x}_{N_{\text{el}}}) \tag{7.49}$$

ซึ่งเราสรุปได้ว่า ***พลังงานรวม (Eigenvalue) ของระบบหรือโมเลกุลของเรานั้นจริง ๆ แล้วมีค่าเท่ากับผลรวมของพลังงานของ Spin Orbital ของอิเล็กตรอนแต่ละตัวรวมกัน***

ประเด็นต่อมาก็คือตามสมการที่ (7.49) นั้น เราสามารถเขียนผลคูณของ Spin Orbitals แต่ละตัวให้เป็น Wavefunction ของระบบที่อิเล็กตรอนเป็นอิสระต่อกันได้ ดังนี้

$$\Psi_{\text{eff}} = \chi_a(\boldsymbol{x}_1)\chi_b(\boldsymbol{x}_2)\cdots\chi_z(\boldsymbol{x}_{N_{\text{el}}}) \tag{7.50}$$

โดยเราเรียก Wavefunction ในสมการที่ (7.50) นี้ว่า “ผลคูณฮาร์ทรี (Hartree Product)” ซึ่งการเขียน Wavefunction แบบนี้ถูกต้องตามหลักคณิตศาสตร์ทุกประการ แต่ทว่าตามหลักกลศาสตร์ควอนตัม Hartree Product นั้นขัดแย้งกับคุณสมบัติข้อหนึ่งของ Wavefunction นั่นก็คือ Wavefunction จะต้องมีปฏิสมมาตร (Antisymmetry) กล่าวคือถ้าเรามีการสลับตำแหน่งของ Spatial Coordinate หนึ่งครั้ง เครื่องหมายของ Wavefunction ก็จะเปลี่ยนไป (จากลบเป็นบวกหรือจากบวกเป็นลบ) แต่ว่าในกรณีของ Hartree Product นั้นเนื่องจากว่าอิเล็กตรอนทุกตัวเป็นอิสระต่อกัน จึงทำให้ Wavefunction ที่เป็นผลคูณระหว่าง Spin Orbitals นั้นไม่มีคุณสมบัติ Antisymmetry ดังนั้นเป้าหมายต่อไปของเราก็คือการหารูปแบบ (Form) ทางคณิตศาสตร์แบบอื่นที่สามารถอธิบาย Wavefunction และมีคุณสมบัติของ Antisymmetry อยู่ด้วย

7.4.9 จากผลคูณฮาร์ทรีสู่ดีเทอร์มิแนนต์ของสเลเตอร์

เมื่อผลคูณฮาร์ทรี (Hartree Product) ไม่เหมาะสมที่จะถูกนำมาใช้ในการสร้าง Wavefunction ดังนั้นจึงได้มีการพัฒนาสิ่งที่เรียกว่าดีเทอร์มิแนนต์ของสเลเตอร์ (Slater Determinant) ขึ้นมา โดยตั้งชื่อตามนามสกุลของศาสตราจารย์ John C. Slater นักฟิสิกส์ชาวอเมริกา ซึ่งแนวคิดของ Slater Determinant นั้นก็คือการใช้ผลรวมเชิงเส้น (Linear Combination) ของ Hartree Product นั่นเอง[84] ตัวอย่างเช่นกรณีที่ระบบมีอิเล็กตรอนสองตัว เราจะได้ว่า Wavefunction ที่เขียนในรูปของผลรวมเชิงเส้นของ Hartree Product ที่มีคุณสมบัติ Antisymmetry มีดังนี้

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2) = \chi_1(\boldsymbol{x}_1)\chi_2(\boldsymbol{x}_2) - \chi_2(\boldsymbol{x}_1)\chi_1(\boldsymbol{x}_2) \tag{7.51}$$

ซึ่งสามารถเขียนเป็นดีเทอร์มีแนนต์ (Determinant) ของเมทริกซ์ออร์บิทัลเชิงสปินได้ดังนี้

$$\begin{aligned}\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2) &= \det\begin{pmatrix}\chi_1(\boldsymbol{x}_1) & \chi_2(\boldsymbol{x}_1)\\ \chi_1(\boldsymbol{x}_2) & \chi_2(\boldsymbol{x}_2)\end{pmatrix}\\ &= \begin{vmatrix}\chi_1(\boldsymbol{x}_1) & \chi_2(\boldsymbol{x}_1)\\ \chi_1(\boldsymbol{x}_2) & \chi_2(\boldsymbol{x}_2)\end{vmatrix}\end{aligned} \tag{7.52}$$

โดยเราเรียกสมการที่ (7.52) ที่เป็น Wavefunction นี้ว่า Slater Determinant ซึ่งสามารถที่จะทำให้มีรูปแบบทั่วไป (Generalized) สำหรับระบบที่มีอิเล็กตรอนกี่ตัวก็ได้ (N-electron System) นอกจากนี้ ตามที่ได้อธิบายไว้ก่อนหน้านี้ว่า Slater Determinant นั้นมีคุณสมบัติ Antisymmetry ซึ่งทำให้ Wavefunction นั้นจะมีการเปลี่ยนเครื่องหมายทุกครั้งที่เราทำการสลับแถวหรือสลับหลัก

สำหรับ Slater Determinant ของระบบที่มี N อิเล็กตรอนนั้นเราสามารถกำหนดได้ดังนี้[55]

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \frac{1}{\sqrt{N!}}\begin{vmatrix}\chi_1(\boldsymbol{x}_1) & \chi_2(\boldsymbol{x}_1) & \cdots & \chi_N(\boldsymbol{x}_1)\\ \chi_1(\boldsymbol{x}_2) & \chi_2(\boldsymbol{x}_2) & \cdots & \chi_N(\boldsymbol{x}_2)\\ \vdots & \vdots & & \vdots\\ \chi_1(\boldsymbol{x}_N) & \chi_2(\boldsymbol{x}_N) & \cdots & \chi_N(\boldsymbol{x}_N)\end{vmatrix} \tag{7.53}$$

โดยที่ $1/\sqrt{N!}$ คือค่าคงที่ของการทำ Normalization นอกจากนี้เรายังสามารถเขียนสมการที่ (7.53) ให้สั้นลงได้โดยใช้สัญกรณ์เค็ท (Ket Notation) ดังนี้

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N) = |\chi_1(\boldsymbol{x}_N)\chi_2(\boldsymbol{x}_N)\cdots\chi_N(\boldsymbol{x}_N)\rangle \tag{7.54}$$

หรือจะเขียนให้สั้นและกระชับกว่านี้ก็ได้ ดังนี้

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = |1, 2, \cdots, N\rangle \tag{7.55}$$

คุณสมบัติอีกอย่างหนึ่งที่ Slater Determinant มีนั่นก็คือประพฤติตัวตามหลักของเพาลี (Pauli Principle) นั่นคือ Spatial Orbital นั้นจะมีอิเล็กตรอนได้สูงสุดไม่เกิน 2 ตัว และอิเล็กตรอนทั้งสองตัวนั้นจะต้องมีสปินที่ตรงข้ามกัน[58]

ณ จุด ๆ นี้เมื่อเราสามารถใช้ Slater Determinant เป็น Wavefunction ได้แล้ว เราก็สามารถหาความหนาแน่นของอิเล็กตรอนได้เช่นกัน โดยสำหรับระบบที่มีจำนวนอิเล็กตรอนเป็นเลขคู่ (Closed-shell) และอิเล็กตรอนทุกตัวถูกบรรจุอยู่ใน Spatial Orbitals จะได้ว่า

$$n(\boldsymbol{r}) = 2 \sum_{i=1}^{N_{el}/2} |\psi_i(\boldsymbol{r})|^2 \tag{7.56}$$

ดังนั้นเราจึงสรุปได้ว่า ***ความหนาแน่นของอิเล็กตรอนของฟังก์ชันคลื่นแบบ Slater Determinant นั้นมีค่าเท่ากับผลรวมของยกกำลังสองของออร์บิทัลที่บรรจุอิเล็กตรอน (Occupied Orbital)***

7.4.10 Orbital-free DFT ปะทะ Kohn-Sham DFT

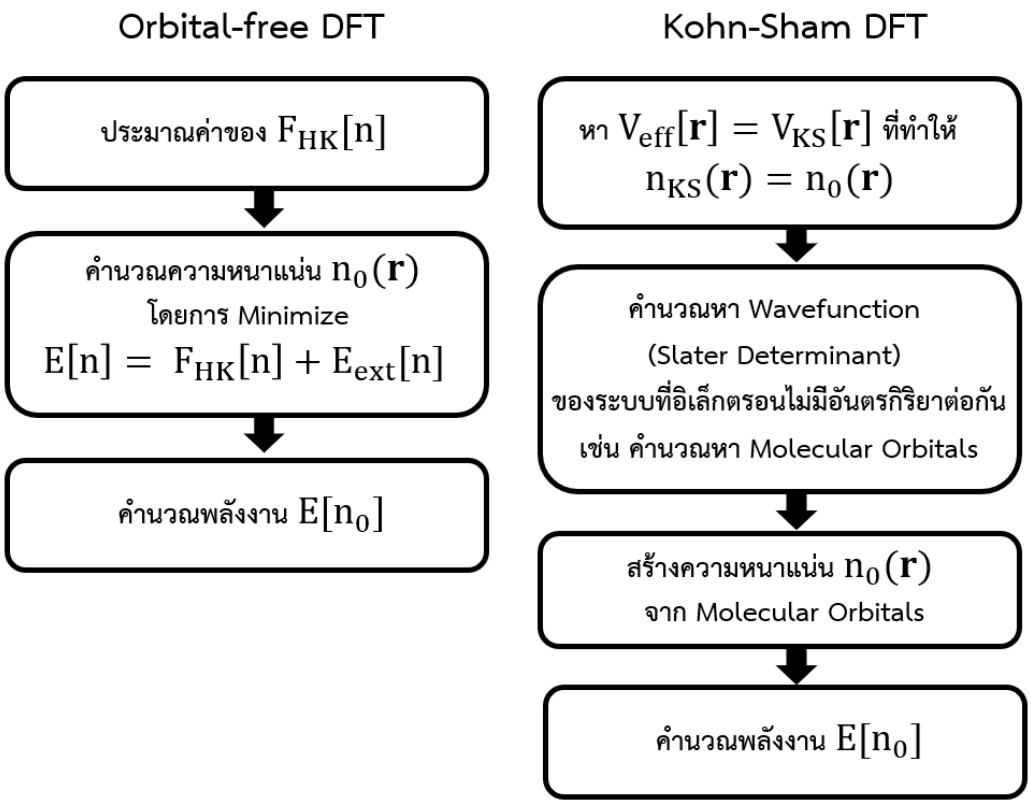
ภาพที่ 7.10 แสดงการเปรียบเทียบขั้นตอนการคำนวณพลังงานของระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันด้วยวิธี Orbital-free DFT ซึ่งเป็นการคำนวณผ่าน Universal Function ของ Hohenberg-Kohn และวิธี Kohn-Sham DFT ซึ่งเป็นการคำนวณผ่านความหนาแน่นซึ่งอ้างอิงกับ Molecular Orbitals ผู้อ่านสามารถศึกษาโค้ดของ Kohn-Sham DFT และการเขียนนำฟังก์ชันนอลจาก LibXC มาใช้เพื่อคำนวณพลังงานได้ที่ซอร์สโค้ดของโปรแกรม PySCF (https://github.com/pyscf/pyscf/tree/master/pyscf/dft) ตัวอย่างเช่น โค้ดของฟังก์ชัน `get_veff` บรรทัดที่ 30-105 (https://github.com/pyscf/pyscf/blob/master/pyscf/dft/uks.py#L30-L105) สำหรับ Unrestricted Kohn-Sham ซึ่งเป็นฟังก์ชันที่คำนวณผลรวมของพลังงานศักย์คูลอมป์ (Coulomb Energy) และพลังงานแลกเปลี่ยน-สหสัมพันธ์ (Exchange-Correlation Energy)

จากที่ได้อธิบายมาตั้งแต่ต้นเราสามารถสรุปเปรียบเทียบฟังก์ชันของพลังงานที่คำนวณด้วยวิธี Orbital-free DFT และวิธี Kohn-Sham DFT ได้ดังนี้

- **Orbital-free DFT**

$$E[n] = E_{\text{kin}}[n] + E_{\text{Coul}}[n] + E_{\text{XC}}[n] + E_{\text{Ext}}[n] \tag{7.57}$$

อุปสรรคในการใช้สมการที่ (7.57) ก็คือเราไม่มีผลเฉลยที่ถูกต้องของฟังก์ชันนอลของพลังงานจลน์



ภาพ 7.10 การเปรียบเทียบขั้นตอนการคำนวณพลังงานของระบบด้วยวิธี Orbital-free DFT และวิธี Kohn-Sham DFT

● **Kohn-Sham DFT**

$$E[n] = E_{\text{kin,KS}}[n] + (E_{\text{kin}}[n] - E_{\text{kin,KS}}[n]) + E_{\text{Coul}}[n] + E_{\text{XC}}[n] + E_{\text{Ext}}[n] \quad (7.58)$$

โดยที่ $E_{\text{kin,KS}}[n]$ คือพลังงานของระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกัน (ระบบอิเล็กตรอนของ Kohn-Sham) ซึ่งเรารู้ผลเฉลยแบบแม่นตรงของ $E_{\text{kin,KS}}[n]$ ถึงแม้ว่าจะอยู่ในเทอมของ Molecular Orbitals และไม่ใช่ความหนาแน่นก็ตาม นอกจากนี้เทอมที่น่าสนใจอีกเทอมหนึ่งก็คือความแตกต่างระหว่างพลังงานจลน์จริง ๆ ของระบบกับพลังงานจลน์ของ Kohn-Sham $(E_{\text{kin}}[n] - E_{\text{kin,KS}}[n])$ นั้นมีค่าน้อยกว่าความคลาดเคลื่อนในประมาณค่าของ $E_{\text{kin,KS}}[n]$ ของ Orbital-free DFT มาก ดังนั้น Kohn-Sham DFT จึงสามารถคำนวณพลังงานเชิงอิเล็กทรอนิกส์ได้แม่นยำมากกว่าวิธี Orbital-free DFT

7.4.11 พลังงานของ Kohn-Sham

เพื่อไม่ให้ผู้อ่านสับสนเราสามารถสรุปสมการที่ใช้ในการคำนวณพลังงานของระบบอิเล็กตรอนของ Kohn-Sham ดังนี้

> ฟังก์ชันนอลของพลังงานรวม = พลังงานจลน์ที่เป็นฟังก์ชันนอลของ MOs + พลังงานเทอมอื่น ๆ ที่เหลือที่เป็นฟังก์ชันนอลของความหนาแน่น

ซึ่งเขียนได้เป็นสมการดังนี้

$$E_{\text{KS}}[n] = E_{\text{kin,KS}}[n] + E_{\text{Coul}}[n] + E_{\text{Ext}}[n] + \underbrace{E_{\text{XC}}[n] + (E_{\text{kin}}[n] - E_{\text{kin,KS}}[n])}_{\text{นำมารวมกันได้}} \quad (7.59)$$

$$= 2\sum_{i=1}^{N_{\text{el}}/2} \int \psi_i^*(\boldsymbol{r}) \left(-\frac{1}{2}\nabla^2\right) + E_{\text{Coul}}[n] + E_{\text{Ext}}[n] + E'_{\text{XC}}[n] \quad (7.60)$$

นั่นคือความหนาแน่นของระบบที่อิเล็กตรอนที่มีอันตรกิริยาต่อกันถูกสร้างขึ้นจากออร์บิทัลเชิงโมเลกุลของระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันของ Kohn-Sham (Kohn-Sham MOs) ดังนั้นท้ายที่สุดแล้วเราสามารถสรุปได้อีกว่า *"พลังงานของระบบของ Kohn-Sham นั้นเป็นฟังก์ชันนอลของ Kohn-Sham MOs นั่นเอง"*

ประเด็นก็คือเราจะต้องทำการประมาณค่าของเทอม $E'_{\text{XC}}[n]$ ซึ่งถึงแม้ว่าเทอมนี้จะมีการรวม Contribution จากพลังงานจลน์ของระบบเข้ามาด้วย (จากสมการที่ (7.59) มาเป็นสมการที่ (7.60)) แต่เราก็ยังเรียก

เทอมนี้ว่า Exchange-Correlation Functional อยู่ครับ ซึ่งการหาเทอมนี้เป็นหนึ่งในหัวข้องานวิจัยที่สำคัญและมีกลุ่มวิจัยหลายกลุ่มให้ความสนใจเป็นอย่างมาก

7.4.12 การปรับหาค่าพลังงานให้ต่ำที่สุด

เมื่อเรารู้แล้วว่าพลังงานรวมของระบบที่ถูกคำนวณด้วย Kohn-Sham DFT นั้นเป็นฟังก์ชันนอลของ MOs ของระบบที่อิเล็กตรอนไม่มีอันตรกิริยาต่อกันของ คำถามต่อมาที่เราจะต้องมาหาคำตอบก็คือเราจะคำนวณ Kohn-Sham MOs ได้อย่างไร ในหัวข้อก่อนหน้านี้ผู้อ่านได้ศึกษาไปแล้วว่าในการปรับลดค่าของพลังงานนั้นสามารถทำได้โดยการใช้หลักการแปรค่า (Variation Principle) ซึ่งจะเป็นการปรับลดค่าพลังงานรวมโดยเทียบกับความหนาแน่นเพื่อหาพลังงานของระบบ ณ สถานะพื้นโดยเราเรียกกระบวนการนี้ว่า Energy Minimization อย่างไรก็ตามเราไม่สามารถใช้เทคนิคนี้กับกรณีของ Kohn-Sham DFT ได้ เพราะว่าพลังงานรวมของ Kohn-Sham นั้นเป็นฟังก์ชันนอลของ MOs ดังนั้นเราจะต้องทำการ Minimize พลังงานโดยเทียบกับ MOs แทน ซึ่งความหนาแน่นของอิเล็กตรอนนั้นก็ขึ้นอยู่กับ MOs ด้วย

เนื่องจากว่าปัญหาการปรับลดค่าพลังงานรวมนั้นเป็นปัญหาค่าต่ำสุดที่มีหลายตัวแปรซึ่งเราสามารถใช้ตัวคูณลากรองจ์ (Langrange Multiplier) ได้ เริ่มต้นเรากำหนดชุดของ Langrange Multiplier แล้วทำการ Minimize ดังนี้

$$\Omega_{\mathrm{KS}[n]} = E_{\mathrm{KS}}[n] - 2\sum_{i=1}^{N_{\mathrm{el}}/2}\sum_{j=1}^{N_{\mathrm{el}}/2}\epsilon_{ij}\left(\int \phi_i^*(\boldsymbol{r})\phi_j(\boldsymbol{r})d\boldsymbol{r} - \delta_{ij}\right) \tag{7.61}$$

ในการหาอนุพันธ์เชิงฟังก์ชันนอล (Functional Derivative) ของสมการที่ (7.61) เทียบกับออร์บิทัลเราสามารถใช้กฎลูกโซ่ได้ ดังนี้

$$\frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta\phi_j^*(\boldsymbol{r})} = \frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta n(\boldsymbol{r})}\frac{\delta n(\boldsymbol{r})}{\delta\phi_j^*(\boldsymbol{r})} \tag{7.62}$$

เนื่องจากว่าเราทราบค่าของอนุพันธ์ของความหนาแน่นเทียบกับออร์บิทัล ดังต่อไปนี้

$$\frac{\delta n(\boldsymbol{r})}{\delta\phi_j^*(\boldsymbol{r})} = 2\phi_j^*(\boldsymbol{r}) \tag{7.63}$$

เมื่อเราแทนสมการ (7.63) เข้าไปในสมการที่ (7.62) เราจะได้สมการคือ

$$\frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta\phi_j^*(\boldsymbol{r})} = \frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta n(\boldsymbol{r})}\frac{\delta n(\boldsymbol{r})}{\delta\phi_j^*(\boldsymbol{r})} = \frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta n(\boldsymbol{r})}2\phi_j^*(\boldsymbol{r}) \tag{7.64}$$

นอกจากนี้เรามีเงื่อนไขเพิ่มเติมว่าอนุพันธ์เชิงฟังก์ชันนอลของพลังงานเมื่อเทียบกับออร์บิทัลนั้นจริง ๆ แล้วเท่ากับศูนย์ ดังนี้

$$\frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta\phi_j^*(\boldsymbol{r})} = 0 \tag{7.65}$$

เมื่อรวมทุกอย่างเข้าด้วยกันเราจะพบว่าอนุพันธ์เชิงฟังก์ชันนอลของพลังงานนั้นสามารถเขียนกระจายได้เป็น

$$\begin{aligned}\frac{\delta\Omega_{\mathrm{KS}[n]}}{\delta\phi_j^*(\boldsymbol{r})} &= 2\left(-\frac{1}{2}\nabla^2\phi_j(\boldsymbol{r})\right) \\ &\quad + 2\left(\frac{\delta E_{\mathrm{Coul}}}{\delta n} + \frac{\delta E_{\mathrm{ext}}}{\delta n} + \frac{\delta E_{\mathrm{XC}}}{\delta n}\right)\phi_j(\boldsymbol{r}) \\ &\quad - 2\sum_{i=1}^{N_{\mathrm{el}}/2}\epsilon_{ij}\phi_j(\boldsymbol{r})\end{aligned} \tag{7.66}$$

$$= 0 \tag{7.67}$$

เมื่อถึงขั้นตอนนี้แล้วขั้นตอนต่อไปก็คือเราสามารถทำการรวมออร์บิทัล $\{\phi\}$ ทั้งหมดเข้าด้วยกัน (โดยใช้ Summation สำหรับทุกดัชนี i) ให้เป็นออร์บิทัลใหม่โดยกำหนดด้วยตัวแปร $\{\psi\}$ ซึ่งเราจะได้สมการดังต่อไปนี้

$$\left(-\frac{1}{2}\nabla^2\psi_j(\boldsymbol{r})\right) + \left(\frac{\delta E_{\mathrm{Coul}}}{\delta n} + \frac{\delta E_{\mathrm{ext}}}{\delta n} + \frac{\delta E_{\mathrm{XC}}}{\delta n}\right)\psi_j(\boldsymbol{r}) - \epsilon_j\psi_j(\boldsymbol{r}) = 0 \tag{7.68}$$

เมื่อทำการจัดรูปสมการโดยย้ายเทอมที่มีพลังงานของระบบมาทางด้านขวาของสมการแล้วทำการรวมเทอมฝั่งซ้ายซึ่งเป็นโอเปอเรเตอร์ดังนี้

$$\begin{aligned}\left\{-\frac{1}{2}\nabla^2\left(+\frac{\delta E_{\mathrm{Coul}}}{\delta n} + \frac{\delta E_{\mathrm{ext}}}{\delta n} + \frac{\delta E_{\mathrm{XC}}}{\delta n}\right)\right\}\psi_j(\boldsymbol{r}) &= \epsilon_j\psi_j(\boldsymbol{r}) \\ \left(-\frac{1}{2}\nabla^2 + V_{\mathrm{KS}}(\boldsymbol{r})\right)\psi_j(\boldsymbol{r}) &= \epsilon_j\psi_j(\boldsymbol{r})\end{aligned} \tag{7.69}$$

ซึ่งสมการที่ (7.69) ก็คือ Schrödinger Equation สำหรับหนึ่งอิเล็กตรอนนั่นเอง โดยเราสามารถแก้สมการเพื่อหา Kohn-Sham MOs ได้ อย่างไรก็ตาม เนื่องจากว่าเราไม่รู้ว่า MOs เริ่มต้นนั้นมีหน้าตาเป็นอย่างไร

(Unknown) เราจึงไม่สามารถหาเฉลยแม่นตรงได้ ดังนั้นเราจึงจำเป็นต้องใช้วิธี Self-Consistent Field (SCF) ซึ่งเป็นวิธีวนซ้ำ (Iterative Method) โดนมีขั้นตอนคร่าว ๆ ดังต่อไปนี้

1. ทำการสร้างออร์บิทัลเชิงโมเลกุลเริ่มต้น (Initial Guess of MOs)
↓
2. สร้าง Kohn-Sham Operator
↓
3. คำนวณพลังงาน
↓
4. เปรียบเทียบพลังงานของรอบปัจจุบันกับพลังงานของรอบที่แล้ว
↓
5. ถ้าความแตกต่างของพลังงานและความหนาแน่น
   - $>$ ค่า Cutoff $\rightarrow$ ขั้นตอนที่ 2
   - $<$ ค่า Cutoff $\rightarrow$ สิ้นสุดการคำนวณ

รายละเอียดของทฤษฎี DFT และการคำนวณด้วยคอมพิวเตอร์นั้นมีอีกเยอะมาก เช่น ฟังก์ชันพื้นฐาน (Basis Set) แต่ละประเภทและแต่ละขนาด, Exchange-Correlation Functional แบบต่าง ๆ, Spin-polarised DFT[1], การปรับโครงสร้าง (Geometry Optimization) รวมไปถึงเทคนิคต่าง ๆ ที่ได้มีการพัฒนาเพื่อปรับปรุง DFT ให้มีความแม่นยำในการคำนวณคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลมากขึ้น

ผู้อ่านที่สนใจศึกษาเพิ่มเติมสามารถศึกษาได้จากหนังสือดังต่อไปนี้

1. Introduction to Computational Chemistry[57]
   ผู้เขียน Frank Jensen
2. Electronic Structure - Basic Theory and Practical Methods[83]
   ผู้เขียน Richard M. Martin
3. Density Functional Theory - An Advanced Course[85]
   ผู้เขียน Eberhard Engel และ Reiner M. Dreizler

7.5 การคำนวณพลังงานของระบบอิเล็กตรอนด้วย Kohn-Sham DFT

ในหัวข้อก่อนหน้านี้เป็นการอธิบายทฤษฎีของ Kohn-Sham DFT ซึ่งมีเคล็ดลับคือกำหนดให้อิเล็กตรอนในระบบนั้นไม่มีอันตรกิริยาต่อกัน เพื่อให้ผู้อ่านเห็นภาพมากขึ้น ในหัวข้อนี้ผู้อ่านจะได้ศึกษาการเขียน

[1]Spin-polarised DFT เป็นเทคนิคหนึ่งของ DFT ที่ถูกนำมาใช้ในการคำนวณคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลหรือสารประกอบที่มีความเป็นแม่เหล็กเนื่องจากว่าคุณสมบัติที่เกิดจากสปินขึ้นและสปินลงของโมเลกุลที่มีความเป็นแม่เหล็กนั้นจะต่างกัน

โปรแกรมสำหรับการคำนวณพลังงานของระบบหลายอิเล็กตรอนโดยใช้ Kohn-Sham DFT โดยเราจะสนใจกรณีที่เป็น 1 มิติเท่านั้น (อิเล็กตรอนมีการเคลื่อนที่ตามแกน x เพียงอย่างเดียว)

ในการเขียนโค้ดของ Kohn-Sham (KS) DFT นั้นเราจะใช้ใช้ Hamiltonian ตามที่เราได้ศึกษาไปแล้วตามสมการที่ (7.59) โดยเราสามารถเขียนให้สั้นและกระชับมากขึ้นได้ ดังนี้

$$\hat{H} = -\frac{1}{2}\frac{d^2}{dx^2} + v_{Coul}(x) + v_{LDA}(x) + v_{ext} \tag{7.70}$$

โดยทางด้านขวาของสมการ (7.70) ประกอบไปด้วยเทอมดังต่อไปนี้

1. พลังงานจลน์ (Kinetic Energy)
2. พลังงานศักย์คูลอมป์ (Coulomb Energy) หรือแรงผลักไฟฟ้าสถิตย์ระหว่างอิเล็กตรอน
3. พลังงานแลกเปลี่ยน (Exchange Energy) ซึ่งเราจะใช้การประมาณค่าความหนาแน่นแบบพื้นที่ (Local Density Approximation)
4. พลังงานภายนอก (External Potential) ซึ่งเราจะใช้ฟังก์ชัน Harmonic Oscillator

หมายเหตุ: เราจะไม่พิจารณา Correlation Energy เนื่องจากว่ามีความซับซ้อนมากเกิน

โดยเราจะใช้ภาษา Python ในการเขียน โดยสิ่งที่เราต้องทำหลัก ๆ มีดังนี้

1. สร้าง Hamiltonian
2. คำนวณฟังก์ชันคลื่นของ Kohn-Sham (KS Wavefunction)
3. คำนวณความหนาแน่น (Density)
4. คำนวณพลังงานอิเล็กทรอนิกส์ (Electronic Energy)

เมื่อพร้อมแล้วเราก็มาเริ่มกันได้เลย

1. นำเข้าไลบรารี่และสร้างฟังก์ชันที่จำเป็นต้องใช้

ใช้ไลบรารี่ NumPy สำหรับจัดการกับเมทริกซ์และไลบรารี่ Matplotlib กับ Seaborn สำหรับพลอต

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

ทำการสร้างฟังก์ชันสำหรับการ Integrate ซึ่งก็คือการรวมกันนั่นเอง โดยเราจะนำฟังก์ชันนี้ไปใช้งานต่อในโค้ดด้านล่าง

```
def integral(x, y, axis=0):
    dx = x[1]-x[0]
    return np.sum(y*dx, axis=axis)
```

2. กำหนดโอเปอเรเตอร์เชิงอนุพันธ์สำหรับการสร้าง Hamiltonian ของพลังงานจลน์

```
# Define a real-space grid
n_grid = 200
x = np.linspace(-5, 5, n_grid)
y = np.sin(x)

# First derivative
h = x[1]-x[0]
D = -np.eye(n_grid) + np.diagflat(np.ones(n_grid-1),1)
D = D / h

# Second derivative
D2 = D.dot(-D.T)
D2[-1,-1] = D2[0,0]
```

3. คำนวณพลังงานจลน์

แก้สมการ Kohn-Sham เฉพาะของพลังงานจลน์โดยการทำ Diagonalization (เป็นขั้นตอนที่กำหนด Computational Complexity ของ DFT นั่นคือ $\mathcal{O}(n^3)$

```
# Solve Kohn-Sham equation
eig_non, psi_non = np.linalg.eigh(-D2/2)
```

4. คำนวณพลังงานศักย์ภายนอก

ลำดับต่อไปคือการพิจารณาศักย์ภายนอก (External Potential) ซึ่งเราสามารถใช้ฟังก์ชัน Harmonic Oscillator ง่าย ๆ ได้ ในตัวอย่างนี้ผู้เขียนเลือกใช้ External Potential เป็นฟังก์ชันพหุนาม คือ $v_{ext} = x^2$:

```
# Define external potential with a matrix
X = np.diagflat(x*x)

# Solve Kohn-Sham equation
eig_harm, psi_harm = np.linalg.eigh(-D2/2+X)
```

5. คำนวณพลังงานแลกเปลี่ยน

ลำดับต่อมาคือการคำนวณพลังงานแลกเปลี่ยน (Exchange Energy) โดยเราจะพิจารณาฟังก์ชันนอลแลกเปลี่ยน (Exchange Functional) โดยใช้ Local Density Approximation (LDA) ซึ่งมีสมการดังต่อไปนี้ (จริง ๆ แล้ว LDA มี Correlation Functional ด้วยแต่ว่าเราจะไม่สนใจ)

$$E_X^{LDA}[n] = -\frac{3}{4}\left(\frac{3}{\pi}\right)^{1/3}\int n^{4/3}dx \tag{7.71}$$

โดยที่ Potential นั้นสามารถคำนวณได้จากอนุพันธ์ของ Exchange Energy เทียบกับความหนาแน่น

$$\begin{aligned} v_X^{LDA}[n] &= \frac{\partial E_X^{LDA}}{\partial n} \\ &= -\left(\frac{3}{\pi}\right)^{1/3} n^{1/3} \end{aligned} \tag{7.72}$$

```
def get_exchange(nx, x):
    energy = -3./4.*(3./np.pi)**(1./3.)*integral(x, nx**(4./3.))
    potential = -(3./np.pi)**(1./3.)*nx**(1./3.)
    return energy, potential
```

6. คำนวณพลังงานคูลอมป์

ลำดับต่อมาคือพลังงานคูลอมป์ซึ่งเป็นพลังงานทางไฟฟ้าสถิตย์ (Electrostatic Energy) หรืออาจจะเรียกเรียกว่าพลังงานฮาร์ทรี Hartree Energy ก็ได้ อย่างไรก็ตาม ตามทฤษฎีนั้นพลังงานคูลอมป์สำหรับนั้นลู่เข้า (Converged) เฉพาะกรณี 3 มิติเท่านั้นซึ่งมีสมการดังต่อไปนี้

$$E_{Coul}^{3D} = \frac{1}{2}\iint \frac{n(r)n(r')}{\sqrt{(r-r')^2}} dr dr' \tag{7.73}$$

ดังนั้นในกรณี 1 มิติเราจะต้องทำการโกงนิดหน่อยเพื่อทำให้พลังงานนั้นลู่เข้าโดยการปรับสมการ ดังนี้

$$E_{Coul}^{1D} = \frac{1}{2}\iint \frac{n(x)n(x')}{\sqrt{(x-x')^2+\varepsilon}} dx dx' \tag{7.74}$$

โดยที่ ε คือคงที่ที่เป็นบวกที่มีค่าน้อย ๆ ซึ่งทำให้ฟังก์ชันนี้ลู่เข้าได้ง่ายขึ้น

ดังนั้นพลังงานศักย์จึงมีสมการดังต่อไปนี้:

$$v_{Coul} = \int \frac{n(x')}{\sqrt{(x-x')^2+\varepsilon}} dx' \tag{7.75}$$

```
def get_coulomb(nx, x, eps=1e-1):
    h = x[1]-x[0]
```

```
    energy = np.sum(nx[None,:]*nx[:,None]*h**2 /
np.sqrt((x[None,:]-x[:,None])**2 + eps)/2)
    potential =
np.sum(nx[None,:]*h/np.sqrt((x[None,:]-x[:,None])**2+eps),
axis=-1)
    return energy, potential
```

7. คำนวณความหนาแน่น

เนื่องจากว่าเราจะต้องทำการรวม Coulomb Energy และ LDA Exchange โดยที่ทั้งคู่นั้นเป็นฟังก์ชันนอลของความหนาแน่น ดังนั้นเราจึงจำเป็นต้องคำนวณความหนาแน่นของอิเล็กตรอน (Electron Density) โดยเรามีเงื่อนไขของการทำ Normalization ดังนี้

$$\int |\psi|^2 dx = 1 \tag{7.76}$$

ซึ่งเราสามารถเขียนความหนาแน่นให้อยู่ในรูปของผลรวมเชิงเส้นของออร์บิทัลยกกำลังสองได้ ดังนี้

$$n(x) = \sum_n f_n |\psi(x)|^2 \tag{7.77}$$

โดยที่ f_n คือ Occupation Number (จำนวนอิเล็กตรอนในออร์บิทัลที่ n) ซึ่งแต่ละ State นั้นจะมีอิเล็กตรอนที่มีสปินขึ้นและสปินลง โดยใน DFT นั้นเราคำนวณสถานะพื้นของระบบ

กำหนดจำนวนอิเล็กตรอน เช่น 17 ตัว

```
num_electron = 17
```

ทำการคำนวณความหนาแน่น

```
def get_nx(num_electron, psi, x):
    # Normalization
    I = integral(x, psi**2, axis=0)
    normed_psi = psi/np.sqrt(I)[None, :]

    # Occupation Number
    fn=[2 for _ in range(num_electron//2)]
    if num_electron % 2:
        fn.append(1)

    # Density
```

```
    res = np.zeros_like(normed_psi[:,0])
    for ne, psi in zip(fn, normed_psi.T):
        res += ne*(psi**2)

    return res
```

8. คำนวณพลังงานอิเล็กทรอนิกส์ของระบบ

เมื่อเราเตรียมองค์ประกอบทุกอย่างพร้อมแล้ว ขั้นตอนต่อไปนี้สำคัญมากเพราะว่าเป็นขั้นตอนสุดท้ายที่เราจะนำฟังก์ชันทั้งหมดที่เราได้เขียนไว้มาแก้สมการ Kohn-Sham (KS) โดยการวนซ้ำเทียบกับตัวเอง (Self-Consistency) มีขั้นตอนดังนี้

1. เริ่มต้นด้วยการกำหนดเมทริกซ์ความหนาแน่นของอิเล็กตรอน (Initialize) ซึ่งเราสามารถใช้ค่าคงที่อะไรก็ได้ (เพื่อให้ง่ายต่อการคำนวณ)
2. คำนวณพลังงานศักย์แลกเปลี่ยน (Exchange) และศักย์คูลอมบ์ (Coulomb Potential)
3. คำนวณ Hamiltonian
4. แก้สมการ KS เพื่อคำนวณหา Wavefunctions และ Eigenvalues (พลังงาน)
5. ตรวจสอบการลู่เข้า ถ้าไม่ลู่เข้า ให้อัพเดทความหนาแน่นและกลับไปที่ขั้นตอนที่ 2

ก่อนอื่นให้สร้างฟังก์ชันสำหรับแสดงผลการคำนวณพลังงานในระหว่างการวนซ้ำ (Iteration)

```
def print_log(i, log):
    print(f"step: {i:<5} energy: {log['energy'][-1]:<10.4f}
   energy_diff: {log['energy_diff'][-1]:.10f}")
```

กำหนดพารามิเตอร์เพิ่มเติม เช่น จำนวนรอบสูงสุดในการวนซ้ำและค่า Cutoff ของความแตกต่างระหว่างพลังงานจากรอบที่ n และรอบที่ $n+1$

```
max_iter = 1000
energy_tolerance = 1e-5
# A dictionary to save energies
log = {"energy":[float("inf")], "energy_diff":[float("inf")]}
```

กำหนดค่าความหนาแน่นเริ่มต้นซึ่งจะถูกมานำใช้เป็นค่าเริ่มต้นในการประมาณค่าหาความหนาแน่นโดยการทำปรับค่าเทียบค่าความหนาแน่นที่ได้จากลูปในรอบก่อนหน้า โดยค่าความหนาแน่นเริ่มต้นนั้นเราจะกำหนดโดยใช้ค่าคงที่อะไรก็ได้ ในตัวอย่างนี้ผู้เขียนใช้ความหนาแน่นเท่ากับ 0 และสิ่งที่เกิดขึ้นภายในลูปนั้นเราจะทำการคำนวณพลังงาน Exchange และพลังงาน Coulomb ก่อนแล้วก็สร้าง Hamiltonian ขึ้นมาแล้วก็ทำการ Diagonalize Hamiltonian เพื่อให้ได้ Eigenvalue ออกมาซึ่งนั่นก็คือพลังงานของเรานั่นเอง หลังจากนั้นเราจะทำการเก็บค่าพลังงานที่ได้แล้วก็ตรวจสอบว่าส่วนต่างของพลังงานที่ได้จากการวนลูปรอบปัจจุบันที่ n กับรอบที่ $n-1$ นั้นต่ำกว่าค่า Cutoff แล้วหรือยัง ถ้าหากว่ายังก็ให้ทำการอัพเดทค่าความหนาแน่นแล้วทำการคำนวณพลังงานอีกรอบ

```python
# Initialize density
nx = np.zeros(n_grid)

for i in range(max_iter):
    ex_energy, ex_potential = get_exchange(nx, x)
    ha_energy, ha_potential = get_coulomb(nx, x)

    # Hamiltonian
    H = -D2/2 + np.diagflat(ex_potential + ha_potential + x*x)

    energy, psi = np.linalg.eigh(H)

    # Collect energy and eenrgy difference
    log["energy"].append(energy[0])
    energy_diff = energy[0] - log["energy"][-2]
    log["energy_diff"].append(energy_diff)
    print_log(i, log)

    # Check if the calculation is converged
    if abs(energy_diff) < energy_tolerance:
        print("Converged!   :)")
        break

    # Update the density
    nx = get_nx(num_electron, psi, x)
else:
    print("Not Converged   :(")
```

เมื่อทำการรันโค้ดด้านบนแล้วจะได้เอาต์พุตดังต่อไปนี้

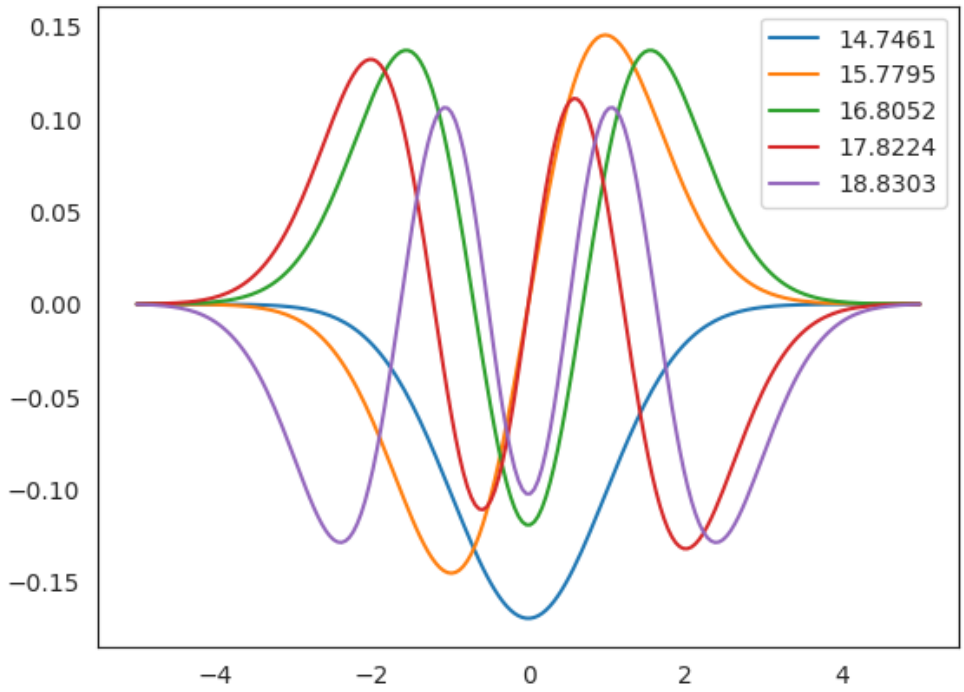
```
step: 0     energy: 0.7069     energy_diff: -inf
step: 1     energy: 16.3625    energy_diff: 15.6555321919
step: 2     energy: 13.8021    energy_diff: -2.5603559494
step: 3     energy: 15.3002    energy_diff: 1.4980525863
step: 4     energy: 14.4119    energy_diff: -0.8882287680
step: 5     energy: 14.9470    energy_diff: 0.5350438262
step: 6     energy: 14.6242    energy_diff: -0.3228271880
step: 7     energy: 14.8201    energy_diff: 0.1959328656
step: 8     energy: 14.7011    energy_diff: -0.1190355457
step: 9     energy: 14.7735    energy_diff: 0.0724651058
step: 10    energy: 14.7294    energy_diff: -0.0441312736
step: 11    energy: 14.7563    energy_diff: 0.0268946713
```

```
step: 12    energy: 14.7399    energy_diff: -0.0163922405
step: 13    energy: 14.7499    energy_diff: 0.0099933983
step: 14    energy: 14.7438    energy_diff: -0.0060926001
step: 15    energy: 14.7475    energy_diff: 0.0037147279
step: 16    energy: 14.7452    energy_diff: -0.0022649307
step: 17    energy: 14.7466    energy_diff: 0.0013810031
step: 18    energy: 14.7458    energy_diff: -0.0008420446
step: 19    energy: 14.7463    energy_diff: 0.0005134280
step: 20    energy: 14.7460    energy_diff: -0.0003130574
step: 21    energy: 14.7462    energy_diff: 0.0001908842
step: 22    energy: 14.7461    energy_diff: -0.0001163900
step: 23    energy: 14.7461    energy_diff: 0.0000709679
step: 24    energy: 14.7461    energy_diff: -0.0000432721
step: 25    energy: 14.7461    energy_diff: 0.0000263849
step: 26    energy: 14.7461    energy_diff: -0.0000160880
step: 27    energy: 14.7461    energy_diff: 0.0000098095
Converged!   :)
```

เมื่อทำการแก้หาค่าพลังงานไปทั้งหมด 27 รอบจะพบว่าพลังงานนั้นลู่เข้า โดยค่าพลังงานสุดท้ายที่ได้คือ 14.7461 และมีค่าความแตกต่างระหว่างพลังงานของรอบที่ 26 กับพลังงานของรอบที่ 27 เท่ากับ 0.0000098095 ซึ่งน้อยกว่า Cutoff ที่กำหนดไว้คือ 0.00001

นอกจากนี้เราสามารถพลอต Wavefunction ซึ่งเป็นฟังก์ชันของ Real-space Grid และระบุพลังงานได้ด้วย ดังนี้

```python
for i in range(5):
    plt.plot(x, psi[:,i], label=f"{energy[i]:.4f}")
    plt.legend(loc=1)
```



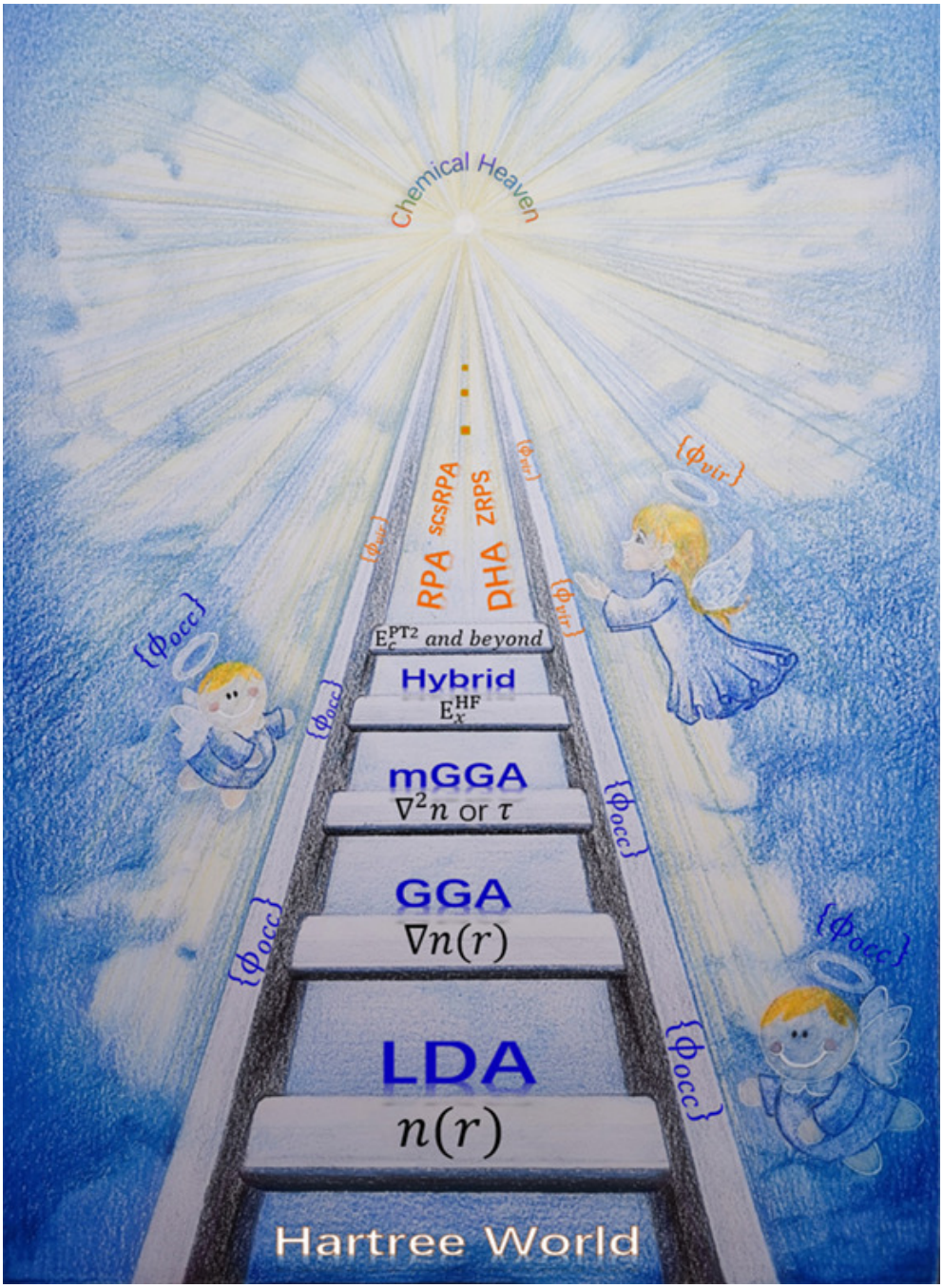
ภาพ 7.11 Wavefunction และพลังงานที่ได้จากการคำนวณ Kohn-Sham DFT สำหรับกรณี 1 มิติ

ผู้อ่านที่ต้องการศึกษาโค้ดฉบับสมบูรณ์สามารถดูได้ที่ไฟล์ `6_1D_DFT.ipynb` ใน Code Repository ของหนังสือที่ https://github.com/rangsimanketkaew/ml-qm-book-code

7.6 บันไดของ Jacob สู่สรวงสวรรค์ของความถูกต้องของ DFT

ในปี 2001 ได้มีบทความงานวิจัยที่นำเสนอไอเดียที่มีการเปรียบเทียบความถูกต้องของวิธี DFT ด้วยบันไดของ Jacob หรือ Jacob's Ladder สำหรับแสดงความถูกต้องและความแม่นยำของการคำนวณโดยไล่ระดับตามขั้นของฟังก์ชันนอล[86] ซึ่งฟังก์ชันนอลของ DFT นั้นเป็นการผสมผสานกันของเทอมหลาย ๆ เทอมที่ใช้ในการอธิบายพลังงานต่าง ๆ ภายในโมเลกุล โดยหลัก ๆ ก็จะมีเทอมที่เป็นพลังงานของ Hartree-Fock รวมอยู่ด้วย เมื่อมีการเพิ่มความซับซ้อนให้กับฟังก์ชันนอลโดยรวมเทอมพลังงานอื่น ๆ เข้าไปก็จะเพิ่มความถูกต้องให้กับการคำนวณมากขึ้นเรื่อย ๆ

โดยภาพที่ 7.12 แสดงขั้นบันไดที่เริ่มจาก Hartree-Fock (HF) ซึ่งบางบทความวิจัยก็เปรียบเทียบว่าเป็นนรก (วิธี HF นั้นให้ความแม่นยำต่ำที่สุดเมื่อเปรียบเทียบกับขั้นบันไดขั้นอื่น ๆ) และขั้นบันไดสุดท้ายคือสววรค์ทางเคมีซึ่งเปรียบเสมือนกับเป็นค่าที่ได้จากการทดลอง (References) จริง ๆ แล้ววิธี HF นั้นแม่นยำในระดับหนึ่งเลยนะครับ โดยผู้เขียนนั้นเคยได้อ่านบทความวิจารณ์ของศาสตราจารย์ Kieron Burke (University of California Irvine) ซึ่งได้สรุปไว้ว่าโดยเฉลี่ยแล้ววิธี HF สามารถทำนายพลังงานรวมของโมเลกุลได้ถูกต้องมาก



ภาพ 7.12 ขั้นบันไดของ Jacob โดยแต่ละขั้นคือฟังก์ชันนอลของ DFT โดยไล่จากขั้นที่อยู่ด้านล่างสุดคือเทอม Hartree-Fock ซึ่งเป็นองค์ประกอบพื้นฐานของฟังก์ชันนอลทั้งหมดและขั้นสูงสุดคือสวรรค์ทางเคมีซึ่งเปรียบเสมือนเป็นจุดที่แม่นยำที่สุดของวิธี DFT

ถึง 99% ส่วนอีก 1% ที่หายไปนั้นก็คือพลังงานสหสัมพันธ์ (Correlation Energy) อย่างไรก็ตาม วิธี HF นั้นก็มักจะให้ผลการคำนวณการปรับโครงสร้างของโมเลกุลที่ไม่ถูกต้องแม่นยำ เช่น มักจะให้ผลการคำนวณความยาวพันธะหรือการคำนวณความถี่เชิงการสั่นที่คลาดเคลื่อนไปเยอะมาก

เมื่อเวลาผ่านไปก็ได้มีการพัฒนาฟังก์ชันนอลสำหรับ DFT โดยมีการใช้พารามิเตอร์ต่าง ๆ เข้ามาช่วยเพื่อเพิ่มความถูกต้อง เช่น ในยุคแรก ๆ นั้นเริ่มจาก Local Density Approximation (LCD) ซึ่งใช้เพียงแค่ความหนาแน่นของอิเล็กตรอนเพียงอย่างเดียว แล้วในเวลาต่อมาก็ได้มีการใช้ Gradient ของความหนาแน่นของอิเล็กตรอนเข้ามาช่วยจนได้ออกมาเป็นฟังก์ชันนอลแบบ Generalized Gradient Approximation (GGA)[87] แล้วตามด้วยการใช้ Hessian ซึ่งได้เป็น Meta GGA (mGGA) แล้วก็มาถึงจุดเปลี่ยนสำคัญคือฟังก์ชันนอลแบบไฮบริด (Hybrid Functional) แล้วก็หลังจากนั้นก็จะเป็นการเพิ่ม Correction เทอมอื่น ๆ โดยใช้วิธีต่าง ๆ เช่น Perturbation หรือ Multireference โดยบันไดขั้นสุดท้ายที่เข้าใกล้ (ต้องใช้คำว่าเข้าใกล้) สวรรค์หรือความถูกต้องมากที่สุดนั้นก็คือฟังก์ชันนอลแบบที่ไม่จำเพาะต่อพื้นที่ (Non-local) นั่นเอง

นอกจากนี้แล้วยังมี Jacob's Ladder ที่เปรียบเทียบวิธีการคำนวณหลาย ๆ วิธีด้วย เช่น เปรียบเทียบระหว่างวิธี DFT กับวิธี Post-HF เช่น เปรียบเทียบกับ Full Configuration Interaction (FCI) ซึ่งเป็นวิธีที่รวมการกระตุ้น (Excitation) ที่มากที่สุดเท่าที่เราต้องการเพื่อทำให้การคำนวณพลังงานสหสัมพันธ์นั้นลู่เข้าได้ง่ายขึ้น

สุดท้ายนี้ผู้เขียนขอให้ความเห็นเพื่อให้ผู้อ่านได้เก็บไว้เป็นแนวคิด 3 ข้อดังนี้

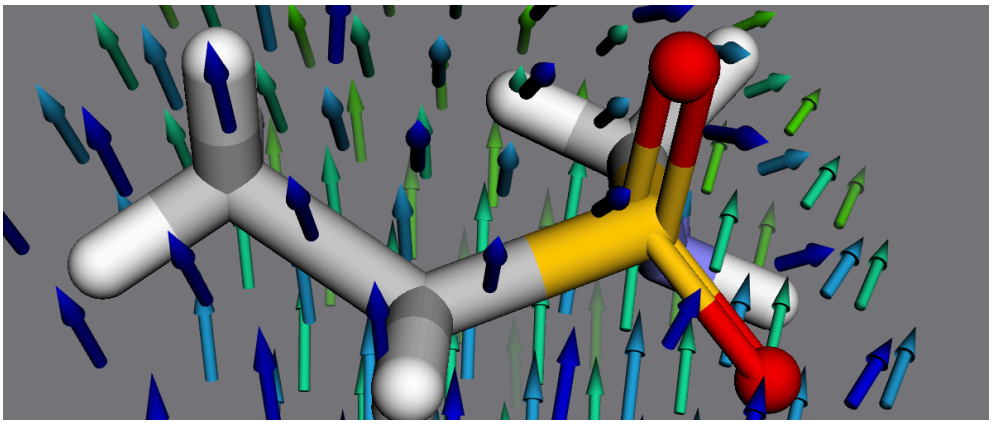
1. DFT ไม่ใช่วิธีที่ให้ผลการคำนวณที่ถูกต้องและแม่นยำมากที่สุด ยังมีวิธีอื่น ๆ อีกเยอะแยะมากมายที่ให้ผลการคำนวณที่ถูกต้องกว่า ขึ้นอยู่กับโมเลกุลและสิ่งที่เราต้องการคำนวณด้วย

2. ไม่มีฟังก์ชันนอลไหนที่ดีที่สุดในโลก (ในแง่ของการทำนายคุณสมบัติต่าง ๆ ของโมเลกุล) กล่าวคือบางฟังก์ชันนอลที่แม่นยำในการคำนวณคุณสมบัติหนึ่ง ๆ ของโมเลกุลก็อาจจะไม่แม่นยำในการคำนวณคุณสมบัติอื่น ๆ

3. การเลือกฟังก์ชันนอลเป็นศิลปะอย่างหนึ่ง ถ้าฟังก์ชันนอลไหนที่ให้ผลที่แม่นยำและใกล้เคียงกับค่าทางการทดลองที่มากที่สุดก็ใช้อันนั้นครับ ดังนั้นจึงเกิดคำถามตามมาว่า "แล้วเราจะเชื่อถือฟังก์ชันนอลที่เราใช้ได้มากน้อยแค่ไหนถ้าหากสมมติว่าไม่มีผลการทดลองของโมเลกุลนั้น ๆ ให้เราได้เปรียบเทียบ?"

บทที่ 8

คุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุล



ภาพ 8.1 สนามเวกเตอร์ของโมเลกุลเมื่ออยู่ในสนามแม่เหล็ก

เนื้อหาในบทนี้จะต่อเนื่องจากบทที่ 7 โดยเราจะมาดูรายละเอียดของทฤษฎีของคุณสมบัติเชิงอิเล็กทรอนิกส์ (Electronic Properties) ของโมเลกุลแบบต่าง ๆ ซึ่งผู้เขียนคิดว่าเป็นสิ่งที่สำคัญมากนั่นก็เพราะว่าถ้าหากเราต้องการที่จะเชื่อมโยง ML และเคมีควอนตัมเข้าด้วยกันเราควรจะต้องเข้าใจถึงทฤษฎีของคุณสมบัติของโมเลกุลที่เราต้องการที่จะศึกษาเสียก่อน เมื่อเราเข้าใจทฤษฎีแล้วก็จะทำให้เกิดไอเดียที่เราสามารถประยุกต์ใช้ ML ได้อย่างถูกต้อง

8.1 ความหนาแน่นเชิงประจุและเมทริกซ์ความหนาแน่น

ความหนาแน่นเชิงประจุ (Charge Density) เป็นปริมาณที่บ่งบอกถึงประจุของอะตอมที่อยู่ในโมเลกุล ถ้าหากเราทำการอินทิเกรต Charge Density ทั่วทั้งปริมาตรเราจะได้ผลลัพธ์เป็นจำนวนของอิเล็กตรอนในระบบของเรา (โมเลกุล) ดังนี้[55]

$$N = \int \rho(\boldsymbol{r}) dV \tag{8.1}$$

โดยนิยามของ Charge Density จะเป็นผลรวมของโอกาสที่เราจะพบอิเล็กตรอนที่อยู่ภายใน Molecular Orbitals ของทั้งระบบ ดังนี้

$$\rho(\boldsymbol{r}) = 2 \sum_{i=1}^{N/2} \int |\varphi_i(\boldsymbol{r})|^2 \tag{8.2}$$

โดยเลข 2 ด้านหน้าเครื่องหมาย Summation ก็คือ Occupation Number สำหรับกรณีที่ Molecular Orbital (i) นั้นมีอิเล็กตรอนทั้งแบบ Spin Up และ Spin Down และ $\varphi_i(\boldsymbol{r})$ คือ Wavefunction ซึ่งเราสามารถเขียน Wavefunction ให้อยู่ในรูปผลรวมเชิงเส้น (LCAO) ของ Basis Function (ϕ_i) ซึ่ง Basis Function นี้จะเป็นฟังก์ชันอะไรก็ได้ที่สามารถอธิบายการมีอยู่ของ Molecular Orbital โดยในกรณีแบบที่ง่ายที่สุดคือเราจะมองว่า Molecular Orbital นั้นเกิดขึ้นจากการรวมกันของ Atomic Orbitals ดังนั้นเราจะกำหนดให้ Atomic Orbitals เป็น Basis Function[1] ดังนั้นเราสามารถเขียน LCAO ได้ดังต่อไปนี้ ตามสมการดังต่อไปนี้

$$\rho(\boldsymbol{r}) = 2 \sum_i \left(\sum_\mu c_{\mu i} \phi_\mu^* \right) \left(\sum_\nu c_{\nu i}^* \phi_\nu \right) \tag{8.3}$$

โดยที่ c คือสัมประสิทธิ์ของ LCAO ลำดับต่อมาคือเมื่อเราจัดรูปให้มีเทอมที่เป็นผลคูณของ Basis Function $(\phi_\mu^* \phi_\nu)$ เราจะกำหนดให้เทอมนี้เป็นสิ่งที่เรียกว่าเมทริกซ์ซ้อนทับ (Overlap Matrix) หรือ $S_{\mu\nu}$ โดยจะได้สมการที่จัดรูปแล้ว ดังนี้

$$\rho(\boldsymbol{r}) = 2 \sum_i \sum_{\mu\nu} c_{\mu i} c_{\nu i}^* S_{\mu\nu} \tag{8.4}$$

หลังจากนั้นเราจะพบว่าจะมีเทอมที่เป็นผลคูณระหว่าง c ซึ่งเราเรียกผลคูณแบบนี้ว่าเมทริกซ์ความหนาแน่น (Density Matrix)

$$P_{\mu\nu} = c_{\mu i} c_{\nu i}^* \tag{8.5}$$

ซึ่งเราจะได้สมการของความหนาแน่นเชิงประจุในรูปของเมทริกซ์ความหนาแน่นดังต่อไปนี้

[1] Basis Function ในที่นี้คือ Atomic Orbitals ที่ถูกกำหนดให้มีจุดศูนย์กลางอยู่ที่อะตอมนั้น ๆ

$$\rho(\boldsymbol{r}) = 2\sum_{i}\sum_{\mu\nu} P_{\mu\nu}S_{\mu\nu} \tag{8.6}$$

8.2 ประจุย่อย

การวิเคราะห์ Wavefunction หลังจากการคำนวณเป็นสิ่งที่สำคัญมากเพราะจะช่วยให้เราเข้าใจถึงพฤติกรรมเชิงอิเล็กทรอนิกส์ของอะตอมภายในโมเลกุล โดยหนึ่งในคุณสมบัติเชิงอิเล็กทรอนิกส์ที่นักเคมีทฤษฎีมักจะทำการวิเคราะห์เป็นอันดับแรกเสมอ (นอกจากพลังงานเชิงอิเล็กทรอนิกส์) นั้นก็คือประจุย่อยของแต่ละอะตอม (Partial Atomic Charge) ซึ่งมีความสำคัญมากเพราะว่าถ้าหากเราทราบค่าประจุย่อยของอะตอมในโมเลกุลจะช่วยทำให้สามารถเข้าใจว่าอะตอมแต่ละตัวส่งผลหรือมี Contribution มากน้อยเพียงใดเมื่อเทียบกับอะตอมอื่น ๆ ภายในโมเลกุลเดียวกัน

ต้องอธิบายก่อนว่าประจุรวมของโมเลกุลเกิดจากผลรวมของประจุย่อยของอะตอมแต่ละตัว (Partial Atomic Charge) ซึ่งในทางทฤษฎีและการทดลองเราไม่มีนิยามที่แน่นอนในการระบุหรือกำหนดประจุย่อยของอะตอมแต่ละตัวในโมเลกุล อย่างไรก็ตามได้มีทฤษฎีต่าง ๆ มากมายที่ถูกเสนอและพัฒนามาเพื่อใช้ในการคำนวณประจุย่อย โดยแนวคิดแรก ๆ ก็ได้ถูกพัฒนามานานหลายสิบปีแล้ว หนึ่งในนั้นก็คือทฤษฎีที่ตั้งอยู่บนแนวคิดพื้นฐานของ Wavefunction ซึ่งใช้ Population ของ Atomic Orbital (Basis Set) เช่น Mulliken Population ที่เราสามารถนำมาใช้ในการคำนวณประจุอะตอม เรียกว่าประจุของมัลลิเคน (Mulliken Charge) ซึ่งออร์บิทัลของอะตอมที่อยู่ในโมเลกุลถูกกำหนดและแบ่งด้วยเมทริกซ์ซ้อนทับ (Overlap Matrix) เนื่องจากว่าประจุย่อยเป็นปริมาณที่วัดไม่ได้ (Non-observable Property) กล่าวคือไม่สามารถวัดได้ในทางทดลอง นั่นก็เพราะว่าจริง ๆ แล้วโมเลกุลนั้นไม่ได้เกิดขึ้นจากที่อะตอมแต่ละตัวมาต่อติดกัน แต่ว่าเป็นกลุ่มก้อนอะตอมที่มารวมกันแบบราบเรียบ (Smooth) ดังนั้นเราจึงไม่สามารถที่จะระบุได้ว่าอะตอมแต่ละตัวนั้นมีขนาดเล็กหรือใหญ่แค่ไหน กล่าวคือเราไม่รู้ว่าอะตอมและตัวเริ่มต้นตรงไหนและสิ้นสุดตรงไหน นั่นจึงทำให้เราไม่รู้ขอบเขตของแต่ละอะตอมนั่นเอง

นอกจากนี้ยังมีอีกหลายทฤษฎี/วิธีที่ได้ถูกพัฒนาขึ้นมาเพื่อกำหนดนิยามของประจุย่อยของอะตอม ดังนี้

1. วิธีที่ใช้ออร์บิทัล (Orbital-based) ซึ่งเป็นการทำ Population โดยใช้ Basis Functions

    (a) Mulliken Charge[55] เป็นวิธีที่ได้รับความนิยมเป็นอย่างมากสำหรับคำนวณประจุย่อยของอะตอมซึ่งโปรแกรมเคมีเชิงคำนวณหลาย ๆ โปรแกรมมักจะคำนวณ Mulliken Charge ให้โดยอัตโนมัตินั่นก็เพราะว่าสามารถคำนวณได้ง่ายโดยใช้ Density Matrix (P) และ Overlap Matrix (S) ซึ่งได้จากการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์ ซึ่งวิธี Mulliken นั้นจะเกี่ยวข้องกับการแบ่ง (Partition) ผลคูณระหว่าง P และ S ดังนี้

$$\rho_A = \sum_{\alpha \in A}^{M_{\text{basis}}} \sum_{\beta}^{M_{\text{basis}}} P_{\alpha\beta} S_{\alpha\beta} \tag{8.7}$$

โดยที่ ρ_A คือ Population ของอะตอม A และ α และ β คือออร์บิทัลเชิงอะตอม (Atomic Orbitals, AOs) ซึ่งประจุของอะตอม A นั้นสามารถหาจากได้ผลต่างจากเลขนิวเคลียร์ของอะตอม ดังนี้

$$Q_A = Z_A - \rho_A \tag{8.8}$$

โดยการใช้ผลคูณ $\sum P \cdot S$ นั้นเป็นการกระจายอิเล็กตรอนให้อยู่ในรูปของการมีส่วมรวมเชิงอะตอม (Atomic Contributions) นั่นเอง หมายความว่าสมาชิกแนวทแยง $P_{\alpha\alpha} S_{\alpha\alpha}$ นั้นเป็นจำนวนของอิเล็กตรอนและสมาชิกนอกแนวทแยง $P_{\alpha\beta} S_{\alpha\beta}$ เป็นจำนวนครึ่งหนึ่งของจำนวนอิเล็กตรอนที่ถูกแชร์ระหว่าง AOs α และ β

(b) Löwdin Charge[88] วิธีต่อมาที่เปรียบเสมือนเป็นพี่น้องกับ Mulliken Charge นั่นก็คือประจุ Löwdin โดยจะเป็นเมทริกซ์ซ้อนทับแบบแบ่งส่วน (Partition Overlap Matrix) ดังนี้

$$\sum P \cdot S = N_{\text{elec}} \tag{8.9}$$

$$\sum S^{\frac{1}{2}} \cdot (P \cdot S) \cdot S^{-\frac{1}{2}} = S^{\frac{1}{2}} N_{\text{elec}} S^{-\frac{1}{2}} \tag{8.10}$$

$$\sum S^{\frac{1}{2}} \cdot D \cdot S^{\frac{1}{2}} = N_{\text{elec}} \tag{8.11}$$

ถ้าหากผู้อ่านสังเกตดี ๆ จะพบว่าทั้งวิธี Mulliken และ Löwdin นั้นต่างก็เป็นเพียงแค่รูปแบบเฉพาะของการทำ Population Analysis ที่ใช้ $S^n \cdot P \cdot S^{1-n}$ เท่านั้น โดยที่กรณีที่ $n = 0$ นั้นก็จะเป็น Mulliken และกรณีที่ $n = \frac{1}{2}$ นั้นก็จะเป็น Löwdin

อย่างไรก็ตามไม่มีวิธีไหนนั้นดีที่สุด ถึงแม้ว่าการทำ Population Analysis โดยอ้างอิงกับ Basis Function นั้นจะง่ายแต่ก็มีปัญหาอยู่หลายข้อด้วยกัน ดังนี้

(a) สมาชิกแนวทแยงของ Population สำหรับบางออร์บิทัลนั้นอาจจะมีค่ามากกว่า 2 ได้ ซึ่งขัดแย้งกับหลักการของเพาลี

(b) สมาชิกนอกแนวทแยงอาจจะมีค่าเป็นลบได้ ซึ่งบอกเป็นนัย ๆ ว่าจำนวนอิเล็กตรอนระหว่าง Basis Function นั้นเป็นลบด้วยซึ่งไม่มีทางเป็นไปได้

(c) เราไม่สามารถหาเหตุผลมาอธิบายได้ว่าเมื่อไหร่ควรจะใช้ Population Method แบบนี้เพราะว่าเราไม่รู้ว่าการที่เราแบ่งครึ่งออร์บิทัลออกจากกันแบบเท่า ๆ กันนั้นจะสอดคล้องกับค่า Electronegativity ของอะตอมด้วยหรือไม่

2. วิธีที่ใช้ศักย์เชิงไฟฟ้าสถิตย์ (Electrostatic Potential-based)

เป็นวิธีที่ถูกพัฒนาโดยใช้หลักการระหว่างศักย์เชิงไฟฟ้าสถิตย์ (Electrostatic Potential หรือ ESP) กับอนุภาคที่มีประจุ โดย ESP ที่ตำแหน่ง $\boldsymbol{r}$ สามารถคำนวณได้จากผลรวมของ Contribution จากนิวเคลียสและความหนาแน่นของอิเล็กตรอน ดังนี้

$$\phi_{\text{ESP}}(\boldsymbol{r}) = \sum_{A}^{\text{nuclei}} \frac{Z_A}{|\boldsymbol{r} - \boldsymbol{R}_A|} - \int \frac{\rho(\boldsymbol{r}')}{|\boldsymbol{r} - \boldsymbol{r}'|} d\boldsymbol{r}' \tag{8.12}$$

โดยที่

$$\rho(\boldsymbol{r}') = |\Psi(\boldsymbol{r}')|^2 \tag{8.13}$$

เทอมแรกของสมการที่ (8.12) สามารถคำนวณได้อย่างง่าย ๆ จากประจุนิวเคลียร์และตำแหน่งของอะตอมแต่ว่าเทอมที่สองของสมการซึ่งเป็นผลจากอิเล็กตรอนนั้นจำเป็นที่จะต้องใช้ Wavefunction เข้ามาช่วยซึ่งทำได้ไม่ง่าย นอกจากนี้แล้วเทอมที่สองนั้นไม่ได้ถูกรวมเข้าไปในบางวิธี เช่น วิธี Force Field ที่ใช้ในการจำลองแบบ Molecular Dynamics ดังนั้นวิธีที่ง่ายที่สุดในการจำลองเทอมที่สองหรือ Electrostatic Potential นั้นก็คือการกำหนดค่าประจุย่อยให้แต่ละอะตอมโดยตรงเลย โดยพารามิเตอร์ของ Force Field นั้นสามารถหาได้จากการ Fit ค่าเทียบกับข้อมูลเชิงการทดลอง เช่น Dipole Moment, Quadrupole Moment, หรือ Octopole Moment

นอกจากนี้แล้วยังมีวิธีอื่น ๆ อีกหลายวิธีในการกำหนด ESP สำหรับการคำนวณค่าประยุย่อยเชิงอะตอม โดยวิธีเหล่านั้นก็มีความแตกต่างกันที่วิธีที่ใช้ในการกำหนดจำนวนและตำแหน่งของจุด (Point) ที่ใช้ในการสุ่มตัวอย่าง (Sampling) ESP โดยจำนวนจุดที่ใช้ในการ Sampling นั้นมักจะอยู่ที่ประมาณหลักร้อยจุดรอบ ๆ นิวเคลียส โดยระยะห่างระหว่างจุดถึงนิวเคลียสนั้นมักจะถูกกำหนดให้ไม่เกินสองเท่าของรัศมี van Der Waals สำหรับวิธีอื่น ๆ ของ ESP นั้นมีดังต่อไปนี้

(a) Merz-Singh-Kollman (MSK) Charge[89]
(b) Electrostatic Potential (CHELP) Charge[90]
(c) Electrostatic Potential on a Grid (CHELPG) Charge[91]
(d) Restrained Electrostatic Potential (RESP)[92]

3. วิธีที่ใช้ความหนาแน่นของอิเล็กตรอน (Electron Density-based)

ในหัวข้อก่อนหน้านี้ผู้อ่านได้ศึกษาการหา Population โดยอ้างอิงกับ Wavefunction ไปแล้ว ในหัวข้อนี้จะเป็นการวิเคราะห์ Population โดยใช้ความหนาแน่นของอิเล็กตรอน ซึ่งตามที่เราได้ศึกษาไปในหัวข้อ 7.4 แล้วว่าความหนาแน่นของอิเล็กตรอนนั้นคือ Wave Function ที่ถูกยกกำลังสองซึ่งถูกอินทิเกรตทั่วทั้ง Coordinate ของอิเล็กตรอนจำนวน $N_{\text{elec}}-1$ โดยความยากในการแบ่ง(Partitioning) ความหนาแน่นของอิเล็กตรอนให้เป็นผลรวมของการมีส่วนร่วมเชิงอะตอม (Atomic Contribution)[1] นั้นคือขึ้นอยู่กับว่าเราจะกำหนดคำว่าอะตอมที่อยู่ภายในโมเลกุลอย่างไร ซึ่งตรงจุดนี้แหละที่เป็นความ

[1]ผู้เขียนแปลคำว่า Contribution เป็นการมีส่วนร่วมก็เพราะว่าเป็นการบ่งบอกว่าอะตอมแต่ละอะตอมนั้นส่งผลในเชิงอิเล็กทรอนิกส์ต่อโมเลกุลเทียบกับอะตอมตัวอื่น ๆ มากน้อยแค่ไหน

ยากเพราะว่าจริง ๆ แล้วรอยต่อระหว่างอะตอมนั้นมันราบเรียบและเราไม่สามารถหารอยต่อหรือขอบเขตที่ชัดเจนและแน่นอนได้

ไอเดียก็คือถ้าหากว่าผลรวมของปริมาตรเชิงโมเลกุลนั้นสามารถถูกแบ่งออกเป็นปริมาตรส่วนย่อย ๆ ได้แล้วปริมาตรย่อย ๆ แต่ละส่วนนั้นก็เป็นส่วนหนึ่งของนิวเคลียส เราจะสามารถอินทิเกรตความหนาแน่นของอิเล็กตรอนและคำนวณจำนวนของอิเล็กตรอนในรูปของเชิงอะตอมได้ (Ω) แล้วสิ่งที่เราจะทำได้เพิ่มเติมก็คือประจุเชิงอะตอม Q นั้นสามารถหาได้จากประจุเชิงนิวเคลียร์ Z ดังนี้

$$N_A = \int_{\Omega} \rho(\boldsymbol{r}) d\boldsymbol{r} \tag{8.14}$$

$$Q_A = Z_A - N_A \tag{8.15}$$

โดยสมการที่ (8.14) สามารถถูกทำให้อยู่ในรูปทั่วไป (Generalized) ได้ตามสมการดังต่อไปนี้

$$N_A = \int_{\Omega} w_A \boldsymbol{r} \rho(\boldsymbol{r}) d\boldsymbol{r} \tag{8.16}$$

โดยที่ $w_A(\boldsymbol{r})$ คือฟังก์ชันถ่วงน้ำหนักที่กำหนดค่าสัดส่วนของความหนาแน่นของอิเล็กตรอนที่ตำแหน่ง $\boldsymbol{r}$ ที่ขึ้นอยู่กับอะตอม A

นอกจากนี้ยังมีทฤษฎีเพิ่มเติมที่ได้มีการนำเสนอแนวคิดที่น่าสนใจเกี่ยวกับการแบ่งโมเลกุลออกเป็นอะตอม เช่น

(a) Hirshfeld Charge[93]

(b) Atoms in Molecules (AIM) หรือ Bader Charge[94,95]

ซึ่งผมขอไม่ลงรายละเอียดครับ

4. เทนเซอร์เชิงขั้ว (Polar Tensor)[96,97]

อีกหนึ่งวิธีที่น่าสนใจที่ถูกพัฒนาขึ้นมาเพื่อคำนวณประจุย่อยเชิงอะตอมนั้นก็คือเทนเซอร์เชิงขั้วของอะตอม (Atomic Polar Tensor หรือ APT) โดยถูกพิสูจน์มาจากอนุพันธ์ของ Dipole Moment เทียบกับตำแหน่งของนิวเคลียสซึ่งเป็นตัวที่กำหนดความเข้ม (Intensity) ของการดูดกลืนแบบ Infrared โดยสมการสำหรับการคำนวณประจุ APT นั้นมีดังต่อไปนี้

$$q_i^{\text{APT}} = \frac{1}{3}\left(\frac{\partial \mu_x}{\partial x_i} + \frac{\partial \mu_y}{\partial y_i} + \frac{\partial \mu_z}{\partial z_i}\right) \tag{8.17}$$

การคำนวณประจุย่อยเชิงอะตอมด้วยวิธี APT นั้นมีข้อดีอย่างหนึ่งคือเราสามารถเชื่อมโยงและหาความสัมพันธ์กับค่าที่ได้จากการทดลองได้นั่นก็คือ Infrared Spectrum ซึ่งเป็นปริมาณที่ตรวจวัดและสังเกตได้ (Observable Property) อย่างไรก็ตามการคำนวณประจุเชิงอะตอมด้วย APT นั้นมีความสิ้นเปลืองสูงมากและค่าประจุที่ได้นั้นขึ้นอยู่กับปริมาณของเทอมสหสัมพันธ์ระหว่างอิเล็กตรอนที่อยู่ใน Wavefunction จึงทำให้การใช้งาน APT นั้นไม่ค่อยแพร่หลายมากนัก

8.3 พลังงานของออร์บิทัล

8.3.1 พลังงานของ HOMO และ LUMO

8.3.2 ผลต่างของพลังงานของ HOMO และ LUMO

8.4 พื้นผิวพลังงานศักย์

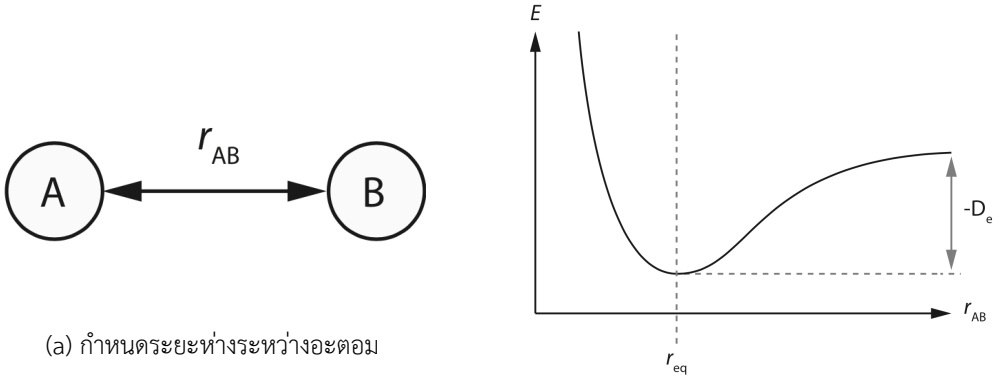
หนึ่งในหัวข้อที่สำคัญของเคมีเชิงคำนวณก็คือพื้นผิวพลังงานศักย์ (Potential Energy Surface) ซึ่งเป็นสิ่งที่อธิบายความสัมพันธ์ระหว่างรูปร่างเชิงเรขาคณิตของโมเลกุล (Molecular Geometry) เช่น ตำแหน่งที่สัมพันธ์กันของอะตอมในโมเลกุลและพลังงานเชิงโมเลกุล พื้นผิวพลังงานศักย์ที่เราจะมาศึกษากันในบทนี้จะเป็นพื้นผิวแบบง่ายสำหรับโมเลกุลเล็ก ๆ เช่น โมเลกุลอะตอมคู่ (Diatomic Molecular) และ โมเลกุลที่มีสามอะตอม เพื่อให้ง่ายต่อการอ่านและเพื่อความกระชับ ผู้เขียนจะขอใช้ตัวย่อ PES ซึ่งย่อมาจาก Potential Energy Surface แทนการเรียกพื้นผิวพลังงานศักย์ซึ่งจะยาวเกินไป

8.4.1 พื้นผิวพลังงานศักย์สำหรับโมเลกุลอะตอมคู่

โดยทั่วไปแล้ว PES สำหรับระบบที่ประกอบไปด้วยอะตอมหลายอะตอมนั้นจริง ๆ แล้วก็เป็นฟังก์ชันหลายมิติเชิงซ้อนแบบหนึ่ง ซึ่งเรียกเป็นภาษาอังกฤษว่า Complex Multidimensional Function ตัวอย่างเช่นเรามีระบบ (โมเลกุล) ที่มีอะตอม N อะตอม ความสัมพันธ์ระหว่างอะตอมภายในระบบนี้สามารถถูกอธิบายได้ด้วย Degree of Freedom ซึ่งมีจำนวนเท่ากับ $3N-6$ สำหรับกรณีโมเลกุลที่ไม่เป็นเชิงเส้น เช่น โมเลกุลน้ำ (H_2O) และมีจำนวนเท่ากับ $3N-5$ สำหรับกรณีที่โมเลกุลเป็นแบบเชิงเส้น เช่น โมเลกุลแก๊สคาร์บอนไดออกไซด์ (CO_2) ซึ่งการที่ฟังก์ชัน Degree of Freedom มีจำนวนมิติที่มากเกินกว่า 3 มิตินี้ทำให้ยากต่อมองและวิเคราะห์ PES ดังนั้นวิธีที่ง่ายที่สุดคือเรามักจะทำการพิจารณาเฉพาะ Degree of Freedom ที่สำคัญและเกี่ยวข้องกับการเปลี่ยนแปลงของระบบและพลังงาน โดยที่เราเรียกพารามิเตอร์ที่เราทำการเปลี่ยนค่าไปเรื่อย ๆ เพื่อดูผลต่อการเปลี่ยนแปลงพลังงานของโมเลกุลนี้ว่าพิกัดของปฏิกิริยา (Reaction Coordinates)

เรามาเริ่มกันด้วยตัวอย่างแรกด้วย PES ของอะตอมคู่ตามที่แสดงในภาพที่ 8.2a

สำหรับคู่อะตอม A และ B มี Degree of Freedom เพียงแค่ 1 Degree เท่านั้น และกำหนดให้ระยะห่างระหว่างอะตอมเป็น r_{AB} ถ้าอะตอม A มีการสร้างพันธะกับอะตอม B สิ่งที่เกิดขึ้นคือเราสามารถคำนวณ PES ได้โดยใช้วิธีการคำนวณทางเคมีควอนตัมทั่วไปหรือทำนายโดยใช้เทคนิค ML ซึ่งจะได้ PES ที่มีลักษณะตามที่แสดงในภาพที่ 8.2b โดยที่แกน x คือ Degree of Freedom ซึ่งก็คือระยะห่างระหว่างอะตอม r_{AB} และแกน y คือพลังงานของโมเลกุล โดย PES นี้มีจุดต่ำสุดของค่าพลังงานเพียงแค่จุดเดียวที่ตำแหน่งระยะห่าง



(a) กำหนดระยะห่างระหว่างอะตอม

(b) พลังงานศักย์ของโมเลกุลคู่

ภาพ 8.2 โมเลกุลคู่

สมดุล r_{eq} ซึ่งเป็นระยะห่างที่เหมาะสมที่สุดระหว่างอะตอม A กับอะตอม B

ถ้าหากเราขยับให้อะตอม A กับอะตอม B เข้ามาใกล้กันสิ่งที่เกิดขึ้นคือพลังงานของโมเลกุลจะเพิ่มขึ้นอย่างต่อเนื่องทันทีเนื่องจากว่ามีแรงผลักระหว่างนิวเคลียสที่เพิ่มมากขึ้นนั่นเอง ซึ่งแรงผลักดังกล่าวที่ระยะห่างระหว่างอะตอมสั้น ๆ นี้คือแรงผลักเพาลี (Pauli Repulsion) ที่อ้างอิงด้วยหลักกีดกันของเพาลีนั่นเอง

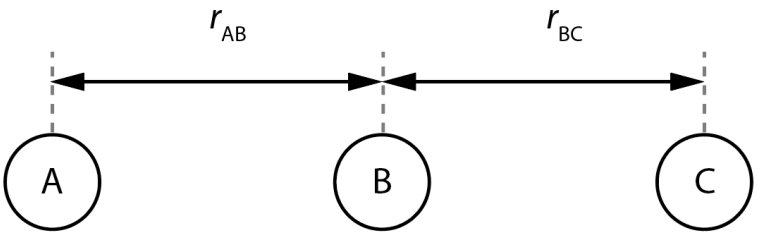
ในทำนองเดียวกัน เมื่อเราทำการขยับอะตอมทั้งสองให้ห่างออกจากกันพลังงานของโมเลกุลก็จะเพิ่มขึ้นเช่นเดียวกัน แต่จะแตกต่างจากกรณีที่ขยับอะตอมเข้าหากันก็คือพลังงานที่เพิ่มขึ้นในกรณีนี้จะเพิ่มขึ้นในอัตราที่ช้ากว่าและจะเพิ่มขึ้นไปถึงค่า ๆ หนึ่งเท่านั้นซึ่งจะสอดคล้องกับการที่พันธะระหว่างอะตอมทั้งสองนั้นได้สลายไป โดยความแตกต่างระหว่างพลังงานที่ต่ำที่สุดของโมเลกุล ณ ระยะห่างสมดุลกับพลังงานที่สูงที่สุดของโมเลกุลได้ลู่เข้า ณ ระยะห่างที่มาก ๆ นั้นจะแทนด้วย $-D_e$ ซึ่งเป็นความลึกของบ่อพลังงานศักย์ของพันธะระหว่างอะตอม A กับอะตอม B

หมายเหตุ ค่าพลังงาน D_e นั้นไม่ใช่ค่าพลังงานการแตกสลายของพันธะ (Bond Dissociation Energy หรือ D_0) เสียทีเดียวเพราะว่าเราไม่ได้พิจารณาพลังงานจุดศูนย์เชิงการสั่น (Vibrational Zero-point Energy) เข้าไปด้วย โดยค่า D_e นั้นจะมีค่าที่สูงกว่า D_0 อยู่เล็กน้อย

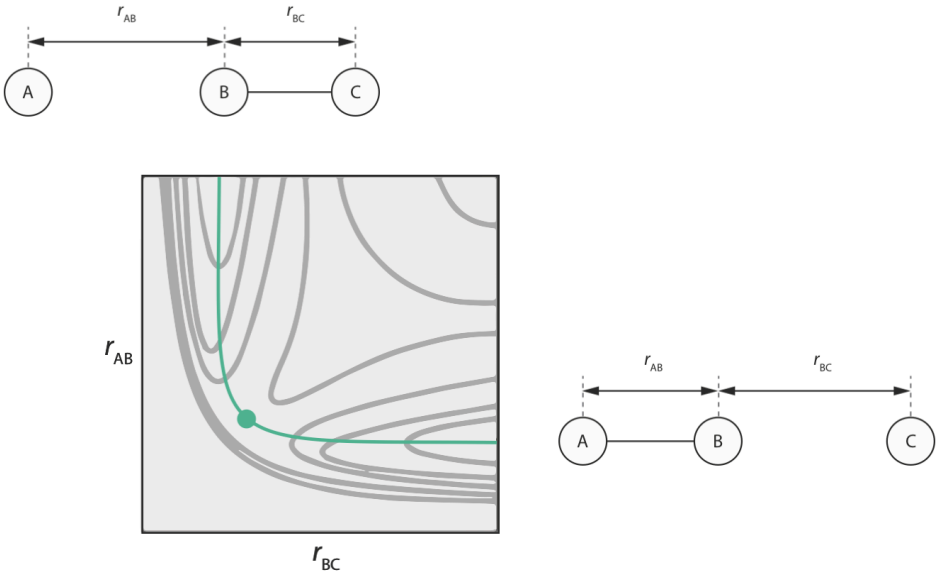
8.4.2 พื้นผิวพลังงานศักย์สำหรับโมเลกุลที่มีสามอะตอม

กรณีที่เราจะศึกษากันเป็นลำดับต่อไปก็คือระบบโมเลกุลที่มีสามอะตอมซึ่งจะแบ่งออกได้เป็นสองกรณีคือกรณีที่โมเลกุลเป็นเส้นตรง (อะตอมทั้งหมดเรียงอยู่ในแนวเส้นตรงเดียวกัน) และกรณีที่โมเลกุลไม่เป็นเส้นตรง (อะตอมไม่ได้อยู่ในแนวเส้นตรงเดียวกัน)

สำหรับกรณีแรกนั้นดูได้ตามภาพที่ 8.3 โดย PES ที่แสดงนั้นเป็นแบบสองมิติ กล่าวคือเป็นพื้นผิวที่เป็นฟังก์ชันกับระยะห่างระหว่างอะตอม A กับอะตอม B (r_{AB}) และระยะห่างระหว่างอะตอม B กับอะตอม C



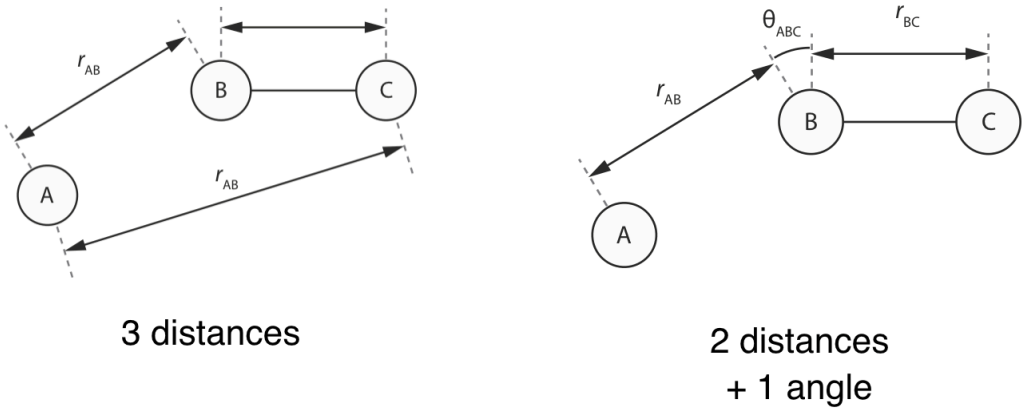
(a) กำหนดระยะห่างระหว่างอะตอม



(b) พลังงานศักย์ของโมเลกุลสามอะตอมแบบเชิงเส้นตรงร่วม

ภาพ 8.3 โมเลกุลที่มีสามอะตอม

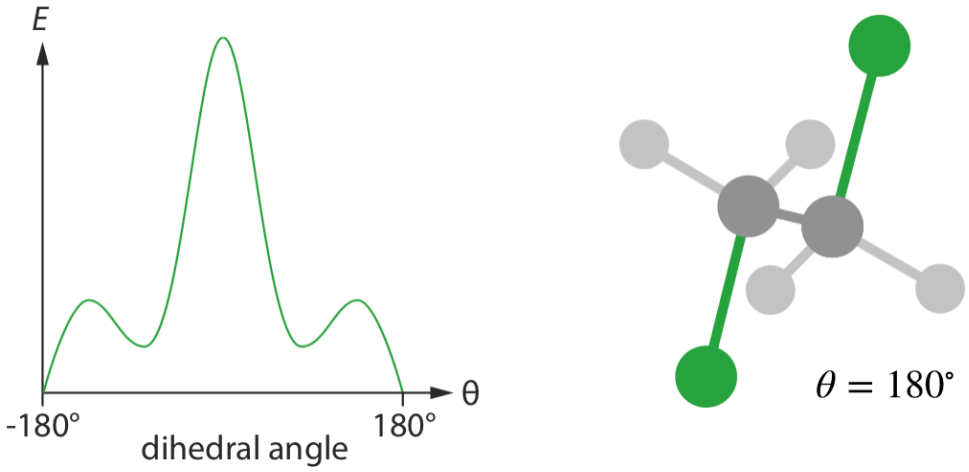
(r_{BC})



ภาพ 8.4 โมเลกุลสามอะตอมแบบไม่เป็นเชิงเส้นตรงร่วม

สำหรับกรณีที่อะตอมทั้งสามอะตอมนั้นไม่ได้เรียงอยู่ในแนวเส้นตรงเดียวกันนั้นจะมีความซับซ้อนมากกว่ากรณีแบบแรกมากนั่นก็เพราะว่ากรณีนี้เรามีจำนวน Degree of Freedom ทั้งหมดคือ 3 ทำให้ PES นั้นมีจำนวณ 4 มิติ (3 มิติแรกคือจำนวน Degree of Freedom และมิติที่ 4 คือพลังงาน) โดยภาพที่ 8.4 แสดงระบบพิกัดที่ประกอบไปด้วยระยะห่างและมุมระหว่างอะตอมที่อธิบายเรขาคณิตของโมเลกุล

8.4.3 พื้นผิวพลังงานศักย์สำหรับโมเลกุลที่มีมากกว่าสามอะตอม



ภาพ 8.5 พลังงานศักย์ของโมเลกุล $C_{22}H_4Cl_2$

สำหรับระบบที่มีจำนวนอะตอมมากกว่า 3 อะตอมนั้นจำนวน Degree of Freedom จะเพิ่มขึ้นอย่างรวดเร็วและจะเพิ่มความลำบากและความซับซ้อนให้กับเราในการอธิบาย PES ของโมเลกุลอย่างมหาศาล ดัง

นั้นเรามักจะทำการเลือกเฉพาะ Descriptor หรือ Degree of Freedom ที่สำคัญบางอันที่สามารถอธิบายการเปลี่ยนแปลงของโมเลกุลได้เป็นอย่างดีในการนำมาคำนวณ PES ตัวอย่างเช่นภาพที่ 8.5 ซึ่งเป็น PES ของโมเลกุล $C_{22}H_4Cl_2$ ที่เราเลือกใช้ Dihedral Angle ของอะตอม Cl—C—C—Cl มาใช้ในการอธิบาย PES

8.5 ไดโพลโมเมนต์

ไดโพลโมเมนต์ (Dipole Moment) คือสภาพมีขั้วไฟฟ้าที่เกิดขึ้นจากการที่กลุ่มของอิเล็กตรอนในโมเลกุลนั้นมีการกระจายตัวที่ไม่สม่ำเสมอ โดยบริเวณที่มีอิเล็กตรอนหนาแน่นมากกว่าจะประพฤติตัวเป็นขั้วลบ ส่วนบริเวณที่มีความหนาแน่นอิเล็กตรอนน้อยกว่าจะประพฤติตัวเป็นขั้วบวก ซึ่งมักจะอยู่ทางด้านเดียวกับนิวเคลียสด้วย โดยขั้วไฟฟ้าทั้งสองนี้จะอยู่ด้วยกันเป็นคู่ ๆ และอยู่ตรงข้ามกันเสมอ

ตามธรรมชาตินั้นประจุไฟฟ้าที่วางอยู่ในสนามไฟฟ้าจะถูกออกแรงกระทำทั้งในรูปของแรงผลักและแรงดูดซึ่งขึ้นอยู่กับชนิดของประจุไฟฟ้า กล่าวคือสนามไฟฟ้าออกแรงผลักแก่ประจุไฟฟ้าบวกและออกแรงดึงดูดแก่ประจุไฟฟ้าลบให้เคลื่อนที่ตามแนวของเส้นสนามไฟฟ้า และสำหรับไดโพลโมเมนต์เมื่อวางอยู่ในสนามไฟฟ้าก็จะถูกสนามไฟฟ้านี้เหวี่ยงด้วยแรงบิดหรือทอร์กทำให้ขั้วไฟฟ้าทั้งสองของอะตอมเรียงตัวใหม่ หากสนามไฟฟ้ามีลักษณะเอกรูป (Uniform) หรือมีความสม่ำเสมอ แรงลัพธ์ทางไฟฟ้าที่เกิดขึ้นต่อประจุทั้งสองจะทำให้ไดโพลโมเมนต์นี้เคลื่อนที่ด้วยความเร่งออกไปตามเส้นของสนามไฟฟ้าต่อไปอย่างช้า ๆ ได้บ้างถ้าหากเป็นไดโพลโมเมนต์ของอะตอมอิสระ

8.6 สภาพการเกิดขั้ว

สภาพการเกิดขั้ว (Polarizability) นั้นเป็นการบ่งบอกถึงแนวโน้มที่โมเลกุลนั้นจะมีการสร้างการเหนี่ยวนำไดโพลโมเมนต์เชิงไฟฟ้า (Induced Electric Dipole Moment) ขึ้นมาเมื่อมีการใส่สนามไฟฟ้าเข้าไปให้กับโมเลกุล ซึ่งแนวคิดของการพัฒนาทฤษฎีที่ใช้ในการอธิบาย Polarizability นั้นเริ่มจากการตั้งสมมติฐานว่าทั้งนิวเคลียสและอิเล็กตรอนนั้นต่างก็สามารถเคลื่อนที่ได้อย่างอิสระ ไม่มีการถูกตรึง (Fixed) ไว้อยู่กับที่ เมื่อโมเลกุลนั้นถูกรบกวนด้วยสนามไฟฟ้าภายนอก ทำให้เกิดการเกาะกลุ่มรวมกันของนิวเคลียสและอิเล็กตรอนภายในโมเลกุล แยกกันไปคนละฝั่ง ทำให้เกิดประจุที่ถูกเหนี่ยวนำขึ้นมาภายในโมเลกุล ดังนั้นการที่เราเข้าใจ Polarizability ของโมเลกุลนั้นก็จะช่วยให้เราเข้าใจอันตรกิริยาระหว่างอะตอมและโมเลกุลที่ไม่มีขั้ว รวมถึงโมเลกุลที่มีขั้วทางไฟฟ้าด้วย เช่น โมเลกุลที่มีไดโพลโมเมนต์

8.7 เทคนิคสเปกโทรสโกปีแบบสั่น

สเปกโทรสโกปี (Spectroscopy) เป็นการศึกษาอันตรกิริยา (Interaction) ระหว่างสสารกับรังสีแม่เหล็กไฟฟ้า (Electromagnetic Radiation) ที่เกิดจากการเปลี่ยนระดับพลังงานของอิเล็คตรอน การเปลี่ยนระดับพลังงานการหมุน (Rotation) และการสั่นสะเทือน (Vibration) ของโมเลกุล ซึ่งการที่เราทราบจากสเปกตรัมของโมเลกุลจะทำให้เราทราบข้อมูลหลายอย่างเกี่ยวกับโครงสร้างของโมเลกุลของสสารและสมบัติทางเคมี เช่น

- สมมาตรของโมเลกุล (Symmetry)
- ความยาวพันธะ (Bond Length)
- มุมพันธะ (Bond Angle)
- ความแข็งแรงของพันธะ (Bond Strength)
- การเปลี่ยนแปลงภายในโมเลกุล
- การเปลี่ยนแปลงระหว่างโมเลกุล

โดยในหัวข้อนี้เราจะมาดูรายละเอียดเกี่ยวกับการคำนวณความเข้มของการดูดกลืนสำหรับเทคนิค Infrared (IR) และรามาน (Raman) ซึ่งทั้งสองเทคนิคนี้ต่างก็เป็นเทคนิคสเปกโทรสโกปีแบบสั่น (Vibrational Spectroscopy) ซึ่งมีการนำมาใช้ในการทำงานวิจัยสำหรับการศึกษาคุณสมบัติของโมเลกุลอย่างแพร่หลาย

8.7.1 อินฟราเรดสเปกโทรสโกปี

อินฟราเรดสเปกโทรสโกปี (IR Spectroscopy) เป็นการวัดการดูดกลืนของการแผ่รังสีของโมเลกุลในช่วงอินฟราเรดซึ่งเกี่ยวข้องกับการเปลี่ยนแปลงของอิเล็กทริกไดโพลโมเมนต์ (Electric Dipole Moment) ของโมเลกุลที่ศึกษา สำหรับการคำนวณความเข้มของการดูดกลืน IR ในรูปแบบของวิธีแบบ Dynamic นั้นสามารถทำได้โดยใช้สมการ (ความสมพันธ์) ดังต่อไปนี้[98]

$$I_{IR}(\omega) \propto \int \langle \dot{\boldsymbol{\mu}}(\tau)\dot{\boldsymbol{\mu}}(\tau+t)|\dot{\boldsymbol{\mu}}(\tau)\dot{\boldsymbol{\mu}}(\tau+t)\rangle_{\tau} e^{-i\omega t} dt \tag{8.18}$$

โดยที่ $\dot{\boldsymbol{\mu}}$ คืออนุพันธ์ของไดโพลโมเมนต์เทียบกับเวลา, ω คือความถี่เชิงการสั่น (Vibrational Frequency), τ คือเวลาที่เปลี่ยนแปลงไปอย่างช้า ๆ และ t คือเวลาสำหรับการทำ Integration นอกจากนี้ยังจะสังเกตได้ว่าจะมีเทอม $\langle \dot{\boldsymbol{\mu}}(\tau)\dot{\boldsymbol{\mu}}(\tau+t)|\dot{\boldsymbol{\mu}}(\tau)\dot{\boldsymbol{\mu}}(\tau+t)\rangle_{\tau}$ ซึ่งจะเป็นตัวที่บ่งบอกถึงสหสัมพันธ์ของเวลา (Time Correlation) ของ $\dot{\boldsymbol{\mu}}$

สำหรับกรณีที่เป็นแบบ Static นั้น สเปกตรัมของ IR สามารถคำนวณได้ผ่านอนุพันธ์ของไดโพลโมเมนต์เทียบกับพิกัดหรือตำแหน่งของโหมดการสั่นแบบปกติ (Normal Coordinates) ซึ่งจะไม่ขึ้นกับเวลา ด้วยสมการดังต่อไปนี้

$$\boldsymbol{\mu} = \sum_{\mu\nu} P_{\mu\nu} \langle \phi_\mu | \boldsymbol{r} | \phi_\nu | \phi_\mu | \boldsymbol{r} | \phi_\nu \rangle \tag{8.19}$$

$$\boldsymbol{\mu} = \sum_{J} q_J \boldsymbol{R_J} \tag{8.20}$$

โดยที่สมการ (8.19) จะเป็นสำหรับกรณีแบบควอนตัมซึ่งจะคำนวณผ่านเมทริกซ์ความหนาแน่นและ Basis Function แต่สมการ (8.20) จะเป็นสำหรับกรณีแบบดั้งเดิมซึ่งจะคำนวณผ่านจุดประจุ (Point Charge) และพิกัดคาร์ทีเซียนของอะตอม

8.7.2 รามานสเปกโทรสโกปี

รามานสเปกโทรสโกปี (Raman Spectroscopy) เป็นเทคนิคหนึ่งที่เปรียบเสมือนเป็นพี่น้องกับเทคนิคอินฟราเรดสเปกโทรสโกปี โดยที่ Raman Spectroscopy จะเป็นผลมาจากการเกิดการกระเจิงของแสงแบบไม่ยืดหยุ่นในช่วงอินฟราเรด (Infraref), วิสิเบิล (Visible), และอัลตราไวโอเล็ต (Ultraviolet) ซึ่งเกี่ยวข้องกับการเปลี่ยนแปลงสภาพการเกิดขั้ว (Polarizability) แบบอิเล็กทริกไดโพล-อิเล็กทริกไดโพล (Electric-dipole–electric-dipole) ของสสาร โดยความเข้มของการกระเจิงแบบรามาน (I_{Raman}) สามารถคำนวณได้ด้วยความสัมพันธ์ดังต่อไปนี้[98]

$$I_{Raman}(\omega) \propto \frac{(\omega_{in} - \omega)^4}{\omega} \frac{1}{1 - \exp\left(-\frac{\hbar\omega}{k_B T}\right)} S(a^2, \gamma^2) \tag{8.21}$$

โดยที่ $S(a^2, \gamma^2)$ คือตัวแปรที่เป็นผลจากการรวมกันของความคงที่ (ไม่เปลี่ยนแปลง) แบบไอโซโทรปิค (Isotropic) และแอนิโซโทรปิค (Anisotropic)[1] ของเทนเซอร์แบบ Placzek-type Polarizability $(\boldsymbol{\alpha})$,[99] ω คือความถี่เชิงการสั่น, ω_{in} คือความถี่ของแสดง, k_B คือค่าคงที่ของโบลทซ์มานน์ (Boltzmann Constant) และ T คืออุณหภูมิของระบบในหน่วย Kelvin โดยสมการที่จะใช้ในการอธิบาย $S(a^2, \gamma^2)$ จะขึ้นอยู่กับรูปแบบของการทดลองและสมการของ Time Correlation[100]

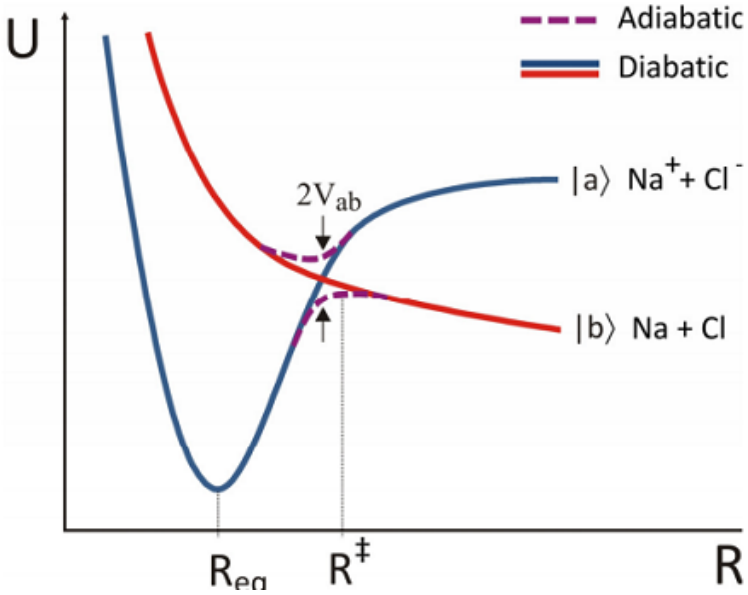
[1] คำจำกัดความ: คุณสมบัติที่เท่ากันทุกทิศทาง (Isotropic) และคุณสมบัติที่ขึ้นอยู่กับทิศทาง (Anisotropic)

8.8 การถ่ายโอนอิเล็กตรอน

การถ่ายโอนอิเล็กตรอน (Electron Transfer) เป็นกระบวนการที่อิเล็กตรอนเปลี่ยนตำแหน่งหรือเคลื่อนย้ายจากอะตอมหนึ่งไปยังอีกอะตอมหนึ่ง (Transfering) โดยเราสามารถแบ่งการถ่ายโอนอิเล็กตรอนออกได้เป็นสองกรณีคือการถ่ายโอนระหว่างโมเลกุล (Intermolecular Electron Transfer) และการถ่ายโอนภายในโมเลกุล (Intramolecular Electron Transfer) สำหรับการถ่ายโอนกรณีแรกนั้นมีสิ่งเร้าภายนอกเป็นปัจจัยหลัก ตัวทำละลายหรือสิ่งแวดล้อมภายนอกเป็นตัวกระตุ้นหรือตัวขับเคลื่อน (Driving Force) ที่ทำให้เกิดการถ่ายโอนจากโมเลกุลหนึ่งไปยังโมเลกุลหนึ่ง สำหรับการถ่ายโอนกรณีที่สองนั้นจริง ๆ แล้วมีปัจจัยหลายอย่างที่ทำให้เกิดกระบวนการนี้ เช่น ความเสถียรเชิงโครงสร้างของโมเลกุล (Stability) ซึ่งเกิดจากการรบกวนจากภายนอกที่ส่งผลให้โครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลเปลี่ยนไป

ในการพิจารณาการถ่ายโอนอิเล็กตรอนทั้งสองกรณีนั้นสามารถอธิบายได้ดังนี้ ให้ผู้อ่านลองจินตนาการโดยสมมติว่ามีกล่องอยู่สองกล่อง โดยกล่องซ้ายใส่ลูกบอลไว้ ส่วนกล่องขวานั้นว่างเปล่า หลังจากนั้นเราทำการหยิบลูกบอลจากกล่องซ้ายแล้วนำไปใส่ไว้ในกล่องขวา ซึ่งนี่คือการจำลองการถ่ายโอนอิเล็กตรอน จากเหตุการณ์ดังกล่าวเราแบ่งออกได้เป็นสองเหตุการณ์ย่อยคือ

1. เหตุการณ์ที่เกิดขึ้นก่อนที่จะเกิดการถ่ายโอนอิเล็กตรอน
2. เหตุการณ์ที่เกิดหลังจากถ่ายโอนอิเล็กตรอนแล้ว



ภาพ 8.6 แผนภาพแสดงพื้นผิวพลังงานศักย์ของกระบวนการถ่ายโอนอิเล็กตรอน (เครดิตภาพ: https://chem.libretexts.org)

8.8.1 ค่าคู่ควบของการถ่ายโอนอิเล็กตรอน

ค่าคู่ควบของการถ่ายโอนอิเล็กตรอน (Electron Transfer Coupling) เป็นค่าคู่ควบที่เกิดขึ้นจากการถ่ายโอนอิเล็กตรอน

8.8.2 พลังงานการปรับเปลี่ยนโครงสร้าง

พลังงานการปรับเปลี่ยนโครงสร้าง (Reorganization Energy) คือพลังงาน(ที่น้อยที่สุด)ที่ใช้ในการปรับเปลี่ยนโครงสร้างของโมเลกุลเพื่อทำให้เกิดการถ่ายโอนอิเล็กตรอนได้

8.9 คุณสมบัติของสถานะกระตุ้น

คุณสมบัติของอิเล็กตรอน ณ สถานะกระตุ้น (Excited State Properties)

8.9.1 พลังงานของสถานะกระตุ้น

พลังงานของสถานะกระตุ้น (Excited State Energies)

8.9.2 ค่าคู่ควบของกระบวนการนอนอะเดียแบติก

ค่าคู่ควบแบบนอนอะเดียแบติก (Nonadiabatic Coupling)

8.10 การคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุล

ในหัวข้อนี้เราจะมาดูการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลกันครับ ซึ่งสิ่งที่เราจะคำนวณนั้นก็คือคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลนั่นเอง โดยโปรแกรมเคมีเชิงคำนวณที่ผู้เขียนเลือกมาให้ผู้อ่านศึกษาเพื่อเป็นตัวอย่างนั้นก็คือโปรแกรม PySCF ซึ่งเป็นโปรแกรมที่ติดตั้งและใช้งานได้ง่าย มีฟังก์ชันที่หลากหลาย รองรับการคำนวณหลากหลายวิธี โดยผู้อ่านสามารถศึกษารายละเอียดเพิ่มเติมได้ในหัวข้อที่ 4

โค้ดต่อไปนี้เป็นการคำนวณคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุล HF ด้วยวิธี DFT โดยใช้ฟังก์ชันนอล PBE0 และเซตพื้นฐาน 6-31G(d)

```python
import pyscf

mol = pyscf.M(
    atom = 'H 0 0 0; F 0 0 1.1',  # in Angstrom
    basis = '631g(d)',
    symmetry = True,
)

mf = mol.KS()
mf.xc = 'pbe0'
mf.kernel()

# Orbital energies, Mulliken population etc.
mf.analyze()
```

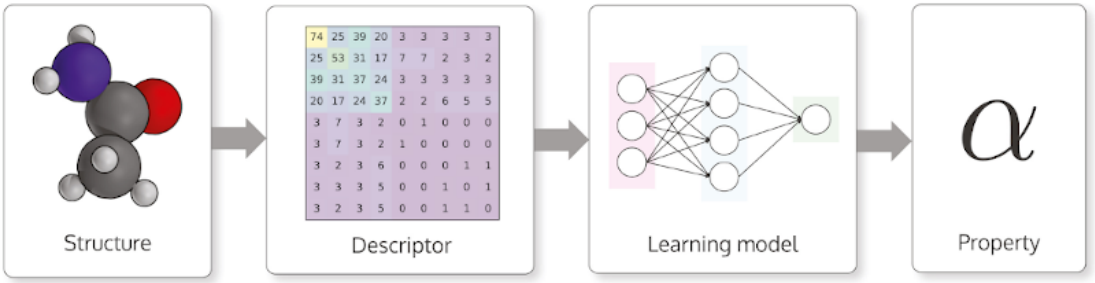
ซึ่งจะได้เอาต์พุตดังต่อไปนี้

```
converged SCF energy = -100.302481944224
Wave-function symmetry = Coov
occupancy for each irrep:     A1  E1x  E1y  E2x  E2y
                               3    1    1    0    0
**** MO energy ****
MO #1 (A1 #1), energy= -24.7448119170483 occ= 2
MO #2 (A1 #2), energy= -1.15590146781068 occ= 2
MO #3 (A1 #3), energy= -0.497762978336231 occ= 2
MO #4 (E1x #1), energy= -0.378844054318716 occ= 2
MO #5 (E1y #1), energy= -0.378844054318716 occ= 2
MO #6 (A1 #4), energy= 0.0180305141394873 occ= 0
MO #7 (A1 #5), energy= 0.718896484194941 occ= 0
MO #8 (E1x #2), energy= 1.21692188697545 occ= 0
MO #9 (E1y #2), energy= 1.21692188697545 occ= 0
MO #10 (A1 #6), energy= 1.31220491703922 occ= 0
MO #11 (A1 #7), energy= 1.62220484001697 occ= 0
MO #12 (E1x #3), energy= 1.84298258830569 occ= 0
MO #13 (E1y #3), energy= 1.84298258830569 occ= 0
MO #14 (E2x #1), energy= 1.89656974390515 occ= 0
MO #15 (E2y #1), energy= 1.8965699570922 occ= 0
MO #16 (A1 #8), energy= 2.33936741542906 occ= 0
    ** Mulliken atomic charges  **
charge of  0H =      0.37993
charge of  1F =     -0.37993
Dipole moment(X, Y, Z, Debye):  0.00000,  0.00000, -2.08373
```

โดยสรุปผลการคำนวณได้ดังนี้โมเลกุล HF มีพลังงานเชิงอิเล็กทรอนิกส์ ($E_{HF}+E_{Exchange}+E_{Correlation}$) เท่ากับ -100.302481944224 Hartree และมีพลังงานของออร์บิทัลเชิงโมเลกุล (MO) ตามที่แสดงทั้ง 16 ออร์บิทัล

บทที่ 9

ลักษณะเฉพาะของอะตอมและโมเลกุล



ภาพ 9.1 ขั้นตอนแสดงการสร้างโมเดลการเรียนรู้ของเครื่องเพื่อใช้ในการทำนายคุณสมบัติของโมเลกุล เริ่มจากการเปลี่ยนข้อมูลทางเคมีจากจากโครงสร้างของโมเลกุลไปเป็นข้อมูลแบบดิจิทัลที่คอมพิวเตอร์สามารถเข้าใจและประมวลผลต่อได้ ตามด้วยขั้นตอนการสร้างโมเดลสำหรับการเรียนรู้ และขั้นตอนสุดท้ายคือการทำนายคุณสมบัติของโมเลกุล (เครดิตภาพ: https://chemintelligence.com)

ในบทนี้เราจะมาดูความสำคัญของลักษณะเฉพาะ (Feature) ของโมเลกุลต่อประสิทธิภาพของโมเดล ML ในการทำนายคุณสมบัติของโมเลกุล[101] ซึ่งการคำนวณ Feature เป็นหนึ่งในขั้นตอนที่สำคัญมากของ ML ดังแสดงในภาพที่ 9.1 ซึ่งแสดงขั้นตอน (Workflow) ในการสร้างโมเดลเพื่อเชื่อมโยงความสัมพันธ์ระหว่างโครงสร้างของโมเลกุลกับคุณสมบัติทางเคมีของโมเลกุลนั้น ๆ จะเห็นได้ว่าทั้งสองอย่างนี้จริง ๆ แล้วมีความเชื่อมโยงกันผ่านโครงสร้างเชิงอิเล็กทรอนิกส์แต่ว่าความสัมพันธ์นั้นมีความซับซ้อนและการที่จะหาสมการทางคณิตศาสตร์ที่อธิบายความเชื่อมโยงนี้ได้นั้นเป็นเรื่องอาจจะทำได้ไม่ง่ายนัก ซึ่งปัญหาตรงนี้ ML ก็ได้เข้ามาช่วยในฐานะที่เป็นเครื่องมือหนึ่งที่พยายามสร้างสมการทางคณิตศาสตร์ในรูปแบบของฟังก์ชันที่ขึ้นอยู่กับตัวแปรที่อ้างอิงกับรูปแบบที่เกิดจากข้อมูลภายในชุดข้อมูลขนาดใหญ่โดยเชื่อมโยงผ่าน Feature นั่นเอง

Feature หรือ Representation คือคุณลักษณะที่บ่งบอกความเฉพาะตัวของอะตอมหรือโมเลกุลนั้น ๆ ซึ่งอาจจะเรียกว่าเป็นคุณลักษณะแบบพิเศษ (Special Attributes) ก็ได้ นอกจากนี้เราสามารถตีความได้ว่า Feature นั้นจริง ๆ แล้วก็เปรียบเสมือนเป็นตัวแทนของสิ่งที่เราสนใจอีกด้วย ซึ่งในบริบททางเคมีนั้นเราจะเรียกสิ่งที่เป็นตัวแทนของโมเลกุลว่า Molecular Representation อย่างไรก็ตามผู้เขียนมีความเห็นว่าจริง ๆ

แล้ว Feature กับ Representation ก็ไม่ได้มีความหมายเหมือนกันเสียทีเดียว ขึ้นอยู่กับประเภทของข้อมูลที่ใช้เป็น Feature แต่ผู้เขียนจะใช้คำว่า Representation ในการอ้างถึงลักษณะเฉพาะของระบบที่เรากำลังศึกษา (อะตอม, โมเลกุล, และสารประกอบ) เพราะว่าให้ความหมายที่สื่อการเป็นตัวแทนของระบบที่เราสนใจได้ดีกว่า[102]

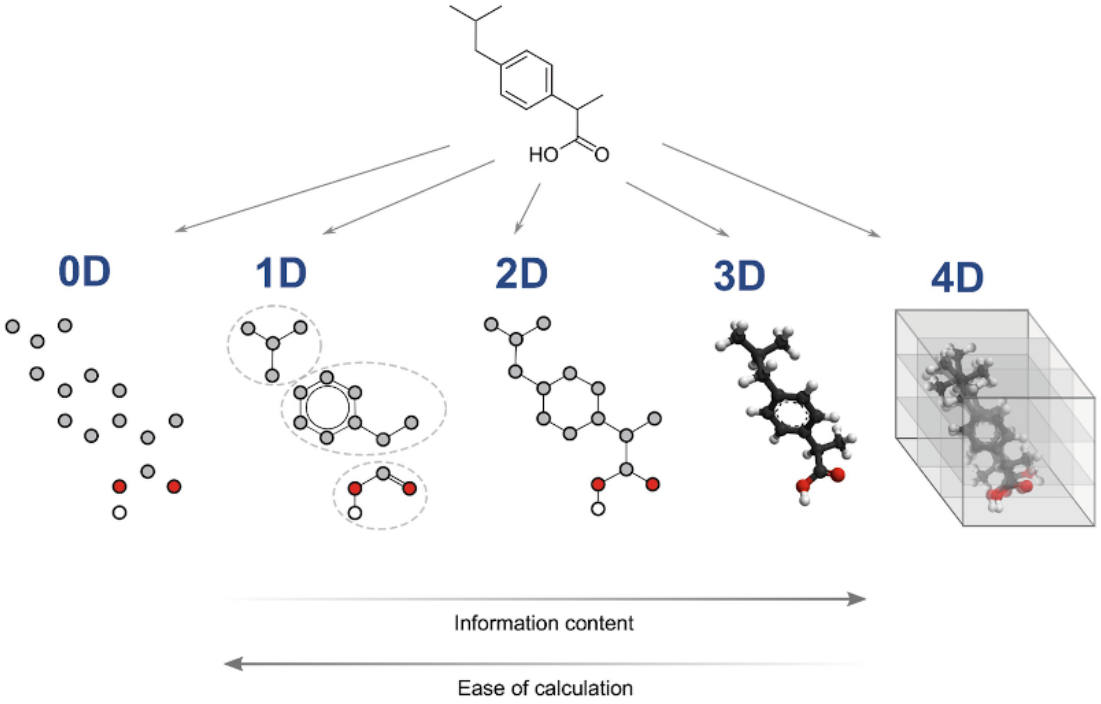
การแบ่งประเภทของ Feature หรือ Representation ในทางเคมีนั้นสามารถแบ่งได้หลายประเภท ขึ้นอยู่กับเกณฑ์ที่จะใช้ในการแบ่ง ความคิดเห็นส่วนตัวของผู้เขียนก็คือเราสามารถแบ่ง Feature ได้อย่างง่ายที่สุดเลยก็คือแบ่งตามสเกลของความจำเพาะเจาะจงของ Feature ที่คำนวณมาจากโมเลกุล กล่าวคือการที่เราต้องการที่จะทำนายคุณสมบัติของโมเลกุลนั้น เราควรจะต้องทราบก่อนว่าคุณสมบัติของโมเลกุลชนิดนั้นอยู่ในสเกลไหน โดยสเกลในที่นี้แบ่งง่าย ๆ เป็นระดับเฉพาะที่ (Local) หรือแบบทั่วทั้งพื้นที่ (Global) ตัวอย่างเช่น พลังงานรวมของโมเลกุลกับความสามารถในการละลายในน้ำเป็นคุณสมบัติในระดับ Global เพราะว่าแต่ละโมเลกุลมีค่าเหล่านี้ได้เพียงแค่หนึ่งค่าเท่านั้น แต่ถ้าหากเป็นคุณสมบัติอย่างเช่นแรงเชิงอะตอมหรือประจุย่อย คุณสมบัติเหล่านี้จะถูกจัดให้เป็นคุณสมบัติระดับ Local เพราะว่าอยู่ในสเกลระดับอะตอม เมื่อเราทราบแล้วว่าคุณสมบัติของเราเป็นแบบ Local หรือ Global เราก็สามารถที่จะพิจารณาเลือก Feature ที่อยู่ในระดับเดียวกันเพื่อมาใช้ในการฝึกสอนโมเดลได้ เพราะการที่เราใช้ Feature ที่อยู่ในระดับเดียวกับเอาต์พุตที่ต้องการทำนายนั้นจะทำให้การหาความสัมพันธ์ระหว่างสองสิ่งนี้ทำได้ง่ายและสมเหตุสมผล

9.1 ความสำคัญของลักษณะเฉพาะ

คำถามที่ตามมาก็คือ “*Molecular Representation มีความสำคัญมากไหม และมีความสำคัญอย่างไร*” แน่นอนว่าคำตอบคือต้องมีความสำคัญอยู่แล้ว (เพราะถ้าหากไม่สำคัญผู้เขียนก็คงไม่เขียนหัวข้อนี้ใช่ไหมครับ) และมีความสำคัญมากด้วย โดยส่วนตัวของผู้เขียนนั้นคิดว่า Molecular Representation มีความสำคัญมากที่สุดเลยก็ว่าได้ เพราะ Representation ก็คืออินพุตที่เรานำมาใช้ฝึกสอนโมเดล ML นั่นเอง ดังนั้น Molecular Representation จึงเป็นปัจจัยหลักที่กำหนดประสิทธิภาพในการทำนายของโมเดลด้วย

เนื่องจากว่ามนุษย์สามารถแยกแยะโมเลกุลแต่ละตัวออกจากกันได้ แต่ว่าคอมพิวเตอร์ไม่สามารถทำได้ เพราะว่าคอมพิวเตอร์เข้าใจข้อมูลที่เป็นแบบดิจิทัลในภาษาเครื่องจักรเท่านั้น (Machine Language) ดังนั้นเราจึงต้องมีการใช้ Representation เพื่ออธิบายโมเลกุลในรูปแบบของค่าพารามิเตอร์ที่คอมพิวเตอร์สามารถเข้าใจได้ เช่น แปลงโมเลกุลเป็นข้อมูลเชิงตัวเลข (Numeric) ให้อยู่ในรูปของเวกเตอร์หรือเมทริกซ์ สำหรับการอธิบายโมเลกุลแบบง่าย ๆ นั้นสามารถทำได้โดยใช้ Representation เพื่อมาอธิบายข้อมูลเชิงโครงสร้าง (Structural Properties) ซึ่งสามารถใช้ข้อมูลทางเคมีทั่วไปได้ ยกตัวอย่างเช่น รูปร่างของโมเลกุล, จำนวนหมู่ฟังก์ชัน, ชนิดของพันธะระหว่างอะตอมคาร์บอน, และจำนวนวงเบนซีน ฯลฯ ซึ่งข้อมูลเหล่านี้เราสามารถคำนวณออกมาได้ง่าย ๆ ไม่มีความซับซ้อนอะไร แต่ปัญหาคือ Representation ที่เป็น Structure-based นั้นมีข้อมูลที่น้อยเกินไป จึงทำให้ไม่สามารถถูกนำมาใช้เป็นอินพุตสำหรับการสร้างโมเดลเพื่อทำนายคุณสมบัติหรือพารามิเตอร์ทางเคมีที่ซับซ้อนหรือละเอียดกว่าได้ เช่น พลังงานพันธะ (Bond Energy), พลังงานของออร์บิทัล (Orbital Energy), ความถี่เชิงการสั่น (Vibrational Frequency), ไดโพลโมเมนต์ (Dipole Moment),

ฯลฯ นั่นก็เพราะว่าอินพุตของเราเป็น Representation ที่ไม่มีความสัมพันธ์กับเอาต์พุตที่เราต้องการทำนาย (จริง ๆ ก็อาจจะสอดคล้องแต่ว่าไม่ได้สอดคล้องกันแบบโดยตรง)

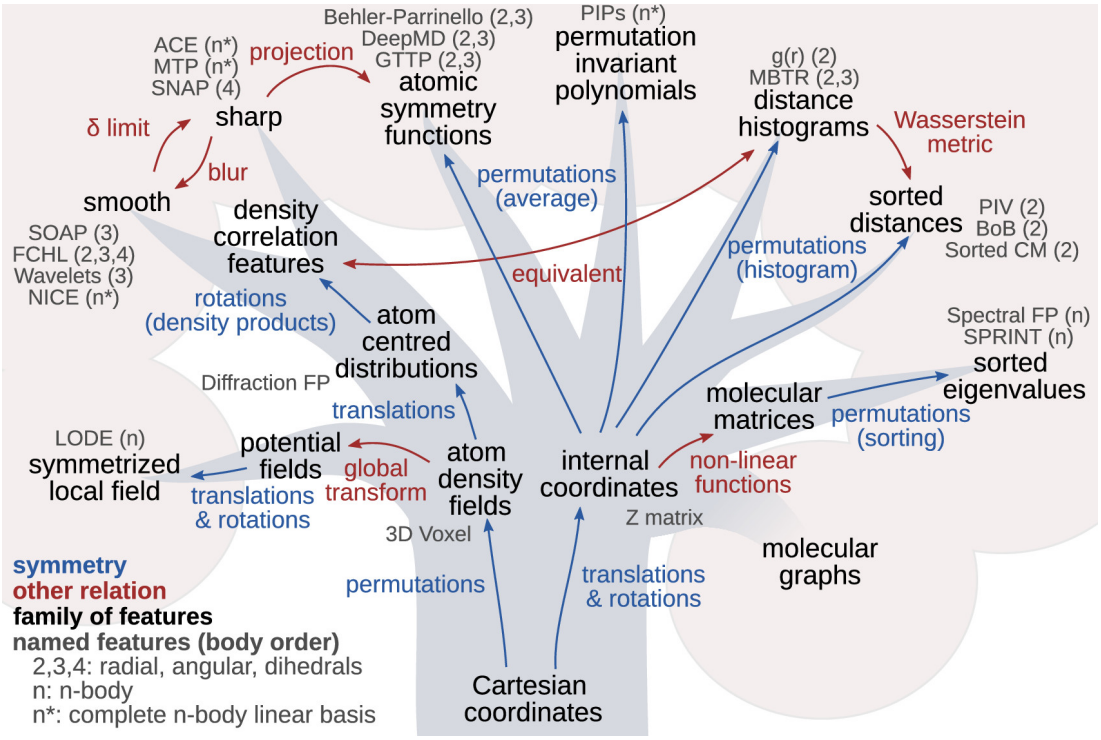


ภาพ 9.2 แผนภาพแสดงการแบ่งประเภทของ Descriptor หรือตัวคำนวณคุณลักษณะเฉพาะของโมเลกุลโดยแบ่งตามจำนวนมิติของโมเลกุล (เครดิตภาพ: https://chemintelligence.com)

ดังนั้นถ้าหากเราต้องการที่จะทำนายเอาต์พุตที่เป็นปริมาณที่มีความละเอียดอยู่ในระดับอะตอม (ปริมาณเชิงอิเล็กทรอนิกส์) เราควรจะใช้ Representation ที่อยู่ในระดับเดียวกันและ Representation ควรจะต้องให้ Feature Vectors ที่เก็บข้อมูลทางเคมีควอนตัมและฟิสิกส์ไว้ด้วย โดยการพัฒนา Representation โดยใช้องค์ความรู้ทางฟิสิกส์ (Physics-inspired Representation) ก็เป็นหนึ่งในหัวข้องานวิจัยที่กำลังมาแรงในขณะนี้ ข้อมูลทางฟิสิกส์ที่เราเพิ่มเข้าไปก็เปรียบเสมือนเป็นส่วนเติมเต็มที่เพิ่มความถูกต้อง (เรียกอีกอย่างว่าการทำ Correction) ให้กับ Representation มากขึ้น โดยเราสามารถใส่ความเป็นสมมาตร (Symmetricity) หรือคุณสมบัติจากปรากฏการณ์ทางฟิสิกส์เชิงควอนตัมของโมเลกุลเข้าไปได้ เป็นต้น

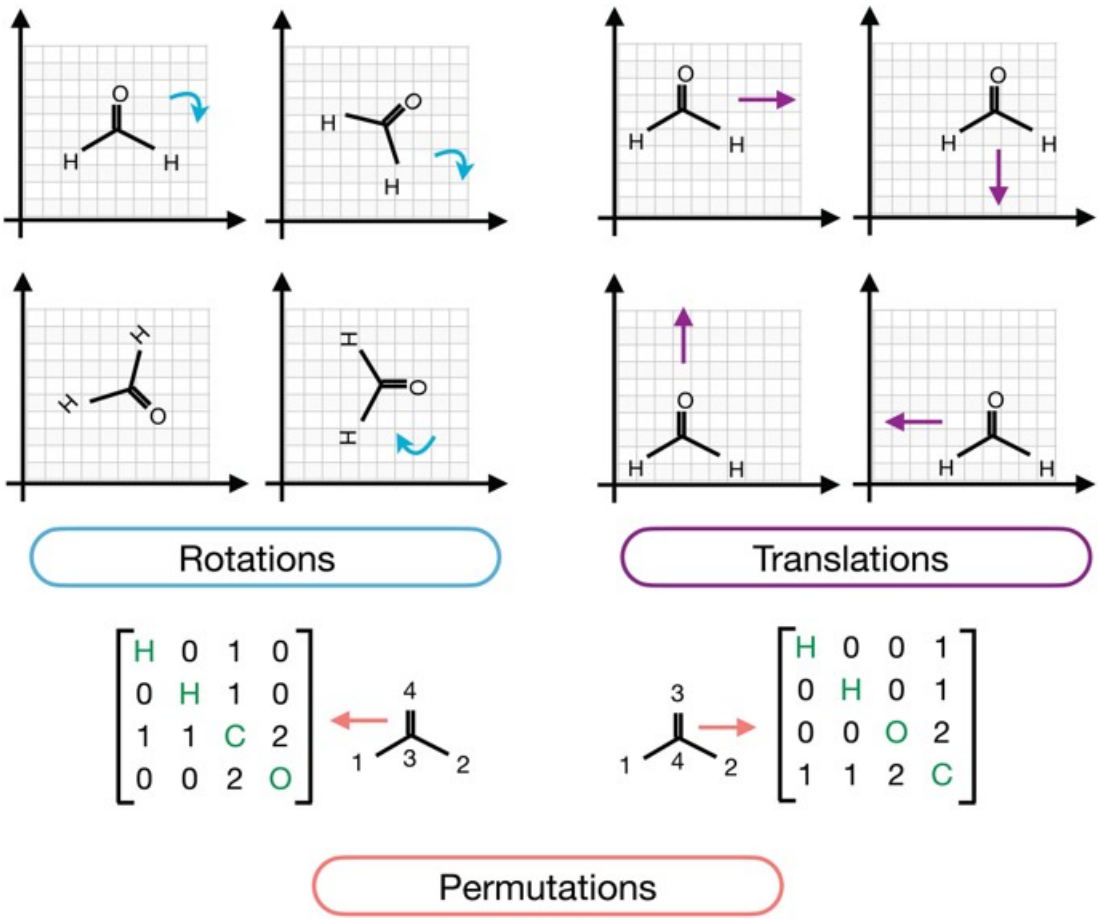
9.2 การแปลงข้อมูลเชิงโมเลกุล

โมเลกุลประกอบไปด้วยอะตอมหลายอะตอมมารวมกัน เราจึงเปรียบเทียบโมเลกุลเป็นประโยคหรือข้อความและเปนียบเทียบอะตอมเป็นคำแต่ละคำได้ ดังที่บอกไปในข้างต้นว่าการทำให้คอมพิวเตอร์เข้าใจความเชื่อมโยงระหว่างอะตอมในโมเลกุลนั้นต้องมีการเลือกใช้ Representation ที่เหมาะสม โดยคุณสมบัติของ Representation ที่ดีนั้นไม่เพียงแต่จะต้องไม่ขึ้นกับการเคลื่อนที่เชิงการหมุน (Rotational Motion) และการเคลื่อนที่เชิงเส้น (Translational Motion) เท่านั้น แต่ควรจะต้องมีความเรียบง่ายและไม่ซับซ้อนหรือยุ่งยาก



ภาพ 9.3 แผนผังแสดงความเชื่อมโยงของ Descriptor ทั้งเชิงโครงสร้างและอิเล็กทรอนิกส์โดยเริ่มจาก Cartesian Coordinates ของโมเลกุลแล้วพัฒนาต่อเป็น Descriptor แบบต่าง ๆ (เครดิตภาพ: *Chem. Rev.* **2021**, 121, 16, 9759-9815[103])

เกินไปในการคำนวณเพื่อสร้าง Machine Code จากพิกัดตำแหน่งคาร์ทีเซียน (Cartesian Coordinates) ด้วย



ภาพ 9.4 Invariance ของการเคลื่อนที่เชิงเส้น (Translation), การเคลื่อนที่เชิงการหมุน (Rotation) และการเปลี่ยนลำดับ (Permutation) ของโมเลกุล Formaldehyde

ภาพที่ 9.4 แสดงตัวอย่างของการเคลื่อนที่แบบต่าง ๆ รวมถึงการเปลี่ยนลำดับซึ่งคุณสมบัติเหล่านี้จะสอดคล้องกับสมมาตรของโมเลกุลด้วย โดยเงื่อนไขในการพิจารณาว่า Molecular Representation ไหนมีความเหมาะสมในการนำมาคำนวณ Feature Vectors มีดังต่อไปนี้

1. การไม่ขึ้นกับการหมุน (Rotational Invariance): Representation จะต้องไม่ขึ้นกับโอเปอเรเตอร์ของการหมุน

2. การไม่ขึ้นกับการเลื่อนตำแหน่ง (Translational Invariance): Representation จะต้องไม่เปลี่ยนไปเมื่อมีการเลื่อนตำแหน่งแบบเชิงเส้นภายในปริภูมิ

3. การไม่ขึ้นกับการเปลี่ยนลำดับ (Permutation Invariance): Representation จะต้องไม่เปลี่ยนไปเมื่อมีการเปลี่ยนลำดับหรือสลับอะตอม

9.3 ลักษณะเฉพาะเชิงโครงสร้างแบบทั่วไป

9.3.1 Internal Coordinates

พิกัดภายในของโมเลกุล (Internal Coordinates) หรือเรียกอย่างว่า Z Matrix เป็น Representation พื้นฐานที่สุดในการอธิบายโครงสร้างของโมเลกุล (ไม่นับการใช้พิกัดของอะตอมแต่ละอะตอมโดยตรง) อาจจะเรียกได้ว่าเป็น Representation ที่เรียบง่ายที่สุดเลยก็ว่าได้ โดยถูกใช้อย่างแพร่หลายในยุคแรก ๆ ที่มีการนำ ML มาใช้สำหรับเคมีและยังถูกใช้มาอย่างยาวนานจึงถึงปัจจุบัน ข้อดีอย่างหนึ่งของ Internal Coordinates คือสามารถอธิบายได้ทั้งโมเลกุล โดยองค์ประกอบของ Representation อันนี้มีความยาวพันธะระหว่างอะตอม (Bond Distance) หรือ d, มุมพันธะ (Bond Angle) หรือ α, และมุมบิดเบี้ยว (Dihedral Angle) หรือ θ ซึ่งอะตอมที่ถูกเลือกมาคำนวณ Internal Coordinates นั้นมักจะเป็นอะตอมที่เรียงติดกัน (มีพันธะเคมีระหว่างกัน) หรืออยู่ใกล้กัน อย่างไรก็ตามเราสามารถคำนวณหา Internal Coordinates ของโมเลกุลได้โดยพิจารณาอะตอมทุก ๆ คู่หรือทุกความเป็นไปได้ทั้งหมดภายในโมเลกุล โดยเซตของ Internal Coordinates สามารถเขียนได้ดังนี้

$$Z = \{d, \alpha, \theta\} \tag{9.1}$$

9.3.2 Geometric Descriptors

Geometric Descriptors (ลักษณะเฉพาะเชิงเรขาคณิต) เป็น Descriptor (จะเรียกแทนด้วย Representation ก็ได้) ที่อ้างอิงกับข้อมูลตำแหน่งของอะตอมในโมเลกุล โดยมักจะเชื่อมโยงกับ Representation แบบที่เป็นสัญลักษณ์ (Symbolic Representation) เช่น SMILES ซึ่ง Geometric Descriptors ก็สามารถแบ่งออกเป็นได้หลาย Descriptor ซึ่งก็รวมไปถึง Z Matrix, Coordination Number, Adjacency Matrix อย่างไรก็ตาม Representation ในกลุ่มนี้มักจะให้ผลการทำนายด้วย ML ไม่ค่อยดีนัก นั่นก็เพราะว่าความสามารถในการกักเก็บข้อมูลเชิงอิเล็กทรอนิกส์นั้นน้อยมากเมื่อเทียบกับ Representation ประเภทที่เป็นแบบเชิงอะตอม (Atom-wise Descriptor) และยังมีโมเลกุลบางประเภทที่ Geometric Descriptors ไม่สามารถนำไปใช้ได้ อย่างเช่นโมเลกุลไอโซเมอร์ เช่น cis/trans สเตอรีโอไอโซเมอร์ ดังนั้น Representation ประเภทนี้จึงไม่เป็นที่นิยมในการนำมาฝึกสอนโมเดล ML โดยเฉพาะโมเดลที่ใช้ในงานวิจัยทางด้านเคมีควอนตัม[104,103]

9.4 ลักษณะเฉพาะเชิงโครงสร้างสำหรับโมเลกุล

Representation ประเภทนี้จะเป็นการอธิบายสภาพแวดล้อม (Environment) ของอันตรกิริยาระหว่างอะตอมทุกอะตอมในโมเลกุล โดยมักจะอยู่ในรูปของเมทริกซ์ เช่น เมทริกซ์ของส่วนกลับของระยะห่างระหว่าง

อะตอม (Inverse Distance Matrix) และเมทริกซ์คูลอมบ์ (Coulomb Matrix)

9.4.1 Inverse Distance Matrix

Inverse Distance Matrix (เมทริกซ์ของส่วนกลับของระยะห่างระหว่างอะตอม) เป็น Representation Matrix แบบที่ง่ายที่สุดและมีความหมายทางเคมีที่ชัดเจน นั่นก็คือการใช้ส่วนกลับของระยะห่างระหว่างนิวเคลียสของอะตอมนั้นเป็นการจำลองเทอมของอันตรกิริยาระหว่างนิวเคลียสที่อยู่ใน Hamiltonian ของพลังงาน นิยามทางคณิตศาสตร์ของ Inverse Distance Matrix (D) อธิบายได้ตามสมการต่อไปนี้

$$D_{ij} = \frac{1}{||r_i - r_j||} \tag{9.2}$$

เมื่อเราคำนวณ Inverse Distance ออกมาเป็นเมทริกซ์ขนาด $i \times j$ แล้ว เราจะพบว่าสมาชิกของเมทริกซ์ในแนวทแยง (Diagonal Elements) นั้นจะไม่มีความหมาย ดังนั้นเราจึงสนใจเฉพาะสมาชิกนอกแนวทแยง (Off-diagonal Elements)

9.4.2 Coulomb Matrix

Coulomb Matrix (เมทริกซ์คูลอมบ์) เป็น Molecular Representation ที่ถูกเสนอครั้งแรกในปี ค.ศ. 2012 โดย Matthias Rupp และทีมวิจัย[105] โดยได้ถูกนำมาใช้อย่างแพร่หลายในงานวิจัยทางด้าน ML นั่นก็เพราะว่าไม่มีความสิ้นเปลืองในการคำนวณและให้ความแม่นยำในการทำนายค่าพลังงานของโมเลกุลสูง ซึ่ง Coulomb Matrix นั้นถูกพัฒนาขึ้นมาโดยมีพื้นฐานมาจาก Inverse Distance Matrix ซึ่งเป็นการแก้ปัญหาที่พบใน Distance Matrix สองส่วนดังนี้

1. มีการกำหนดเงื่อนไขในการคำนวณสมาชิกโดยกำหนดค่าของสมาชิกในแนวทแยง
2. รวมประจุของอะตอมเข้าไปด้วย ซึ่งเป็นพารามิเตอร์สำคัญในการพัฒนา Force Field สำหรับการจำลองพลวัตเชิงโมเลกุล (Molecular Dynamics หรือ MD)

สมการสำหรับการคำนวณสมาชิกของ Coulomb Matrix คือ

$$C_{ij} = \begin{cases} \frac{1}{2} Z_i^{2.4} & \text{if } i = j \\ \frac{Z_i Z_j}{R_{ij}} & \text{if } i \neq j \end{cases} \tag{9.3}$$

จากสมการที่ (9.3) จะเห็นได้ว่าเราได้แบ่งเงื่อนไขในการคำนวณสมาชิกของ Coulomb Matrix ออกเป็นสองเงื่อนไข สำหรับกรณีคู่อะตอมเหมือนกันและต่างกัน โดยที่พลังงานศักย์เชิงไฟฟ้าสถิตย์ของอะตอม

แต่ละคู่นั้นจะถูกเข้ามาด้วย (ผ่านเทอมของประจุ) นั่นก็คือสมาชิกนอกแนวทแยงจะแสดงถึงแรงผลักคูลอมบ์ระหว่างอะตอม ในขณะที่สมาชิกในแนวทแยงจะเป็นการเทียบเคียงค่าเลขอะตอมกับพลังงานของอะตอมในกรณีที่ไม่ได้มีอันตรกิริยากับอะตอมตัวอื่น

ถึงแม้ว่า Coulomb Matrix จะเป็น Representation ที่ไม่ขึ้นกับการเลื่อนตำแหน่งและการหมุนของโมเลกุล แต่ว่าก็ยังขึ้นอยู่กับการเปลี่ยนตำแหน่งหรือการสลับที่กันของอะตอม เพื่อแก้ปัญหาดังกล่าว ได้มีนักวิจัยได้นำเสนอ Representation ตัวใหม่อีกหลายตัวที่เปรียบเสมือนเป็น Coulomb Matrix ที่ถูกปรับปรุงให้ดีขึ้น เช่น Sine Matrix,[106] Ewald Sum Matrix,[106] Permutation-Invariant Polynomials (PIP),[107] Randomly Sorted Coulomb Matrices (RSCM),[108] Bag of Bonds (BoB),[108] Permutation Invariant Vectors (PIV)[109] โดย Representation เหล่านี้ได้แก้ปัญหาเกี่ยวลำดับของอะตอม ทำให้ Representation ประเภทนี้มีประสิทธิภาพมากขึ้นและลด Bias ที่อาจจะเกิดขึ้นด้วย ซึ่งจะผู้อ่านจะได้ศึกษาต่อไป

ตัวอย่างโค้ดของการคำนวณ Coulomb Matrix โดยใช้ไลบรารี่ molml[110]

```
>>> from molml.features import CoulombMatrix
>>> feat = CoulombMatrix()
>>> H2 = (
...         ['H', 'H'],
...         [
...             [0.0, 0.0, 0.0],
...             [1.0, 0.0, 0.0],
...         ]
... )
>>> HCN = (
...         ['H', 'C', 'N'],
...         [
...             [-1.0, 0.0, 0.0],
...             [ 0.0, 0.0, 0.0],
...             [ 1.0, 0.0, 0.0],
...         ]
... )
>>> feat.fit([H2, HCN])
CoulombMatrix(input_type='list', n_jobs=1, sort=False,
   eigen=False, drop_values=False, only_lower_triangle=False)
>>> feat.transform([H2])
array([[ 0.5,  1. ,  0. ,  1. ,  0.5,  0. ,  0. ,  0. ,  0. ]])
>>> feat.transform([H2, HCN])
array([[0.5, 1. , 0. , 1. , 0.5,
        0. , 0. , 0. , 0. ],
       [0.5, 6. , 3.5, 6. , 36.8581052,
        42., 3.5, 42., 53.3587074]])
```

9.4.3 Sine Matrix

Sine Matrix เป็น Representation ที่คำนวณอันตรกิริยาระหว่างอะตอมของระบบที่เป็นแบบ Perodic System และรวมผลของ Coulomb เข้าไปด้วย[106] มีสมการดังนี้

$$M_{ij}^{\text{sine}} = \begin{cases} \frac{1}{2} Z_i^{2.4} & \text{for } i = j \\ \dfrac{Z_i Z_j}{\left| \boldsymbol{B} \cdot \sum_{k=\{x,y,z\}} \hat{\boldsymbol{e}}_k \sin^2\left(\pi \boldsymbol{B}^{-1} \cdot (\boldsymbol{R}_i - \boldsymbol{R}_j)\right) \right|} & \text{for } i \neq j \end{cases} \tag{9.4}$$

โดยที่ $\boldsymbol{B}$ คือพารามิเตอร์ที่ถูกคำนวณมาจาก Lattice Vectors และ $\hat{\boldsymbol{e}}_k$ คือ Cartesian Unit Vectors

ตัวอย่างของโค้ดสำหรับการคำนวณ Sine Matrix Feature

```python
from dscribe.descriptors import SineMatrix

sm = SineMatrix(
    n_atoms_max=6,
    permutation="sorted_l2",
    sparse=False,
    flatten=True
)
```

9.4.4 Ewald Sum Matrix

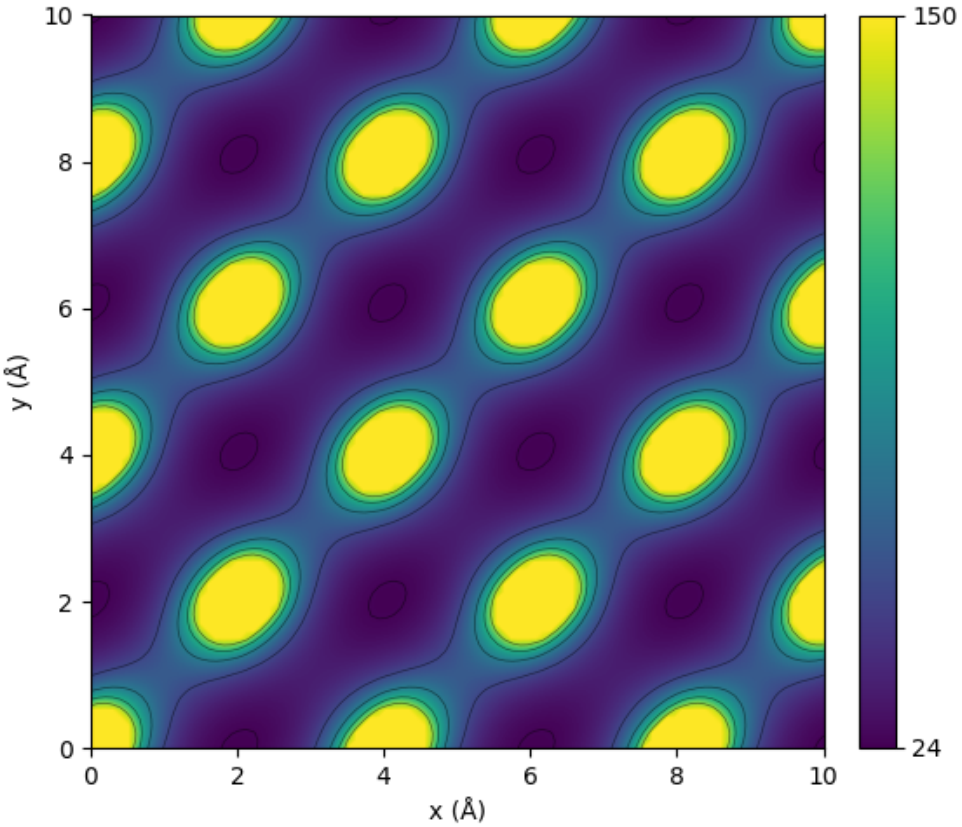
Ewald Sum Matrix เป็น Representation ที่พัฒนาต่อมาจาก Coulomb Matrix ซึ่งได้รวมผลของอันตรกิริยาระหว่างอะตอมแบบ Periodic โดยใช้อันตรกิริยาไฟฟ้าสถิตย์เข้าไปด้วย[106]

$$\phi_{ij}^{\text{self}} + \phi_{ij}^{\text{bg}} = -\frac{\alpha}{\sqrt{\pi}}(Z_i^2 + Z_j^2) - \frac{\pi}{2V\alpha^2}(Z_i + Z_j)^2 \; \forall \; i \neq j \tag{9.5}$$

โดยที่ ϕ_{ij}^{bg} คือ Background Charge

ตัวอย่างของโค้ดสำหรับการคำนวณ Ewald Sum Matrix

```python
from dscribe.descriptors import EwaldSumMatrix

atomic_numbers = [1, 8]
rcut = 6.0
```



ภาพ 9.5 อันตรกิริยาแบบ Periodic ที่คำนวณด้วย Sine Matrix

```
nmax = 8
lmax = 6

# Setting up the Ewald sum matrix descriptor
esm = EwaldSumMatrix(
    n_atoms_max=6,
)
```

9.4.5 Bag of Bonds

นอกจากนี้ยังมี Bag of Bonds (BoB)[111] ซึ่งเป็น Representation ที่พัฒนาต่อจาก Coulomb Matrix โดยจะมีการจัดกลุ่มประเภทของพันธะเข้าด้วยกัน ซึ่งได้แรงบันดาลใจในการพัฒนามาจาก Bag of Words ที่ใช้ใน Natural Language Processing (NLP) โดยคำว่า *Bag* ในที่นี้คือประเภทของพันธะเคมีที่แตกต่างกันในโมเลกุลนั้น ๆ เช่น C—C, C—O, และ C—H ส่วน *Bond* นั้นจะแยกตามชนิดของพันธะ คือพันธะเดี่ยว พันธะคู่ และพันธะสาม

ตัวอย่างโค้ดของการคำนวณ Bag of Bonds ของโมเลกุล Methane[1] โดยใช้ไลบรารี่ qml[112] โดยสามารถดูรายละเอียดเพิ่มเติมได้ที่[2]

```
>>> import qml
>>> CH4 = qml.Compound(xyz="methane.xyz")
>>> CH4.generate_bob(asize={"C":4, "H":8})
>>> print(CH4.representation)
[36.8581052   0.          0.          0.          0.          0.
  0.          0.
  0.          0.          5.50964209  5.50964187  5.50963981
  5.50963981  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.
  0.          0.          0.5         0.5         0.5
  0.5         0.          0.
  0.          0.          0.56232548  0.56232539  0.56232539
  0.56232533  0.56232532  0.56232532
  0.          0.          0.          0.          0.          0.
  0.          0.
```

[1] พิกัดคาร์ทีเซียนดูได้ที่ https://en.wikipedia.org/wiki/Z-matrix_(chemistry)
[2] https://www.qmlcode.org

```
13  0.          0.          0.          0.          0.          0.
          0.          0.
14  0.          0.          0.          0.          0.          0.
          ]
```

9.5 ลักษณะเฉพาะเชิงอิเล็กทรอนิกส์สำหรับอะตอม

นอกเหนือจาก Representation ที่เรานำมาอธิบายโมเลกุลแบบทั้งโมเลกุล ยังมี Representation แบบอื่นซึ่งมีความซับซ้อนมากกว่าที่ถูกพัฒนาขึ้นมาเพื่ออธิบาย Environment ของโมเลกุลในระดับอะตอมแบบเฉพาะเจาะจง ซึ่ง Representation แบบนี้จะเป็นการรวมความสำคัญของกฎทางฟิสิกส์และเคมีแบบต่าง ๆ เข้ามาไว้ด้วยกัน จึงทำให้ Representation ประเภทนี้มีความสามารถในการที่จะอธิบายโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลได้ดีมาก ๆ โดยเฉพาะการทำนายคุณสมบัติในระดับอะตอม โดยในปัจจุบันก็ได้มีการพัฒนา Representation ที่สามารถอธิบายอะตอมแบบทีละอะตอม ซึ่งเราเรียก Representation ประเภทนี้ว่า Atom-wise ดังนั้นผู้เขียนจะขออธิบายเฉพาะ Representation ที่มีความโดดเด่นน่าสนใจและเป็นที่นิยมในการนำมาใช้ในงานวิจัยเท่านั้น

การพัฒนา Representation สำหรับอธิบายโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุลนั้นสามารถใช้องค์ความรู้ที่เกี่ยวข้องกับความหนาแน่น (Density) ของโมเลกุล,[113] เทคนิค Linear Scaling ในเชิงคำนวณ,[114,115] หลักการมองเห็นระยะสั้น (Nearsightedness Principle)[116] รวมไปถึงการพิจารณาสมมาตร (Symmetry) ของโมเลกุล เพื่อมาใช้ในการออกแบบและปรับปรุงประสิทธิภาพของ Representation เพื่อให้ครอบคลุมและอธิบายปรากฏการณ์ทางควอนตัมให้ได้มากที่สุด[117]

9.5.1 Smooth Overlap of Atomic Positions

Representation ที่เป็นประเภทเชิงอิเล็กทรอนิกส์สำหรับอะตอมอันแรกที่จะพูดถึงก็คือ Smooth Overlap of Atomic Positions (SOAP) ซึ่งเป็นตัวที่โด่งดังมากแล้วก็มีการนำมาใช้อย่างแพร่หลาย เรียกได้ว่านักวิจัยเคมีควอนตัมและปัญญาประดิษฐ์ต่างก็รู้จัก SOAP กันเป็นอย่างดี โดยบทความงานวิจัยของ SOAP ได้ถูกตีพิมพ์ครั้งแรกในปี ค.ศ. 2013 ซึ่งไอเดียของ SOAP ก็คือจะเป็นการนำข้อมูลโครงสร้างของอะตอมมาเข้ารหัส (Encoding) ไว้ได้โดยได้รวมสภาพแวดล้อมทางเคมีโดยการใช้ความหนาแน่นเชิงอะตอมแบบเกาส์เซียน (Gaussian Smeared Atomic Density) ซึ่ง Atomic Density ที่ว่านี้ก็ถูกคำนวณมาจากฟังก์ชันพื้นฐานเชิงรัศมี (Radial Basis Function) หรือ $g_n(r)$ และฟังก์ชันฮาร์โมนิคเชิงทรงกลม (Real Spherical Harmonic Functions) หรือ $Y_{lm}(\theta, \phi)$[118,119] โดย SOAP นั้นเหมาะสำหรับนำมาใช้ทำนายคุณสมบัติของโมเลกุลแบบเฉพาะที่ (Local Properties) อย่างเช่นแรงเชิงอะตอม (Atomic Force) หรือ Chemical Shift

การคำนวณ SOAP นั้นจริง ๆ แล้วสามารถทำได้ผ่านการคำนวณ Kernel ของ Atomic Environment 2 อันเข้าด้วยกัน ($\mathcal{X}$ และ $\mathcal{X}'$) ซึ่งเขียนออกมาได้ในรูปของ Polynomial Kernel (K^{SOAP} ของพารามิเตอร์

ที่ชื่อว่า Partial Power Spectra ($\boldsymbol{p}$ และ $\boldsymbol{p}'$)[1]

โดยสมการที่เรานำมาใช้ในการคำนวณ SOAP นั้นคือ

$$K^{\text{SOAP}}(\boldsymbol{p}, \boldsymbol{p'}) = \left(\frac{\boldsymbol{p} \cdot \boldsymbol{p'}}{\sqrt{\boldsymbol{p} \cdot \boldsymbol{p} \quad \boldsymbol{p'} \cdot \boldsymbol{p'}}} \right)^{\xi} \tag{9.6}$$

โดยที่กำหนดให้ ξ เป็นจำนวนเต็มบวกและสมาชิกของเวกเตอร์ $\boldsymbol{p}$ มีนิยามคือ

$$p_{nn'l}^{Z_1 Z_2} = \pi \sqrt{\frac{8}{2l+1}} \sum_m {c_{nlm}^{Z_1}}^{\dagger} c_{n'lm}^{Z_2} \tag{9.7}$$

โดยที่ n และ n' เป็นดัชนีสำหรับ Radial Basis Function ซึ่งมีค่าได้สูงสุดถึง n_{max}, l เป็นดีกรีเชิงมุม (Angular Degree) ของ Spherical Harmonics ซึ่งมีค่าได้สูงสุดถึง l_{max}, m เป็นจำนวนเต็มที่สอดคล้องกับเงื่อนไขคือ $|m| \leq l$, และ Z_1 และ Z_2 เป็นสปีชีส์เชิงอะตอม (Atomic Species) นอกจากนี้ค่าสัมประสิทธิ์ $c_{n'lm}^{Z}$ และ ${c_{nlm}^{Z}}^{\dagger}$ ถูกนิยามให้เป็นผลคูณภายในของ Spherical Harmonic Functions กับความหนาแน่นเชิงอะตอมแบบเกาส์เซียน (ρ^Z) และคอนจูเกตเชิงซ้อน (Complex Conjugate) ตามลำดับ[119] ดังนี้

$$c_{nlm}^{Z}(\boldsymbol{r}) = \iiint_{\mathcal{R}^3} dV \, g_n(r) Y_{lm}(\theta, \phi) \rho^Z(\boldsymbol{r}) \tag{9.8}$$

โดยที่ $\boldsymbol{r}$ คือตำแหน่งในปริภูมิและ $\rho^Z(\boldsymbol{r})$ คือความหนาแน่นเชิงอะตอมแบบเกาส์เซียน (Gaussian Smoothed Atomic Density) สำหรับอะตอม Z

ในการคำนวณ $c_{nlm}^{Z}(\boldsymbol{r})$ นั้นเราจะต้องนั้นเราจะต้องมีการกำหนด Radial Basis Function (RBF) ที่เราต้องการใช้ด้วย โดยเราสามารถใช้ฟังก์ชัน Spherical Gaussian-type Orbitals (สมการที่ (9.9)) หรือฟังก์ชัน Polynomial Basis (ตามสมการที่ (9.10)) ก็ได้

$$g_{nl}(r) = \sum_{n'=1}^{n_{\text{max}}} \beta_{nn'l} r^l e^{-\alpha_{n'l} r^2} \tag{9.9}$$

$$g_n(r) = \sum_{n'=1}^{n_{\text{max}}} \beta_{nn'} (r - r_{\text{cut}})^{n'+2} \tag{9.10}$$

และในส่วนของ Spherical Harmonics นั้นเราสามารถใช้เฉพาะส่วนจริง (Real Part) ได้ (เพราะว่าง่ายต่อการ Implement) โดยรูปแบบหรือสมการของ Spherical Harmonics มีดังนี้

[1] ผู้เขียนไม่แน่ใจว่าจะแปลเป็นภาษาไทยอย่างไรดี ต้องขออภัยผู้อ่านด้วยครับ

$$Y_{\ell m} = \begin{cases} \frac{i}{\sqrt{2}}\left(Y_\ell^m - (-1)^m\, Y_\ell^{-m}\right) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \frac{1}{\sqrt{2}}\left(Y_\ell^{-m} + (-1)^m\, Y_\ell^m\right) & \text{if } m > 0 \end{cases} \tag{9.11}$$

$$= \begin{cases} \frac{i}{\sqrt{2}}\left(Y_\ell^{-|m|} - (-1)^m\, Y_\ell^{|m|}\right) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \frac{1}{\sqrt{2}}\left(Y_\ell^{-|m|} + (-1)^m\, Y_\ell^{|m|}\right) & \text{if } m > 0 \end{cases} \tag{9.12}$$

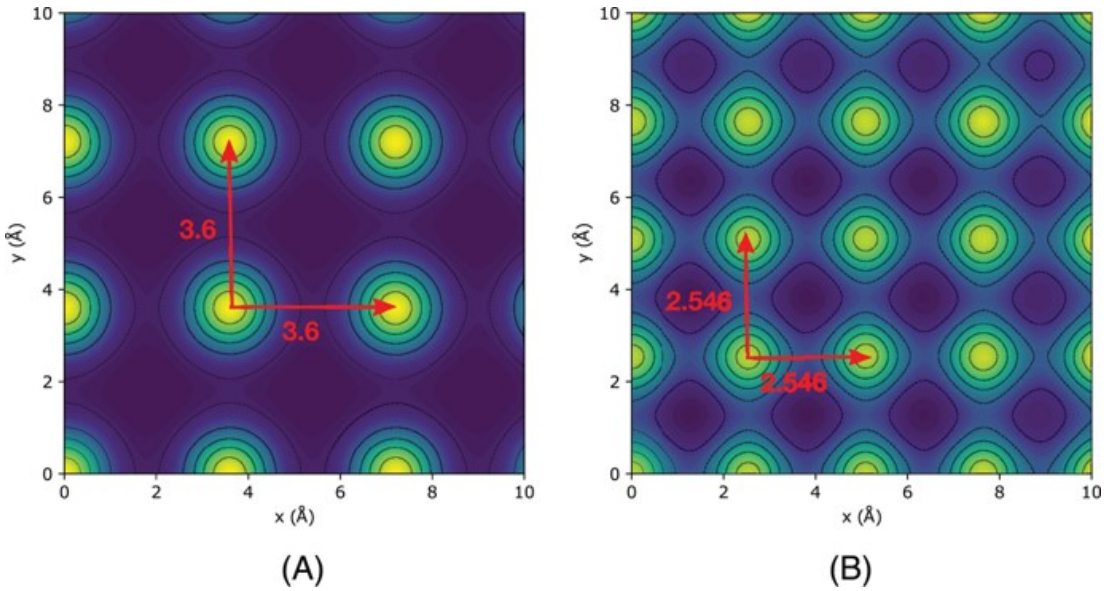
$$= \begin{cases} \sqrt{2}\,(-1)^m\ \mathrm{Im}[Y_\ell^{|m|}] & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \sqrt{2}\,(-1)^m\ \mathrm{Re}[Y_\ell^m] & \text{if } m > 0 \end{cases} \tag{9.13}$$

เนื่องจากว่ารายละเอียดในการพิสูจน์สมการ Kernel ของ SOAP นั้นมีความซับซ้อนพอสมควร ผู้เขียนจึงขอแนะนำให้ผู้อ่านที่สนใจศึกษาเพิ่มเติมทฤษฎีของ SOAP ได้ที่บทความต้นฉบับ[118]

ตัวอย่างโค้ดสำหรับการตั้งค่า SOAP Representation และคำนวณ SOAP ของโมเลกุลน้ำโดยใช้ไลบรารี่ DScribe และ ASE

```python
from dscribe.descriptors import SOAP
from ase.build import molecule

species = ["H", "C", "O", "N"]
r_cut = 6.0
n_max = 8
l_max = 6

# Setting up the SOAP descriptor
soap = SOAP(
    species=species,
    periodic=False,
    r_cut=r_cut,
    n_max=n_max,
    l_max=l_max,
)

# Molecule created as an ASE.Atoms
water = molecule("H2O")

```

```
# Create SOAP output for the system
soap_water = soap.create(water, positions=[0])

print(soap_water)
print(soap_water.shape)
```



ภาพ 9.6 ตัวอย่างของ SOAP Matrix ของระบบ Cu/fcc Lattices สำหรับ (A) Cubic และ (B) Orthorhombic

รายละเอียดเพิ่มเติมสำหรับการคำนวณและการเขียนโปรแกรมสำหรับคำนวณ SOAP นั้นสามารถดูได้ที่ https://singroup.github.io/dscribe/latest/tutorials/descriptors/soap.html

9.5.2 Atom-centered Symmetry Functions

Representation ลำดับถัดมาคือฟังก์ชันสมมาตรแบบมีศูนย์กลางบนอะตอม (Atom-centered Symmetry Functions หรือ ACSF) เป็นวิธีที่ทำการสร้างผลลัพธ์หรือคำตอบจากฟังก์ชันหลายวัตถุ (Many-body Functions) อีกทีหนึ่งเพื่อทำการประมาณค่า Electronic Environment รอบ ๆ อะตอมที่เราสนใจในโมเลกุล ซึ่ง ACSF นี้ได้ถูกพัฒนาโดยศาสตราจารย์ Jörg Behler ซึ่งตีพิมพ์ในปี ค.ศ. 2011 โดยถูกออกแบบเพื่อนำมาใช้กับโมเดลประเภท Neural Network[120]

ไอเดียของ ACSF ก็คือจะเป็นการแปลง (Transformation) พิกัดคาร์ทีเซียนให้เป็นฟังก์ชันสมมาตร (Symmetry Function) โดยจะมีการกำหนดฟังก์ชันสำหรับ Cutoff ขึ้นมาก่อน ดังนี้

$$f_c(R_{ij}) = \begin{cases} \frac{1}{2}\left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right], & R_{ij} \leq R_c \\ 0, & R_{ij} \geq R_c \end{cases} \tag{9.14}$$

โดยที่ R_{ij} คือระยะห่างระหว่างอะตอม i กับ j และ R_c คือค่า Cutoff

Symmetry Function นั้นเป็นฟังก์ชันที่สามารถอธิบายกรณีแบบหลายวัตถุ (Many-body) แต่ว่าจริง ๆ แล้วมีชื่อเรียก Symmetry Function เฉพาะ นั่นคือ Symmetry Function ที่เป็นฟังก์ชันแบบสองวัตถุ (Two-body Function) นั้นเราจะเรียกว่าฟังก์ชันเชิงรัศมี (Radial Function) และเรียกที่เป็นฟังก์ชันแบบสามวัตถุ (Three-body Function) ว่าฟังก์ชันเชิงมุม (Angular Function) ตามลำดับ ซึ่งความแตกต่างหลัก ๆ ก็คือ Radial Function จะเป็นผลรวมเชิงเส้นของเทอมที่เป็น Two-body ส่วน Angular Function นั้นจะมีการเพิ่มเทอมที่เป็น Three-body เข้ามาด้วยนั่นเอง

โดยเรามาดูกันที่ Radial Function ก่อนซึ่งจะมีด้วยกัน 3 ฟังก์ชันย่อย ดังนี้

$$G_i^1 = \sum_j f_c(R_{ij}) \tag{9.15}$$

$$G_i^2 = \sum_j e^{-\eta(R_{ij}-R_s)^2} f_c(R_{ij}) \tag{9.16}$$

$$G_i^3 = \sum_j \cos(\kappa R_{ij}) f_c(R_{ij}) \tag{9.17}$$

ซึ่ง G_i^1 ก็คือฟังก์ชันผลรวมของ Cutoff Function (สมการที่ (9.14)), G_i^2 คือผลรวม Gaussian Function คูณด้วย Cutoff Function และ G_i^3 คือฟังก์ชัน Cosine ที่ถูกปรับการหน่วงโดยค่าพารามิเตอร์ κ

นอกจากนี้ยังมี Angular Function อีกสองฟังก์ชันย่อยด้วยกัน นั่นคือฟังก์ชัน G_i^4

$$\begin{aligned} G_i^4 = \; & 2^{1-\xi} \sum_{j,k\neq i}^{\text{all}} (1 + \lambda \cos\theta_{ijk})^{\xi} \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)} \\ & \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}) \end{aligned} \tag{9.18}$$

และฟังก์ชัน G_i^5

$$G_i^5 = 2^{1-\xi} \sum_{j,k\neq i}^{\text{all}} (1 + \lambda \cos\theta_{ijk})^{\xi} \cdot e^{-\eta(R_{ij}^2 + R_{ik}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik})) \tag{9.19}$$

สำหรับการคำนวณหาแรง (Force) และเทนเซอร์แรงผลัก (Stress Tensor) สำหรับกรณีของ ACSF นั้นสามารถทำได้โดยใช้กฎลูกโซ่ (Chain Rule) ซึ่งต้องการทำพิสูจน์จากสมการของพลังงานที่เกิดขึ้นจากการคำนวณใน Neural Network โดยสำหรับค่าพลังงานย่อยต่อหนึ่งหน่วย Neuron นั้นมีนิยามตามนี้

$$E_{\text{Neuron}} = f(wG + b) \tag{9.20}$$

โดยที่ f คือฟังก์ชันกระตุ้น, w คือค่าน้ำหนัก (Weight Parameter), b คือค่าความอคติ (Bias Parameter) และ G คืออินพุตของ Neuron นั้น ๆ ซึ่งก็คือ Symmetry Function ตามด้านบนที่ได้อธิบายไป โดยที่พลังงานรวมทั้งหมดของโมเลกุล (Total Energy) หรือ E สามารถคำนวณได้จากผลรวมของพลังงานย่อย ๆ ที่เกิดขึ้นจากแต่ละอะตอม (สมการที่ (9.20)) ซึ่งมีสมการดังนี้

$$E = \sum_i E_i \tag{9.21}$$

นอกจากนี้ยังมี Representation อื่น ๆ ที่คล้ายกับ ACSF นั่นคือถูกพัฒนาขึ้นมาด้วยไอเดีย Symmetry Function เหมือนกัน เช่น Spectrum of London and Axilrod-Teller-Muto (SLATM) ซึ่งถูกใช้อย่างแพร่หลายเหมือนกันแต่จะนิยมใช้กับ KRR มากกว่า[121,122] แล้วก็ยังมีการใช้ฟังก์ชันพหุนาม (Polynomial functions) ของส่วนกลับของระยะห่างระหว่างอะตอม (Inverse Bond Distance) ด้วย[123,103]

ตัวอย่างโค้ดสำหรับการตั้งค่า ACSF Representation และคำนวณ ACSF ของโมเลกุลน้ำโดยใช้ไลบรารี่ DScribe และ ASE

```python
from dscribe.descriptors import SOAP
from ase.build import molecule

# Setting up the ACSF descriptor
acsf = ACSF(
    species=["H", "O"],
    rcut=6.0,
    g2_params=[[1, 1], [1, 2], [1, 3]],
    g4_params=[[1, 1, 1], [1, 2, 1], [1, 1, -1], [1, 2, -1]],
)

# Molecule created as an ASE.Atoms
```

```
water = molecule("H2O")

# Create ACSF output for the hydrogen atom at index 1
acsf_water = acsf.create(water, positions=[1])

print(acsf_water)
print(acsf_water.shape)
```

นอกจากนี้แล้ว ACSF ยังได้ถูกนำมาพัฒนาและปรับปรุงให้มีประสิทธิภาพในการทำนายคุณสมบัติเชิงอิเล็กทรอนิกส์แบบเฉพาะ เช่น Weighted ACSF,[124] การใช้ Polynomial Function มาเป็นฟังก์ชันเสริม,[125] Physically-inspired ACSF[126] ซึ่งเป็นการช่วยลดตัวแปรที่ต้องกำหนดสำหรับ Symmetry Function, และ Spin-dependent ACSF สำหรับทำนายโมเลกุลที่มีคุณสมบัติเชิงแม่เหล็ก[127]

9.5.3 Gaussian-type Orbital-based Density Vectors

เวกเตอร์ของความหนาแน่นเชิงออร์บิทัลประเภทเกาส์เซียน (Gaussian-type Orbital-based Density Vector หรือเรียกสั้น ๆ ว่า GTB-based Density Vector) เป็น Representation อีกอันหนึ่งที่ใช้หลักการของ Molecular/Atomic Orbitals ซึ่งถูกพัฒนาขึ้นมาเพื่อให้เป็นอีกทางเลือกหนึ่งนอกเหนือจาก ACSF และ Symmetric Polynomial Function[128] โดยสมการที่ใช้คำนวณ GTO-based Density Vector คือ

$$\rho^{i}_{L,\alpha,r_s} = \sum_{l_x,l_y,l_z}^{l_x+l_y+l_z=L} \frac{L!}{l_x!l_y!l_z!} \left(\sum_{t=1}^{n_{type}} c_t \sum_{j=1}^{N^t_{atom}} \phi^{\alpha,r_s}_{l_x l_y l_z}(\boldsymbol{r}_{ij}) \right) \tag{9.22}$$

โดยกำหนดให้ $l_x + l_y + l_z = L$ เป็นเลข Angular Momentum ของออร์บิทัล, n_{type} เป็นชนิดของอะตอมในโมเลกุล, c_t เป็นค่าถ่วงน้ำหนักที่ขึ้นกับชนิดของอะตอม (Type-dependent Weight), N^t_{atom} คือจำนวนของอะตอมชนิดนั้น ๆ และ $\phi^{\alpha,r_s}_{l_x l_y l_z}(r_{ij})$ เป็นออร์บิทัลแบบเกาส์เซียนของแต่ละอะตอม นอกจากนี้ยังมีการกำหนดพารามิเตอร์เพิ่มเติมคือ α และ r_s ซึ่งเป็นตัวกำหนดฟังก์ชันออร์บิทัลของ Radial Distribution[128] โดยออร์บิทัลแบบเกาส์เซียนมีสมการดังต่อไปนี้

$$\phi^{\alpha,r_s}_{l_x l_y l_z}(\boldsymbol{r}_{ij}) = x^{l_x} y^{l_y} z^{l_z} e^{-\alpha|r-r_s|^2} \tag{9.23}$$

โดยกำหนดให้ $\boldsymbol{r}_{ij} = (x, y, z)$ เป็นเวกเตอร์จากอะตอม i ไปยังอะตอม j และ r เป็นขนาด (Magnitude) ของ $\boldsymbol{r}_{ij}$ โดยทั่วไปแล้วเรามักจะกำหนดให้ค่า $L = 0, 1$ สำหรับการสร้าง Density Vector ด้วย GTO สำหรับโมเลกุลขนาดเล็ก เช่น โมเลกุลเคมีอินทรีย์ที่ประกอบไปด้วยอะตอมขนาดเล็ก เช่น คาร์บอน (C), ไฮโดรเจน (H), ออกซิเจน (O), และไนโตรเจน (N)

9.5.4 ลักษณะเฉพาะเชิงอิเล็กทรอนิกส์อื่น ๆ

นอกจากนี้ยังมีลักษณะเฉพาะเชิงอิเล็กทรอนิกส์อีกมากมายที่ถูกพัฒนาขึ้นมา[121] ดังนี้

- Bonds and Angles based Machine Learning (BAML)[129]
- Histogram of Distances, Angles, and Dihedrals (HDAD)[130]
- Many-body Tensor Representation (MBTR)[131,132]
- Faber-Christensen-Huang-Lilienfeld (FCHL)[121]
- Atomic Cluster Expansion (ACE)[133,134]
- N-body iterative contraction of equivariants (NICE)[135]

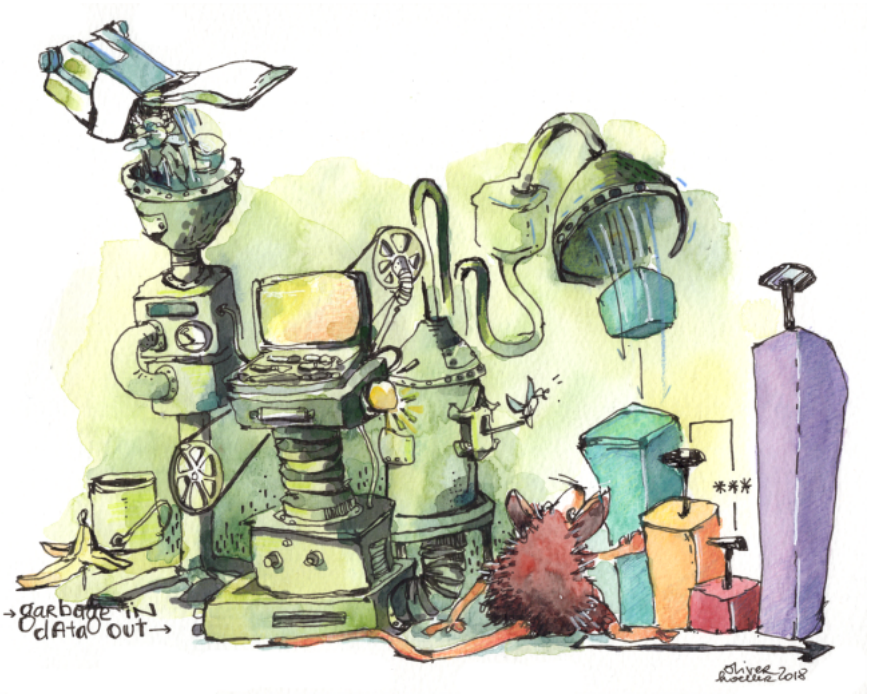
เมื่ออ่านบทความวิจารณ์ (Review) ต่าง ๆ แล้วผู้อ่านจะพบว่าไม่มี Descriptor ไหนที่ให้ผลการทำนายคุณสมบัติเชิงโมเลกุลทุกคุณสมบัติได้แม่นยำที่สุด กล่าวคือ Descriptor แต่ละอันนั้นมีความสามารถในการทำนายคุณสมบัติที่แตกต่างกันไป ถ้าเปรียบเทียบง่าย ๆ ก็เหมือนกับ Functional ของวิธี DFT ซึ่งแต่ละ Functional ก็มีข้อดีและข้อเสียแตกต่างกันไป ดังนั้นผู้เขียนควรศึกษาความแตกต่างของ Descriptor แต่ละอันว่ามีทฤษฎีที่อยู่เบื้องหลังที่ใช้ในการพัฒนา Descriptor นั้น ๆ เป็นอย่างไร จะช่วยทำให้เราเข้าใจและสามารถเลือกใช้ Descriptor ที่เหมาะกับคุณสมบัติเชิงโมเลกุลได้อย่างเหมาะสมและให้ผลการทำนายที่ถูกต้องและแม่นยำ ซึ่งจะช่วยประหยัดเวลาโดยไม่ต้องทำการสุ่มหรือทดลองเปลี่ยน Descriptor ไปเรื่อย ๆ นั่นเอง

นอกจากนี้ผู้เขียนขอแนะนำผู้อ่านที่สนใจการศึกษา Descriptor แบบเชิงลึกได้อ่านบทความงานวิจัย “Physics-Inspired Structural Representations for Molecules and Materials”[103] ที่ได้สรุป Descriptor (ในบทความต้นฉบับผู้วิจัยใช้คำว่า Representation ซึ่งเป็นสิ่งเดียวกัน) ที่เกี่ยวข้องกับ Atomic Composition สำหรับศึกษาคุณสมบัติเชิงเคมีควอนตัมของโมเลกุลไว้อย่างครบถ้วน รวมไปถึงสรุปคุณสมบัติของ Descriptor ที่ดีไว้ด้วย

บทที่ 10

ชุดข้อมูลทางเคมี

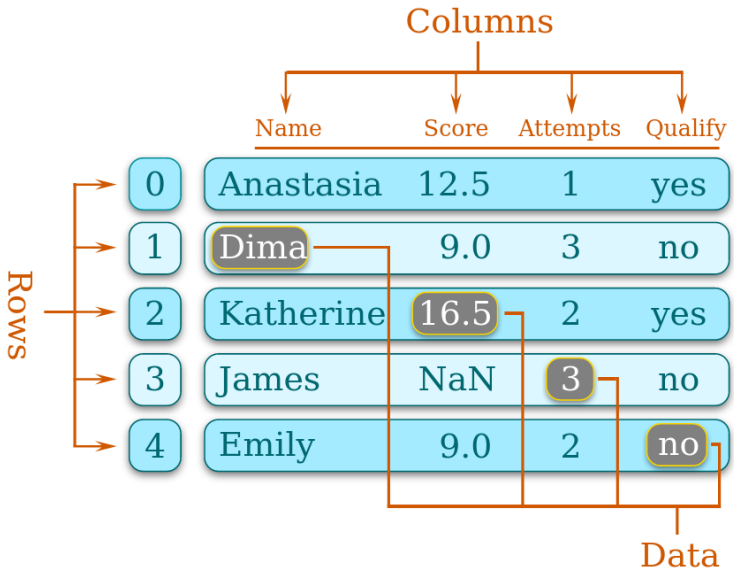
10.1 ชุดข้อมูล



ภาพ 10.1 แนวคิดของ Garbage In, Garbage Out (GIGO) (เครดิตภาพ: http://oliverhoeller.com)

ชุดข้อมูล (Data Set) คือการเก็บข้อมูลให้อยู่ในรูปแบบที่สามารถแบ่งประเภทของข้อมูลในชุดข้อมูลได้ โดยส่วนใหญ่แล้วเรามักจะเก็บข้อมูลในรูปแบบของตาราง โดยตารางของชุดข้อมูลนั้นอาจจะมีหลายคอลัมน์ก็ได้ โดยแต่ละคอลัมน์จะแสดงถึงตัวแปรเฉพาะของข้อมูล ข้อมูลนั้นเป็นสิ่งที่สำคัญและเป็นองค์ประกอบที่

ขาดไม่ได้เลยในการสร้างโมเดลปัญญาประดิษฐ์ ซึ่งการที่เรามีข้อมูลมหาศาลในทุกวันนี้ก็อาจจะเรียกได้ว่าเป็นสาเหตุหลักที่ทำให้เกิด ML ขึ้นมาได้ ชุดข้อมูลถือได้ว่าเป็นหัวใจสำคัญของ ML เลยก็ว่าได้ ถ้าหากเรามีชุดข้อมูลที่มีคุณภาพดี เราก็จะสามารถสร้าง Feature Input Vector ที่มีคุณภาพให้กับโมเดลต้องการฝึกสอนได้ แต่ถ้าหากชุดข้อมูลของเราไม่ดี สิ่งที่ตามมาก็คือโมเดลที่ถูกฝึกสอนออกมานั้นก็จะมีประสิทธิภาพในการทำนายที่ต่ำมาก (Garbage In, Garbage Out)



ภาพ 10.2 ตัวอย่างชุดข้อมูลแบบ 2 มิติ โดยมี Feature คือ Score, Attempts, และ Qualify (เครดิตภาพ: w3resource.com)

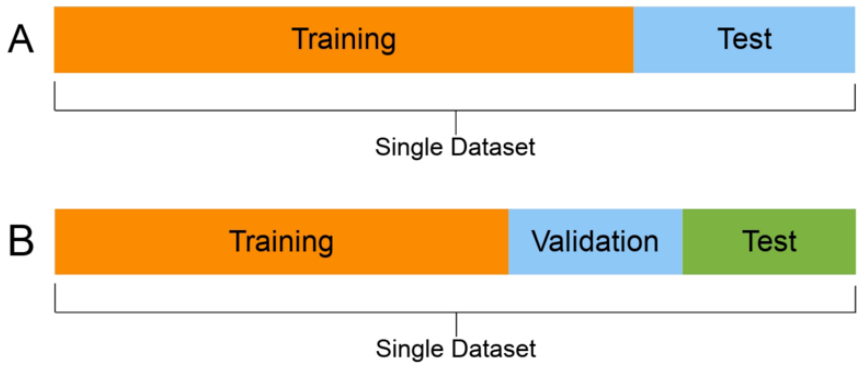
เรามาดูรายละเอียดของชุดข้อมูลกันมากกว่านี้ดีกว่าครับ เริ่มจากเราต้องทำความเข้าใจมิติของชุดข้อมูล (Dimensionality of Dataset) กันก่อน ซึ่งมิติของชุดข้อมูลก็จะมีตั้งแต่ 1 มิติ, 2 มิติ, 3 มิติ, 4 มิติ หรือสูงมากกว่านั้นก็ได้ ให้ลองนึกถึง Tensor ซึ่งเราสามารถเพิ่มจำนวนมิติของข้อมูลได้ สำหรับกรณีที่ชุดข้อมูลมี 1 มิตินั้นจะง่ายที่สุดเพราะเราจะมองว่าชุดข้อมูลแบบนี้เป็นเวกเตอร์ก็ได้ ชุดข้อมูล 1 มิติก็คือข้อมูลที่มีหลายแถวแต่มีแค่ 1 หลัก หรือจะเป็นชุดข้อมูลที่มีเพียงแค่ 1 แถวแต่มีหลายหลักก็ได้ สำหรับชุดข้อมูลแบบ 2 มิตินั้นให้เปรียบเทียบกับตาราง ซึ่งตารางประกอบไปด้วยแถวและหลัก โดยเราจะมองว่าตารางนั้นจริง ๆ แล้วก็คือเมทริกซ์ก็ได้ ซึ่งชุดข้อมูล 2D ประกอบด้วยมิติของแถว (Row) และมิติของหลักหรือคอลัมน์ (Column) เมื่อนำจำนวนของแถวคูณกับจำนวนของหลัก (Row x Column) จะได้ขนาดของชุดข้อมูล (Size) ซึ่งสอดคล้องกับขนาดของเมทริกซ์ เช่น ชุดข้อมูลขนาด 125 x 50 หมายความว่าชุดข้อมูลนี้คือชุดข้อมูลขนาด 2 มิติ ที่มีจำนวนข้อมูลทั้งหมด 125 แถว แต่ละแถวมี 50 หลัก สำหรับชุดข้อมูล 3 มิติ ก็จะมีอีก 1 มิติเพิ่มเข้ามานอกเหนือจากแถวกับหลักซึ่งจะถูกเรียกว่าอะไรนั้นก็ขึ้นอยู่กับว่าข้อมูลนั้นเป็นข้อมูลประเภทไหน เพราะว่าบางครั้งเราก็ไม่ได้ใช้คำว่าแถวกับหลัก เช่น ถ้าเป็นชุดข้อมูลรูปภาพ ก็จะใช้ ความสูง x ความกว้าง x ความลึก (Height x Width x Depth) ซึ่งก็สามารถเรียงสลับได้

โดยชุดข้อมูลพื้นฐานที่ใช้กันอย่างแพร่หลายทางด้าน ML ได้แก่ชุดข้อมูลดอกไม้ (Iris Dataset) กับชุด

ข้อมูลลายมือตัวเลขอารบิก (MNIST Dataset)

10.2 ประเภทและการแบ่งชุดข้อมูล

ชุดข้อมูลที่ใช้ใน ML นั้นโดยทั่วไปแล้วมักจะมีอยู่ 2 ประเภทคือชุดข้อมูลสำหรับการฝึกสอน (Training Set) และชุดข้อมูลสำหรับการทดสอบ (Test Set) ซึ่งวัตถุประสงค์ของชุดข้อมูลทั้งสองประเภทนี้ก็ตรงตัวเลยก็คือ Training Set จะถูกนำมาใช้ในการฝึกสอนโมเดล ส่วน Test Set จะถูกเก็บไว้ใช้ในการทดสอบโมเดลหรือการทำนายคำตอบที่โมเดลถูกสอนมา (Prediction) อย่างไรก็ตาม การฝึกสอนโมเดลโดยการใช้ Train Set ทั้งหมดนั้นมักจะทำให้เกิดความโน้มเอียง (Bias) ที่เกิดขึ้นจากชุดข้อมูลและส่งผลให้เกิด Biased ในขั้นตอน Prediction ด้วย เพราะป้องกันเหตุการณ์ดังกล่าวและทำให้เกิด Bias น้อยที่สุด เรามักจะทำการแบ่ง (Split) ชุดข้อมูลฝึกสอนให้เป็นชุดข้อมูลสำหรับการฝึกสอนจริง ๆ (Actual Training Set) และชุดข้อมูลสำหรับการตรวจสอบและยืนยันความถูกต้องซึ่งเรียกอีกอย่างว่า Validation Set



ภาพ 10.3 การแบ่งชุดข้อมูลทั้งหมดออกเป็น (A) Training Set และ Test Set และ (B) Training Set, Validation Set และ Test Set (เครดิตภาพ: Wikimedia Commons)

ภาพที่ 10.3 แสดงสัดส่วนแบบคร่าว ๆ ในการแบ่งชุดข้อมูลหลักออกเป็น Training Set และ Test Set และแสดงการแบ่งชุดข้อมูล Training Set อีกครั้งให้เป็น Actual Training Set ที่จะถูกนำไปใช้ในการฝึกสอนโมเดลจริง ๆ และ Validation Set ที่จะถูกนำมาทดสอบโมเดลเพื่อเป็นการหยั่งเชิงความสามารถของโมเดลก่อนที่จะนำไปใช้ทำนายค่าของเอาต์พุตของข้อมูลใน Test Set โดยทั่วไปแล้วหลาย ๆ คนมักจะทำการแบ่งชุดข้อมูลโดยใช้อัตราส่วนคือ 80% (สำหรับ Training Set) และ 20% (สำหรับ Test Set) ตามหลักการของ Pareto[1]

แล้วขั้นตอนการลด Bias นั่นมันเกิดขึ้นได้อย่างไร คำตอบก็คือในการแบ่งข้อมูลออกมาเป็น Validation Set (เช่นแบ่งออกมา 20% จากทั้งหมด) โดยทำการสุ่มเลือกบางส่วนของข้อมูลออกมา ซึ่งถ้าหากเราทำวนไปแบบนี้ไปเรื่อย ๆ เราจะเรียกว่าเป็นการทำ Validation แบบข้ามไปมาทั่วทั้ง Training Set ซึ่งเมื่อเรานำ

[1] https://en.wikipedia.org/wiki/Pareto_principle

Training Set แต่ละชุดไปฝึกสอนโมเดล เราจะได้ประสิทธิภาพของโมเดลแบบเฉลี่ย เปรียบเสมือนเป็นการเกลี่ยหาความเท่ากันของข้อมูลนั่นเอง (กระจายออกไปให้เสมอกัน) ท้ายที่สุดแล้วถ้าเราแบ่งชุดข้อมูลตามที่ได้อธิบายมา เราจะมีอัตราส่วนของชุดข้อมูลย่อย ๆ แต่ละประเภท ดังนี้

- Training: 80%
  - Actual Training Set: 60%
  - Cross Validation: 20%
- Testing: 20%

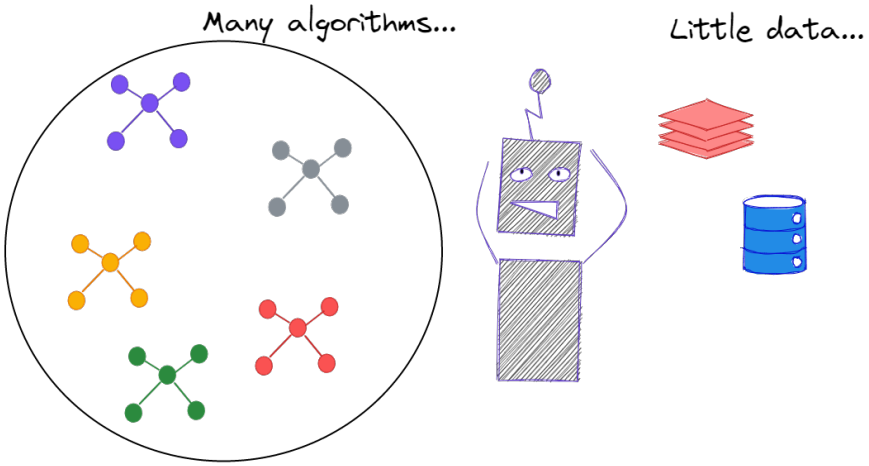
นอกจากนี้แล้วเรายังสามารถแบ่งชุดข้อมูลตามประเภทของอัลกอริทึมของ ML ได้ด้วย โดยแบ่งออกตามประเภทของการเรียนรู้ ดังนี้

- การเรียนรู้แบบมีผู้สอน : นำข้อมูลที่ใช้ในการฝึกสอนมาแยกประเภทผลลัพธ์ด้วยการติดป้ายกำกับ (Labels/Class) เป็นผลเฉลย จากนั้นนำข้อมูลที่ติดป้ายแล้วไปใช้ในการฝึกโมเดลที่ทำงานผ่านอัลกอริทึมสำหรับสร้างโมเดลที่ใช้ในการทำนายผลลัพธ์
- การเรียนรู้แบบไม่มีผู้สอน : ชุดข้อมูลสำหรับ ML ประเภทนี้จะเป็นแบบไม่ถูกจัดประเภทหรือติดป้ายกำกับข้อมูล วิธีนี้โมเดลของเราจะคาดเดาข้อมูลที่ได้รับและทำความเข้าใจถึงโครงสร้างที่ซ่อนอยู่แต่ไม่สามารถหาผลลัพธ์ที่ถูกต้องได้ 100% แต่จะใช้วิธีสำรวจข้อมูลและใช้การประมาณการว่าข้อมูลนั้นคืออะไร

10.3 การสร้างชุดข้อมูล

โดยทั่วไปแล้วการเลือกใช้อัลกอริทึม ML นั้นควรจะต้องสอดคล้องกับขนาดของชุดข้อมูล (ปริมาณข้อมูล) ปัญหาก็คือว่าเรามีอัลกอริทึม ML ให้เลือกใช้เยอะมากแต่ว่าบางครั้งเรามีชุดข้อมูลที่มีขนาดเล็กมากหรือไม่มีชุดข้อมูลเลย ดังนั้นสิ่งที่เราต้องทำก็คือสร้างชุดข้อมูลขึ้นมาเองจากข้อมูลดิบหรือทำการเพิ่มปริมาณข้อมูลเพื่อให้ได้ชุดข้อมูลที่มีขนาดที่เหมาะสมเพราะว่าอัลกอริทึม ML หลาย ๆ อัลกอริทึมนั้นต้องการชุดข้อมูลที่มีขนาดใหญ่ ซึ่งถ้าถามว่าเราต้องการชุดข้อมูลที่ใหญ่ขนาดไหนนั้นไม่มีใครสามารถตอบได้ เพราะว่าชุดข้อมูลแต่ละประเภทนั้นไม่เหมือนกัน แต่ถ้าหากเรามีชุดข้อมูลที่ใหญ่ไว้ก่อนก็จะได้เปรียบ เพราะว่าการที่เรามีข้อมูลที่เยอะนั้นย่อมทำให้การฝึกสอนโมเดลนั้นเป็นไปอย่างมีประสิทธิภาพ ขั้นตอนการสร้างชุดข้อมูลประกอบไปด้วย 3 ขั้นหลักดังนี้

1. รวบรวมข้อมูล (Data Collection) สิ่งแรกที่เราจะทำในการมองหา Dataset ก็คือแหล่งข้อมูลที่เราสามารถนำข้อมูลมาใช้ได้ ถ้าหากแหล่งข้อมูลไม่น่าเชื่อถือเราก็มักจะได้ชุดข้อมูลที่มีคุณภาพต่ำซึ่งอาจจะมีข้อผิดพลาดในชุดข้อมูลด้วยเช่นกัน เช่น ข้อมูลที่ถูกสร้างขึ้นมา (ข้อมูลปลอมหรือ Fake Data)



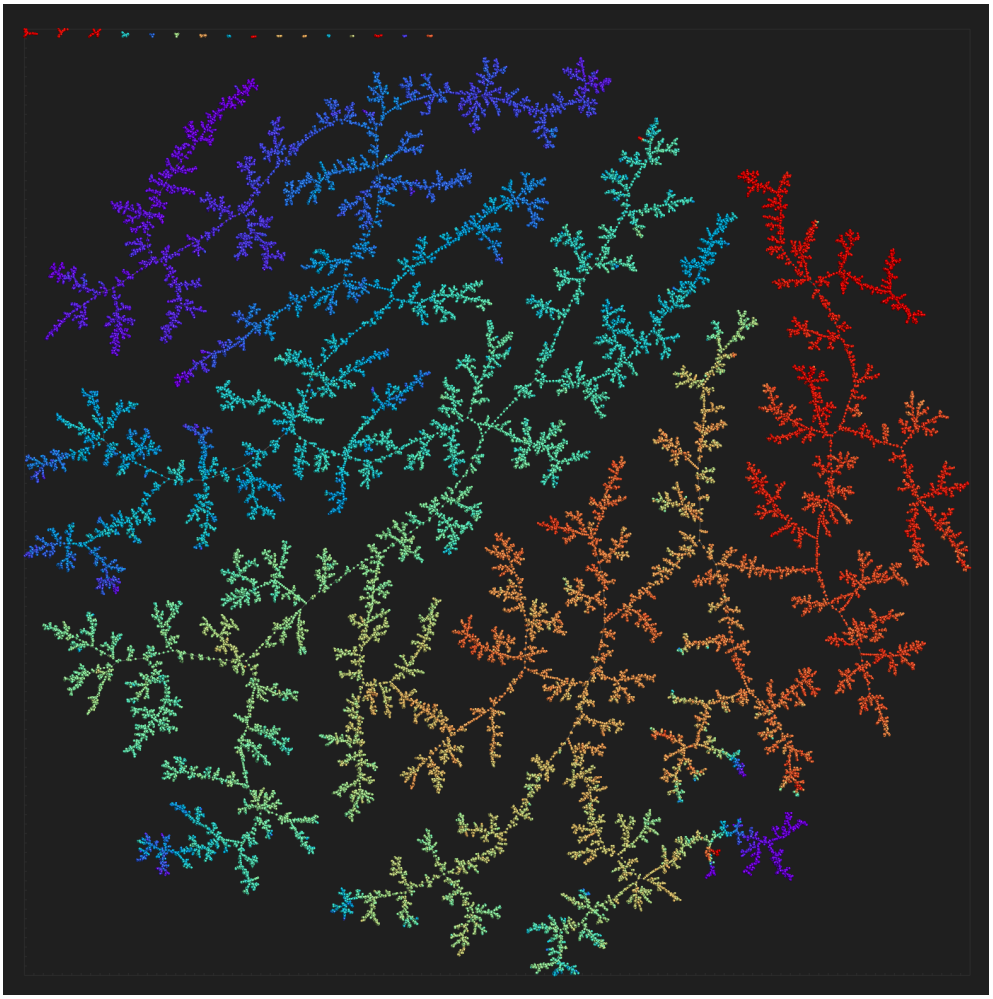
ภาพ 10.4 แสดงปัญหาของการที่เรามีชุดข้อมูลที่มีขนาดเล็กเกินไป ซึ่งอัลกอริทึม ML ส่วนใหญ่นั้นต้องการชุดข้อมูลที่มีขนาดใหญ่ (เครดิตภาพ: https://www.exxactcorp.com)

2. ประมวลผลข้อมูลก่อน (Data Preprocessing) หลักการสำคัญข้อหนึ่งของวิทยาศาสตร์ข้อมูลก็คือทำความสะอาดชุดข้อมูล (Data Cleaning) รวมไปถึงการทราบที่มาที่ไปและการทำความเข้าใจชุดข้อมูล เราต้องถามตัวเองก่อนว่าชุดข้อมูลที่เราสนใจนั้นเคยถูกใช้มาก่อนหน้านี้แล้วหรือยัง ถ้าหากว่ายังและเป็นชุดข้อมูลใหม่ เราควรจะต้องตั้งข้อสังเกตหรือสมมติฐานไว้ก่อนว่าชุดข้อมูลชุดนี้อาจจะมีข้อมูลที่ผิดปกติซ่อนอยู่ได้ หรืออาจจะมีข้อมูลที่ไม่ครบถ้วนขาดหายไป เราสามารถทำการประมวลผลชุดข้อมูลก่อนนำไปใช้งานจริงได้โดยการดูที่คุณภาพของ Features รวมไปถึง Bias ภายในชุดข้อมูล บางชุดข้อมูลมี Features เยอะมากแต่ว่ามี Bias เยอะมาก ๆ กับ Features เพียงแค่ 2-3 Features นอกจากนี้แล้วปริมาณของชุดข้อมูลก็มีผลด้วยเพราะว่าถ้าหากเรามีปริมาณข้อมูลที่น้อยเกินไปก็อาจจะเกิดปัญหา Overfitting ได้ในภายหลัง

3. การทำคำอธิบายประกอบ (Annotatation) หลังจากทำความสะอาดข้อมูลเสร็จเรียบร้อยแล้วสิ่งที่เราควรจะต้องในลำดับต่อไปคือ Annotate นั่นคือเป็นการทำให้มั่นใจว่าข้อมูลของเรานั้นมันสามารถนำไปใช้ในการสอนเครื่องจักรเข้าใจได้ อธิบายง่าย ๆ คือทำให้คอมพิวเตอร์สามารถเรียนรู้จากข้อมูลได้ นั่นก็เพราะว่าเครื่องจักรไม่สามารถเข้าใจข้อมูลเหมือนอย่างที่มนุษย์เข้าใจ ดังนั้นเราควรจะต้องทำการเพิ่มคำอธิบายเชิงดิจิทัลให้กับข้อมูลนั่นก็คือการระบุค่าเฉพาะสำหรับข้อมูลนั้น ๆ หรือที่เรียกว่าการติดป้าย (Labeling)

10.4 ปริภูมิเคมี

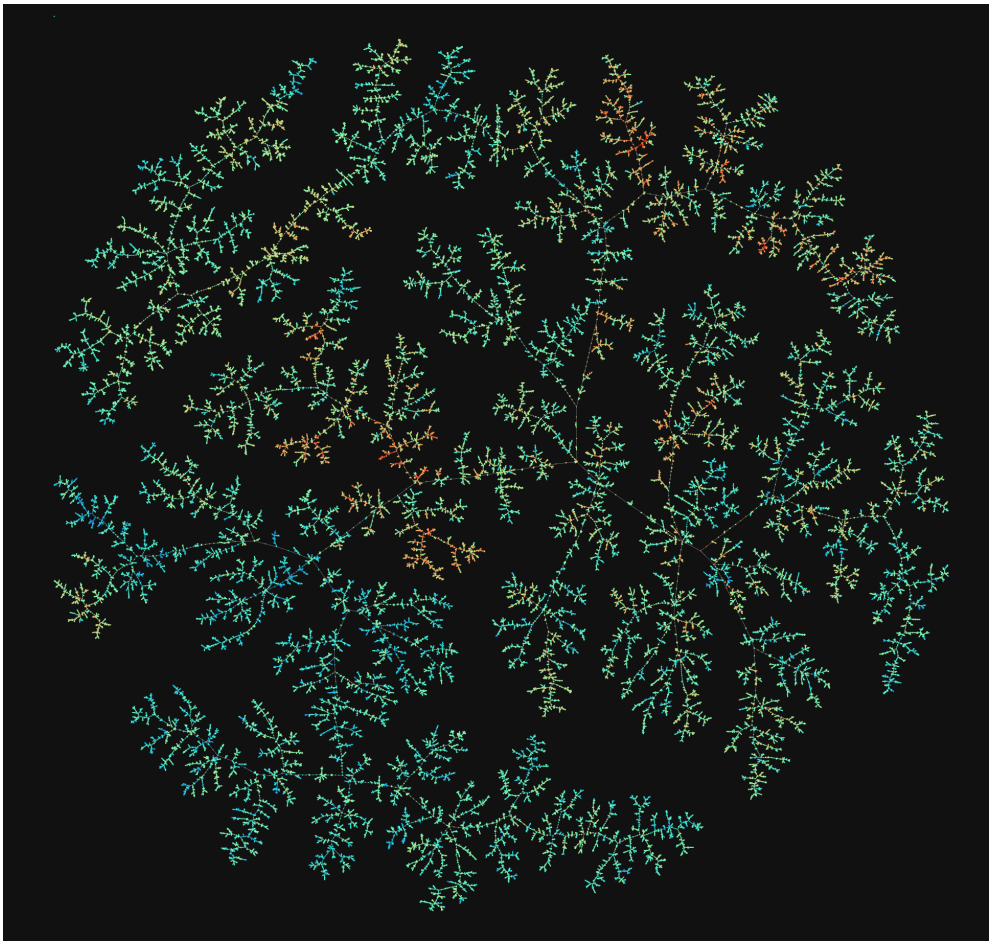
ปริภูมิเคมี (Chemical Space)[136] เป็นแนวคิดที่อธิบายว่าจำนวนและชนิดของโมเลกุลนั้นไม่ที่สิ้นสุด (Infinity) ซึ่งเปรียบเสมือนจำนวนดวงดาวในจักรวาลที่ก็ไม่มีที่สิ้นสุดเหมือนกัน ในขณะที่นักดาราศาสตร์



ภาพ 10.5 ปริภูมิเคมีของชุดข้อมูลโปรตีน Protein Data Bank (PDB) โดยใช้ไลบรารี่ TMAP ในการคำนวณหาความเชื่อมโยง

พยายามสำรวจค้นหาดาวดวงใหม่นั้น นักเคมีก็สำรวจหาโมเลกุลชนิดใหม่ที่ซ่อนอยู่ในปริภูมิเคมี การค้นพบโมเลกุลใหม่นั้นอาจนำมาซึ่งคุณสมบัติเชิงเคมีที่น่าสนใจที่เราสามารถนำเอาองค์ความรู้ไปพัฒนาและต่อยอดในการศึกษาเคมีแขนงต่าง ๆ ได้ เช่น นำโมเลกุลที่ค้นพบใหม่นี้ไปศึกษาปฏิกิริยาใหม่ ๆ หรือรวมไปถึงการนำไปใช้ประโยชน์ในระยะยาวและใช้จริงในระดับอุตสาหกรรม เช่น การพัฒนาวัสดุหรือผลิตภัณฑ์ชนิดใหม่

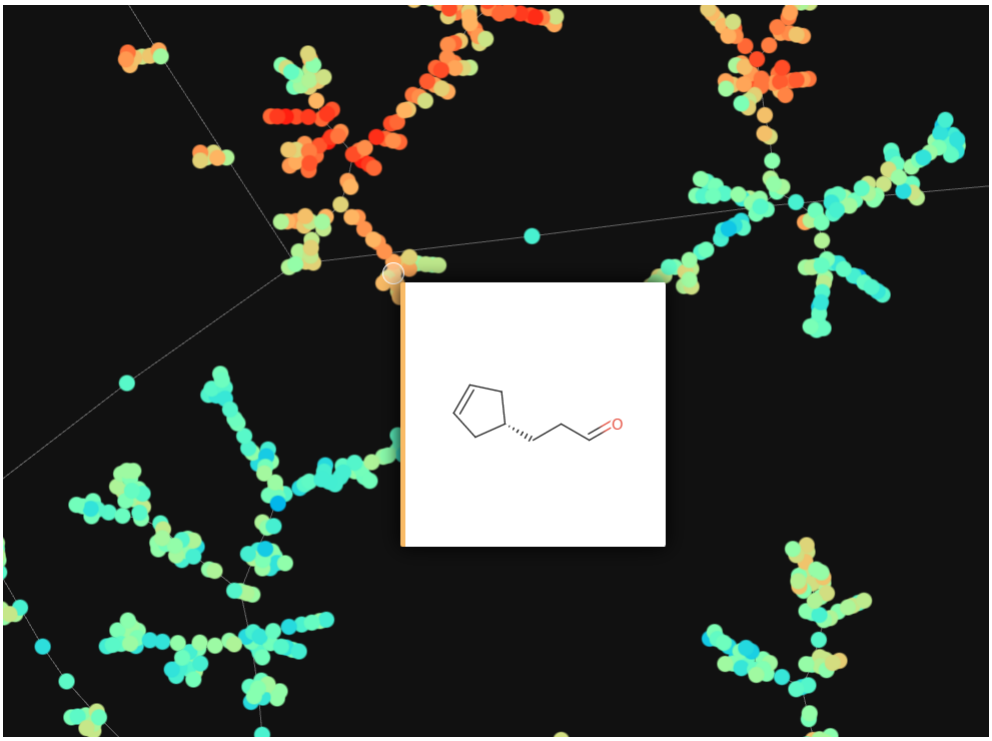
ภาพที่ 10.5 แสดงปริภูมิเคมีของชุดข้อมูล Protein Data Bank (PDB) ซึ่งเป็นชุดข้อมูลที่มีตำแหน่งของอะตอมของโปรตีน ข้อมูลที่เกี่ยวข้องกับโครงสร้างและข้อมูลที่เกี่ยวกับ NMR[1] โดยปริภูมินี้ถูกคำนวณด้วยวิธี Tree MAP (TMAP)[137] ซึ่งเป็นการจัดเรียงและอธิบายความสัมพันธ์ระหว่างข้อมูลแต่ละจุดในรูปแบบของแผนภาพต้นไม้



ภาพ 10.6 ปริภูมิเคมีของชุดข้อมูลโมเลกุลขนาดเล็ก QM9 โดยใช้ไลบรารี่ TMAP ในการคำนวณหาความเชื่อมโยง

นอกจากนี้ยังมีอีกหนึ่งตัวอย่างนั่นคือปริภูมิเคมีของชุดข้อมูล QM9 ซึ่งแสดงในภาพที่ 10.6 ถ้าเราขยายเข้าไปใกล้ ๆ จะพบว่าจุดแต่ละจุดนั้นคือโมเลกุล เช่น โมเลกุลที่แสดงในภาพที่ 10.7 โดยเราสามารถตีความ

[1]ดูรายละเอียดของ PDB ได้ที่ https://www.rcsb.org/



ภาพ 10.7 ภาพขยายของโมเลกุลที่สุ่มเลือกจากปริภูมิเคมีของชุดข้อมูลโมเลกุลขนาดเล็ก QM9

แผนภาพนี้ได้ว่ายิ่งจุดอยู่ใกล้กันมากเท่าไหร่ก็หมายความว่าโมเลกุลเหล่านั้นมีความเชื่อมโยงกันมากเท่านั้น และถ้าอยู่ห่างกัน เช่น อยู่กันคนละกลุ่ม ก็หมายความว่าโมเลกุลมีความสัมพันธ์กันน้อย สำหรับสีที่ใช้ในการไฮไลท์จุดแต่ละจุดนั้นแสดงถึงค่าของ cLogP ซึ่งเป็นสเกลที่บอกถึงความสามารถในการละลายน้ำซึ่งก็คือชอบน้ำ (Hydrophilic) หรือไม่ชอบน้ำ (Hydrophobic) นั่นเองซึ่งมีค่าตั้งแต่ -3.16 ถึง 3.76 และเป็นปริมาณที่ไม่มีหน่วย (Unitless)

10.5 การสร้างชุดข้อมูลเคมีควอนตัม

การสร้างชุดข้อมูลเคมีควอนตัมสามารถแบ่งออกได้เป็น 2 ประเภท ดังนี้

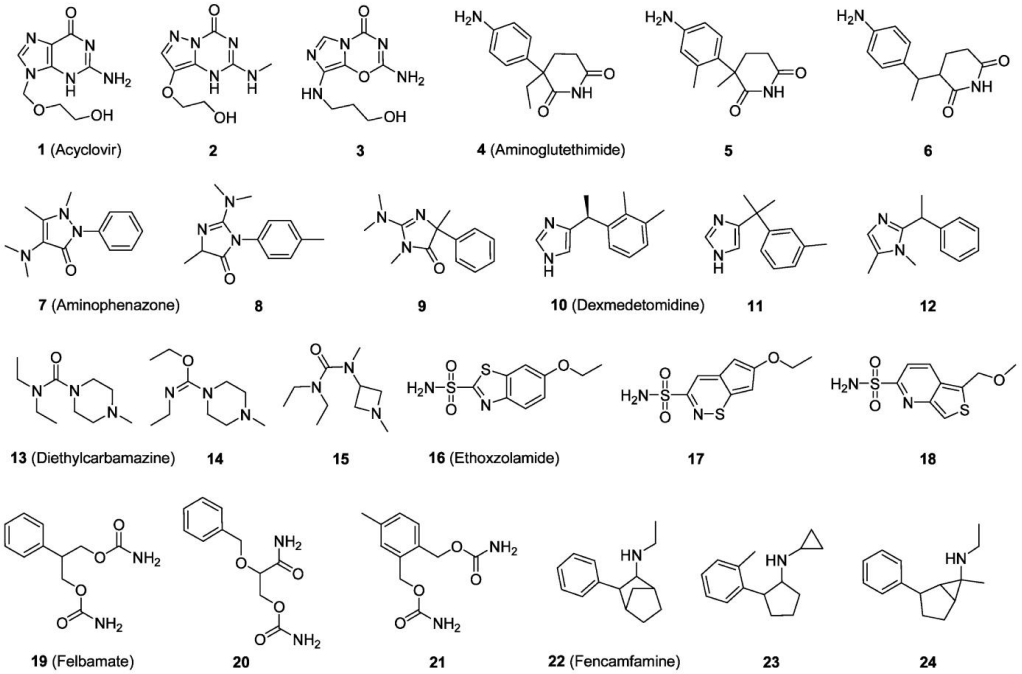
1. ชุดข้อมูลแบบที่มีเพียงหนึ่งโมเลกุลแต่มีหลาย Configuration: เป็นชุดข้อมูลที่โมเลกุลเพียงแค่หนึ่งโมเลกุลนั้นมีการจัดรูปแบบเชิงโครงสร้างที่หลากหลายแตกต่างกัน โดยเราสามารถใช้การจำลองด้วยวิธี Molecular Dynamics แล้วนำ Trajectory มาใช้เป็นชุดข้อมูลก็ได้ นอกจากนี้เรายังสามารถนำ Configuration (หรือจะเรียกว่า Conformer ก็ได้) ซึ่งเป็นโครงสร้างใน Trajectory มาคำนวณหาคุณสมบัติเชิงอิเล็กทรอนิกส์เพิ่มเติมด้วยวิธีการคำนวณที่มีความแม่นยำมากกว่า เช่น DFT หรือ MP2

2. ชุดข้อมูลแบบที่มีหลายโมเลกุล โดยแต่ละโมเลกุลมีเพียงหนึ่ง Configuration: ชุดข้อมูลแบบนี้จะมี

ความซับซ้อนในการสร้างที่มากกว่าชุดข้อมูลแบบแรกนั่นก็เพราะว่าเราจะต้องมีการสร้างอัลกอริทึมที่สามารถสร้างโมเลกุลที่มีความหลากหลายแตกต่างกันได้และโมเลกุลที่สร้างขึ้นมานั้นจะต้องเป็นโมเลกุลที่เหมาะสม มีโครงสร้างที่เหมาะสม แต่ถูกต้องตามหลักเคมี เช่น มีจำนวนพันธะที่เหมาะสม สามารถนำมาสังเคราะห์หรือศึกษาได้ในเชิงการทดลอง

10.6 ชุดข้อมูลเคมีควอนตัมมาตรฐาน

QM9 เป็นหนึ่งในชุดข้อมูลเคมีควอนตัมที่ได้รับความนิยมและถูกนำมาใช้ในงานวิจัย ML สำหรับเคมีควอนตัมเป็นอย่างมาก ซึ่งถูกใช้อย่างแพร่หลายตั้งแต่ปี ค.ศ. 2014 เป็นต้นมา[138,139] โดยบทความงานวิจัยแรกที่นำ QM9 มาใช้ในการทดสอบประสิทธิภาพของโมเดล ML นั้นได้รายงานค่าความผิดพลาดของโมเดล ML ที่ใช้ในการทดสอบว่ามีค่าคลาดเคลื่อนไม่เกิน 10 kcal/mol ซึ่งในเชิงการวัดนั้นถือว่ามีคลาดเคลื่อนที่เยอะมาก ๆ และในเวลาต่อมาก็ได้มีการพัฒนาระเบียบวิธีวิจัยรวมไปถึงโมเดล ML และ Descriptor ใหม่ ๆ จนทำให้ในปัจจุบันนั้นนักวิจัยสามารถที่จะทำนายหรือพยากรณ์ค่าพลังงานของโมเลกุลทางเคมีอินทรีย์ขนาดเล็กได้แม่นยำมากโดยมีค่าความคลาดเคลื่อนประมาณ 1 kcal/mol หรือต่ำกว่านั้น ซึ่งค่า 1 kcal/mol นี้ถือได้ว่าเป็น *ค่าความถูกต้องทางเคมี (Chemical Accuracy)* ซึ่งเป็นค่ามาตรฐานที่ต่ำที่สุดที่เทคนิคทางการทดลองสามารถวัดได้ โดยถ้าหากค่าความคลาดเคลื่อนทางการคำนวณที่ต่ำไปกว่า 1 kcal/mol แล้วเทคนิคต่าง ๆ ในเชิงการทดลองจะไม่สามารถระบุความแตกต่างของความคลาดเคลื่อนที่แม่นยำได้อีกต่อไป



ภาพ 10.8 โมเลกุลเพียงบางส่วนของชุดข้อมูล QM9

ตาราง 10.1 ข้อมูล Feature ของชุดข้อมูล QM9

ดัชนี	ชื่อ	หน่วย	คำอธิบาย
0	index	-	Consecutive, 1-based Integer Identifier of Molecule
1	A	GHz	ค่าคงที่การหมุน (Rotational Constant) A
2	B	GHz	ค่าคงที่การหมุน (Rotational Constant) B
3	C	GHz	ค่าคงที่การหมุน (Rotational Constant) C
4	mu	Debye	ไดโพลโมเมนต์ (Dipole Moment)
5	alpha	$Bohr^3$	Isotropic Polarizability
6	homo	Hartree	พลังงานของ Highest Occupied Molecular Orbital (HOMO)
7	lumo	Hartree	พลังงานของ Lowest Unoccupied Molecular Orbital (LUMO)
8	gap	Hartree	พลังงานระหว่าง LUMO and HOMO (Energy Gap)
9	r2	$Bohr^2$	Electronic Spatial Extent
10	zpve	Hartree	Zero Point Vibrational Energy
11	U0	Hartree	พลังงานภายใน (Internal Energy) ที่ 0 K
12	U	Hartree	พลังงานภายใน (Internal Energy) ที่ 298.15 K
13	H	Hartree	Enthalpy at 298.15 K
14	G	Hartree	พลังงานอิสระ (Free Energy) ที่ 298.15 K
15	Cv	cal/(mol K)	ความจุความร้อน (Heat Capacity) ที่ 298.15 K

QM9 ประกอบไปด้วยข้อมูลคุณสมบัติอิเล็กทรอนิกส์ของโมเลกุลมากถึง 134,000 โมเลกุล โดยภาพที่ 10.8 แสดงตัวอย่างของโมเลกุลเพียงบางส่วนของ QM9 ซึ่งทุกโมเลกุลในชุดข้อมูลนั้นมีธาตุพื้นฐานเป็นองค์ประกอบ ประกอบไปด้วย คาร์บอน (C), ไนโตรเจน, (N), ออกซิเจน (O), ไฮโดรเจน (H), และฟลูออรีน (F) โดย Feature หลักของ QM9 ก็จะมีพิกัดคาร์ทีเซียนของอะตอมทุกอะตอมในโมเลกุลซึ่งได้มาจากการคำนวณการปรับโครงสร้าง (Geometry Optimization) ด้วยระเบียบวิธี B3LYP/6-31G(2df,p) กับ G4MP2 และนอกจากนี้ยังมีค่า Label หรือค่าที่ไว้ใช้ในการเปรียบเทียบการพยากรณ์ดังแสดงในตารางที่ 10.1[1]

ชุดข้อมูล QM9 มีข้อมูลพิกัดคาร์ทีเซียน (Cartesian Coordinates), ลักษณะเฉพาะ (Features), และค่าพลังงานซึ่งเป็น Target ของข้อมูล โดยผู้อ่านสามารถดาวน์โหลดมาใช้งานได้ฟรีจากเว็บไซต์ http://quantum-machine.org/datasets/ คราวนี้เราลองมาดูโค้ดสำหรับการใช้งาน QM9 โดยใช้ภาษา Python ตามด้านล่างนี้ได้เลย

ทำการเรียกใช้ไลบรารี่และอ่านไฟล์ของชุดข้อมูล

```
# Import libraries
import ase.io as aio
import pandas as pd

# Read qm9.csv
qm9_data = pd.read_csv('./qm9.csv', index_col=0)
```

[1]โมเดล ML ที่เหมาะสมสำหรับการฝึกสอนด้วย QM9 นั้นจะต้องไม่ขึ้นกับ Translation, Rotation และ Permutation

เราสามารถใช้คำสั่งด้านล่างในการแสดง Target ได้

```
# Convert energy from Hartree to kcal/mol
target = qm9_data['u0'] * 627.5096080305927
print(target)

# Output
0         -25400.917498
40       -121896.331092
80       -146555.740566
120      -122481.233425
160      -168344.348805
               ...
119800   -242900.012196
119840   -230424.279698
119880   -266202.856340
119920   -252980.566249
119960   -288738.466448
Name: u0, Length: 3000, dtype: float64
```

อ่านพิกัดคาร์ทีเซียนของโมเลกุล

```
# We read xyz coorinates of all the molecules with ase aio.read
    tool
ase_mols = [aio.read('data/qm9/qm9_xyz/' + mol + '.xyz') for mol
    in qm9_data.mol_id]
```

ตรวจสอบขนาดของโมเลกุลที่ใหญ่ที่สุดในชุดข้อมูล

```
# Check the size of molecules in the dataset
size=[]
for mol in ase_mols:
    num = len(mol.get_atomic_numbers())
    size.append(num)
# Show maximum size of the molecule in the dataset
max(size)

# Output
27
```

โดยโค้ดด้านบนที่เราใช้สำหรับการโหลดชุดข้อมูล QM9 นั้นเราจะนำไปใช้ต่อในบทที่ 11.3 สำหรับการทำนายพลังงานรวมของโมเลกุลของชุดข้อมูล QM9

นอกจาก QM9 แล้วยังมีชุดข้อมูลอื่น ๆ ที่นักวิจัยมักจะนำมาใช้ในการฝึกสอนโมเดลและทำวิจัย เช่น

- QM7[140,105] เป็นส่วนหนึ่งของชุดข้อมูล GDB-13 ซึ่งเป็นชุดข้อมูลที่มีโมเลกุลเคมีอินทรีย์เกือบหนึ่งพันล้านโมเลกุล โดยชุดข้อมูล QM7 ประกอบไปด้วยโมเลกุลที่มีจำนวนอะตอมรวมกันสูงสุดถึง 23 อะตอม โดย Feature ที่ชุดข้อมูลนี้มีคือ Coulomb Matrix และพลังงานการทำให้เป็นอะตอม (Atomization Energy)
- QM7b[140,141] เป็นส่วนขยายของชุดข้อมูล QM7 สำหรับใช้กับโมเดล ML ที่เป็นแบบ Multitask Learning โดยมี Feature เช่น Polarizability, ค่าไอเกนของ HOMO และ LUMO, และพลังงานกระตุ้น
- QM8[138,142] เป็นชุดข้อมูลสำหรับการพัฒนาโมเดล ML สำหรับการเรียนรู้สเปกตรัมเชิงอิเล็กทรอนิกส์ของโมเลกุล โดยชุดข้อมูลนี้มี Feature คือพลังงานกระตุ้นที่ถูกคำนวณด้วยวิธี Second-order Approximate Coupled-cluster (CC2)
- ISO17[143,144,139] เป็นชุดข้อมูลที่ถูกสร้างขึ้นโดยใช้การจำลอง MD ด้วยโปรแกรม Fritz-Haber Institute ab initio Simulation (FHI-aims) โดยมี Feature คือพลังงานและแรงของแต่ละโมเลกุล

ซึ่งก็จะมี Label สำหรับวัตถุประสงค์ในการฝึกสอนโมเดลในการเพิ่มความสามารถการพยากรณ์คุณสมบัติเคมีของโมเลกุลที่ต่างกันออกไป โดยตารางที่ 10.2 แสดงการเปรียบเทียบชุดข้อมูลเคมีควอนตัม

ตาราง 10.2 เปรียบเทียบชุดข้อมูลเคมีควอนตัม

Dataset	Data Type	จำนวน Tasks	จำนวนโมเลกุล	Rec-Split	Heavy Atoms
QM7	SMILES, 3D Coordinates	1	7,165	Stratified	$\leq$7
QM7b	3D Coordinates	14	7,211	Random	$\leq$7
QM8	SMILES, 3D Coordinates	12	21,786	Random	$\leq$8
QM9	SMILES, 3D Coordinates	12	133,885	Random	$\leq$9

ชุดข้อมูลที่รายงานในตารางที่ 10.2 สามารถดาวน์โหลดมาใช้งานได้ฟรีจากเว็บไซต์ http://quantum-machine.org/datasets

นอกจากนี้ยังมีชุดข้อมูลเคมีควอนตัมใหม่ ๆ อีกหลายชุดข้อมูลที่ได้ถูกสร้างขึ้นที่ถูกพัฒนาขึ้นมา โดยมีรายละเอียดของคุณสมบัติเชิงอิเล็กทรอนิกส์ของชุดข้อมูลแต่ละชุด ดังนี้

- **ANI-1**[145] DFT: Total Energy
- **QMugs**[146]

  - GFN2 + DFT: Total, Internal Atomic และ Formation Energies, Dipole, Rotational Constants, HOMO/LUMO/Gap Energies, Mulliken Partial Charges
  - GFN2: Total Enthalpy, Total Free Energy, Quadrupole, Enthalpy, Heat Capacity, Entropy, Fermi Level, Covalent Coordination Number, Molecular Dispersion Coefficient, Atomic Dispersion Coefficients, Molecular Polarizability, Atomic Polarizabilities, Wiberg Bond Orders, Total Wiberg Bond Orders
  - DFT: Electrostatic Potential, Löwdin Partial Charges, Exchange Correlation Energy, Nuclear Eepulsion Energy, One-electron Energy, Two-electron Energy, Mayer Bond Orders, Wiberg-Löwdin Bond Orders, Total Mayer Bond Orders, Total Wiberg-Löwdin Bond Orders, Density/Orbital Matrices, Atomic-orbital-to-symmetry-orbital Transformer Matrix

- ∇DFT[147] DFT: Electrostatic Potential, Löwdin Partial Charges, Exchange Correlation Energy, Nuclear Repulsion Energy, One-electron Energy, Two-electron Energy, Mayer Bond Orders, Wiberg-Löwdin Bond Orders, Total Mayer Bond Orders, Total Wiberg-Löwdin Bond Orders, Density/Orbital Matrices, Atomic-orbital-to-symmetry-orbital Transformer Matrix, Hamiltonian Matrix

10.7 การวิเคราะห์ชุดข้อมูล

หลังจากที่เราเลือกชุดข้อมูลที่ต้องการนำมาศึกษาได้แล้ว ลำดับต่อไปคือการคำนวณ Input Feature หรือจะเรียกว่า Feature Vector ก็ได้ ซึ่งจะถูกนำไปใช้ในการฝึกสอนโมเดล ML ต่อไป โดย Feature Vector ที่ผู้เขียนจะยกตัวอย่างให้ได้ศึกษานั้นก็จะเป็น Feature ที่ใช้ Descriptor แบบง่ายนั่นก็คือ Coulomb Matrix (CM) ผู้เขียนจะยังคงใช้ชุดข้อมูล QM9 และใช้โค้ดต่อไปนี้ในการคำนวณ CM ของโมเลกุลตัวอย่างเพียงแค่ 3,000 โมเลกุลเท่านั้น

```
import numpy as np
from qml.representations import *

cm = []
size = 27 # Maximum size of molecule in the set

# Run for loop over every molecule in the database
for structure in ase_mols:
    # ASE prints atomic numbers
    atomic_numbers = structure.get_atomic_numbers()
    # ASE prints coordinates
    coordinates=structure.get_positions()
    cm1 = generate_coulomb_matrix(atomic_numbers,
    # CM representation is saved into cm1
    coordinates, size = size, sorting="row-norm")
```

```
    # All CM representations are added into one variable
    cm.append(cm1)

# Transform cm into numpy array
cm = np.array(cm)
# Check size of cm
print(cm.shape)

# Output
(3000, 378)
```

หลังจากที่เราคำนวณ CM ของโมเลกุลในชุดข้อมูลเสร็จเรียบร้อยแล้ว สิ่งที่หลายคนทำในลำดับต่อไปก็คือสร้างโมเดลแล้วนำ Feature Vector หรืออินพุตที่ได้ไปใช้ในการฝึกสอนโมเดลทันทีเลย ซึ่งการทำแบบนี้นั้นจริง ๆ แล้วไม่เหมาะสมเท่าไหร่นัก นั่นก็เพราะว่าเราควรจะต้องทำความเข้าใจ Feature ที่เราคำนวณออกมาก่อนโดยทำการวิเคราะห์เพื่อดูลักษณะการกระจายตัวหรือการจัดกลุ่มซึ่งสามารถบอกแนวโน้มรวมไปถึง Bias ได้

เราสามารถใช้เทคนิค Unsupervised ML แบบง่าย ๆ ที่ไม่ซับซ้อน เช่น Principal Component Analysis (PCA) ซึ่งเป็นวิธีที่ลดจำนวนมิติ (Dimensionality Reduction) ของข้อมูลให้อยู่ในรูปขององค์ประกอบเชิงตั้งฉาก (Orthogonal Component) ที่อธิบายปริมาณของความแปรปรวน (Variance) ที่มากที่สุด หรือจะใช้วิธี t-distributed Stochastic Neighbor Embedding (t-SNE) ซึ่งเป็นวิธีที่สามารถแสดงข้อมูลที่มีมิติสูง ๆ (Highd-dimensional Data) ได้เช่นเดียวกัน[148,149] โดยจะทำเปลี่ยนความเหมือนกันระหว่างข้อมูลสองชุดให้เป็นความน่าจะเป็นร่วม (Joint Probability) แล้วทำการปรับค่า Kullback-Leibler Divergence ระหว่างความน่าจะเป็นร่วมของข้อมูลที่อยู่ในมิติต่ำและมิติสูงให้น้อยที่สุด (Minimization)[1] ซึ่งเราสามารถใช้ทั้ง PCA และ t-SNE ในการวิเคราะห์เพื่อดูลักษณะหรืออธิบายง่าย ๆ คือดู “*รูปร่างหน้าตา*” ของ CM ของทั้ง 3,000 โมเลกุลที่คำนวณออกมาได้โดยใช้โค้ดต่อไปนี้

สร้างโมเดล t-SNE

```
import sklearn

tsne_cm = sklearn.manifold.TSNE(n_components=2)
tsne_cm_data = tsne_cm.fit_transform(cm)
```

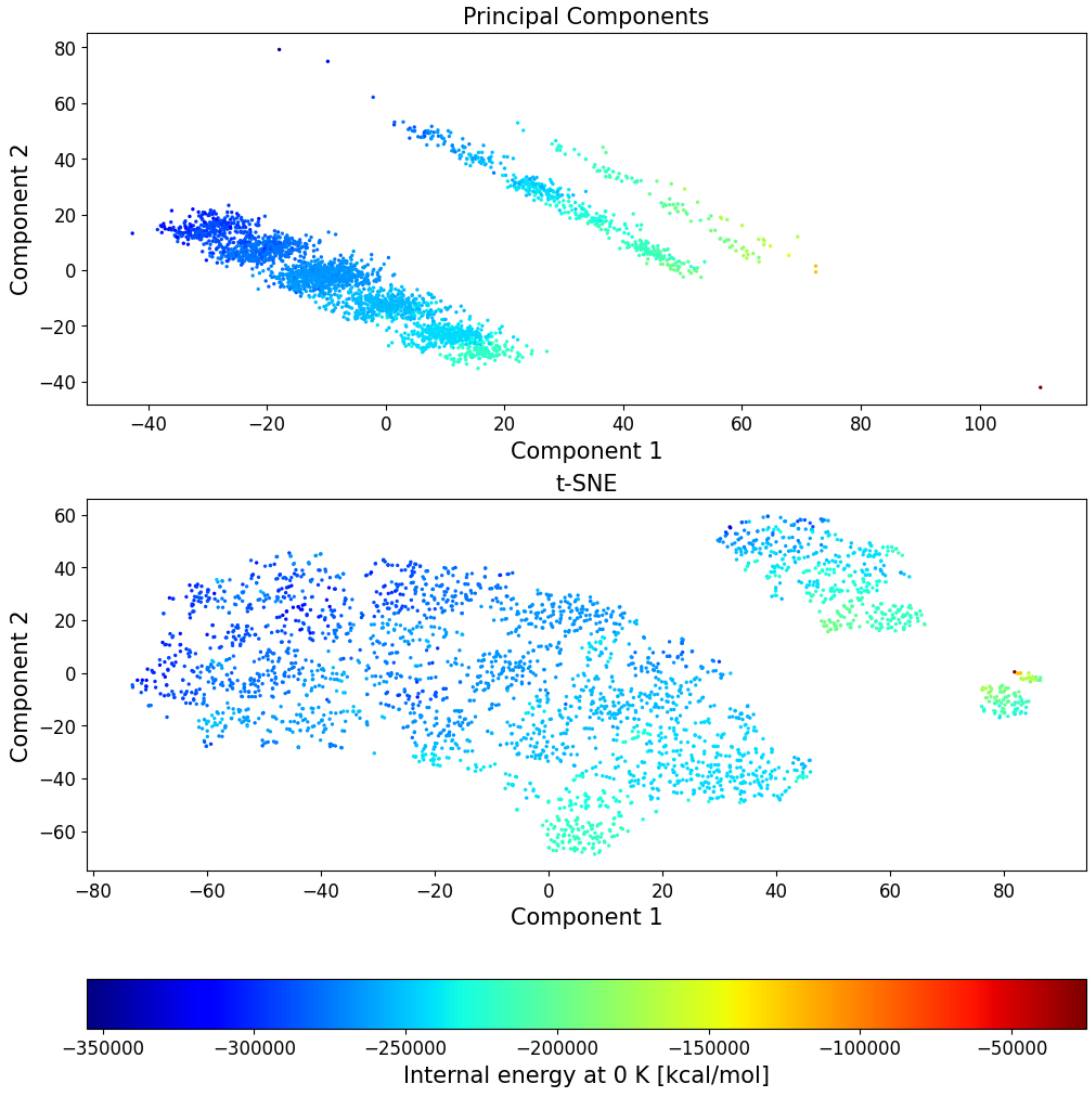
สร้างโมเดล PCA

```
pca_cm = sklearn.decomposition.PCA(n_components=2)
pca_cm_data = pca_cm.fit_transform(cm)
```

[1] เทคนิค t-SNE ถูกพัฒนาต่อมาจากเทคนิค SNE ซึ่งแรกเริ่มนั้นพัฒนาโดย Geoffrey Hinton และ Sam Roweis แห่งมหาวิทยาลัยโทรอนโต ประเทศแคนาดา[150] หลังจากนั้น Laurens van der Maaten ได้ทำการเพิ่ม t-distributed เข้าไป

พล็อตกราฟค่าที่ได้จากการ Fit ข้อมูล

```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(10,10),
   constrained_layout=True)

axs[0].set_title('Principal Components', fontsize=15)
axs[1].set_title('t-SNE', fontsize=15)
axs[0].set_xlabel('Component 1', fontsize=15)
axs[0].set_ylabel('Component 2', fontsize=15)
axs[1].set_xlabel('Component 1', fontsize=15)
axs[1].set_ylabel('Component 2', fontsize=15)
axs[0].tick_params(axis='both', which='major', labelsize=12)
axs[1].tick_params(axis='both', which='major', labelsize=12)

plot1 = axs[0].scatter(pca_cm_data[:, 0], pca_cm_data[:, 1],
   c=target, cmap='jet', s=2)
plot2 = axs[1].scatter(tsne_cm_data[:, 0], tsne_cm_data[:, 1],
   c=target, cmap='jet', s=2)

cbar = fig.colorbar(plot2, ax=axs, orientation="horizontal",
   pad=0.05);
cbar.set_label('Internal energy at 0 K [kcal/mol]', fontsize=15)
cbar.ax.tick_params(labelsize=12)

plt.show()
```

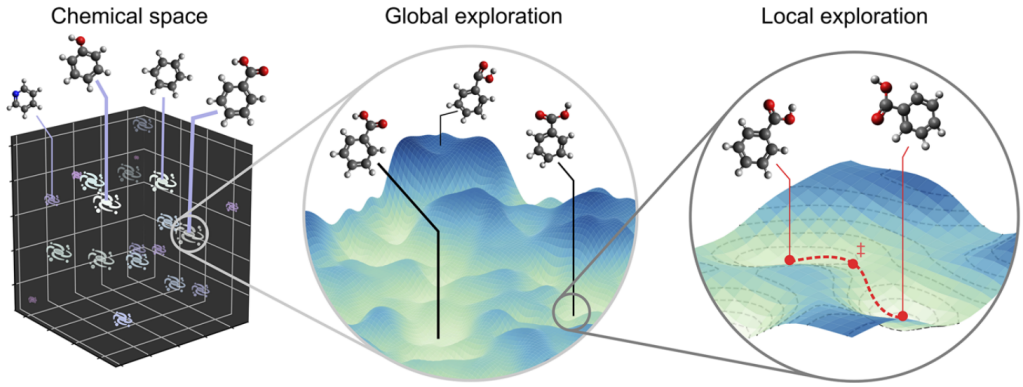


ภาพ 10.9 การกระจายตัวของ Coulomb Matrix Feature ที่ถูกลดจำนวนมิติให้เหลือเพียงแค่ 2 มิติ (Principal Components = 2)

โดยเราจะได้พล็อตตามที่ได้ในภาพที่ 10.9 โดยข้อมูลแต่ละจุดนั้นจะถูกไฮไลต์ด้วยสีที่มีสเกลแตกต่างกันไปซึ่งแสดงค่าของพลังงานภายในของโมเลกุล

บทที่ 11

การทำนายคุณสมบัติของโมเลกุล



ภาพ 11.1 การสำรวจคุณสมบัติเชิงโมเลกุลในปริภูมิเชิงเคมีแบบ Local และ Global (เครดิตภาพ: *J. Chem. Phys.* 2021, 154, 230903[151])

ในบทนี้เราจะมาศึกษาการทำนายสมบัติของโมเลกุลด้วย ML ซึ่งถือได้ว่าเป็นหัวใจสำคัญของส่วนที่สองของหนังสือเล่มนี้เลยก็ว่าได้ โดยผู้อ่านจะได้เรียนรู้และทำความเข้าใจลำดับขั้นตอนในการใช้โมเดล ML ในการทำนายคุณสมบัติเชิงอิเล็กทรอนิกส์ทางควอนตัมต่าง ๆ ที่ได้ศึกษาไปแล้วในบทที่ 8 ซึ่งเป้าหมายหลักของการทำวิจัยในสาขานี้ก็คือการศึกษาและพัฒนา Representation และโมเดล ML ที่สามารถทำนายคุณสมบัติเชิงควอนตัมของโมเลกุลที่ต้องการได้อย่างแม่นยำและเทียบเคียงได้กับผลจากการคำนวณด้วยวิธีเคมีควอนตัมแบบมาตรฐาน เช่น วิธี DFT และวิธีเชิงฟังก์ชันคลื่น[117] นอกจากนี้ผู้อ่านจะได้ศึกษาตัวอย่างโค้ดที่สามารถนำไปใช้งานในการทำนายได้จริงซึ่งผู้เขียนเชื่อว่าการศึกษาทฤษฎีควบคู่ไปพร้อมกับการเขียนโปรแกรมนั้นจะช่วยทำให้เข้าใจทฤษฎีได้อย่างถูกต้องและรวดเร็วเพราะการเขียนโปรแกรมช่วยให้เราคิดอย่างเป็นระบบและเข้าใจสิ่งที่ศึกษาอย่างเป็นขั้นเป็นตอน

11.1 แนวทางการปฏิบัติ

แนวทางปฏิบัติ (Best Practice) หรือลำดับขั้นตอนสำหรับการนำ ML มาประยุกต์กับเคมีควอนตัม สามารถแบ่งออกเป็น 6 ขั้นตอนง่าย ๆ ได้ดังนี้

1. วิเคราะห์ชุดข้อมูลดิบและทำความสะอาดข้อมูล (Data Analysis และ Data Cleaning)
2. เลือก Representation/Descriptor ที่จะนำมาคำนวณ Feature
3. เลือกอัลกอริทึม ML ที่เหมาะสมกับโจทย์หรือชุดข้อมูล
4. ฝึกสอนโมเดลและทำนายคำตอบ
5. ศึกษาผลกระทบจากการเปลี่ยน Hyperparameter และทำ Validation ต่อความถูกต้องในการทำนายเพื่อประเมินประสิทธิภาพของโมเดล
6. สรุปปัจจัยและพารามิเตอร์ที่ทำให้โมเดลมีประสิทธิภาพดีที่สุดและวิเคราะห์ผลการทำนาย

ขั้นตอนแรกของการนำ ML มาประยุกต์กับเคมีควอนตัมก็คือการเตรียมข้อมูลดิบ (Raw Data) นั่นก็คือข้อมูลทางเคมีเบื้องต้นของโมเลกุลที่เรามี โดยส่วนใหญ่แล้วนักเคมีเชิงคำนวณมักจะเริ่มต้นด้วยการเตรียม Cartesian Coordinates ของชุดโมเลกุลที่ต้องการศึกษา เช่น โมเลกุลอินทรีย์ สารประกอบโลหะ ฯลฯ เมื่อเรามีชุดข้อมูลแล้วสิ่งที่เราควรทำต่อไปก็คือการวิเคราะห์ข้อมูลแบบเชิงลึก (Exploratory Data Analysis หรือ EDA) ซึ่งในขั้นตอนนี้เราควรจะต้องทำความสะอาดข้อมูลดิบด้วย เช่น การนำค่าผิดปกติออกไป การแปลงพารามิเตอร์บางค่าให้อยู่ในรูปแบบที่เหมาะสม โดยเราสามารถใช้เทคนิค Unsupervised ML เช่น Elliptic Envelope, Isolation Forest, และ Local Outlier Factor (LOF) ในการตรวจหาค่าผิดปกติได้

ลำดับถัดมาคือเราจะต้องเลือก Representation หรือ Descriptor ที่เราต้องการนำมาคำนวณมาคุณสมบัติต่าง ๆ ของโมเลกุล ซึ่งเรามักจะได้มาจากการคำนวณด้วยวิธีแบบดั้งเดิม โดยการคำนวณ Representation นั้นก็จะมีให้เลือกมากมาย ขึ้นอยู่กับความสอดคล้องของอินพุต (Feature ที่เราคำนวณ) กับเอาต์พุตที่เราต้องการจะทำนาย ซึ่งขั้นตอนนี้จะเป็นการสร้าง Feature Vector สำหรับการฝึกโมเดล ML นั่นเอง เมื่อเราได้ชุดข้อมูลที่มี Input Feature แล้ว อาจจะมีขั้นตอนที่เพิ่มเข้ามาเพื่อช่วยให้เราเข้าใจชุดข้อมูลได้มากขึ้น เช่น เราอาจจะใช้วิธีทางสถิติเข้ามาช่วยคำนวณค่าทางสถิติของชุดข้อมูลก่อนนำไปฝึกสอนโมเดล เช่น ค่าเฉลี่ย (Mean), ค่าเบี่ยงเบน (Deviation), การแจกแจงความถี่, (Frequency), ความแปรปรวน (Variance), และสหสัมพันธ์ของเพียร์สัน (Pearson Correlation) ซึ่งค่าทางสถิติเหล่านี้จะช่วยให้เข้าใจการกระจายตัวในชุดข้อมูลรวมไปถึงความสำคัญ (Importance) ของ Feature แต่ละตัวในชุดข้อมูล ซึ่งนำไปสู่การตัดสินใจและวิเคราะห์ว่า Feature ตัวไหนที่น่าจะมีผลต่อประสิทธิภาพของโมเดลเรามากที่สุด

เมื่อเราได้ชุดข้อมูลที่มีความเหมาะสมแล้ว ขั้นตอนต่อมาคือการเลือกเทคนิค ML ที่เราต้องการจะใช้สำหรับฝึกสอน ข้อแนะนำก็คือในช่วงเริ่มต้นเราอาจจะยังไม่ต้องไปใช้เทคนิคที่ซับซ้อนหรืออลังการมากก็ได้

เพราะการที่เราใช้เทคนิคที่ซับซ้อนหรือมีความสิ้นเปลืองสูงตั้งแต่แรกนั้นอาจจะไม่ได้การันตีว่าเราจะได้โมเดลที่ดีเสมอไปและยังเสียเวลาอีกด้วย ดังนั้นการเลือกใช้เทคนิคง่าย ๆ เช่น Ridge Regression ในช่วงเริ่มต้นก็อาจจะทำให้เรามีโมเดล MLP ที่มีประสิทธิภาพมาก ๆ แล้วก็ได้ นอกจากนี้ผู้เขียนมักจะพบเห็นผู้ที่เพิ่งเริ่มศึกษา ML หลาย ๆ คนที่เริ่มฝึกสอนโมเดลด้วย Deep Neural Network โดยการใช้เทคนิคขั้นสูงกับข้อมูลที่มีความเรียบง่ายซึ่งตรงจุดนี้บางครั้งมันก็มีความไม่เหมาะสมระหว่างเทคนิคและข้อมูลที่เรามี ซึ่งการทำแบบนี้เราอาจจะเรียกว่าขี่ช้างจับตั๊กแตน อย่างไรก็ตามประเด็นการเลือกใช้เทคนิค ML นี้เป็นเพียงความเห็นของผู้เขียนซึ่งท้ายที่สุดแล้วก็ขึ้นอยู่กับวิจารณญานของแต่ละคนครับ

เมื่อเราเลือกเทคนิค ML ได้แล้ว ขั้นตอนต่อมาก็คือการสร้างโมเดลและฝึกสอนกับ Training Set โดยในขั้นตอนนี้เราอาจจะลองสร้างหลาย ๆ โมเดลและทำการปรับ Hyperparameter ไปด้วยก็ได้ (ควรเปลี่ยนค่าอย่างเป็นระบบและให้สอดคล้องกัน) นอกจากนี้เราอาจจะทำ Validation เพิ่มด้วยก็ได้เพื่อเป็นการทดสอบความสามารถของเทคนิค ML ที่เราได้เลือกมาใช้ว่ามีประสิทธิภาพอย่างไร เมื่อเราได้โมเดลที่ถูกฝึกสอนมาแล้ว ลำดับต่อมาก็คือการทำนายหรือพยากรณ์คำตอบนั่นเอง โดยเราควรจะต้องมาวิเคราะห์ถึงปัจจัยที่ส่งผลต่อการทำนาย พยายามหาความเชื่อมโยงระหว่าง Feature ที่เลือกใช้, เทคนิค ML และ Hyperparameters ต่าง ๆ ที่เราได้กำหนดและลองปรับเปลี่ยนค่าในระหว่างการฝึกสอนโมเดล เมื่อเราได้โมเดลที่ถูกฝึกสอนมาอย่างดีและมีประสิทธิภาพที่อยู่ในเกณฑ์ที่ยอมรับได้แล้วนั้น เราก็จะมีโมเดลที่พร้อมจะไปใช้งานจริงครับ

11.2 การเลือกโมเดลที่เหมาะสม

ปัจจัยที่เราควรพิจารณาในการเลือกอัลกอริทึม ML สำหรับทำการฝึกสอนโมเดลมีดังนี้

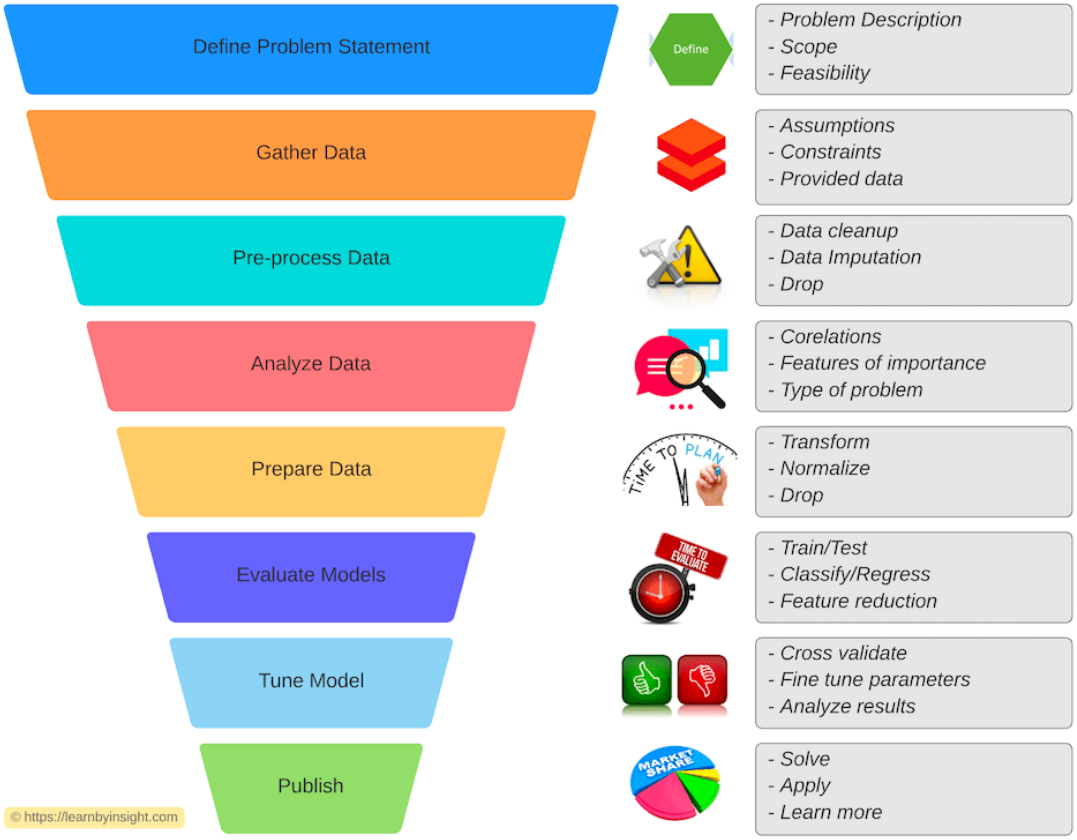
- การตีความของอัลกอริทึม (Interpretability): เมื่อเราพูดถึง Interpretability ของอัลกอริทึม ML หมายความว่าเรากำลังพูดถึงความสามารถของอัลกอริทึมในการอธิบายการทำนายผล ซึ่งเราเรียกอัลกอริทึมที่ขาดความสามารถในการอธิบายดังกล่าวว่ากล่องดำ (Black-box)

  อัลกอริทึมอย่างเช่น KNN มี Interpretability ที่สูงมากผ่านการทำ Feature Importance และอัลกอริทึมอย่างเช่นโมเดลเชิงเส้นมี Interpretability สูงผ่านการคำนวณน้ำหนัก (Weights) ของแต่ละ Feature

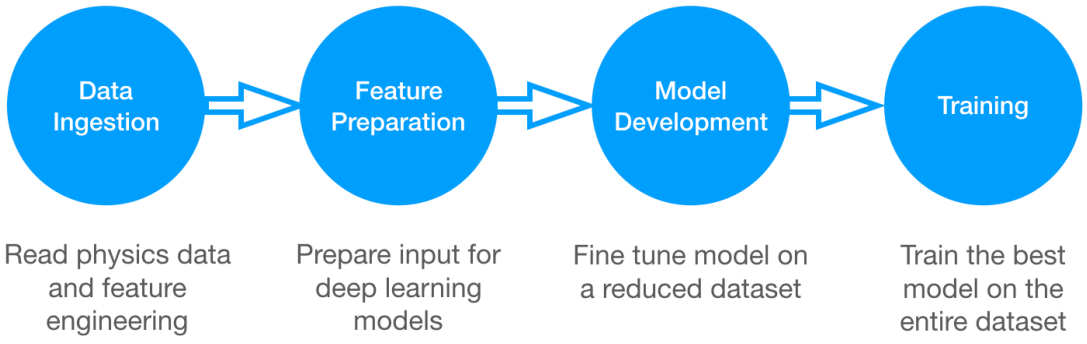
- จำนวนของข้อมูลในชุดข้อมูลและจำนวน Features: ขนาดของชุดข้อมูลมีผลอย่างมากต่อการเลือกอัลกอริทึม ML เช่น อัลกอริทึม Neural Network นั้นทำงานได้ดีกับชุดข้อมูลที่มีขนาดใหญ่และมีปริมาณ Feature ที่เยอะ

- ชนิดของข้อมูล (Data Format): ปกติแล้วชนิดของข้อมูลนั้นมีหลากหลายรูปแบบ โดยชนิดของข้อมูลส่วนใหญ่จะเป็นแบบหมวดหมู่ (Categorical) และแบบตัวเลข (Numerical) ซึ่งบางชุดข้อมูลอาจจะมีแค่หมวดหมู่หรืออาจจะมีแค่ตัวเลข และอาจจะมีทั้งสองแบบก็ได้

- ความเป็นเส้นตรงของข้อมูล (Linearity of Data): การเข้าใจความเป็นเชิงเส้นของข้อมูลนั้นสำคัญมากเพราะว่าจะเป็นการช่วยให้เราสามารถวิเคราะห์ขอบเขตการตัดสินใจและเส้น Regression Line



ภาพ 11.2 แนวทางและขั้นตอนการสร้างโมเดลปัญญาประดิษฐ์แบบที่ 1 (เครดิตภาพ: https://learnbyinsight.com)



ภาพ 11.3 แนวทางและขั้นตอนการสร้างโมเดลปัญญาประดิษฐ์แบบที่ 2 (เครดิตภาพ: https://cds.cern.ch)

ตัวอย่างเช่นถ้าข้อมูลสามารถถูกแยกได้แบบเชิงเส้นหรือเส้นตรงเราก็สามารถใช้อัลกอริทึมแบบเชิงเส้นอย่างเช่น SVM, Linear Regression หรือ Logistic Regression ได้ แต่ถ้าโมเดลมีความไม่เป็นเชิงเราก็ควรใช้อัลกอริทึมอย่างเช่น Neural Network

- ระยะเวลาในการฝึกสอนโมเดล (Training Time): ระยะเวลาในการฝึกสอนโมเดลของแต่ละอัลกอริทึมนั้นต่าง อัลกอริทึมพื้นฐานเช่น KNN และ Logistic Regression นั้นใช้ระยะเวลาไม่นานมากเมื่อเทียบกับอัลกอริทึมอื่น ส่วนอัลกอริทึมที่มีความซับซ้อนเช่น Neural Network นั้นมักจะใช้เวลานานในการฝึกสอน นอกจากนี้ยังมีอัลกอริทึมบางประเภทที่ระยะเวลาในการฝึกสอนนั้นขึ้นอยู่กับจำนวนของ CPU Cores ที่ใช้ในการรัน เช่น Random Forest

- ระยะเวลาในการทำนาย (Prediction Time): นอกเหนือจากระยะเวลาในการฝึกสอนเราก็ควรคำนึงถึงระยะเวลาที่ใช้ในการทำนายด้วย โดยอัลกอริทึมอย่างเช่น Linear Regression, Logistic Regression และ Neural Network บางประเภทนั้นสามารถทำนายได้อย่างรวดเร็ว อย่างไรก็ตามอัลกอริทึมอย่างเช่น KNN หรือการทำ Ensemble Model นั้นใช้เวลาเยอะกว่ามากในการทำนาย

- หน่วยความจำที่ต้องใช้ (Memory Requirements): ถ้าหากชุดข้อมูลของเรามีขนาดใหญ่ก็จะทำให้พารามิเตอร์หรือตัวแปรที่ถูกสร้างขึ้นและถูกคำนวณในระหว่างการฝึกสอนโมเดลนั้นเยอะตามไปด้วย ส่งผลให้ปริมาณของหน่วยความจำ (Memory) นั้นต้องเพียงพอสำหรับการฝึกสอนโมเดล ดังนั้นถ้าหากเรามีข้อมูลที่มีขนาดใหญ่ การเลือกอัลกอริทึม ML ที่เหมาะสมและไม่ซับซ้อนมากก็จะทำให้ไม่มีปัญหาเกี่ยวกับ Memory

นอกจากนี้แล้วผู้อ่านขออธิบายเพิ่มเติมในส่วนของการเลือกเคอร์เนล (Kernel) สำหรับการทำ Regression โดยเคอร์เนลที่มักถูกใช้นั้นมีดังต่อไปนี้

Linear Kernel :	$K(x_i, x_j) = x_i \cdot x_j$
Polynomial Kernel :	$K(x_i, x_j; a, b) = (x_i \cdot x_j + a)^b$
Gaussian Kernel :	$K(x_i, x_j; w, \sigma) = \exp\left(-\frac{\lvert x_i - x_j\rvert^2}{2\sigma^2}\right)$
Laplacian Kernel :	$K(x_i, x_j; w, \gamma) = \exp\left(-\lvert x_i - x_j\rvert\right)$

การเลือก Kernel ที่เหมาะสมนั้นจริง ๆ แล้วขึ้นกับอัลกอริทึมของ ML ที่เราจะเลือกใช้ด้วย โดย Kernel ที่สามารถนำไปใช้แล้วทำให้เกิดการเรียนรู้ Regression ได้อย่างดีเยี่ยมก็คือ Gaussian Kernel นั่นก็เพราะว่ามีคุณสมบัติ อย่างเช่น ความสมมาตรและความต่อเนื่องตลอดช่วงของฟังก์ชัน

11.3 การทำนายพลังงานรวมของโมเลกุล

การคำนวณพลังงานรวมเชิงอิเล็กทรอนิกส์ของโมเลกุลถือได้ว่าเป็นหนึ่งในการคำนวณพื้นฐานที่สุดในการศึกษาความเสถียรของโมเลกุลเลยก็ว่าได้ โดยเราจะมาดูการเขียนโค้ดสำหรับการทำนายพลังงานของโมเลกุลจากชุดข้อมูล QM9 โดยใช้ Feature ที่เป็น Coulomb Matrix (CM) ซึ่งเราได้ดูรายละเอียดการคำนวณ CM รวมถึงการวิเคราะห์ไปแล้วในหัวข้อที่ 10.7 โดยโมเดล ML ที่ผู้เขียนเลือกคือ Molecular Kernel สำหรับการทำ Regression ซึ่งสามารถเรียกใช้ได้จากไลบรารี่ QML

ไอเดียของ Molecular Kernel ก็คือเริ่มต้นด้วยการกำหนด Kernel ขึ้นมาก่อน ดังนี้

$$\alpha = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y} \tag{11.1}$$

เป้าหมายของเราก็คือการหาค่าของพารามิเตอร์ α ซึ่งเป็นสัมประสิทธิ์ที่เหมาะสมที่สุด (Best Fit for Regression) นั่นหมายความว่าเราจะต้องทำการแก้สมการที่ (11.1) ซึ่งสามารถทำได้โดยใช้ Cholesky Decomposition[1]

เราเริ่มต้นเขียนโค้ดสำหรับแบ่งชุดข้อมูลออกเป็นชุดข้อมูลสำหรับการฝึกสอนและการทดสอบตามลำดับ

```
from sklearn.model_selection import train_test_split

X_cm_train, X_cm_test, Y_cm_train, Y_cm_test =
   train_test_split(cm, target, test_size=0.2, random_state=42)
```

ตามด้วยการกำหนด Kernel ซึ่งเราจะใช้ Gaussian Kernel

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right) \tag{11.2}$$

พร้อมกับสร้างลิสต์ของ σ หลาย ๆ ค่า ซึ่งเราจะทำการหาค่าที่เหมาะสมที่สุดโดยการคำนวณระยะห่างแบบคู่ (Pairwise) ระหว่าง CM เพื่อเพิ่มความเร็วในการคำนวณเพราะว่าเราจะทำการแก้ Kernel หลาย ๆ อันพร้อมกับเปลี่ยนค่า σ หลาย ๆ ค่าไปพร้อม ๆ กัน

```
# Generates a list of different sigma values
sigmas = np.arange(100,5000,500)
test_maes = []

```

[1]การแยกส่วนประกอบโชเลสกี (Cholesky Decomposition เป็นวิธีการแยกเมทริกซ์ของเมทริกซ์สมมาตรที่เป็นบวกแน่นอน (Symmetric Positive-definite Matrix) ไปเป็นเมทริกซ์สามเหลี่ยมล่าง (Lower Triangular Matrix, L) และเมทริกซ์สลับเปลี่ยนของเมทริกซ์สามเหลี่ยมล่าง (L^T)

```
# Compute the pairwise distances between cm representations
dm_train_train = sklearn.metrics.pairwise_distances(
    X_cm_train,
    X_cm_train,
    n_jobs=-1
    )
dm_train_test = sklearn.metrics.pairwise_distances(
    X_cm_train,
    X_cm_test,
    n_jobs=-1
    )
```

หลังจากนั้นจึงเริ่มทำการหาค่า σ โดยใช้ Loop

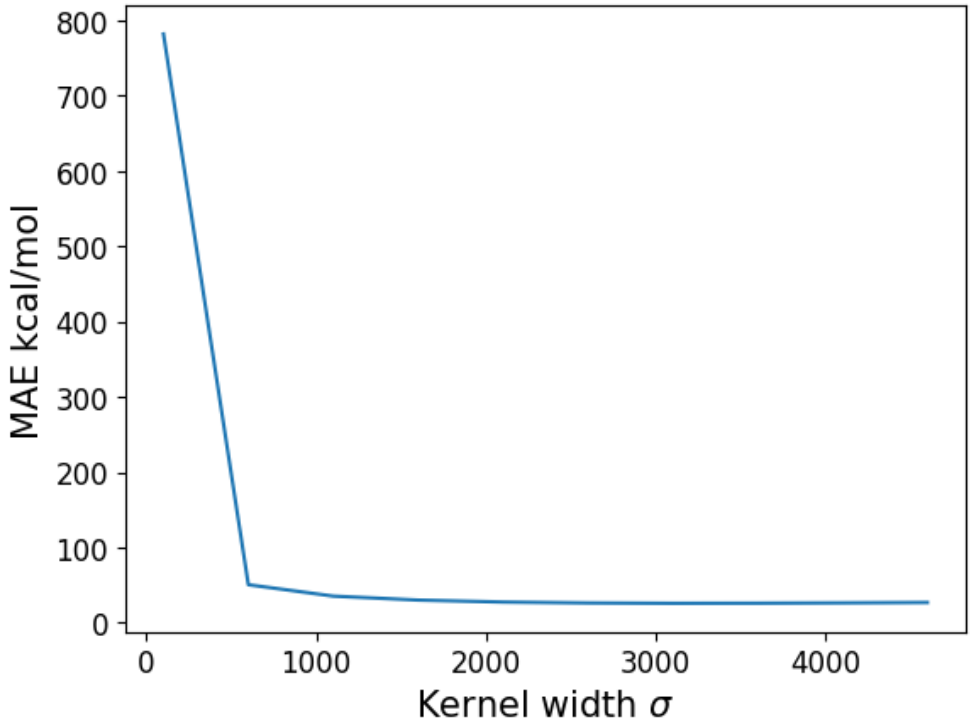
```
for sigma in sigmas:
    # Step 1
    K_cm = np.exp( - dm_train_train ** 2 / (2 * sigma ** 2))
    # Step 2
    K_cm[np.diag_indices_from(K_cm)] += 1e-8
    # Step 3
    alpha_cm = cho_solve(K_cm, Y_cm_train)
    # Step 4
    K_cm_test = np.exp( - dm_train_test ** 2 / (2 * sigma ** 2))
    # Step 5
    Y_cm_predicted = np.dot(K_cm_test.T, alpha_cm)
    # Step 6
    test_MAE = np.mean(np.abs(Y_cm_predicted - Y_cm_test))
    test_maes.append(test_MAE)
```

โดยสิ่งที่เกิดขึ้นภายใน Loop ของโค้ดด้านบนมีขั้นตอนดังนี้

1. เราเริ่มด้วยการคำนวณ Kernel ซึ่งถูกกำหนดด้วยสมการ (11.1)
2. เพิ่มค่า λ ที่มีค่าน้อย ๆ เข้าไปในสมาชิกแนวทแยงของ Kernel Matrix
3. แก้สมการโดยใช้ Cholesky Decomposition
4. คำนวณ Kernel Matrix ระหว่างชุดข้อมูลที่ใช้ฝึกสอนกับชุดที่ใช้ทดสอบโดยใช้ค่า σ เดียวกัน
5. ทำการพยากรณ์หรือทำนายค่าพลังงานภายในของโมเลกุล
6. คำนวณค่าความคลาดเคลื่อน Mean Absolute Error (MAE)

เมื่อทำ Regression เสร็จแล้ว เราสามารถพล็อตกราฟเพื่อดูความสัมพันธ์ค่า σ ของ Kernel กับค่าความคลาดเคลื่อน MAE ได้ดังนี้

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.plot(sigmas, test_maes)
ax.set_ylabel('MAE kcal/mol', fontsize=15)
ax.set_xlabel('Kernel width $\sigma$', fontsize=15)
ax.tick_params(axis='both', which='major', labelsize=12)

plt.show()
```



ภาพ 11.4 ค่าความคลาดเคลื่อน MAE กับค่าความกว้างของ Kernel (σ)

นอกจากนี้เรายังสามารถทำการวิเคราะห์เพิ่มเติมได้ เช่น แสดงค่า σ ที่เหมาะสมที่สุด (ให้ MAE น้อยที่สุด)

```python
best_sigma = sigmas[np.argmin(test_maes)]
print(best_sigma)

# Output
3100
```

แล้วนำค่า σ ที่เหมาะสมที่สุดนี้ไปใช้ในการคำนวณ Gaussian Kernel ต่อไป ดังนี้

```
# Create Gaussian Kernel
K_cm = gaussian_kernel(X_cm_train, X_cm_train, sigma)

# Add a small lambda to the diagonal of the kernel matrix
K_cm[np.diag_indices_from(K_cm)] += 1e-8

# Use the built-in Cholesky-decomposition to solve
alpha_cm = cho_solve(K_cm, Y_cm_train)
print(alpha_cm)

# Output
[-1.72214025e+09 -1.37779221e+09 -6.40402006e+08 ...
   2.42741968e+09  5.94869282e+08
 -1.19747799e+09]
```

แล้วคำนวณต่อ Kernel ระหว่างชุดข้อมูลฝึกสอนกับชุดข้อมูลทดสอบได้โดยใช้ค่า σ เดียวกัน

```
# Calculate a kernel matrix between test and training data,
   using the same sigma
K_cm_test = gaussian_kernel(X_cm_test, X_cm_train, sigma)

# Make the predictions
Y_cm_predicted = np.dot(K_cm_test, alpha_cm)

# Calculate mean-absolute-error (MAE), the units are Hartree
print('MAE: ', np.mean(np.abs(Y_cm_predicted - Y_cm_test)),
   'kcal/mol')

# Output
MAE:  25.908959327933474 kcal/mol
```

ขั้นตอนสุดท้ายคือการพล็อต Correlation ระหว่างค่าอ้างอิง (Reference) กับค่าที่ได้จากการทำนาย (Prediction) และแสดง Histogram ของความคลาดเคลื่อนเพื่อตรวจสอบประสิทธิภาพของโมเดล

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(ncols=1, nrows=2, figsize=[6,8])

ax = axes[0]
ax.scatter(Y_cm_test, Y_cm_predicted, s=16)
ax.set_ylabel(
```

```
    'CM-KRR internal energies at 0 K \n prediction [kcal/mol]',
    fontsize=15
    )
ax.set_xlabel(
    'Internal energies at 0 K [kcal/mol]',
    fontsize=15
    )

ax = axes[1]
ax.hist(Y_cm_test - Y_cm_predicted, bins=50, range=[-100,100],
   density=True)
ax.set_ylabel('Histogram density', fontsize=15)
ax.set_xlabel(
    'CM-KRR internal energies at 0 K errors [kcal/mol]',
    fontsize=15
    )

plt.tight_layout()
plt.show()
```

11.4 การทำนายพื้นผิวพลังงานศักย์

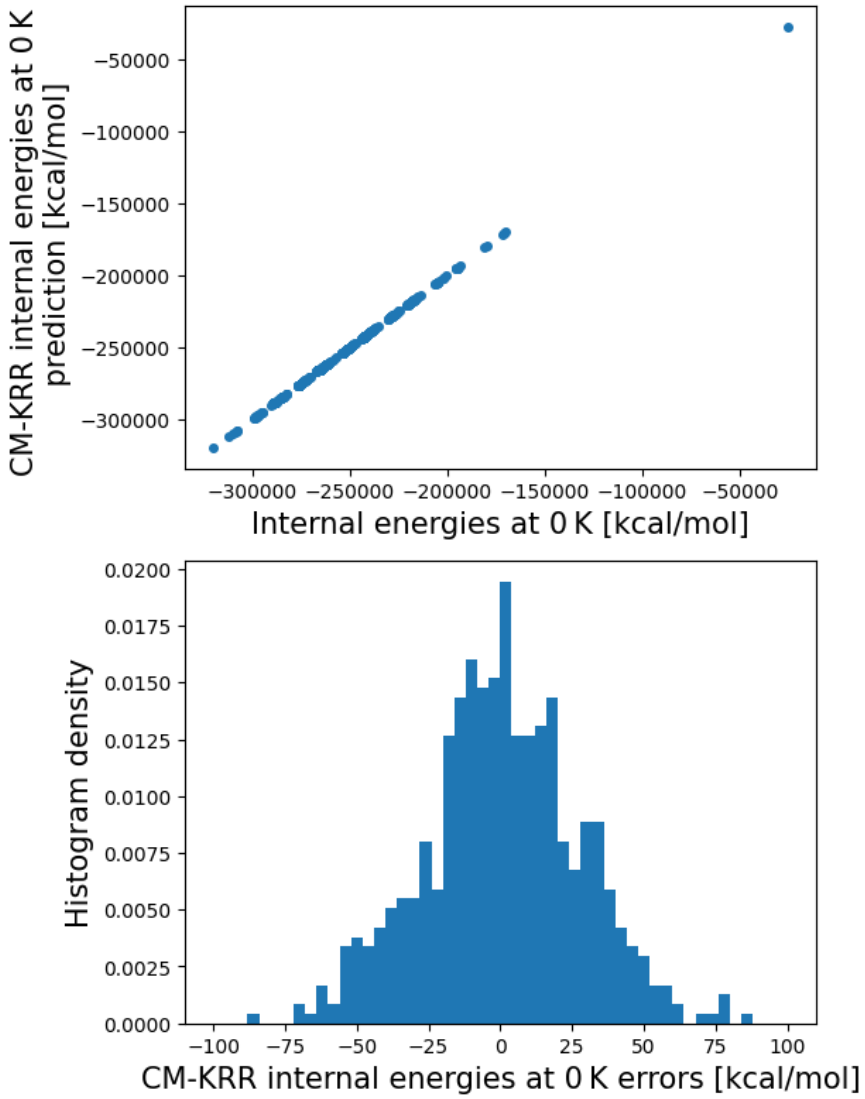
การทำนายพื้นผิวพลังงานศักย์ (Potential Energy Surface หรือ PES)

Machine Learning Potentials (MLP) แปลได้ตรงตัวคือ *การเรียนรู้พลังงานศักย์ของเครื่อง* ซึ่งเป็นอีกหนึ่งเครื่องมือสำคัญสำหรับการจำลองในระดับอะตอม (Atomistic Simulation) โดยเฉพาะการศึกษาพลังงานศักย์ของโมเลกุล โดยมีงานวิจัยที่ได้มีการพัฒนาทั้ง Representation และอัลกอริทึม ML[152,153,154] ตัวอย่างของงานวิจัยเฉพาะทางที่ใช้ MLP เช่น การศึกษาพลังงานศักย์ระหว่างอะตอมเพื่อเพิ่มความแม่นยำในการจำลองพลวัตเชิงโมเลกุล (Molecular Dynamics หรือ MD),[155,156,157,158] หรือการพัฒนา Force Field สำหรับการจำลองแบบดั้งเดิม (Classical MD)[159] และสำหรับการจำลองแบบเริ่มแรก (*ab initio* MD หรือ AIMD)[160,161,162]

MLP แบ่งออกได้เป็นสองประเภทตามรูปแบบของอัลกอริทึมของ ML คือแบบเคอร์เนล (Kernel-based Potentials) และแบบโครงข่ายประสาท (Neural Network-based Potentials)

11.4.1 การทำนายพลังงานศักย์ด้วยวิธีเชิงเคอร์เนล

เรามาดูรายละเอียดของ Kernel-based Potentials กันก่อน ซึ่งผู้เขียนขอยกมาให้ดู 3 วิธี



ภาพ 11.5 ซ้าย: Correlation ของค่าพลังงานภายในของโมเลกุลที่เป็นค่าอ้างอิงและค่าที่ได้จากการทำนาย, ขวา: Histrogram Density ของค่าคลาดเคลื่อน

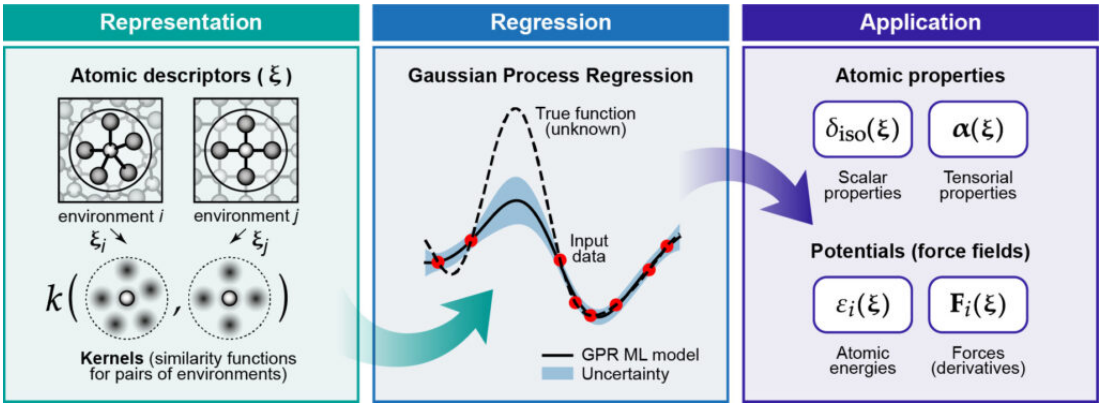
Gaussian Approximation Potentials (GAP) เป็นวิธีที่นำเสนอโดย Albert P. Bartók และคณะ ซึ่งตีพิมพ์บทความวิจัยเรื่อง “Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons” ในวารสาร Physical Review Letters[28] GAP เป็นวิธีคำนวณที่ใช้สำหรับการทำนายพลังงานและแรงของแต่ละอะตอมภายในโมเลกุลโดยที่ GAP จะทำการเรียนรู้ค่าทั้งสองจาก PES ซึ่งมักจะได้มาจากการคำนวณด้วยวิธีง่าย ๆ ทั่วไป เช่น DFT

GAP นั้นเป็นการทำ Kernel Regression ของ PES แบบ Non-parametric ซึ่งเป็นการประมาณค่าของพลังงานแบบเฉพาะที่ (Local Energy) ดังนี้

$$\bar{\epsilon}_* = \delta^2 \sum_{s=1}^{N_s} \alpha_s k(*, s) \tag{11.3}$$

โดยที่ $*$ หมายถึง Environment ของอะตอมที่เราต้องการทำนาย, δ คือพารามิเตอร์ที่กำหนดสเกลของพลังงาน, s คือจำนวนของ Configuration (ขนาดของ Training Set), α_s คือสัมประสิทธิ์ของการ Fitting และ $k(*, s)$ คือเคอร์เนล

โดยทริคของ GAP คือทำการแปลง PES ที่ปกติจะเป็นฟังก์ชันแบบไม่เป็นเชิงเส้นที่ขึ้นกับตำแหน่งของอะตอม $(E = E(\{\boldsymbol{r}_i\}))$ ให้กลายเป็นฟังก์ชันแบบเชิงเส้นที่ขึ้นกับเคอร์เนลแทน พอเรามีฟังก์ชันแบบเชิงเส้นแล้วเราก็สามารถทำ Linear Regression เพื่อหาค่า Parameter ต่าง ๆ ที่ใช้ในการ Fit ได้นั่นเอง โดยเคอร์เนลในที่นี้คือการวัดความคล้ายคลึงกันระหว่าง Feature ของอะตอมแต่ละตัวในโมเลกุล (Atomic Environment)



ภาพ 11.6 ขั้นตอนการคำนวณ Representation และการใช้ Gaussian Approximation Potentials ในการทำนายพลังงานและแรง (เครดิตภาพ: *Chem. Rev.*, **2021**, 121, 10073[155])

ขั้นตอนของการทำ GAP นั้นสามารถดูได้ตามภาพที่ 11.6 โดยมีขั้นตอนดังต่อไปนี้

1. เริ่มต้นเราทำการคำนวณหา Representation ก่อนโดยใช้ Atomic Descriptor ซึ่งเป็นข้อมูลหรือ Feature ในระดับอะตอม

2. หลังจากนั้นทำ Gaussian Process Regression ซึ่งเป็นหัวใจของ GAP

3. เราจะได้ฟังก์ชันที่สามารถนำไปใช้ในการทำนายพลังงานและแรงเชิงอะตอมได้ ซึ่งสุดท้ายก็สามารถนำมาหาพลังงานรวมของโมเลกุลได้

เนื่องจากว่า GAP ไม่ได้ขึ้นกับฟังก์ชันที่มีรูปแบบตายตัวเหมือนกับ Harmonic Potentials ดังนั้น GAP จึงมีความยืดหยุ่นต่อชุดข้อมูลที่ใช้ในการฝึกสอน สามารถอธิบายการเปลี่ยนแปลงทางเคมี เช่น การสร้างพันธะหรือการสลายพันธะ ซึ่งเป็นสิ่งที่สนามแรง (Force Field) แบบดั้งเดิมหรือทั่วไปนั่นทำได้ยาก[117]

Moment Tensor Potentials (MTP)[163]

Spectral Neighbor Analysis Potentials (SNAP)[164,165,166,167]

11.4.2 การทำนายพลังงานศักย์ด้วยวิธีเชิงโครงข่ายประสาท

High-dimensional Neural Network Potentials (HDNNP)[168] เป็นสถาปัตยกรรม Neural Network ที่ได้รับความนิยมมากในการทำนายพลังงานหรือศักย์ของโมเลกุล ซึ่งโมเดล HDNNP นั้นมีชื่อเรียกอีกชื่อว่า Behler-Parrinello Neural Network (BPNN) แนวคิดของ BPNN นั้นก็คือการอธิบายพลังงานรวมของโมเลกุลโดยทำให้อยู่ในรูปของผลรวมเชิงเส้นของพลังงานย่อยของแต่ละอะตอมหรือ Atomic Contribution โดย Neural Network ที่ถูกสร้างขึ้นมาและใช้ใน BPNN นั้นมีขนาดหรือจำนวนของหน่วยการเรียนรู้ที่สอดคล้องกับจำนวนอะตอมภายในโมเลกุล

ANAKIN-ME เรียกสั้น ๆ ว่า ANI (ชื่อเต็มคือ Accurate NeurAl networK engINe for Molecular Energies) เป็นโมเดล Neural Network ที่ถูกพัฒนาขึ้นมาโดยใช้ BPNN โดยโมเดลในตระกูลของ ANI นั้นมีด้วยกันหลายโมเดลซึ่งสามารถทำนายพลังงานของโมเลกุลได้เทียบเท่าหรือเทียบเคียงกับการคำนวณด้วยวิธีทางเคมีควอนตัมแบบดั้งเดิม เช่น วิธี Coupled Cluster

- ANI-1x[169]
- ANI-1ccx[170]
- ANI-2x[171,172]

11.5 การทำนายพลังงานการทำให้เกิดอะตอมและพลังงานของออร์บิทัล

พลังงานการทำให้เกิดอะตอม (Atomization Energy) และพลังงานของออร์บิทัล HOMO และ LUMO เป็นคุณสมบัติของโมเลกุลที่นักเคมีให้ความสนใจเพราะว่าทั้งสองพลังงานนี้เป็นสิ่งสำคัญที่ช่วยให้เราเข้าใจความเสถียรของโมเลกุลและออร์บิทัลที่เกี่ยวข้องกับการสร้างพันธะในปฏิกิริยาเคมี

ตาราง 11.1 ค่าความคลาดเคลื่อน (Mean Absolute Error หรือ MAE) ของพลังงานการทำให้เกิดอะตอม (U_0) ในหน่วย kcal/mol และพลังงานของออร์บิทัลชั้นที่สูงสุดที่มีอิเล็กตรอน (HOMO) และชั้นที่ต่ำที่สุดที่ไม่มีอิเล็กตรอน (LUMO) ในหน่วย eV ที่ทำนายด้วยอัลกอริทึม Machine Learning (ML) เช่น Kernel Ridge Regression (KRR), Elastic Net (EN), Gaussian Process Regression (GPR)

โมเดล ML	U_0	HOMO	LUMO	**อ้างอิง**
KRR/CM[1]	9.9	-	-	Rupp และคณะ[105]
Multilayer NN/Random CM[1]	3.5	-	-	Montavon และคณะ[173]
Multitask NN/Random CM[2]	3.7	0.15	0.12	Montavon และคณะ[141]
KRR/Random CM[1]	3	-	-	Hansen และคณะ[108]
KRR/BoB[1]	1.5	-	-	Hansen และคณะ[111]
KRR/BoB[2]	1.8	0.15	0.16	Huang และ von Lilienfeld[129]
KRR/BAML[2]	1.2	0.1	0.11	Huang และ von Lilienfeld[129]
KRR/REMatch-SOAP[2]	0.92	0.11	0.08	De และคณะ[119]
EN/CM[3]	21	0.34	0.63	Faber และคณะ[130]
EN/BoB[3]	13.9	0.28	0.52	Faber และคณะ[130]
KRR/CM[3]	3	0.13	0.18	Faber และคณะ[130]
KRR/BoB[3]	1.5	0.09	0.12	Faber และคณะ[130]
KRR/BAML[3]	1.2	0.09	0.12	Faber และคณะ[130]
KRR/HDAD[3]	0.6	0.07	0.08	Faber และคณะ[130]
KRR/MBTR[2]	0.6	-	-	Huo และ Rupp[131]
GPR/SOAP-GAP[2]	0.4	-	-	Bartók และคณะ[174]
KRR/SOAP multi-kernel[3]	0.14	-	-	Willatt และคณะ[175]
KRR/FCHL19[3]	0.25			Christensen และคณะ[176]
AML/amon[3]	<1.0			Huang และ von Lilienfeld[122]

[1] ชุดข้อมูล QM7
[2] ชุดข้อมูล QM7b
[3] ชุดข้อมูล QM9

ตารางที่ 11.1 และ 11.2 แสดงการเปรียบเทียบค่าความถูกต้องของโมเดล ML และ NN ในการทำนายพลังงาน โดยค่า Mean Absolute Error (MAE) ซึ่งเป็นค่าที่บอกความคลาดเคลื่อนนั้นแสดงให้เห็นว่าอัลกอ

ริทึม KRR สามารถทำนายค่าพลังงานทำให้เกิดอะตอมและพลังงาน HOMO และ LUMO ได้แม่นยำมากโดยมีค่าที่ใกล้เคียงกับความถูกต้องทางเคมี (1 kcal/mol) โดยใช้ BoB[129] และ BAML[130] เป็น Representation ในการฝึกสอน

ตาราง 11.2 ค่าความคลาดเคลื่อน (Mean Absolute Error หรือ MAE) ของพลังงานการทำให้เกิดอะตอม (U_0) ในหน่วย kcal/mol และพลังงานของออร์บิทัลชั้นที่สูงสุดที่มีอิเล็กตรอน (HOMO) และชั้นที่ต่ำที่สุดที่ไม่มีอิเล็กตรอน (LUMO) ในหน่วย eV ที่ทำนายด้วยอัลกอริทึม Neural Network (NN) เช่น Message-passing Neural Network (MPNN)

โมเดล NN	U_0	HOMO	LUMO	**อ้างอิง**
MPNN[1]	0.45	0.99	0.87	Gilmer และคณะ[177]
ANI-1a NN/ACSF[2]	<1.5	-	-	Smith และคณะ[169,170,171]
HDNN/w-ACSF[1]	<1.8	-	-	Gastegger และคณะ[124]
Multitask NN/CM[1]	44	0.38	0.63	Hou และคณะ[178]
SchNet NN[1]	0.32	0.04	0.03	Schütt และคณะ[179]
HIP-NN[1]	0.26	-	-	Lubbers และคณะ[180]
HDNN[1]	0.41	-	-	Unke และ Meuwly[181]
RNN/VAE[1]	-	0.16	0.16	Gómez-Bombarelli และคณะ[182]
PhysNet NN[1]	0.14	-	-	Unke และ Meuwly[183]
SchNOrb NN[3]	<0.046	<0.02	<0.1	Schütt และคณะ[184]

[1] ชุดข้อมูล QM9
[2] ชุดข้อมูล COMP6
[3] ชุดข้อมูล MD17

185,186,187,188,189,190

11.6 การจำลองสนามแรง

การทำนายหรือพยากรณ์แรง (Forces) และพลังงาน (Energies) ของโมเลกุลนั้นเรียกอีกอย่างหนึ่งว่าการสร้างโมเดล Machine Learning Force Field (MLFF) เริ่มต้นสมมติว่าเรามีชุดข้อมูลที่มี Feature Vector ซึ่งเขียนแทนด้วย $\boldsymbol{D}$ และมีอนุพันธ์แบบเวกเตอร์ (Divergence) เป็น $\nabla_{\boldsymbol{r}_i}\boldsymbol{D}$ และมีข้อมูลเพิ่มเติมคือพลังงาน E และแรง $\boldsymbol{F}$ ของระบบ (โมเลกุล) สำหรับการฝึกสอนเราจะสร้าง Neural Network (f) เพื่อสร้างโมเดลสำหรับการทำนายพลังงาน $(\hat{E} = f(\boldsymbol{D}))$[1] ซึ่งเราสามารถคำนวณแรงได้โดยตรงจากค่าติดลบของ Gradient ของพลังงานเทียบกับพิกัดตำแหน่งของอะตอมนั้น ๆ ดังนั้นแรงที่ได้จะเป็นปริมาณต่ออะตอม ยกตัวอย่างเช่นแรงของอะตอม i สามารถคำนวณได้จากสมการต่อไปนี้ (โดยใช้เวกเตอร์แบบแถว)

[1]เครื่องหมาย ^ (อ่านว่า “hat”) ที่อยู่ด้านบนของตัวแปรเป็นส่งที่บ่งบอกว่าค่าตัวแปรนั้นสิ่งที่เราทำนายออกมา

$$\hat{\boldsymbol{F}}_i = -\nabla_{\boldsymbol{r}_i} f(\boldsymbol{D}) \tag{11.4}$$

$$= -\nabla_{\boldsymbol{D}} f \cdot \nabla_{\boldsymbol{r}_i} \boldsymbol{D} \tag{11.5}$$

$$= -\begin{bmatrix} \frac{\partial f}{\partial D_1} & \frac{\partial f}{\partial D_2} & \cdots \end{bmatrix} \begin{bmatrix} \frac{\partial D_1}{\partial x_i} & \frac{\partial D_1}{\partial y_i} & \frac{\partial D_1}{\partial z_i} \\ \frac{\partial D_2}{\partial x_i} & \frac{\partial D_2}{\partial y_i} & \frac{\partial D_2}{\partial z_i} \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{11.6}$$

จากสมการ (11.4) นั้นเราอธิบายได้ว่า $\nabla_{\boldsymbol{D}} f$ เป็นค่าอนุพันธ์ของคำตอบของโมเดล ML ซึ่งจะเทียบกับ Descriptor $\boldsymbol{D}$ และ $\nabla_{\boldsymbol{r}_i} \boldsymbol{D}$ เป็นอนุพันธ์ของ Descriptor ที่เทียบกับตำแหน่งของอะตอมซึ่งตามที่เราได้ศึกษามาก่อนหน้านี้ว่า Neural Network นั้นจะให้คำตอบที่เป็นแบบ Analytical Solution

โค้ดดังต่อไปนี้คือการใช้ไลบรารี่ DScribe ในการสร้างชุดข้อมูลโดยมี SOAP เป็น Descriptor สำหรับคำนวณ Feature Vector แล้วฝึกสอนโมเดล Fully Connected Neural Network เพื่อเรียนรู้ MLFF ต่อไป

ขั้นตอนที่ 1 สร้างชุดข้อมูลของพลังงานและแรงของ Lennard-Jones

```python
import numpy as np
import ase
from ase.calculators.lj import LennardJones
import matplotlib.pyplot as plt
from dscribe.descriptors import SOAP

# Setting up the SOAP descriptor
soap = SOAP(
    species=["H"],
    periodic=False,
    rcut=5.0,
    sigma=0.5,
    nmax=3,
    lmax=0,
)

n_samples = 200
traj = []
n_atoms = 2
energies = np.zeros(n_samples)
forces = np.zeros((n_samples, n_atoms, 3))
r = np.linspace(2.5, 5.0, n_samples)

for i, d in enumerate(r):
```

```
    a = ase.Atoms('HH', positions = [[-0.5 * d, 0, 0], [0.5 * d,
    0, 0]])
    a.set_calculator(LennardJones(epsilon=1.0 , sigma=2.9))
    traj.append(a)
    energies[i] = a.get_total_energy()
    forces[i, :, :] = a.get_forces()

# Plot the energies to validate them
fig, ax = plt.subplots(figsize=(8, 5))
plt.subplots_adjust(left=0.1, right=0.95, top=0.95, bottom=0.1)
line, = ax.plot(r, energies)
plt.xlabel("Distance (Å)")
plt.ylabel("Energy (eV)")
plt.show()

# Create the SOAP desciptors and their derivatives for all
   samples.
# One center is chosen to be directly between the atoms.
derivatives, descriptors = soap.derivatives(
    traj,
    positions=[[[0, 0, 0]]] * len(r),
    method="analytical"
)

# Save to disk for later training
np.save("r.npy", r)
np.save("E.npy", energies)
np.save("D.npy", descriptors)
np.save("dD_dr.npy", derivatives)
np.save("F.npy", forces)
```

ขั้นตอนที่ 2 นำเข้าชุดข้อมูลและเตรียมอินพุตสำหรับฝึกสอนโมเดล

```
import numpy as np
import torch
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
torch.manual_seed(7)

# Load the dataset (We only have one SOAP center)
D_numpy = np.load("D.npy")[:, 0, :]
```

```
n_samples, n_features = D_numpy.shape
E_numpy = np.array([np.load("E.npy")]).T
F_numpy = np.load("F.npy")
dD_dr_numpy = np.load("dD_dr.npy")[:, 0, :, :, : ]
r_numpy = np.load("r.npy")

# Select equally spaced points for training
n_train = 30
idx = np.linspace(0, len(r_numpy) - 1, n_train).astype(int)
D_train_full = D_numpy[idx]
E_train_full = E_numpy[idx]
F_train_full = F_numpy[idx]
r_train_full = r_numpy[idx]
dD_dr_train_full = dD_dr_numpy[idx]

# Standardize input for improved learning.
# Fit is done only on training data,
# scaling is applied to both descriptors and
# their derivatives on training and test sets.

scaler = StandardScaler().fit(D_train_full)
D_train_full = scaler.transform(D_train_full)
D_whole = scaler.transform(D_numpy)
dD_dr_whole = dD_dr_numpy / scaler.scale_[None, None, None, :]
dD_dr_train_full = dD_dr_train_full / scaler.scale_[None, None,
    None, :]

# Calculate the variance of energy and force values for
# the training set. These are used to balance their
# contribution to the MSE loss
var_energy_train = E_train_full.var()
var_force_train = F_train_full.var()

# Subselect 20% of validation points for early stopping.
D_train, D_valid, E_train, E_valid, F_train, F_valid,
    dD_dr_train, dD_dr_valid = train_test_split(
     D_train_full,
     E_train_full,
     F_train_full,
     dD_dr_train_full,
     test_size=0.2,
     random_state=7,
)
```

```
52
53 # Create tensors for pytorch
54 D_whole = torch.Tensor(D_whole)
55 D_train = torch.Tensor(D_train)
56 D_valid = torch.Tensor(D_valid)
57 E_train = torch.Tensor(E_train)
58 E_valid = torch.Tensor(E_valid)
59 F_train = torch.Tensor(F_train)
60 F_valid = torch.Tensor(F_valid)
61 dD_dr_train = torch.Tensor(dD_dr_train)
62 dD_dr_valid = torch.Tensor(dD_dr_valid)
```

ขั้นตอนที่ 3 สร้างโมเดลและกำหนด Loss Function

```
1  class FFNet(torch.nn.Module):
2      """A simple feed-forward network with one hidden layer,
       randomly
3      initialized weights, sigmoid activation and a linear output
       layer.
4      """
5      def __init__(self, n_features, n_hidden, n_out):
6          super(FFNet, self).__init__()
7          self.linear1 = torch.nn.Linear(n_features, n_hidden)
8          torch.nn.init.normal_(self.linear1.weight, mean=0,
       std=1.0)
9          self.sigmoid = torch.nn.Sigmoid()
10         self.linear2 = torch.nn.Linear(n_hidden, n_out)
11         torch.nn.init.normal_(self.linear2.weight, mean=0,
       std=1.0)
12
13     def forward(self, x):
14         x = self.linear1(x)
15         x = self.sigmoid(x)
16         x = self.linear2(x)
17
18         return x
19
20
21 def energy_force_loss(E_pred, E_train, F_pred, F_train):
22     """Custom loss function that targets both energies and
       forces.
23     """
24     energy_loss = torch.mean((E_pred - E_train)**2) /
```

```
    var_energy_train
    force_loss = torch.mean((F_pred - F_train)**2) /
    var_force_train
    return energy_loss + force_loss

# Initialize model
model = FFNet(n_features, n_hidden=5, n_out=1)

# The Adam optimizer is used for training the model parameters
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)
```

ขั้นตอนที่ 4 ฝึกสอนโมเดล Neural Network โดยใช้ PyTorch

```
n_max_epochs = 5000
batch_size = 2
patience = 20
i_worse = 0
old_valid_loss = float("Inf")
best_valid_loss = float("Inf")

# We explicitly require that the gradients should be
# calculated for the input variables. PyTorch will
# not do this by default as it is typically not needed.
D_valid.requires_grad = True

# Epochs
for i_epoch in range(n_max_epochs):
    # Batches
    permutation = torch.randperm(D_train.size()[0])
    for i in range(0, D_train.size()[0], batch_size):
        indices = permutation[i:i+batch_size]
        D_train_batch, E_train_batch = D_train[indices],
    E_train[indices]
        D_train_batch.requires_grad = True
        F_train_batch, dD_dr_train_batch = F_train[indices],
    dD_dr_train[indices]

        # Predict energies from the descriptor input
        E_train_pred_batch = model(D_train_batch)

        # Get derivatives of model output w.r.t. input variables.
        df_dD_train_batch = torch.autograd.grad(
            outputs=E_train_pred_batch,
```

```
            inputs=D_train_batch,
            grad_outputs=torch.ones_like(E_train_pred_batch),
            create_graph=True,
        )[0]

        # Get derivatives of input variables (=descriptor)
        # w.r.t. atom positions = forces
        F_train_pred_batch = -torch.einsum('ijkl,il->ijk', dD_dr_train_batch, df_dD_train_batch)

        # Zero gradients, perform a backward pass,
        # and update the weights.
        # D_train_batch.grad.data.zero_()
        optimizer.zero_grad()
        loss = energy_force_loss(E_train_pred_batch, E_train_batch, F_train_pred_batch, F_train_batch)
        loss.backward()
        optimizer.step()

    # Check early stopping criterion and save best model
    E_valid_pred = model(D_valid)
    df_dD_valid = torch.autograd.grad(
        outputs=E_valid_pred,
        inputs=D_valid,
        grad_outputs=torch.ones_like(E_valid_pred),
    )[0]
    F_valid_pred = -torch.einsum('ijkl,il->ijk', dD_dr_valid, df_dD_valid)
    valid_loss = energy_force_loss(E_valid_pred, E_valid, F_valid_pred, F_valid)
    if valid_loss < best_valid_loss:
        # print("Saving at epoch {}".format(i_epoch))
        torch.save(model.state_dict(), "best_model.pt")
        best_valid_loss = valid_loss

    if valid_loss >= old_valid_loss:
        i_worse += 1
    else:
        i_worse = 0

    if i_worse > patience:
        print("Early stopping at epoch {}".format(i_epoch))
        break
```

```
    old_valid_loss = valid_loss

    if i_epoch % 500 == 0:
        print("  Finished epoch: {} with loss:
    {}".format(i_epoch, loss.item()))
```

ขั้นตอนที่ 5 ประเมินโมเดลและวิเคราะห์พลังงานและแรงที่ทำนายได้

```
model.load_state_dict(torch.load("best_model.pt"))
model.eval()

# Calculate energies and force for the entire range
E_whole = torch.Tensor(E_numpy)
F_whole = torch.Tensor(F_numpy)
dD_dr_whole = torch.Tensor(dD_dr_whole)
r_whole = r_numpy
D_whole.requires_grad = True
E_whole_pred = model(D_whole)
df_dD_whole = torch.autograd.grad(
    outputs=E_whole_pred,
    inputs=D_whole,
    grad_outputs=torch.ones_like(E_whole_pred),
)[0]

F_whole_pred = -torch.einsum('ijkl,il->ijk', dD_dr_whole,
   df_dD_whole)
```

ขั้นตอนที่ 6 พลอตค่าพลังงาน, แรง, และกราฟการฝึกสอนโมเดล

```
# Plot energies for the whole range
E_whole_pred = E_whole_pred.detach().numpy()
E_whole = E_whole.detach().numpy()
order = np.argsort(r_whole)
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,
   10))
ax1.plot(r_whole[order], E_whole[order], label="True",
   linewidth=3, linestyle="-")
ax1.plot(r_whole[order], E_whole_pred[order], label="Predicted",
   linewidth=3, linestyle="-")
ax1.set_ylabel('Energy', size=15)
mae_energy = mean_absolute_error(E_whole_pred, E_whole)
ax1.text(0.95, 0.5, "MAE: {:.2} eV".format(mae_energy), size=16,
   horizontalalignment='right', verticalalignment='center',
```

```
    transform=ax1.transAxes)

# Plot forces for whole range
F_x_whole_pred = F_whole_pred.detach().numpy()[order, 0, 0]
F_x_whole = F_whole[:, 0, 0][order]
ax2.plot(r_whole[order], F_x_whole, label="True", linewidth=3,
    linestyle="-")
ax2.plot(r_whole[order], F_x_whole_pred, label="Predicted",
    linewidth=3, linestyle="-")
ax2.set_xlabel('Distance', size=15)
ax2.set_ylabel('Forces', size=15)
mae_force = mean_absolute_error(F_x_whole_pred, F_x_whole)
ax2.text(0.95, 0.5, "MAE: {:.2} eV/Å".format(mae_force),
    size=16, horizontalalignment='right',
    verticalalignment='center', transform=ax2.transAxes)

# Plot training points
F_x_train_full = F_train_full[:, 0, 0]
ax1.scatter(r_train_full, E_train_full, marker="o", color="k",
    s=20, label="Training points", zorder=3)
ax2.scatter(r_train_full, F_x_train_full, marker="o", color="k",
    s=20, label="Training points", zorder=3)

# Show plot
ax1.legend(fontsize=12)
plt.subplots_adjust(left=0.08, right=0.97, top=0.97,
    bottom=0.08, hspace=0)
plt.show()
```

จากตัวอย่างโค้ดด้านบนทั้ง 6 ขั้นตอนนั้นเป็นวิธีการใช้ ML ในการสร้างชุดข้อมูลและฝึกสอนโมเดลเพื่อสร้างโมเดล Force Field ที่สามารถนำไปใช้งานต่อได้จริงในการจำลองเชิงโมเลกุลต่อไป เช่น นำไปใช้กับ Molecular Dynamics เพื่อเพิ่มความเร็วในการคำนวณ

นอกจากนี้ยังมีโมเดล ML ที่น่าสนใจที่ถูกพัฒนาขึ้นมาเพื่อเรียนรู้ Force Field โดยเฉพาะ ดังนี้

1. FFLUX[191]

2. TensorMol[192]

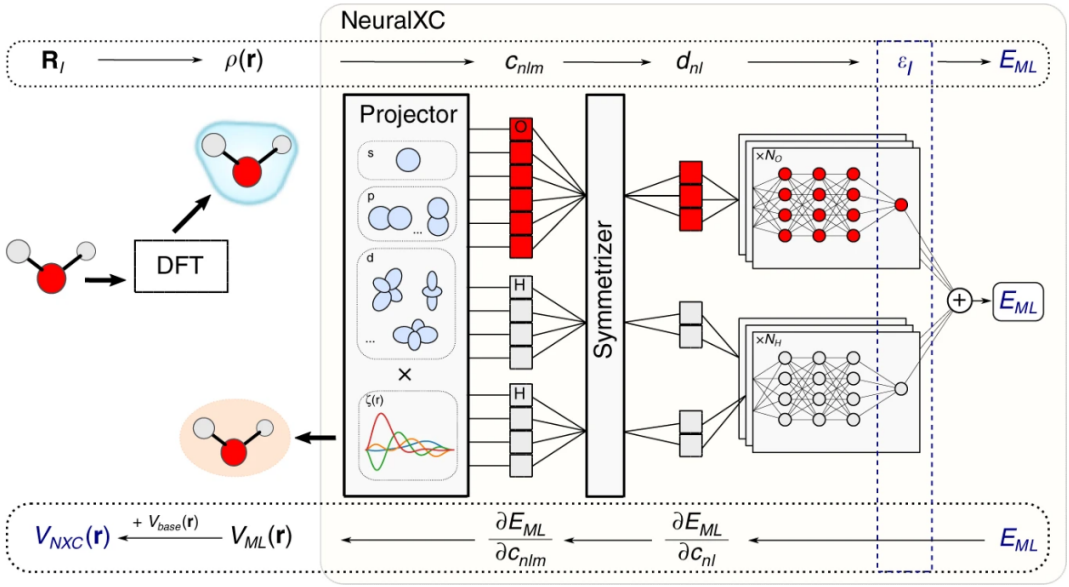
11.7 การทำนายพลังงานสหสัมพันธ์ของอิเล็กตรอน

> ผู้อ่านอยากมี Exchange-Correlation Functional ที่ใช้อธิบายพลังงานอันตรกิริยาระหว่างอิเล็กตรอนสำหรับ DFT เป็นของตัวเองไหมครับ?

เหตุผลที่ผู้เขียนเริ่มต้นหัวข้อนี้ด้วยคำถามนี้ก็เพราะว่าเราสามารถใช้ ML สร้าง Exchange-Correlation หรือ XC Functional ของเราเองได้นั่นเอง โดยไม่ได้เป็นการหลอกลวงผู้อ่านแต่อย่างใดเพราะสามารถทำได้จริง อย่างที่ทราบกันดีว่าในการคำนวณ DFT นั้นเราจะต้องทำการเลือก XC Functional ที่ต้องการใช้ซึ่งปกติแล้วเราก็ลองผิดลองถูกกันไป[1] ถ้า XC Functional อันไหนให้ผลการคำนวณที่แม่นยำและใกล้เคียงกับผลการทดลองมากที่สุดก็ใช้อันนั้นไป เพื่อแก้ปัญหาของการที่ XC Functional แบบดั้งเดิมนั้นไม่มีความสามารถในการส่งต่อความสามารถในการเรียนรู้หรือการนำไปใช้ได้กับระบบที่หลากหลาย ในปัจจุบันได้มีงานวิจัยที่พัฒนาอัลกอริทึม ML สำหรับการเรียนรู้ XC Functional ที่สามารถนำไปใช้งานกับระบบต่าง ๆ ทางเคมีได้อย่างครอบคลุม เช่น

- ฝึกสอนโมเดล ML โดยใช้ Topological Atom[193]
- ฝึกสอนโมเดล ML โดยใช้ Correlation Energy Density ด้วยวิธี CCSD(T)[194]
- ฝึกสอนโมเดล ML โดยใช้ Electron Density สำหรับวิธี DFT[195] และ MP2[196]
- สร้างโมเดล Neural Network แบบซับซ้อนเพื่อทำนาย Exchange-Correlation Holes[197,198]
- สร้างโมเดล Neural Network (DeepMind 21 หรือ DM21) ที่สามารถเรียนรู้และแก้ปัญหา Charge Delocalization Error ของ DFT ได้[199]

[1]จริง ๆ จะบอกว่าลองแบบมั่ว ๆ ก็ไม่ได้เพราะว่าในปัจจุบันมีบทความรีวิวที่อธิบายจุดเด่นและจุดด้อยของแต่ละ XC Functional รวมไปถึง Protocol สำหรับการเลือกใช้ XC Functional ที่เหมาะสมกับระบบโมเลกุลที่เราต้องการจะศึกษา



ภาพ 11.7 สถาปัตยกรรมของอัลกอริทึม NeuralXC โดยเริ่มต้นจากการใช้ DFT คำนวณ Electron Density แบบ Real Space แล้วทำการคำนวณ Feature Vectors จาก Density แล้วก็ใช้เป็นอินพุตสำหรับ BPN โดยเราจะใช้ Network แบบเดียวกันกับอะตอมทุกตัวเพื่อไม่ให้ Descriptor นั้นขึ้นอยู่กับลำดับของอะตอม (Permutation Invariance) ในระหว่างการฝึกสอนโมเดลนั้นเมื่อพลังงานถูกคำนวณแล้วก็จะมีการคำนวณอนุพันธ์ของพลังงานโดยเทียบกับความหนาแน่นอิเล็กตรอนโดยใช้ Backpropagation เพื่อหาเทอม XC Potential ต่อไป (เครดิตภาพ: *Nat Commun* 2020, 11, 3509)

หนึ่งในงานวิจัยที่ผู้เขียนคิดว่าผู้อ่านที่สนใจงานทางด้านการพัฒนา XC Functional สำหรับวิธี DFT ควรอ่านนั้นคืองานที่เสนออัลกอริทึม Supervised ML ที่ชื่อว่า NeuralXC[195] โดยเป็น ML ที่เรียนรู้ทั้งเทอมที่ใช้อธิบายพลังงาน Exchange และพลังงาน Correlation สำหรับ DFT (ปกติแล้วเทอม Correlation นั้นจะมีความซับซ้อนมากกว่าเทอม Exchange) ซึ่งในงานวิจัยนี้ก็ได้มีการฝึกสอนโมเดลโดยใช้ความหนาแน่นอิเล็กตรอน (Electron Density) ซึ่งงานวิจัยอื่น ๆ ก็มักจะใช้ Electron Density มาเป็นอินพุตเริ่มต้น ซึ่ง Representation ที่ใช้ในงานนี้สำหรับการระบุ Electron Density ก็คือ Atom-centered Basis Functions (ACSF) ส่วนอัลกอริทึม ML ที่ใช้คือ Behler-Parrinello Neural Networks (BPNN) โดยสมการของ XC Functional ที่โมเดล NeuralXC สร้างออกมาให้เราก็จะอยู่ในรูปของฟังก์ชันกระตุ้น (Activation Function) ที่ขึ้น Electron Density $(\rho(\boldsymbol{r}))$ ตามสมการดังต่อไปนี้ (ตามสมการที่ 5 ในบทความเต็ม)

$$\begin{aligned} E_{\text{ML}}[\rho(\boldsymbol{r})] &= E_{\text{ML}}(\boldsymbol{d}[\rho(\boldsymbol{r})]) \\ &= \sum_I \epsilon_{\alpha_I}(\boldsymbol{d}[\rho(\boldsymbol{r}), \boldsymbol{R}_I, \alpha_I]) \end{aligned} \tag{11.7}$$

โดยที่ $\boldsymbol{R}_I$ คือตำแหน่งของอะตอม I แล้วก็ใช้กฎลูกโซ่ในการหาอนุพันธ์ของพลังงานเทียบกับ Electron

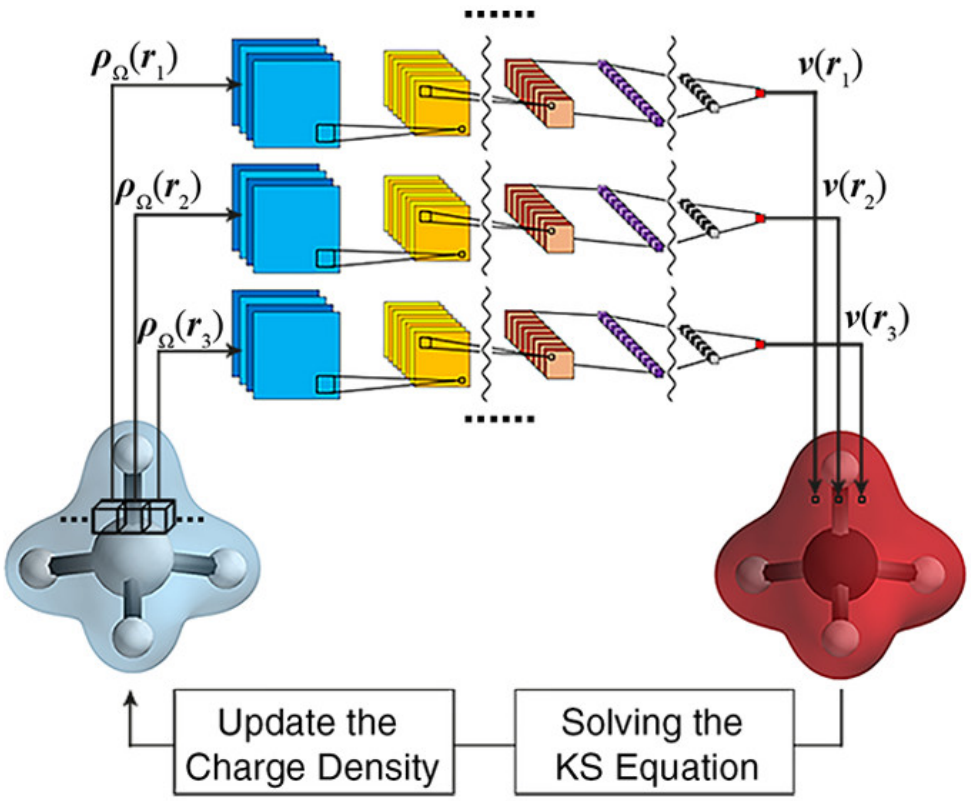
Density ได้ดังนี้

$$
\begin{aligned}
V_{ML}[\rho(\mathbf{r})] &= \frac{\delta E_{ML}[\rho]}{\delta \rho(\mathbf{r})} \\
&= \sum_{\beta} \frac{\partial E_{ML}}{\partial c_{\beta}} \frac{\delta c_{\beta}[\rho]}{\delta \rho(\mathbf{r})}
\end{aligned}
\tag{11.8}
$$

โดยเรากระจายอนุพันธ์ของพลังงานให้อยู่ในรูปของอนุพันธ์ของพลังงานเทียบกับ Descriptor ที่เป็นเกี่ยวข้องกับ Basis Function (c_{nlm})

ผู้วิจัยได้ทดสอบ XC Functional ที่ได้จาก NeuralXC กับชุดข้อมูลทางเคมีและอัลกอริทึม ML อื่น ๆ ด้วย เช่น sGDML และมีการเปรียบเทียบประสิทธิภาพการทำนายกับ Functional อย่าง SCAN และ ωB97M-V ซึ่งเหมาะสำหรับการศึกษาโมเลกุลและโครงสร้างวัสดุโดยเฉพาะ นอกจากนี้ XC Functional ที่ได้นั้นก็สามารถทำนาย Electron Density ของโมเลกุลได้อย่างแม่นยำโดยมีค่าความคลาดเคลื่อนเทียบกับวิธีการคำนวณขั้นสูงแบบคลาสสิค เช่น CCSD(T) ที่น้อยมาก (เมื่อเทียบกับผลการคำนวณด้วยวิธี DFT ที่ใช้ฟังก์ชันนอล PBE เป็นค่าอ้างอิงหลัก) แต่จุดอ่อนอย่างหนึ่งของการใช้ Representation หรือ Loss Function ที่มีความจำเพาะเจาะจงกับค่าเอาต์พุตหรือเป้าหมายมากเกินไปนั้นก็อาจจะส่งผลให้ Transferability ลดลงได้ เมื่อนำโมเดลไปใช้ในการทำนายกับโมเลกุลชนิดอื่นที่มีความแตกต่างไปจากโมเลกุลที่นำมาใช้ในการสร้างชุดข้อมูลที่ใช้ฝึกสอนโมเดล สำหรับผู้อ่านที่สนใจฝึกสอนโมเดลเพื่อคำนวณหา XC Functional เพื่อนำมาใช้ในงานวิจัยสามารถศึกษารายละเอียดในบทความฉบับเต็ม “Machine Learning Accurate Exchange and Correlation Functionals of the Electronic Density”

นอกการพัฒนาโมเดลแล้วในงานวิจัยนี้ยังได้พัฒนาไลบรารี่สำหรับการสร้าง XC Functional อีกด้วย ซึ่งผู้เขียนสนับสนุนงานวิจัยที่เผยแพร่โค้ดเนื่องจากว่างานวิจัยส่วนใหญ่นั้นมักจะโชว์แต่ทฤษฎีแล้วก็ผลการทดลอง แต่มักจะไม่มีโค้ดให้ได้ลองทดสอบ ทำให้เราก็ไม่รู้ว่าวิธีหรือทฤษฎีที่ผู้วิจัยได้เสนอมานั้นถูกต้องไหม หรือมีประสิทธิภาพมากน้อยเพียงใดและยังทำให้นักวิจัยคนอื่น ๆ เข้าถึงงานเราได้มากยิ่งขึ้น



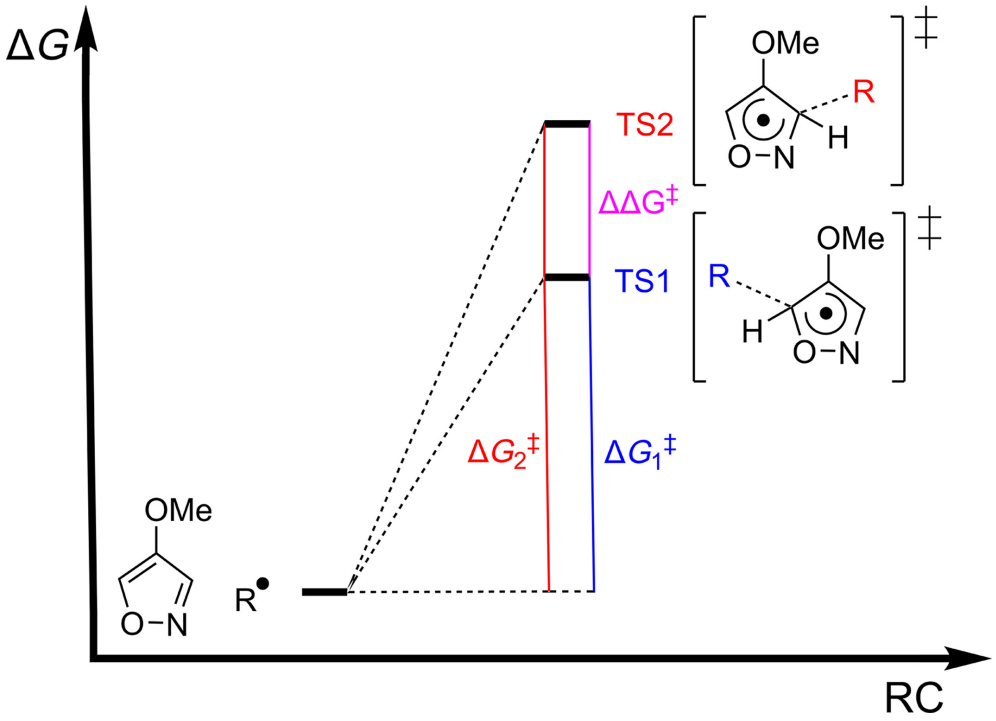
ภาพ 11.8 สถาปัตยกรรมของ Three-dimensional Convolutional Neural Network สำหรับการเรียนรู้ XC Potential โดยที่ ρ คือความหนาแน่นอิเล็กตรอนและ v คือ XC Potential (เครดิตภาพ: *J. Phys. Chem. Lett.* 2019, 10, 22, 7264-7269)

งานวิจัยชิ้นที่สองที่น่าสนใจก็คือ “Toward the Exact Exchange-Correlation Potential: A Three-Dimensional Convolutional Neural Network Construct” โดย Zhou และคณะ[200] โดยเป็นการใช้ Neural Network ที่เป็นแบบคอนโวลูชันแบบสามมิติ (Three-dimensional Convolutional Neural Network หรือ 3D-CNN) ในการเรียนรู้ความสัมพันธ์ระหว่างความแน่นอิเล็กตรอนแบบที่เกือบจะเป็นเชิงพื้นที่ (Quasi-local Electron Density) และพลังงานศักย์แลกเปลี่ยนและสหสัมพันธ์ (XC Potential) โดย Electron Density นั้นได้มาจากการคำนวณด้วยวิธี CCSD ส่วน XC Potential นั้นได้มาจากการคำนวณโดยใช้วิธี Direct Optimization ของ Wu และ Yang[201] ซึ่งเป็นวิธีสำหรับการปรับพลังงานศักย์ (Optimized Effective Potential หรือ OEP)

11.8 การทำนายพลังงานกระตุ้นของปฏิกิริยาเคมี

พลังงานกระตุ้น (Activation Energy) เป็นค่าพลังงานที่ต่ำที่สุดที่ใช้เพื่อกระตุ้นปฏิกิริยาเคมี บ่งบอกถึงความยากง่ายของการเกิดปฏิกิริยาเคมีที่สามารถดำเนินไปได้ ณ สภาวะหนึ่ง ๆ ปัจจัยที่มีผลต่อพลังงาน

กระตุ้น เช่น ความร้อนและตัวเร่งปฏิกิริยา โดยในปัจจุบันนั้นก็ได้มีการนำ ML เข้ามาช่วยในการทำนายพลังงานกระตุ้นอย่างแพร่หลาย[202] โดยหนึ่งในงานวิจัยนั้นก็คือการใช้ Graph Neural Network มาสร้างและฝึกสอนโมเดล[203]



ภาพ 11.9 แสดงค่าความแตกต่างระหว่างค่าพลังงานอิสระ $(\Delta\Delta G^{\ddagger})$ ซึ่งถูกเรียนรู้และทำนายด้วยโมเดล ML

นอกจากนี้ยังมีงานวิจัยของ Xin Li และทีมวิจัยที่ได้ใช้โมเดล ML หลายตัวด้วยกันไม่ว่าจะเป็น Linear Regression, Support Vector Regression, Random Forest Rregression, Kernel Ridge Regression, Gaussian Process Regression, หรือ Gradient boosting มาใช้ในการทำนายการเลือกเกิดเฉพาะที่ของปฏิกิริยาเคมี (Regioselectivity) ของปฏิกิริยาที่เป็นการ Functionalization ของ C—H Radical ของสารประกอบเฮทเทอโรไซคลิกจำนวน 3,406 ปฏิกิริยา[204] โดยค่าพลังงานกระตุ้นนั้นถูกเรียนรู้ด้วยโมเดล ML จากค่าความแตกต่างระหว่างพลังงานอิสระกิบส์ (Gibbs Free Energies) ของสารตั้งต้นและสภาวะทรานซิชั่น (Transition State) ซึ่งถูกปรับโครงสร้างด้วยวิธี DFT โดยใช้ฟังก์ชันนอล B3LYP และ 6-311+G(2d,p) และคำนวณค่าพลังงานด้วย M06-2X และ aug-cc-pVTZ โดย Regioselectivity ของปฏิกิริยาเคมีที่ศึกษานั้นถูกคำนวณโดยใช้ความแตกต่างระหว่างค่าพลังงานอิสระ $(\Delta\Delta G^{\ddagger})$

11.9 การทำนายประจุของอะตอม

อีกหนึ่งหัวข้องานวิจัยทางเคมีควอนตัมที่ได้รับความนิยมในช่วง 3-4 ปี (ตั้งแต่ปี ค.ศ. 2018) คือการใช้ ML สำหรับการระบุประจุย่อยของอะตอมในโมเลกุล (Partial Atomic Charge Assignment) งานวิจัยโดดเด่นและได้รับการตีพิมพ์ในวารสารชั้นแนวหน้าที่ใช้ ML เข้ามาในการทำนายหรือระบุประจุของอะตอมที่ผู้เขียนได้เลือกมาซึ่งเรียงลำดับตาม Timeline มีดังนี้

1. “Fast and Accurate Generation of *ab initio* Quality Atomic Charges Using Nonparametric Statistical Regression”[205]
   หนึ่งในงานวิจัยแรก ๆ ที่เริ่มมีการประยุกต์เทคนิคทางสถิติ (Regression) เข้ามาใช้ในการสร้างโมเดลเรียนรู้การทำนายประจุย่อย

2. “Machine Learning of Partial Charges Derived from High-Quality Quantum-Mechanical Calculations”[206]
   งานวิจัยที่ต่อยอดมาจากงานของ โดยใช้ Feature ที่ชื่อว่า “Atom-centered Atom-pairs Fingerprint”[207] ในการฝึกสอนโมเดลและนำมาใช้ในทำนายประจุย่อยได้อย่างแม่นยำมาก ๆ

3. “Transferable Dynamic Molecular Charge Assignment Using Deep Neural Networks”[208]
   งานวิจัยนี้ใช้ Neural Network ในการทำนายประจุย่อยแบบไดนามิกส์

4. “PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges”[183]
   งานวิจัยนี้นำเสนอโมเดล ML ที่ชื่อว่า PhysNet ซึ่งสามารถทำนายคุณสมบัติเชิงโมเลกุล ประกอบไปด้วย พลังงาน, แรง, ไดโพลโมเมนต์, และประจุย่อย

5. “Fast and Accurate Machine Learning Strategy for Calculating Partial Atomic Charges in Metal-Organic Frameworks”[209]
   งานวิจัยนี้เป็นการทำนายประจุของอะตอมในสารประกอบ Metal-Organic Frameworks (MOFs)

6. “High-Precision Atomic Charge Prediction for Protein Systems Using Fragment Molecular Orbital Calculation and Machine Learning”[210]
   งานวิจัยนี้สนใจการทำนายประจุเชิงอะตอมของโปรตีโดยใช้ Fragment Molecular Orbital

7. “DeepChargePredictor: A Web Server for Predicting QM-based Atomic Charges via State-of-the-art Machine-learning Algorithms”[211]
   งานวิจัยนี้พัฒนาเว็บไซต์ที่ช่วยให้เราสามารถทำนายประจุย่อยได้

11.10 การทำนายไดโพลโมเมนต์

การทำนายไดโพลโมเมนต์นั้นถือว่าเป็นอีกหนึ่งหัวข้องานวิจัยที่ได้รับความนิยมเพราะว่าไดโพลโมเมนต์นี้เป็นคุณสมบัติที่สำคัญมาก ซึ่งเกี่ยวข้องโดยโดยตรงกับเทคนิคทางการทดลอง เช่น Infrared Spectroscopy

ดังนั้นการที่ ML เข้ามาช่วยเราให้สามารถทำนายไดโพลโมเมนต์ได้รวดเร็วและถูกต้องได้นั้นก็จะเป็นการช่วยทำให้เราสามารถศึกษาโมเลกุลได้ง่ายมากยิ่งขึ้นโดยไม่จำเป็นที่จะต้องไปใช้วิธีการคำนวณแบบดั้งเดิมซึ่งมีความซับซ้อนเชิงการคำนวณและสิ้นเปลืองเวลาเยอะกว่ามาก[212,213,214,215,216,217,218]

11.11 การทำนายคุณสมบัติของโมเลกุลที่สถานะกระตุ้น

หนึ่งในงานวิจัยที่น่าสนใจที่ตีพิมพ์ในบทความ “Excited State Non-adiabatic Dynamics of Large Photoswitchable Molecules Using a Chemically Transferable Machine Learning Potential”[219] มีงานวิจัยที่ได้ใช้ ML ในการทำนาย Quantum Yield[1] ของโมเลกุลขนาดใหญ่ ณ สถานะกระตุ้น ซึ่งในงานวิจัยนี้ได้ศึกษาอนุพันธ์ของโมเลกุลเอโซเบนซีน (Azobenzene) รวมถึงไปการทรานซิชันระหว่างคอนฟอร์เมอร์ *cis* กับ *trans* อีกด้วย

11.12 การทำนายค่าคู่ควบเชิงเล็กอิเล็กทรอนิกส์

ค่าคู่ควบเชิงเล็กอิเล็กทรอนิกส์ (Electronic Coupling) คือค่าความเกี่ยวเนื่องเชิงอิเล็กทรอนิกส์ระหว่าง 2 สถานะใด ๆ ของอิเล็กตรอน เช่น สถานะเริ่มต้นและสถานะสิ้นสุดในกระบวนการทางควอนตัม

11.12.1 ค่าคู่ควบของการถ่ายโอนอิเล็กตรอน

11.12.2 ค่าคู่ควบแบบนอนอะเดียแบติก

11.13 การทำนายสเปกตรัม

การทำนายสเปกตรัมเป็นอีกหนึ่งหัวข้องานวิจัยทางด้านเคมีควอนตัมที่ได้รับความสนใจเป็นอย่างมาก นั่นก็เพราะว่าเทคนิคทางสเปกโทรสโกปีนั้นมีประโยชน์อย่างมากในงานทางด้านเคมีสังเคราะห์ ทั้งเคมีอินทรีย์และเคมีอนินทรีย์ รวมไปถึงด้านอื่น ๆ เช่น วัสดุศาสตร์หรือพอลิเมอร์ด้วย

สเปกโทรสโกปีเป็นเทคนิคที่เกี่ยวข้องกับแสงซึ่งเป็นคลื่นแม่เหล็กไฟฟ้าที่มีลักษณะเป็นแถบพลังงาน (Spectrum) โดยมีความยาวคลื่นตั้งแต่ในช่วง คลื่นวิทยุ คลื่นไมโครเวฟ คลื่นอินฟราเรด คลื่นในช่วงที่สายตามนุษย์มองเห็น รวมถึงคลื่นอัลตราไวโอเลต สำหรับคลื่นแม่เหล็กไฟฟ้าที่นักเคมีสนใจนั้นจะเกี่ยวข้องโดยตรงกับการระบุถึงความจำเพาะเจาะจงของโมเลกุล นั่นคือคลื่นแม่เหล็กไฟฟ้าในช่วงอินฟราเรด ซึ่งจะมีความยาวคลื่น

[1]ผู้เขียนขอใช้ทับศัพท์เพราะถ้าหากใช้คำแปลเป็นภาษาไทยจะใช้คำว่า “ผลผลิตควอนตัม” ซึ่งยากต่อการเข้าใจ ดังนั้นจึงขอใช้คำว่า Quantum Yield ตรง ๆ

(Wavelength) ในช่วง 650 - 4,000 nm หรือมีเลขคลื่น (Wavenumber) ในช่วง 14,286-12,800 cm^{-1} โดยหลักการคร่าว ๆ ของเทคนิคอินฟราเรดสเปกโทรโกปีก็คือแสงอินฟราเรดตกกระทบโมเลกุลจะเกิดอันตรกิริยาระหว่างแสงกับโมเลกุล โดยที่แสงอินฟราเรดในบางช่วงที่มีความถี่ตรงกันกับความถี่ของการสั่นของพันธะในโมเลกุลจะถูกดูดกลืนไป ซึ่งทำให้เกิดทรานซิชันการสั่นพร้อมกับทรานซิชันการหมุน ซึ่งทรานซิชันการสั่นนี้จะเราทราบชนิดของหมู่ฟังก์ชัน เช่น พันธะคู่ พันธะสาม หมู่คาร์บอนิล หมู่ไฮดรอกซิล หรือหมู่อะมิโน ภายในโครงสร้างของโมเลกุลอินทรีย์ได้ ดังนั้นความเข้มของแสงอินฟราเรดที่ทะลุผ่านสารตัวอย่าง (Transmitted Infrared) จึงมีความเข้มแสงลดลงในบางช่วงของความถี่ทั้งหมดของอินฟราเรดเนื่องมาจากการถูกดูดกลืนโดยหมู่ฟังก์ชันดังกล่าวนั่นเอง

11.13.1 การทำนายอินฟราเรดสเปกโทรโกปี

การทำนายสเปกตรัมอินฟราเรดด้วย ML นั้นได้รับการค้นคว้ามาอย่างต่อเนื่องเป็นระยะเวลาหลายสิบปี[220] สำหรับงานวิจัยที่ผู้เขียนจะยกมาเป็นกรณีศึกษานั้นเป็นงานวิจัยที่มีชื่อบทความว่า “Infrared Spectra at Coupled Cluster Accuracy from Neural Network Representations” หรือแปลเป็นภาษาไทยคือ “การทำนาย IR Spectrum ที่ระดับความแม่นยำเดียวกับระเบียบวิธี CCSD(T) ด้วย Neural Network”[221] โดยงานวิจัยนี้ได้รับการตีพิมพ์ในวารสาร Journal of Chemical Theory and Computation (JCTC) ซึ่งเป็นวารสารวิชาการแนวหน้าทางด้านเคมีทฤษฎีและการคำนวณทางคอมพิวเตอร์

รายละเอียดงานวิจัยในบทความฉบับนี้มีดังนี้ ผู้วิจัยได้สร้าง Neural Network โดยใช้สถาปัตยกรรมโครงสร้างประสาทของ Behler-Parrinello Neural Network (BPNN)[168,222,223] ซึ่งเป็น Neural Network เชิงโมเลกุลแบบที่มีจำนวนมิติสูง (High-dimensional Molecular Neural Network) ที่มีแนวคิดคือผลรวมของพลังงานทั้งหมดของโมเลกุลเกิดขึ้นจากการรวมกันของพลังงานของแต่ละอะตอม โดยที่ Neural Network ที่ผู้วิจัยได้พัฒนาขึ้นมาได้ถูกนำไปใช้ในการเรียนรู้ (Learn) โครงสร้างของโมเลกุลและ Fit เข้ากับค่า Dipole Moment (μ) ของโมเลกุลนั้น ๆ ซึ่ง μ คือพารามิเตอร์สำคัญที่เราสามารถนำไปใช้ในการคำนวณหาความเข้มหรือ Intensity ของ IR Spectrum (สำหรับแต่ละ Peak) ต่อไปได้

แต่ว่าผู้วิจัยไม่ได้ Fit ข้อมูลเชิงโครงสร้างของโมเลกุลเข้ากับ μ โดยตรงเพราะว่าพารามิเตอร์ทั้งสองนี้ไม่ได้สอดคล้องหรือเกี่ยวข้องกันโดยตรง ดังนั้นเราควรจะต้องทำการ Fit เข้ากับบางสิ่งบางอย่างที่อธิบายเคมีเชิงอิเล็กทรอนิกของโมเลกุลได้ดีกว่าข้อมูลเชิงโครงสร้างทั่วไป ซึ่งสิ่งนั้นเรียกว่า Electronic-based Descriptor โดย Descriptor ที่ผู้วิจัยเลือกใช้ก็คือ Atomic-centered Symmetry Function (ACSF) โดยได้ทำการ Fit ACSF เข้ากับประจุย่อยของแต่ละอะตอม (Atomic Partial Charge) ซึ่งผู้อ่านอาจจะสงสัยว่าทำไมถึงไม่ Fit ACSF กับ μ โดยตรงเลย? เหตุผลก็เพราะว่า μ นั้นถูกคำนวณมาจากผลรวมของผลคูณระหว่างประจุ (q_i) และตำแหน่งของแต่ละอะตอม $(\vec{r}_i)$ ภายในโมเลกุลตามสมการที่ (11.9)

$$\mu = \sum_{i}^{N} q_i \vec{r}_i \tag{11.9}$$

ดังนั้นจึงจะเหมาะกว่าถ้าเรา Fit เข้ากับประจุก่อน หลังจากนั้นเอาต์พุตสุดท้ายที่ถูกทำนายออกมาจาก Neural Network นั้นก็จะเป็น μ นั่นเอง สรุปลำดับการ Fit ข้อมูลมีดังนี้

Structure → ACSF → Atomic Partial Chage → Dipole Moment

นอกจากนี้ผู้วิจัยเลือกใช้ RMSE เป็น Lost Function สำหรับการปรับลด Error ระหว่าง μ ที่ได้จากการทำนายกับค่าอ้างอิงที่ได้จากการคำนวณด้วยวิธี CCSD(T) โดยใช้โปรแกรม Molpro และได้มีการปรับแต่ง Loss Function โดยทำการคำนวณ Error ของทุกคอนฟอร์เมอร์ ซึ่งจะทำการปรับค่าลดค่าคลาดเคลื่อนจาก μ ของทั้งสามทิศทาง (3 Components) นั่นคือ x, y และ z ตามสมการดังต่อไปนี้

$$\mathcal{L} = \frac{1}{3M}\sum_{i}^{M}\sum_{\alpha}^{3}(\mu_{i,\alpha}^{\text{NN}} - \mu_{i,\alpha}^{\text{ref}})^2 \tag{11.10}$$

โดยที่ M คือจำนวนคอนฟอร์เมอร์ และ α คือทิศทาง หลังจากนั้นผู้วิจัยใช้ Autocorrelation Function ในการแปลง μ (เปลี่ยนจาก Time-domain ไปเป็น Frequency-domain) เพื่อคำนวณหาสเปกตรัมของอินฟราเรดต่อไป

สำหรับชุดโมเลกุลที่ผู้วิจัยศึกษาในงานนี้เป็นแค่กลุ่มโมเลกุลง่าย ๆ คือกลุ่มโมเลกุลน้ำ (Water Cluster) โดยมีการศึกษาความสามารถในการเรียนรู้ของโมเดลที่เปลี่ยนแปลงไปตามขนาดของชุดข้อมูลฝึกสอนและชุดข้อมูลทดสอบ โดยผู้วิจัยได้มีการใช้ชุดข้อมูลฝึกสอนขนาดเล็กสุดคือ 5,975 โครงสร้างและใหญ่สุดคือ 18,576 โครงสร้าง ซึ่งประสิทธิภาพในการทำนายถือว่าอยู่ในระดับที่แม่นยำมาก โดยมีค่าคลาดเคลื่อนการทำนาย μ ต่อโมเลกุลอยู่ที่ 0.007 D และต่ออะตอมอยู่ที่ประมาณ 0.002 D

11.13.2 การทำนายรามานสเปกโทรโกปี

11.13.3 วิธีการทำนายสเปกตรัมและโครงสร้าง

ตาราง 11.3 บทความงานวิจัยที่เกี่ยวข้องกับวิธี Struc-to-Spec

อ้างอิง	Learning Target	โมเดล ML	Representation
[224,225]	ความเข้มการดูดกลืน IR	PCA+CPG,[226] CPG[226]	คุณสมบัติเชิงอะตอม, ระยะห่างระหว่างอะตอม
[227,228]	ความเข้มการดูดกลืน IR	Query Driven Selection + RDF + CPG[226]	คุณสมบัติเชิงอะตอม, ระยะห่างระหว่างอะตอม
[229,230]	ความเข้ม IR/Raman	LFFN-EPFs[229]	ความเข้ม IR/Raman
[220]	ประจุย่อยเชิงอะตอม	HDNNP[168] + ED-GEKF[231]	ACSFs,[120] Geometric Descriptors
[232,208]	ประจุย่อยเชิงอะตอม	HIP-NN[180]	เลขอะตอม, ระยะห่างระหว่างอะตอม

ตาราง 11.4 บทความงานวิจัยที่เกี่ยวข้องกับวิธี Spec-to-Struc

อ้างอิง	Learning Target	โมเดล ML	Representation
[233,234]	โครงสร้างและหมู่ฟังก์ชัน	PLS,[23] PCA-MLP, MLP	Band Pattern Full Spectrum
[235]	โมเลกุลทั่วไป	Binary และ Decimal-based NN	Band Pattern
[236]	คอนฟิกูเรชั่นเชิงนิวเคลียร์	Basin-hopping Algorithm[237]	Cosine Distances
[238]	หมู่ฟังก์ชัน	Autoencoder + MLP, RF[25]	Peak ของ FTIR
[239]	โครงสร้างที่มีหมู่ OH และ C=O	LSTM[240]	Peak ของ IR และ Raman ที่คำนวณด้วย DFT

โดยตัวย่อที่ใช้ในตารางที่ 11.3 และ 11.4 มีชื่อเต็มดังต่อไปนี้

- CPG: Counterpropagation
- ED-GEKF: Element-decoupled Global Extended Kalman Filter
- FTIR: Fourier Transform Infrared
- HDNNP: High Dimensional Neural Network Potential
- HIP-NN: Hierarchically Interacting Particle Neural Network
- LFFN-EPFs: Layered Feed Forward Neural Network - Empirical Physical Formulas
- MLP: Multilayer Perceptron
- NMA: N-methylacetamide
- PCA: Principal Component Analysis
- PG-EA: Probability Graph-evolutionary Algorithm
- RDF: Radial Distribution Function

11.14 บทความวิชาการเพิ่มเติม

นอกเหนือจากการนำ ML ไปใช้สำหรับการทำนายพารามิเตอร์ต่าง ๆ แล้ว ถ้าหากผู้อ่านสนใจการประยุกต์ใช้ ML กับงานทางด้านอื่น ๆ ของเคมีควอนตัม สามารถอ่านบทความวิชาการเพิ่มเติมได้จากวารสารวิชาการชั้นแนวหน้า เช่น

1. Journal of Chemical Information and Modeling (*J. Chem. Inf. Model*)
2. Journal of Chemical Physics (*J. Chem. Phys.*)
3. Journal of Chemical Theory and Computation (*J. Chem. Theory Comput.*)
4. Journal of Physical Chemistry A (*J. Phys. Chem. A*)
5. Journal of Physical Chemistry Letters (*J. Phys. Chem. Lett.*)
6. Machine Learning: Science and Technology (*MLST*)
7. Nature Machine Intelligence (*Nat. Mach. Intell*)
8. Nature Communications (*Nat. Commun.*)

9. Proceedings of the National Academy of Sciences of the United States of America (*PNAS*)
10. Science Advances (*Sci. Adv.*)

โดยวารสารอันดับที่ 1-5 เป็นวารสารเฉพาะทางด้านเคมีทฤษฎีและเคมีคอมพิวเตอร์, วารสารอันดับที่ 6-7 เป็นวารสารเฉพาะทางด้านการเรียนรู้ของเครื่อง, และวารสารอันดับที่ 8-10 เป็นวารสารด้านวิทยาศาสตร์ทั่วไป นอกจากนี้แล้วยังมีวารสารอื่น ๆ อีกที่ไม่ได้กล่าวถึงรวมไปถึง Repository ที่ให้บริการเผยแพร่บทความงานวิจัยแบบเข้าถึงได้ฟรี (Open-access) เช่น arXiv (https://arxiv.org/) และ ChemRxiv (https://chemrxiv.org)

11.14.1 บทความเฉพาะทาง

โดยผู้เขียนได้เลือกงานวิจัยที่มีความโดดเด่นและเหมาะสำหรับผู้เริ่มต้นศึกษา ML และเคมีควอนตัม ซึ่งน่าจะช่วยให้ผู้อ่านเห็นภาพรวมของโจทย์งานวิจัยในปัจจุบันที่กำลังมาแรง บทความที่คัดเลือกมาประกอบไปด้วยบทความการทบทวนงานวิจัย (Review) ที่ใช้ ML ในการเรียนรู้ Force Field สำหรับงานทางด้านเคมีควอนตัมและการจำลองพลวัตเชิงโมเลกุล (QM/MD) หรือนำมาใช้ในการทำนายพื้นที่พลังงานอิสระ (Free Energy Landscape) ไปจนถึงการพัฒนาโมเดล ML เพื่อทำนายคุณสมบัติเชิงโมเลกุล เช่น ไดโพลโมเมนต์ (Dipole Moment) และสภาพการเกิดขั้น (Polarizability)

1. “PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges”[183]
   ตีพิมพ์เมื่อวันที่ 01 พฤษภาคม ค.ศ. 2019
2. “Comparison of the Performance of Machine Learning Models in Representing High-Dimensional Free Energy Surfaces and Generating Observables”[241]
   ตีพิมพ์เมื่อวันที่ 10 เมษายน ค.ศ. 2020
3. “Kernel-Based Machine Learning for Efficient Simulations of Molecular Liquids”[242]
   ตีพิมพ์เมื่อวันที่ 13 เมษายน ค.ศ. 2020
4. “Machine Learning Force Fields”[243]
   ตีพิมพ์เมื่อวันที่ 11 มีนาคม ค.ศ. 2021
5. “The Rise of Neural Networks for Materials and Chemical Dynamics”[244]
   ตีพิมพ์เมื่อวันที่ 1 กรกฎาคม ค.ศ. 2021

11.14.2 บทความรีวิว

ในช่วง 5 ปีที่ผ่านมานี้ (ค.ศ. 2017-2022) นอกจากจะมีการตีพิมพ์บทความงานวิจัยเฉพาะทางออกมาอย่างต่อเนื่องแล้ว ยังได้มีการตีพิมพ์บทความวิจารณ์หรือรีวิวซึ่งเป็นการสรุปภาพรวมของการใช้ ML ในหัวข้อต่าง ๆ ของเคมีทฤษฎีซึ่งผู้เขียนได้สรุปไว้ในตารางที่ 11.5

ตาราง 11.5 บทความรีวิวงานวิจัยทางด้าน ML และเคมีทฤษฎี

ปี ค.ศ. ที่ตีพิมพ์	ผู้แต่งและบทความอ้างอิง	หัวข้อที่รีวิว
2017	Behler[245]	พลังงานศักย์ระหว่างอะตอม
2018	Goldsmith และคณะ[246]	ML สำหรับ Catalysis
2019	Carleo และคณะ[247]	ML สำหรับวิทยาศาสตร์เชิงกายภาพ
2019	Yang และคณะ[248]	Drug Discovery
2019	Elton และคณะ[249]	Molecular Design
2019	Schleder และคณะ[250]	ML สำหรับวัสดุศาสตร์
2019	Ceriotti[251]	การเรียนรู้แบบไม่มีผู้สอน
2020	Dral[252]	ML สำหรับเคมีควอนตัม
2020	Noé และคณะ[159]	การจำลองเชิงโมเลกุล
2020	von Lilienfeld และคณะ[253]	ปริภูมิเคมี
2020	Mueller และคณะ[254]	พลังงานศักย์ระหว่างอะตอม
2020	Manzhos และคณะ[255]	ML สำหรับโมเลกุลและปฏิกิริยาขนาดเล็ก
2020	Gkeka และคณะ[256]	Force Fields และ Coarse Graining
2020	Unke และคณะ[243]	Force Fields
2020	Toyao และคณะ[257]	การเร่งปฏิกิริยาเชิงข้อมูล
2020	Manzhos[258]	ML สำหรับโครงสร้างเชิงอิเล็กทรอนิกส์
2020	Westermayr และ Marquetand[259]	ML สำหรับสถานะกระตุ้น
2021	Behler[260]	Neural Network Potentials
2021	Westermayr และคณะ[151]	ML สำหรับเคมีเชิงคำนวณและวัสดุศาสตร์
2021	Zheng และคณะ[261]	วิธีทางควอนตัมและการทำนายพลังงาน
2022	Sajjan และคณะ[262]	การพัฒนา ML สำหรับ Quantum Computing
2022	Lim และคณะ[263]	คุณสมบัติของโมเลกุลและ Drug Discovery
2022	Raghunathan และคณะ[264]	Representation สำหรับ ML
2022	Stuyver และคณะ[265]	ML สำหรับความว่องไวของปฏิกิริยาเคมี
2022	Tavakoli และคณะ[266]	ML (Graph) สำหรับความว่องไวของปฏิกิริยาเคมี
2022	Han และคณะ[267]	ML สำหรับสเปกโทรสโกปี
2022	van Gerwen และคณะ[268]	Representation สำหรับ Chemical Reactivity
2022	Qiao และคณะ[269]	Geometric Deep Learning

นอกจากนี้ยังมีบทความรีวิวที่ผู้เขียนแนะนำให้ผู้สนใจศึกษา ML สำหรับเคมีควอนตัมแบบเชิงลึกโดยเฉพาะการต่อยอดในงานวิจัย ดังนี้

1. “Roadmap on Machine Learning in Electronic Structure”[270]
   อธิบายภาพรวมของ ML กับการศึกษาโครงสร้างเชิงอิเล็กทรอนิกส์ของโมเลกุล

2. “Unsupervised Learning Methods for Molecular Simulation Data”[43]
   อธิบายการใช้เทคนิคการเรียนรู้แบบไม่มีผู้สอนสำหรับการจำลองและการวิเคราะห์ข้อมูลเชิงโมเลกุล

3. “Physics-Inspired Structural Representations for Molecules and Materials”[103]
   อธิบาย Representations สำหรับโมเลกุลและวัสดุ

4. “Combining Machine Learning and Computational Chemistry for Predictive Insights Into Chemical Systems”[104]
   อธิบายการใช้ ML ในการทำนายคุณสมบัติเชิงเคมี

5. “Machine Learning for Electronically Excited States of Molecules”[259]
   อธิบาย ML สำหรับการศึกษาโมเลกุลในสถานะกระตุ้น

6. “Ab Initio Machine Learning in Chemical Compound Space”[271]
   อธิบายการใช้ ML สำหรับการศึกษาปริภูมิเคมีขนาดใหญ่

7. “Four Generations of High-Dimensional Neural Network Potentials”[260]
   อธิบายโครงข่ายประสาทสำหรับการทำนายพลังงานของโมเลกุลทั้ง 4 รุ่น

8. “Gaussian Process Regression for Materials and Molecules”[272]
   อธิบายการใช้เทคนิค GPR ในการทำนายคุณสมบัติของโมเลกุลและวัสดุ

9. “Machine Learning Force Fields”[243]
   อธิบายการเรียนรู้และสร้าง Force Field โดยใช้ ML

10. “Neural Network Potential Energy Surfaces for Small Molecules and Reactions”[255]
    อธิบายการศึกษาโมเลกุลและปฏิกิริยาเคมีระบบเล็กโดยใช้โครงข่ายประสาทของพื้นผิวพลังงานศักย์

11. “Machine Learning for Chemical Reactions”[273]
    อธิบายการใช้ ML สำหรับทำนายปฏิกิริยาเคมี

บทที่ 12

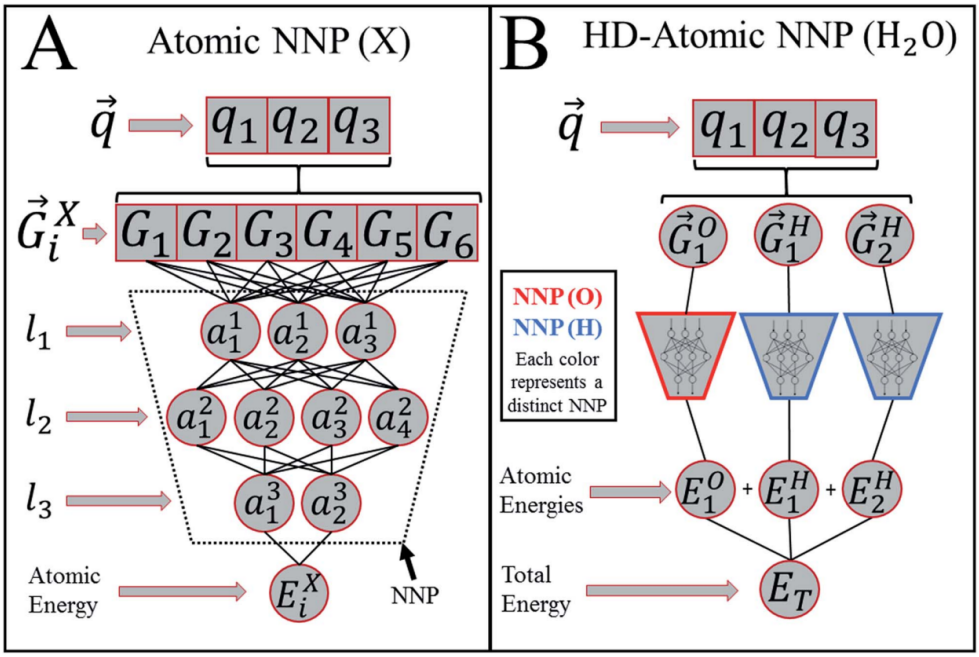
โมเดลการเรียนรู้ของเครื่องสำหรับเคมีควอนตัม

ในบทก่อนหน้านี้เราได้ดูรายละเอียดเกี่ยวกับการสร้างโมเดล ML ทั้งแบบ Parameteric และแบบ Non-parameteric Model ไปแล้ว ในปัจจุบันนั้นก็ได้มีนักวิจัยจากหลายกลุ่มวิจัยที่ได้พัฒนาโมเดล ML ของตัวเองขึ้นมาและตั้งชื่อให้กับโมเดลเหล่านั้น โดยโมเดลแต่ละโมเดลก็มีข้อดีและข้อด้อยแตกต่างกันไป ขึ้นอยู่กับสถาปัตยกรรมที่ใช้ โดยโมเดล ML ที่ได้รับความสนใจมากที่สุดนั้นก็คือ Neural Network โดยได้มีโมเดล Neural Network แบบสำเร็จรูปและพร้อมใช้งานและมีประสิทธิภาพที่สูงมากหลายสิบโมเดล ในบทนี้ผู้อ่านจะได้ศึกษาโมเดล Neural Network เหล่านั้น

12.1 ANI-1

ANI เป็นชื่อย่อสั้น ๆ ของโมเดล ANAKIN-ME ซึ่งย่อมาจาก Accurate NeurAl networK engINe for Molecular Energies อีกทีหนึ่ง ซึ่งผู้เขียนคิดว่าผู้พัฒนาน่าจะตั้งชื่อเพื่อให้มีตัวย่อตรงกับ Artificial Narrow Intelligence (ANI) หรือการเรียนรู้แบบแคบ[1] โดยโมเดลตัวนี้ถูกพัฒนาด้วย Neural Network สำหรับการทำนายค่าพลังงานศักย์ของโมเลกุลโดยที่มีความแม่นยำในระดับเดียวกับการคำนวณด้วยวิธี DFT[169] นอกจากนี้ผู้วิจัยยังได้มีการสร้างชุดข้อมูลสำหรับโมเดลในตระกูล ANI (เช่น โมเดลที่ใช้ในการศึกษาศักย์ของโมเลกุล ANI-1x และ ANI-1ccx) อีกด้วย[274]

[1]Artificial Narrow Intelligence เป็นเทคนิคปัญญาประดิษฐ์แบบหนึ่งซึ่งจะถูกออกแบบให้มีความสามารถและเชี่ยวชาญเฉพาะด้าน

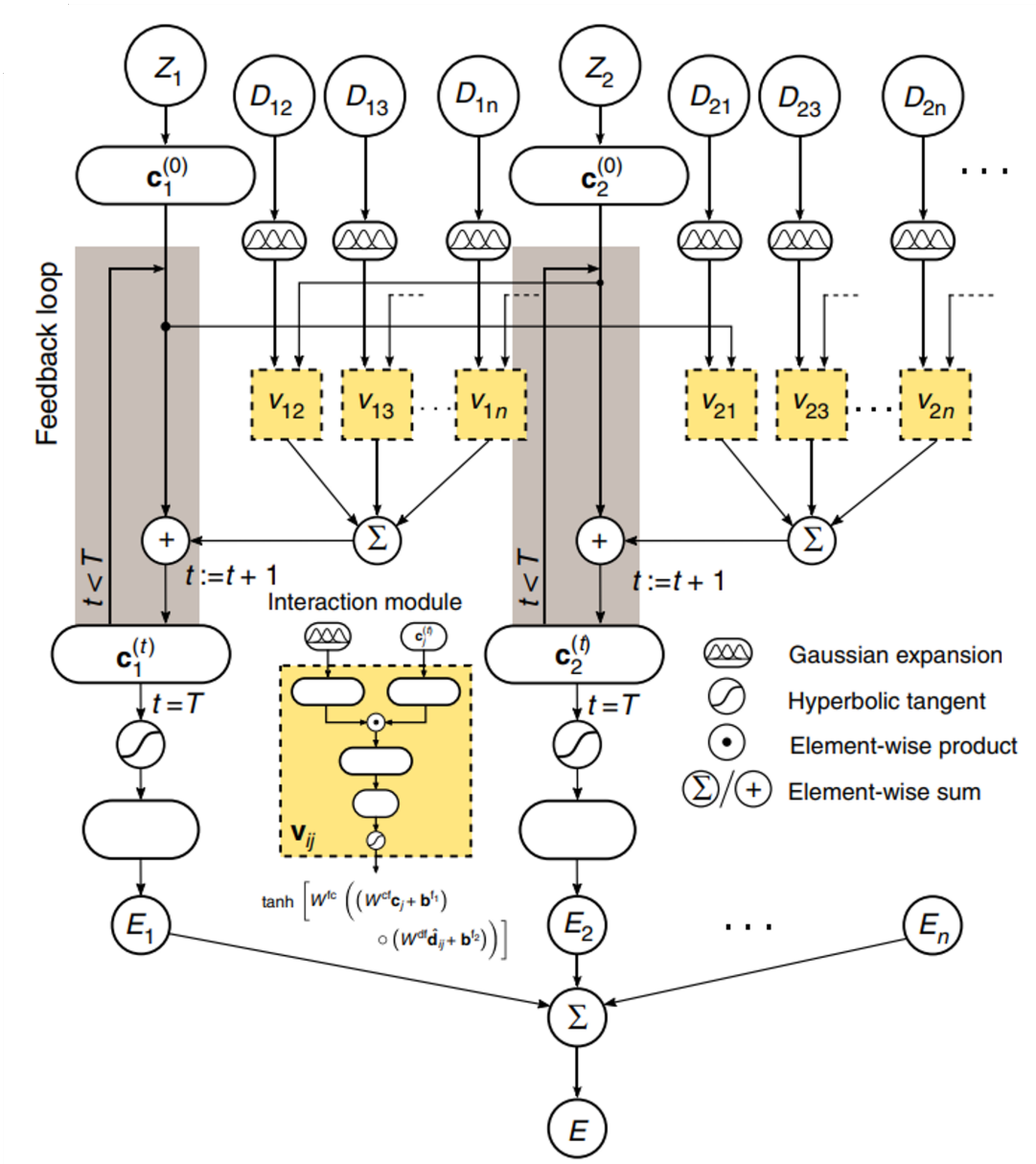


ภาพ 12.1 สถาปัตยกรรมของ Deep Tensor Neural Network (เครดิตภาพ: *Chem. Sci.*, 2017, 8, 3192-3203)

ภาพที่ 12.1 แสดงสถาปัตยกรรมของโมเดล ANI ซึ่งถูกออกแบบโดยใช้ Behler-Parrinello Neural Network (BPNN) เป็นโครงข่ายประสาทแบบพื้นฐาน โดยเริ่มต้นนั้นตามภาพย่อย A และ B อินพุตเริ่มต้นที่ใช้นั้นจะเป็นตำแหน่งของแต่ละอะตอมในโมเดล (q_i) ซึ่งตำแหน่งเหล่านี้จะถูกนำมาใช้ในการคำนวณ Atomic Environment Vector $(\vec{G}_i^X)$ สำหรับอะตอม i ที่มีเลขอะตอม X หลังจากนั้น $(\vec{G}_i^X)$ จะถูกใช้เป็นข้อมูลในการฝึกสอนโมเดลเพื่อทำนาย Atomic Contribution ของแต่ละอะตอม โดยที่แต่ละโหนดในแต่ละชั้นของ Hidden Layer นั้นจะแทนหนึ่งอะตอม

สิ่งที่แตกต่างกันระหว่างภาพย่อย A และ B ก็คือในภาพย่อย A นั้นโครงข่ายประสาทนั้นจะเป็นแบบมิติที่ไม่สูงมากซึ่งจะตรงข้ามกับโครงข่ายประสาทที่แสดงในภาพย่อย B ซึ่งจะมีจำนวนมิติที่สูงกว่า โดยเราเรียกโครงข่ายประสาทในกรณีหลังนี้ว่า High-demensional-atomic BPNN หรือ HD-atomic BPNN โดยตัวอย่างที่แสดงให้ดูก็คือเป็น HD-atomic BPNN ของโมเลกุลน้ำซึ่งจะประกอบไปด้วยโหนดหลัก 3 โหนดในชั้นเริ่มต้น และข้อมูลจะถูกส่งต่อไปยัง Neural Network ย่อย ๆ ของแต่ละอะตอมเพื่อฝึกสอนและเรียนรู้ Atomic Contribution ของอะตอมออกซิเจนและอะตอมไฮโดรเจนนั่นเอง

12.2 Deep Tensor Neural Network



ภาพ 12.2 สถาปัตยกรรมของ Deep Tensor Neural Network (เครดิตภาพ: *Nat Commun*, 2017 8, 13890)

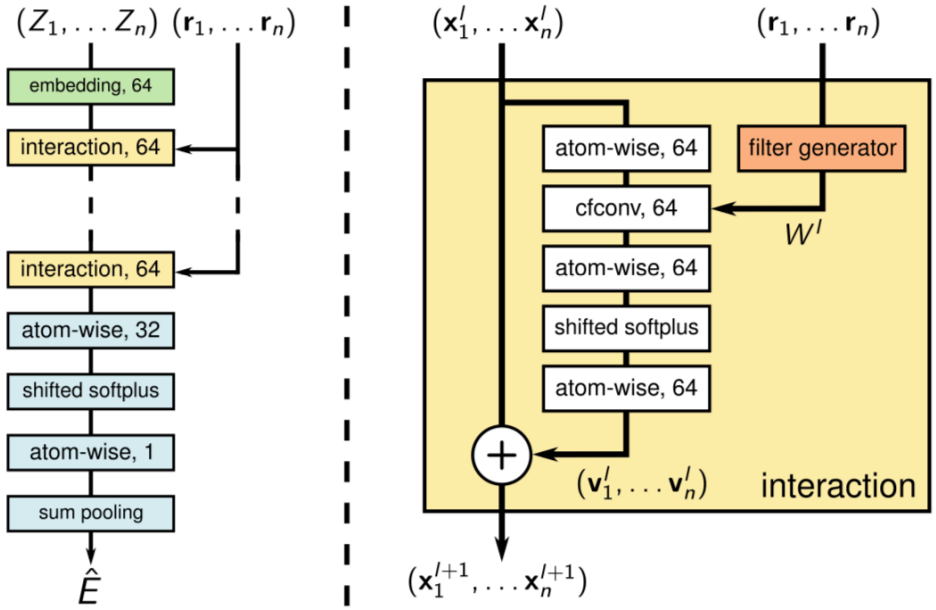
Deep Tensor Neural Network หรือ DTNN[144] เป็น Neural Network ที่ใช้ Many-body Hamiltonian ภาพที่ 12.2 แสดงสถาปัตยกรรมของ DTNN ในการสร้างโมเดลแบบครบวงจร (End-to-end Model) โดยเริ่มต้นนั้น DTNN จะรับอินพุตที่เป็นข้อมูลเชิงโครงสร้างและเชิงอิเล็กทรอนิกส์ของโมเลกุลโดยใช้เมทริกซ์

ของระยะห่างระหว่างอะตอม (Atomic Distance) และประจุนิวเคลียร์ของอะตอม (Z) หลังจากนั้นทำการเปลี่ยนรูปของระยะห่าง (Expand) โดยการใช้ Gaussian Basis Function ซึ่งเป็นไอเดียเดียวกับ Coulomb Matrix นั่นเอง โดยชุดข้อมูลที่ผู้วิจัยหลักใช้ในการทดสอบประสิทธิภาพของโมเลกุลก็คือ GDB-7 และ GDB-9

12.3 SchNet และ SchNOrb

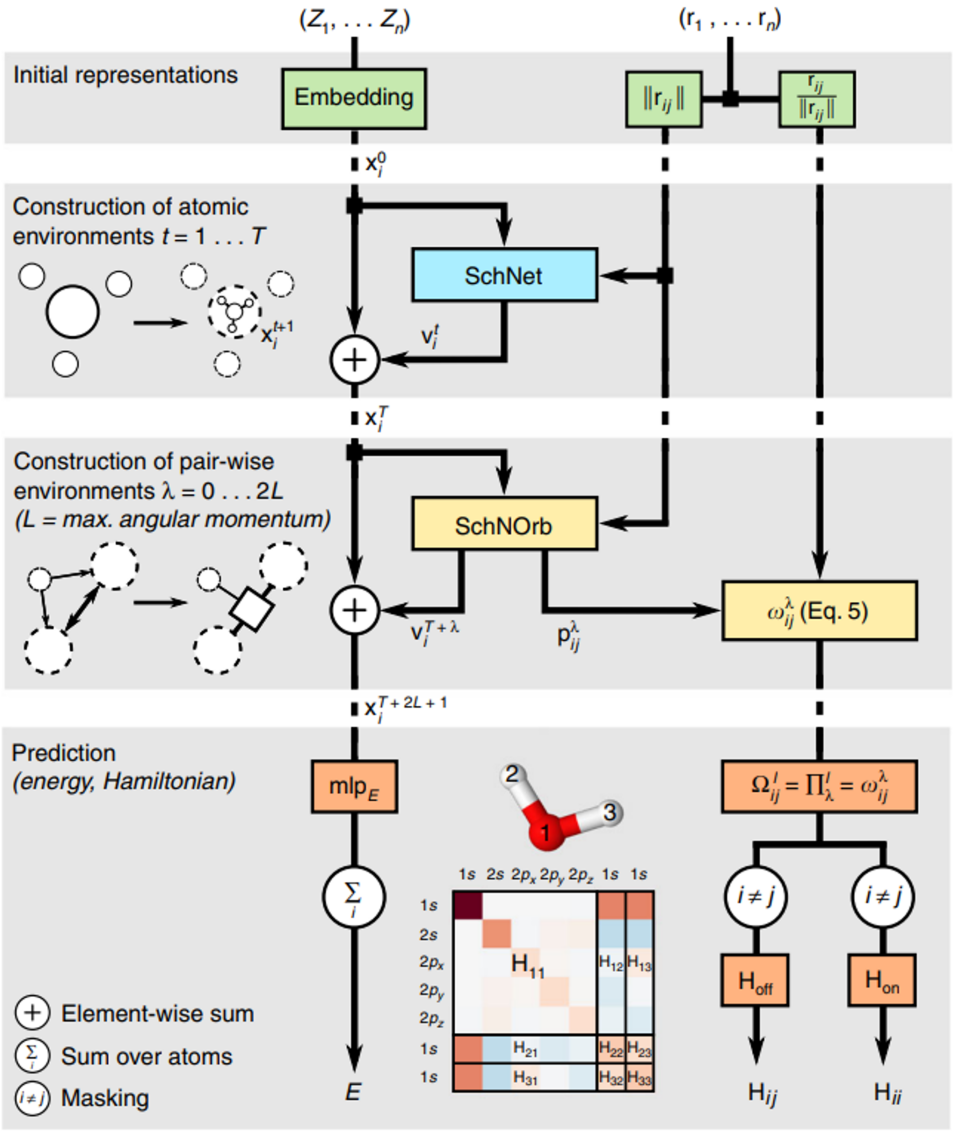
12.3.1 SchNet

SchNet เป็นโมเดลที่ใช้ Continuous Convolutions ซึ่งได้แรงบันดาลใจมาจาก Convolutional Neural Network (CNN) สำหรับการอธิบายอัตรกิริยาระหว่างอะตอม[143,179] ผู้อ่านสามารถดูรายละเอียดและ Source Code ของ SchNet ได้ที่เว็บไซต์ https://github.com/atomistic-machine-learning/SchNet หรือที่เว็บไซต์ https://paperswithcode.com/method/schnet



ภาพ 12.3 สถาปัตยกรรมของ SchNet (เครดิตภาพ: SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions)

12.3.2 SchNOrb



ภาพ 12.4 สถาปัตยกรรมของ SchNorb (เครดิตภาพ: *Chem. Sci.*, 2017, 8, 3192)

SchNOrb ย่อมาจาก “SchNet for Orbitals”[184] เป็นโมเดลที่รวมหรือ Capture ผลของ Local Representation ของออร์บิทัลเชิงอะตอมเข้าไปด้วยเพื่อทำให้การทำนายคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลนั้นมีความแม่นยำมากขึ้น ภาพที่ 12.4 แสดงสถาปัตยกรรมของ Neural Network ที่ใช้ใน SchNorb โดยประกอบไปด้วย 3 ขั้นตอนตามที่แสดงในกล่องสีเทาโดยเริ่มต้นจากการใช้ Representation ในการคำนวณ Feature Vector ตามชนิดและตำแหน่งของอะตอมก่อน ต่อจากนั้นก็ตามด้วยการสร้าง Representation ที่ใช้อธิบายข้อมูลสภาพแวดล้อมเชิงเคมีของอะตอมและคู่ของอะตอม (Atom Pair) ก่อนที่จะใช้ข้อมูลเหล่านี้

ในการทำนายพลังงานและ Hamiltonian ตามลำดับ ซึ่งพลังงานที่ได้ออกมานั้นมีคุณสมบัติที่ไม่ขึ้นอยู่กับการหมุน (Rotationally Invariant) ในขณะที่ Hamiltonian นั้นจะสามารถมีค่า (Valid) ได้ตามค่าของ Angular Moment (L) ที่มากที่สุดของอะตอมที่ต้องการทำนาย นอกจากนี้ยังสามารถทำนาย Overlap Matrix (S) ได้พร้อม ๆ กันอีกด้วย

ผู้อ่านสามารถดูรายละเอียดและ Source Code ของ SchNorb ได้ที่เว็บไซต์ https://github.com/atomistic-machine-learning/SchNOrb

12.4 SchNarc

SchNarc เป็นโมเดล ML สำหรับการศึกษาพลศาสตร์โมเลกุลที่สถานะกระตุ้น (Excited State Dynamics)[275] โดยเป็นการนำเอาโมเดล SchNet[143,179] มารวมกับอัลกอริทึม SCHARC[276,277] ซึ่งเป็นวิธี Semi-classical Surface Hopping

12.5 Symmetric Gradient Domain Machine Learning

Symmetric Gradient Domain Machine Learning หรือ sGDML เป็นหนึ่งในโมเดลที่ผู้เขียนคิดว่ามีประสิทธิภาพมากที่สุดในการสร้าง Force Field ด้วย ML สำหรับการทำนายพลังงานและแรงเชิงอะตอมของโมเลกุล[278,279,280,281,282] โดยไอเดียหลักที่ผู้วิจัยที่พัฒนา sGDML ขึ้นมานั้นได้ใช้ในการพัฒนาและปรับปรุงโมเดลก็คือการรวมความเป็นสมมาตรของโมเลกุลและใช้เทอมที่เป็นอนุพันธ์ของพลังงาน (Gradient) ซึ่งสอดคล้องกับแรงเชิงอะตอมเข้าไปด้วย สำหรับการใช้งาน sGDML นั้นก็ทำได้ไม่ยากเพราะว่าตัวไลบรารี่และคำสั่งนั้นสั้นและกระชับ โดยเราสามารถใช้งานคำสั่งดังต่อไปนี้

นำเข้าชุดข้อมูล

```
sgdml_dataset_via_ase.py <xyz_dataset_file>
```

ฝึกสอนโมเดลและสร้าง Force Field

```
sgdml all <sgdml_dataset_file> <n_train> <n_validate> [<n_test>]
```

นำ Force Field ไปใช้ในการทำนายพลังงานของโมเลกุล

```
import numpy as np
from sgdml.predict import GDMLPredict
from sgdml.utils import io

```

```
model = np.load('model.npz')
gdml = GDMLPredict(model)

r, _ = io.read_xyz('geometry.xyz')
e, f = gdml.predict(r)
```

รายละเอียดเพิ่มเติมเกี่ยวกับ sGDML นั้นสามารถดูได้ที่เว็บไซต์ http://www.sgdml.org/

12.6 ΔML

งานวิจัยที่ใช้ผลต่างระหว่างผลการคำนวณด้วยวิธีที่มีความแม่นยำต่ำและวิธีที่มีความแม่นยำสูงมาเป็นค่าสำหรับการเพิ่มความถูกต้อง (Correction) เพื่อช่วยให้การคำนวณหรือทำนายคุณสมบัติของโมเลกุลมีค่าแม่นยำมากยิ่งขึ้นนั้นนั้นมีมานานแล้ว[283,284,285] โดยไอเดียนี้ก็ได้ถูกนำมาใช้ในงานวิจัย ML เช่นเดียวกัน ซึ่งเทคนิคนี้เรียกว่า Delta-ML (ΔML)

ΔML เป็นเทคนิคที่ใช้ค่าความแตกต่างระหว่างค่าอ้างอิง (Reference หรือจะเรียก Label ก็ได้) จากวิธีการคำนวณทางควอนตัมที่มีความแม่นยำแตกต่างกันมาใช้ในการเทรนโมเดล (จึงเป็นที่มาว่าทำไมถึงเรียกว่า Detla) โดยความแตกต่างนั้นอาจจะแตกต่างกันในบริบทของระดับ (Level) ของวิธีการ เช่น

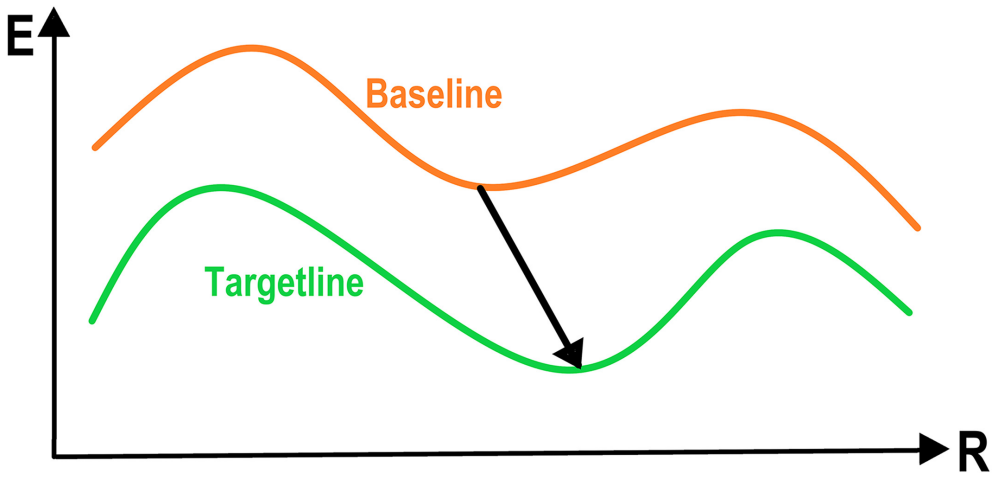
- ใช้วิธี DFT เหมือนกันแต่ว่าใช้ Basis Set ที่ต่างกัน
- หรือจะใช้วิธีที่แตกต่างกันก็ได้ เช่น ใช้วิธี DFT และวิธี Coupled Cluster (ใช้ Basis Set เดียวกัน)

การทำ Correction แบบนี้ด้วย ΔML สามารถช่วยให้โมเดลเพิ่มความสามารถในการถ่ายโอนการเรียนรู้ในการทำนาย (Transferability) ไปยังชุดข้อมูลทดสอบที่ต้องการทำนายได้อย่างถูกต้องและแม่นยำมากขึ้น โดยจะมีความถูกต้องเทียบเคียงกับการใช้วิธีแบบดั้งเดิมที่มีความแม่นยำสูง เช่น Post-HF ตามที่ได้อธิบายไป ดังเช่นที่แสดงในภาพที่ 12.5 ซึ่งเป็นการใช้ ΔML ในการทำนายคุณสมบัติของโมเลกุล (E) ในปริภูมิเคมี (R)

ตัวอย่างของการใช้ ΔML คือการใช้ค่าความแตกต่างของพลังงานที่ได้จากการคำนวณด้วยวิธี DFT และ CCSD(T) ซึ่งวิธี CCSD(T) นี้เปรียบเสมือนเป็นวิธีมาตรฐานที่ให้ผลที่มีความแม่นยำสูงมาก (วิธี CCSD(T) ได้ฉายาว่าเป็น Gold Standard Method) โดยมี Correction ดังนี้

$$y_{\text{DFT}} - y_{\text{CCSD(T)}} \tag{12.1}$$

โดย Raghunathan Ramakrishnan และทีมวิจัยได้นำเทอมในสมการที่ (12.1) มาใช้ในการฝึกสอนโมเดล ML ซึ่งได้รับความสนใจในช่วงเวลาต่อมา[286] นอกจากนี้ยังมีงานวิจัยที่พัฒนา Graph Neural Network



ภาพ 12.5 แสดงค่าที่เป็น Baseline ซึ่งได้จากวิธีการทำนายที่ต่ำกว่าและค่าที่เป็น Targetline ซึ่งได้จากวิธีการทำนายที่สูงกว่า โดยค่าความแตกต่างระหว่างทั้งสองวิธีนี้นั้นจะถูกเรียนรู้โดยโมเดล ΔML และนำมาใช้ในการเป็น Correction ต่อไป

(GNN) และประยุกต์ใช้เข้ากับเทคนิค Δ-learning สำหรับการประมาณค่าพลังงานส่วนต่างของเทอม Triple Excitation (Electronic Correction) เพื่อเพิ่มความแม่นยำในการทำนายค่าพลังงานให้มีความแม่นยำในระดับเดียวกันหรือใกล้เคียงกับวิธี CCSD(T)[287]

จริง ๆ แล้ว ΔML ก็เป็นเทคนิคอันหนึ่งที่มีแนวคิดมาจากความพยายามที่ต้องการจะทำให้โมเดลสามารถเรียนรู้ได้จากค่าความผิดพลาด (Error) โดยเริ่มมีการเอามาใช้กันมากขึ้นในช่วงปีที่ผ่านมา (ในช่วงแรกถูกใช้เยอะแค่ในเฉพาะกลุ่มวิจัยในโซนยุโรป สำหรับการเอามาทำนายพลังงานและเกรเดียนต์ของพลังงาน (Energy Gradient) ซึ่งก็สอดคล้องกับแรงของแต่ละอะตอมในโมเลกุล รวมไปถึง Stationary Points บนพื้นผิวพลังงานศักย์ โครงสร้างที่สภาวะทรานซิชั่น (Transition State Structure) และที่สภาวะสมดุลด้วย

12.7 Graph Neural Network

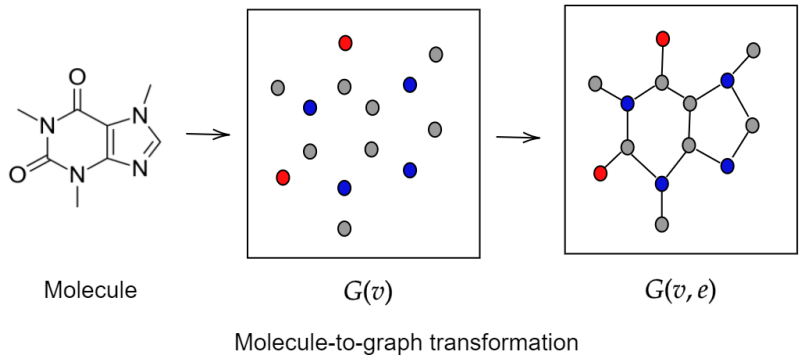
โครงข่ายประสาทแบบกราฟ (Graph Neural Network หรือ GNN) เป็นโครงข่ายประสามแบบหนึ่งซึ่งจะมองความสัมพันธ์ภายในโครงสร้างข้อมูลให้อยู่ในรูปแบบของกราฟแบบ 2 มิติ โดยไอเดียนี้ได้ถูกเสนอตั้งแต่ปี ค.ศ. 2008[288,289] โมเดล GNN นั้นได้รับความนิยมสูงมากในช่วงไม่กี่ปีที่ผ่านมานี้ (นับตั้งแต่ปี ค.ศ. 2018) โดย George Karypis นักวิทยาศาสตร์อาวุโสของบริษัท AWS ได้กล่าวไว้ในการสัมมนาครั้งหนึ่งว่า

“GNNs are one of the hottest areas of deep learning research, and we see an increasing number of applications take advantage of GNNs to improve their performance”

ซึ่งถ้าหากเราไปดูจำนวนบทความงานวิจัยที่ตีพิมพ์ในช่วงที่ผ่านมาจะพบว่า GNN ได้ถูกนำมาใช้เยอะมาก นอกจากนี้ยังได้มีบริษัทชั้นนำทางด้านเทคโนโลยีต่าง ๆ ที่ให้ความสำคัญและความสนใจกับการพัฒนาโมเดล

GNN เป็นอย่างมาก โดยหนึ่งในตัวอย่างที่ชัดเจนก็คือทีม AWS AI ได้พัฒนาไลบรารี่สำหรับ GNN โดยเฉพาะ ชื่อว่า Deep Graph Library (DGL) โดยผู้อ่านสามารถดูรายละเอียดได้ที่ https://www.dgl.ai

สำหรับโมเดล GNN ที่ผู้อ่านจะได้ศึกษาในหัวข้อนี้ก็คือ Message Passing Neural Network ซึ่งเป็นหนึ่งในโมเดล GNN ที่มีประสิทธิภาพในการทำนายคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลสูงมาก



ภาพ 12.6 การแทนโมเลกุลด้วยกราฟ

ภาพ 12.6 แสดงกราฟฟิคของการเปรียบเทียบโมเลกุลและกราฟ ซึ่งจะเห็นได้ว่าการแปลงจากโมเลกุลไปเป็นกราฟนั้นสามารถทำได้ตรงไปตรงมาเพราะว่าเราสามารถแทนอะตอมด้วยโหนด (Node) หรือจุดยอดของกราฟ (Vertex) และแทนพันธะด้วยขอบ (Edge)

12.7.1 MatErials Graph Network

หนึ่งในโมเดล GNN ที่ได้มีประสิทธิภาพในการทำนายคุณสมบัติของโมเลกุลขนาดเล็กและโมเลกุลขนาดใหญ่คือโมเดล MatErials Graph Network (MEGNet)[290]

ผู้อ่านสามารถดูรายละเอียดของซอร์สโค้ดของโปรแกรม MEGNet ได้ที่ https://github.com/materialsvirtuallab/megnet

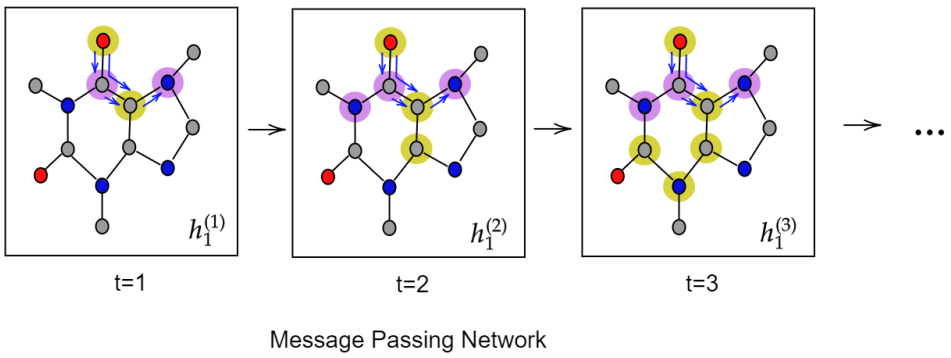
12.8 Message Passing Neural Network

โครงข่ายประสาทแบบการส่งข้อความ (Message Passing Neural Network หรือ MPNN) เป็น GNN ประเภทหนึ่งซึ่งถูกนำเสนอครั้งแรกเมื่อปี ค.ศ. 2017[177] โดยที่ MPNN แบบฉบับดั้งเดิมนั้นจะเป็นการใช้กราฟแบบที่ไม่มีการนำทิศทางหรือ Undirected Graph ซึ่ง MPNN นั้นประกอบไปด้วยเฟสหลัก 2 เฟส ดังนี้

- Message Passing Phase เฟสการส่งผ่านข้อมูลซึ่งอยู่ในรูปของข้อความ เป็นเฟสที่จะทำการขยับหรือแผ่กระจายข้อมูลไปทั่วทั้งกราฟเพื่อจำลองการเกิด Neural Network และการส่งต่อข้อมูลจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง

- Readout Phase เฟสการเรียกใช้ข้อมูล เป็นเฟสที่ Neural Representation ของกราฟจะถูกใช้ในการทำนายค่าของเอาต์พุต เช่น คุณสมบัติของแต่ละโหนด ซึ่งเปรียบเสมือนการทำนายคุณสมบัติของแต่อะตอมแต่ละตัวในโมเลกุลนั่นเอง

12.8.1 สถาปัตยกรรม



ภาพ 12.7 โครงข่ายของการส่งข้อความ

เรามาดูรายละเอียดของสถาปัตยกรรมของ MPNN กันครับ เริ่มที่เฟสแรกซึ่งเป็นการดำเนินการแผ่กระจายข้อมูลทั่วทั้งกราฟ โดยจำนวนก้าวที่ใช้ในการทำ Propagation นั้นจะเขียนแทนด้วย T ซึ่งจริง ๆ แล้วก็คือจำนวนรอบของการฝึกสอนโมเดล (Iteration) นั่นเอง นอกจากนี้เรายังกำหนดตัวแปรแทนพารามิเตอร์อื่น ๆ อีก ดังนี้

- h สถานะซ่อน
- m_v^t ข้อความของโหนด v ณ สเต็ปที่ t
- M ฟังก์ชันที่ใช้ในการอัพเดทข้อความ
- U_t ฟังก์ชันที่ใช้ในการอัพเดทโหนด ณ สเต็ปที่ t

ซึ่งฟังก์ชันที่ใช้ในการอัพเดทข้อความของแต่ละเสต็ปที่เกิดขึ้นในการกระบวน Propagation นั้นมีหน้าตาดังนี้

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h^t + w, e_{vw}) \tag{12.2}$$

ถ้าหากสังเกตสมการที่ (12.2) ดี ๆ จะพบว่าฟังก์ชัน m จะมีการดำเนินการนำข้อมูลของสถานะซ่อน (Hidden States) ของโหนด w เข้าไปกระทำกับโหนด v และยังมีการนำข้อมูลของขอบระหว่างโหนด e_{vw} เข้ามารวมไว้ในฟังก์ชันด้วย นอกจากนี้สมการทางคณิตศาสตร์ของ Hidden State สามารถเขียนให้อยู่ในรูปของฟังก์ชันที่ใช้อัพเดทสถานะของแต่ละโหนดได้ดังนี้

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{12.3}$$

โดยที่ $N(v)$ คือเซตของโหนดข้างเคียงของโหนด v ในกราฟ G และ h_v^0 คือฟังก์ชันเริ่มต้น (ฟังก์ชันอะไรก็ได้) สำหรับการกำหนดค่าเริ่มต้นของข้อมูลของโหนดนั้น ๆ (กำหนด Feature) x_v

$\phi_{1\to3} = f(v_1 \to v_3)$
$\psi_{2\to3} = f(v_2 \to v_3)$
Summarize messages: $\Omega = \phi_{1\to3}$ & $\psi_{2\to3}$

$h_v^{t+1} = \text{Update}(h_v^t, \Omega)$

เมื่อเรามีสถานะซ่อนของแต่ละโหนด ณ เวลาที่แตกต่างกันแล้ว (h_v^t) ลำดับถัดมาคือเราต้องการดำเนินการถดถอย (Regress) สถานะซ่อนทั้งหมดนี้เพื่อคำนวนความสัมพันธ์ไปยังค่าเอาต์พุตนั่นเอง ซึ่งกระบวนการหรือสิ่งที่เราจะนำมาใช้ในการทำดังกล่าวคือต้องใช้ฟังก์ชัน Readout R นั่นเอง หรือพูดง่าย ๆ คือเป็นฟังก์ชันที่ใช้ในการถอดข้อความออกมาเป็นเอาต์พุต เรียกว่าการทำนายสถานะซ่อน โดยมีสมการดังต่อไปนี้

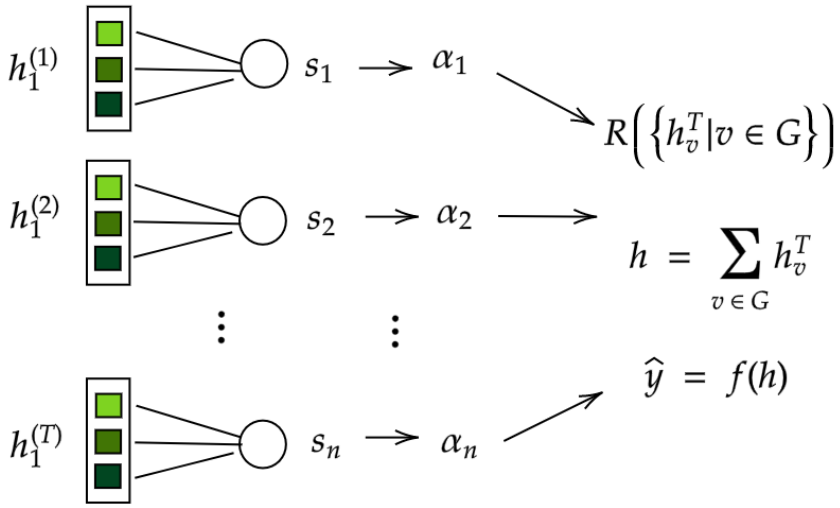
$$\hat{y} = R(h_v^T | v \in G) \tag{12.4}$$

ฟังก์ชัน Readout ที่เราจะนำมาใช้นั้นจะเป็นฟังก์ชันอะไรก็ได้ ขอแค่สามารถรวบรวม (Compose) ข้อความทั้งหมดที่อยู่ในรูปของสถานะซ่อนเข้าไว้ด้วยกันและทำการผสาน (Merge) ข้อความเหล่านั้นให้เป็น $\hat{y}$ ซึ่งวิธีที่ง่ายที่สุดที่สามารถนำมาใช้ในการรวมรวมสถานะต่าง ๆ เข้าไว้ด้วยกันก็คือการใช้ฟังก์ชันการรวมแบบเชิงเส้น (Linear Combination)

$$h = \sum_{v \in G} h_v^T \tag{12.5}$$

$$\hat{y} = f(h) \tag{12.6}$$

สุดท้ายแล้วจุดประสงค์ของการฝึกสอนโมเดล MPNN ก็คือฟังก์ชันที่รับสถานะของแต่ละโหนดเข้ามาที่ผ่านการแผ่กระจายมาจากโหนดอื่น ๆ โดยผ่านขอบระหว่างโหนด แล้วทำการรวมข้อมูลและแปลงให้เป็นคำตอบที่ต้องการทำนาย สมการที่ (12.6) ก็คือการฝึกสอนโมเดล Feed-forward Neural Network โดยใช้ฟังก์ชันแบบไม่เชิงเส้น (Nonlinear) f



ภาพ 12.8 การดำเนินการของการทำการถดถอย (Regression) และการเรียกหรือดึงข้อมูลโดยใช้ Readout

สำหรับ Representation ของโมเลกุลที่เราสามารถเลือกใช้เพื่อนำฝึกสอนโมเดล MPNN นั้นจะแบ่งออกได้ง่าย ๆ ตามลักษณะองค์ประกอบของกราฟ นั่นคือ *โหนด* ซึ่งเรามองเห็นอะตอม และ *ขอบ* ซึ่งเรามองเห็นพันธะระหว่างอะตอม ดังนี้

อะตอม

- ชนิดของอะตอม (Atom Type)
- ขั้นของอะตอม (Atom Degree)
- จำนวนเวเลนซ์อิเล็กตรอน (Valence Electrons)
- ประจุฟอร์มอล (Formal Charge)
- จำนวนอิเล็กตรอนอิสระ (Radical Electrons)
- ไฮบริไดเซชั่น (Hybridization) เช่น SP, SP^2, SP^3, SP^3D, SP^3D^2
- ความเป็นอะโรมาติก (Aromaticity)

- จำนวนไฮโดรเจนอะตอม (Hydrogen Atoms)
- จำนวนหมู่ฟังก์ชัน (Functional Groups)

พันธะเคมี

- ความยาวพันธะ
- ชนิดของพันธะ
  - พันธะเดี่ยว (Single Bond)
  - พันธะคู่ (Double Bond)
  - พันธะสาม (Triple Bond)
  - พันธะแบบอื่น ๆ เช่น พันธะของอะโรมาติก

นอกจากนี้ ในปี ค.ศ. 2021 นักวิจัยจากมหาวิทยาลัยเทคนิคแห่งเบอร์ลิน (Technical University of Berlin) ประเทศเยอรมัน นำโดย Kristof T. Schütt ได้ตีพิมพ์งานวิจัยซึ่งได้นำเสนออัลกอริทึมที่ชื่อว่า Equivariant Message Passing และสถาปัตยกรรมแบบใหม่ชื่อว่า Polarizable Atom Interaction Neural Network (PaiNN)[291] โดยวัตถุประสงค์ก็คือเป็นการแก้ปัญหาที่เกี่ยวข้องความสมมาตรของโมเลกุลของอัลกอริทึม MPNN แบบปกติ

ผู้อ่านสามารถศึกษาโค้ดสำหรับการคำนวณ Feature ของโหนดและขอบและโค้ดการสร้างโมเดล MPNN โดยใช้ TensorFlow และ Keras ได้ที่ https://github.com/rangsimanketkaew/mpnn

12.8.2 งานวิจัยที่เกี่ยวข้องกับ Message Passing Neural Network

เนื่องจากในปัจจุบันนั้นได้มีงานวิจัยที่ได้มีการนำ GNN มาใช้พัฒนาและต่อยอดเป็นจำนวนมาก ผู้เขียนจะขอยกตัวอย่างงานวิจัยที่ผู้เขียนคิดว่าน่าสนใจ โดยหนึ่งในนั้นก็คือ “การพัฒนา Descriptor โดยใช้กราฟสำหรับการทำนายคุณสมบัติของพอลิเมอร์”[292] ซึ่งในงานวิจัยนี้ผู้วิจัยได้เลือกใช้ Weighted Directed Message Passing Neural Network (wD-MPNN) ซึ่งเป็น GNN ประเภทหนึ่ง และ Descriptor ที่ได้พัฒนาขึ้นมานั้นจริง ๆ แล้วก็คือเป็นการนำ Molecular Graph ที่มีอยู่แล้วนำมาปรับเพิ่มความสามารถในการบ่งบอกหรือเป็นตัวแทน (Represent) ความเป็นพอลิเมอร์ให้ดียิ่งขึ้น[1] โดยในงานวิจัยนี้ผู้วิจัยได้เพิ่มพจน์ “Stochastic Edges” เข้าไปในโมเดล wD-MPNN เพื่อให้อธิบาย Repeating Unit ของพอลิเมอร์ดียิ่งขึ้น (พอลิเมอร์เกิดจากมอนอเมอร์ซ้ำ ๆ กัน หลาย ๆ ตัวมาต่อกัน) สิ่งที่น่าสนใจคือปกติแล้ว GNN จะมีประสิทธิภาพในการทำนายคุณสมบัติของโมเลกุลในกรณีที่เป็น Isolated Molecule (โมเลกุลที่อยู่ตัวเดียวโดด ๆ) ได้ดีมาก ๆ แต่กรณีของพอลิเมอร์ที่เป็นโครงข่ายที่มีขนาดใหญ่มากนั้น ความท้าทายคือเราจะจัดการกับอันตรกิริยาหรือปฏิสัมพันธ์ระหว่าง Repeating Unit อย่างไร ? สรุปโจทย์ง่าย ๆ ก็คือเราจะสอนให้โมเดล GNN รู้ได้อย่างไร

[1] บทความงานวิจัย A Graph Representation of Molecular Ensembles For Polymer Property Prediction โดย Matteo Aldeghi และ Connor W. Coley

ว่ามีโมเลกุลอื่น ๆ อีกหลายโมเลกุลที่มาต่อหรือสร้างพันธะกับโมเลกุลเริ่มต้นออกไปทั้งทางซ้ายและขวาเพื่อให้มีความเป็นพอลิเมอร์มากที่สุด

สิ่งที่งานวิจัยชิ้นนี้ทำนายก็คือค่า Electron Affinity (EA) กับ Ionization Potential (IP) ของโมเลกุลโคพอลิเมอร์ (Copolymer) จำนวนทั้งหมด 42,966 พอลิเมอร์ ซึ่งค่าอ้างอิงของ EA กับ IP นั้นก็ถูกคำนวณโดยใช้ทฤษฎี DFT ซึ่งโดยตัวส่วนตัวแล้วผู้เขียนคิดว่างานวิจัยนี้ยังสามารถต่อยอดได้อีกเยอะเพราะว่าจริง ๆ แล้วยังมีคุณสมบัติอื่น ๆ ของพอลิเมอร์ที่ท้าทายกว่า EA กับ IA มาก เช่น คุณสมบัติเชิงกล (Mechanical Properties) ซึ่งถ้าหากเราสามารถพัฒนาโมเดล ML ที่ทำนายคุณสมบัติเหล่านี้ได้อย่างแม่นยำ ก็น่าจะเป็นประโยชน์ต่อฝั่งนักทดลองพอลิเมอร์แน่นอน ถ้าหากผู้อ่านสนใจรายละเอียดเพิ่มเติมก็ไปลองอ่านงานวิจัยฉบับเต็มได้ บทความฉบับนี้อ่านง่าย ไม่ลงทฤษฎีเยอะมาก

12.9 Generative Adversarial Network

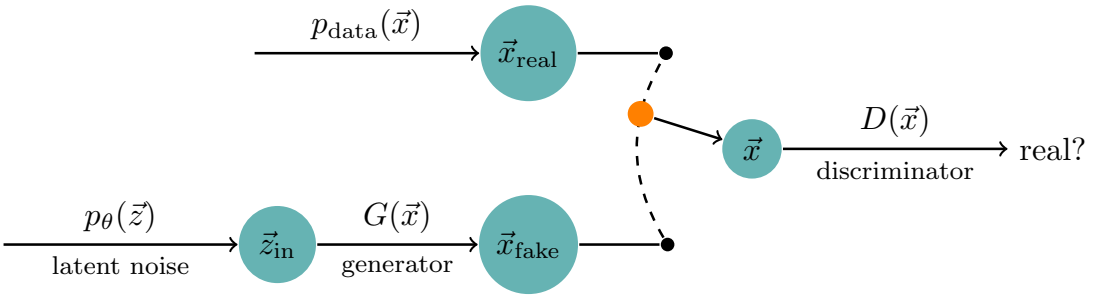
Generative Adversarial Network (GAN) เป็น Generative Model แบบหนึ่งซึ่งเป็น Unsupervised ML โดยผู้ที่นำเสนอไอเดียของ GAN ก็คือ Ian Goodfellow โดยได้เผยแพร่บทความวิชาการในปี ค.ศ. 2014[293] สำหรับผู้อ่านที่สนใจรายละเอียดของ GAN และแอพพลิเคชั่น ผู้เขียนแนะนำให้อ่านบทความ “18 Impressive Applications of Generative Adversarial Networks (GANs)” บทเว็บไซต์ Machine Learning Mastery[1]

โดยรูปแบบโครงสร้างของ GAN ประกอบไปด้วยองค์ประกอบหลัก 2 ส่วน ดังนี้

1. Generator คือโมเดลที่ถูกฝึกสอนให้มีความสามารถในการการสร้างตัวอย่างขึ้นมาใหม่โดนการสุ่มความแปรปรวนจากข้อมูลเริ่มต้น
2. Discriminator คือโมเดลที่ถูกฝึกฝนให้มีความสามารถในการการจำแนกตัวอย่างจริงหรือตัวอย่างปลอมออกจากกัน

หลักการของ GAN คือโมเดลตัวแรกซึ่งเรียกว่า Generator จะสร้างข้อมูล (ปลอม) ที่มีลักษณะใกล้เคียงกันกับข้อมูลจริงขึ้นมา ในขณะที่โมเดลอีกตัวซึ่งเรียกว่า Discriminator จะทำการแยกประเภทของข้อมูลที่เราต้องการโดยประเมินการทำงานของโมเดลตัวแรกโดยการตัดสินว่าข้อมูลที่ Generator สร้างขึ้นมานั้นมีความใกล้เคียงกับข้อมูลจริงมากน้อยแค่ไหน อธิบายให้ง่ายกว่านี้ก็คือโมเดลทั้งสองตัวนี้ถูกฝึกสอนให้ต่อสู้กันโดยที่ Generator นั้นพยายามสร้างข้อมูลหลอก Discriminator และ Discriminator ก็ต้องพยายามจับผิด Generator ให้ได้ โดยในภาพที่ 12.9 แสดงองค์ประกอบของ GAN เริ่มต้นก็คือเรากำหนด Latent Noise ขึ้นมาก่อน แล้วให้ Generator นั้นสร้างข้อมูลปลอม ($\vec{x}_{\text{fake}}$) โดยเรียนรู้จากข้อมูลจริง ($\vec{x}_{\text{real}}$) แล้ว Discriminator จะทำตรวจสอบต่อไป

[1] https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks



ภาพ 12.9 โมเดล Generative Adversarial Network (GAN) ซึ่งประกอบไปด้วย Generator และ Discriminator ซึ่งทั้งสองโมเดลเป็น Neural Network

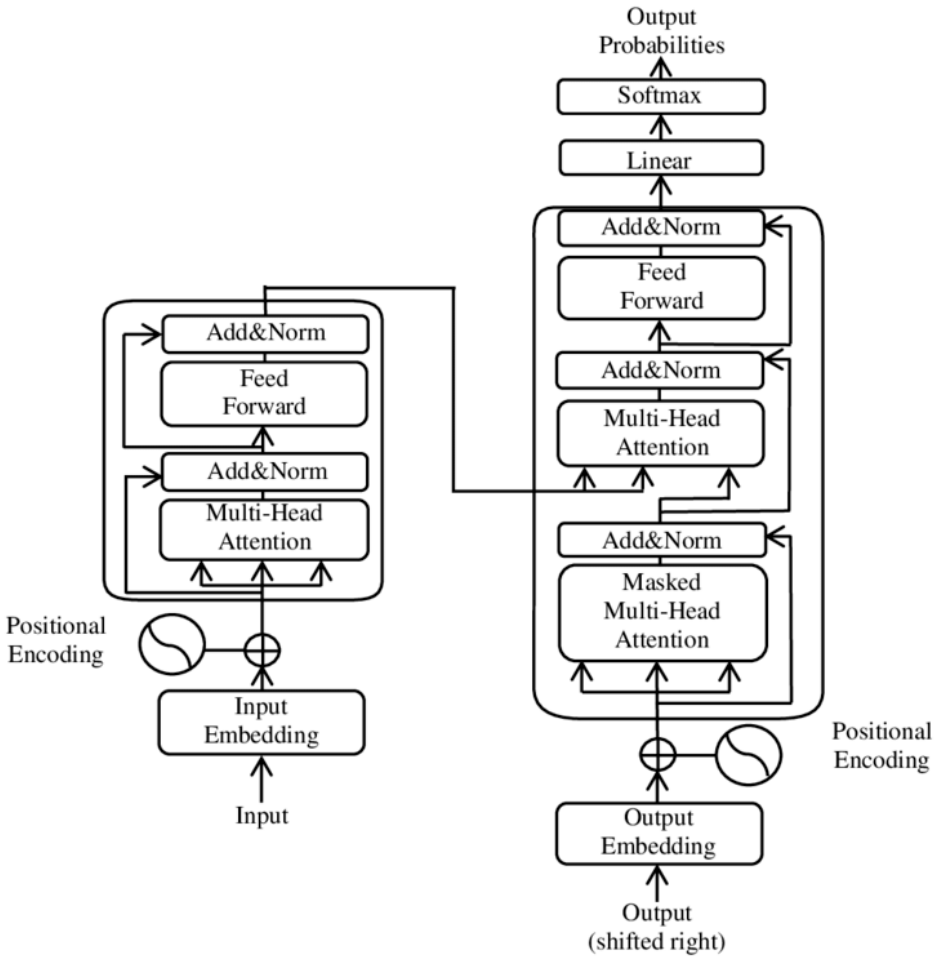
ในปี ค.ศ. 2018 ได้มีการพัฒนา DeepFake ซึ่งเป็นปัญญาประดิษฐ์ที่สามารถสร้างภาพเคลื่อนไหวของบุคคลที่เราต้องการ โดยสามารถเลียนแบบทั้งท่าทาง, น้ำเสียง, ลักษณะการพูด, การขยับปาก แล้วนำข้อความที่เป็นเท็จใส่ลงไปให้เหมือนกับว่าเป็นคำพูดของบุคคลคนนั้น ทั้ง ๆ ที่บุคคลนั้นไม่ได้พูดประโยคเหล่านี้ออกมา สำหรับงานวิจัยทางด้านเคมีนั้นก็ได้มีการนำ GAN มาใช้ด้วยเช่นกัน เช่น การค้นหาโมเลกุลชนิดใหม่,[294,295,296] การทำนายโครงสร้างผลึกที่สภาวะต่าง ๆ,[297] การออกแบบสารประกอบอนินทรีย์,[298] และการปรับปรุงการทำนายสเปกตรัม[299]

สำหรับงานวิจัยที่ใช้ GAN ที่ผู้เขียนจะยกมาให้ผู้อ่านได้ศึกษานั้นคือ “Generative Adversarial Networks for Transition State Geometry Prediction”[300]

12.10 Transformer

ทรานส์ฟอร์มเมอร์ (Transformer)[301] เป็นโมเดล Neural Network ที่ถูกพัฒนาขึ้นมาเพื่อใช้ในการเรียนรู้ข้อมูลที่เป็นลำดับ (Sequential Input) เช่น ข้อความหรือประโยค ซึ่งเหมือนกับโมเดล Recurrent Neural Network (RNN) ซึ่งมีประโยชน์อย่างมากโดยเฉพาะการประยุกต์เพื่อการแปลภาษาจากภาษาหนึ่งไปยังอีกภาษาหนึ่ง รวมไปถึงการสรุปใจความสำคัญของประโยคหรือย่อหน้ายาว ๆ อย่างไรก็ตาม ปัญหาอย่างหนึ่งที่เป็นปัญหาสำคัญของการใช้ RNN ก็คือเรียนรู้ข้อมูลที่มีความต่อเนื่องกันยาว ๆ อย่างเช่นข้อความที่มีความซับซ้อนได้ไม่ดีนัก นั่นก็เพราะว่าสาเหตุก็คือข้อมูลที่ถูกส่งต่อจากอินพุตลำดับหนึ่งไปยังลำดับต่อ ๆ ไปนั้นสูญหายไประหว่างทาง จึงทำให้ประสิทธิภาพของการเรียนรู้ในการส่งต่อข้อมูลนั้นทำได้ไม่ดีนัก

ด้วยเหตุนี้จึงเป็นที่มาของโมเดล Transformer ที่ถูกเสนอในปี ค.ศ. 2017 โดยทีมนักวิจัยจาก Google Brain ซึ่งได้ตีพิมพ์บทความงานวิจัยเรื่อง “Attention Is All You Need”[301] ซึ่งไอเดียของ Transformer ก็คือการใช้ Fully-Connected Neural Network แบบทั่วไปแล้วก็ใช้เพียงกลไกการใส่ใจ (Attention Mechanism) ซึ่งเป็นหัวใจหลักของโมเดลอันนี้ลยก็ว่าได้ โดย Attention Mechanism ได้เข้ามาช่วยทำให้ประสิทธิภาพของโมเดลนั้นดีกว่าโมเดล Sequence to Sequence (Seq2Seq) ที่ใช้ RNN นอกจากนี้ Transformer ยังมีข้อดีคือฝึกสอนโมเดลได้เร็วกว่า ใช้ทรัพยากรน้อยกว่า (ทำงานกับ GPU ได้ดีกว่า RNN ด้วย) และตีความการทำงานภายในได้ง่ายขึ้น รวมไปถึงการแก้ปัญหา Vanishing Gradient ที่มักจะพบเจอใน RNN



ภาพ 12.10 สถาปัตยกรรมของโมเดล Transformer (เครดิตภาพ: https://en.wikipedia.org/wiki/Transformer_(machine_learning_model))

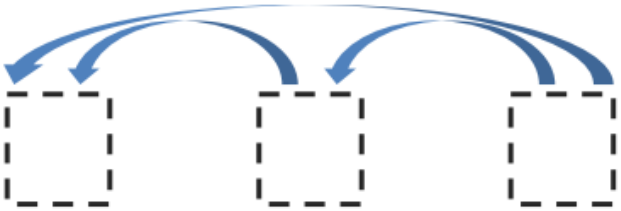
ภาพที่ 12.10 แสดงส่วนประกอบของโมเดล Transformer โดยมี Encoder ที่รับอินพุตที่เป็นลำดับข้อมูลเข้ามาและมี Decoder ที่รับเอาต์พุตที่เป็นลำดับ โดยในขั้นตอนการฝึกสอนโมเดลนี้ ลำดับข้อมูลของเอาต์พุตจะถูกเลื่อนไปทางขวาหนึ่งตำแหน่งซึ่งเป็นการฝึกสอนแบบ Teacher Forcing นั่นเอง

12.10.1 Attention

ตามที่ได้อธิบายไปก่อนหน้านี้ว่าเราสามารถออกแบบโมเดล Neural Machine Translation ให้มีความสามารถในการเรียนรู้ข้อมูลที่มีลำดับยาว ๆ ได้ด้วย Attention Mechanism โดยโมเดลจะโฟกัสเฉพาะบางส่วนของอินพุตเท่านั้น ณ เวลาหนึ่ง ซึ่งพารามิเตอร์ที่จะกำหนดการโฟกัสของโมเดลว่าจะโฟกัสที่ส่วนไหนนั้นก็จะได้จากการฝึกสอนโมเดลนั่นเอง

ประเภทของ Attention ที่ถูกนำมาใช้นั้นมีด้วยกัน 3 ประเภท ดังนี้

ภาพ 12.11 Self-Attention (เครดิตภาพ: Łukasz Kaiser)



ภาพ 12.12 Masked Self-Attention (เครดิตภาพ: Łukasz Kaiser)

ภาพ 12.13 Encoder-Decoder Attention (เครดิตภาพ: Łukasz Kaiser)

1. แบบแรกคือ Self-Attention โดยทุกหน่วยจะทำ Attention กับทุกหน่วย

2. แบบที่สองคือ Masked Self-Attention โดยที่แต่ละหน่วยจะทำ Attention กับเฉพาะข้อมูลที่อยู่ลำดับก่อนหน้าเท่านั้น โดย Attention ชนิดนี้จะใช้กับ Decoder เนื่องจากกระบวนการสร้างเอาต์พุตในตอนใช้งานจริงเป็นการสร้างทีละตัวจึงไม่สามารถนำข้อมูลจากอนาคตมาใช้ได้

3. แบบที่สามคือ Encoder-Decoder Attention เป็น Attention ที่เหมือนกับที่ใช้ในโมเดล seq2seq แบบทั่วไป โดยฝั่ง Decoder จะเป็นตัวเรียกหรือดึงข้อมูล (Query) ข้อมูลมาจากฝั่ง Encoder

12.10.2 Molecule Attention Transformer

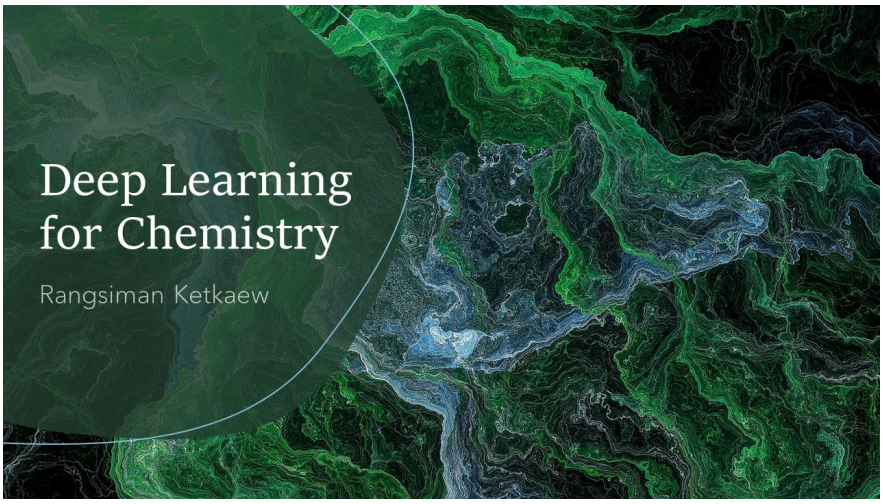
ตัวแปลงความใส่ใจของโมเลกุล (Molecule Attention Transformer หรือ MAT)[302] เป็นโมเดลที่ใช้ Transformer ในการพัฒนา

12.11 โมเดลอื่น ๆ

นอกจากโมเดล ML ที่ได้อธิบายไปก่อนนี้ ยังมีอีกหลายโมเดลที่ถูกพัฒนาขึ้นมาเพื่อวัตถุประสงค์ที่แตกต่างกันไป โดยเฉพาะการทำนายคุณสมบัติของโมเลกุลแบบเฉพาะเจาะจง โมเดล ML ที่ผู้อ่านสามารถศึกษาเพิ่มเติมเองได้มีดังนี้

ตาราง 12.1 โมเดล ML สำหรับเคมีควอนตัม

ปี ค.ศ. ที่ตีพิมพ์	โมเดล ML	รายละเอียด
2018	DeePMD-kit[303]	สำหรับคำนวณ Representation
2019	Cormorant[304]	สำหรับ Many-body System
2020	GMsNN[305]	ใช้ Gaussian Moment
2021	FieldSchNet[306]	ปรับปรุง SchNet สำหรับการจำลองตัวทำละลาย
2022	DimeNet[307]	ใช้ Message-passing



ภาพ 12.14 การเรียนรู้เชิงลึกสำหรับเคมี (TensorFlow)

ผู้อ่านที่ต้องการศึกษาการเขียนโค้ด Deep Learning สำหรับทำนายคุณสมบัติของโมเลกุลสามารถดูเพลลิสต์ที่ผู้เขียนได้จัดทำไว้บน YouTube ที่เว็บไซต์ https://www.youtube.com/playlist?list=PLt-twymrmZ2f5aDzxlmVMKb0-EAkF0KwH

บทที่ 13

ไลบรารี่การเรียนรู้ของเครื่องสำหรับเคมีควอนตัม

13.1 ไลบรารี่สำหรับคำนวณลักษณะเฉพาะเชิงโครงสร้าง

ลักษณะเฉพาะเชิงโครงสร้างของโมเลกุลไม่มีความซับซ้อนและสามารถคำนวณออกมาได้ง่าย จริง ๆ แล้วเราอาจจะเขียนสคริปต์สำหรับคำนวณลักษณะเฉพาะได้โดยที่ไม่ต้องใช้ไลบรารี่เลยก็ได้ อย่างไรก็ตาม ในกรณีที่เราต้องการที่จะคำนวณลักษณะเฉพาะเชิงโครงสร้างหลาย ๆ ตัวสำหรับหลาย ๆ โมเลกุลพร้อมกัน การใช้ไลบรารี่ก็จะสะดวกกว่าในกรณีแบบนี้

13.1.1 RDkit

ไลบรารี่ RDKit[308] เป็นไลบรารี่สำหรับงานทางด้านเคมีสารสนเทศน์ (Cheminformatics)[1] ได้รับความนิยมเป็นอย่างมากเนื่องจากสามารถคำนวณลักษณะเฉพาะเชิงโครงสร้างได้หลากหลาย[2] ตัวอย่างโค้ดสำหรับการใช้งาน RDKit สามารถดูได้ที่ https://www.rdkit.org/docs/Cookbook.html โดยจะมีโค้ดสำหรับการเปลี่ยนข้อมูลเชิงโมเลกุลจากโครงสร้างไปเป็น SMILES และโค้ดสำหรับการแสดงข้อมูลเชิงโครงสร้างของโมเลกุล เช่น จำนวณของอะตอมแต่ละชนิด, จำนวนพันธะคู่, และจำนวนวงเบนซีน

[1]Cheminformatics หรือเรียกอีกอย่างว่า Chemoinformatics เป็นทฤษฎีทางเคมีเชิงฟิสิกส์ที่ใช้คอมพิวเตอร์และข้อมูลสารสนเทศน์ (Informatics) หรือที่เรียกในภาษาละตินว่า *in silico* มาใช้ในการแก้ปัญหาทางเคมี โดยมีการประยุกต์ใช้ Cheminformatics ทั้งในชีววิทยาและเคมี นอกจากนี้ยังถูกใช้ในงานวิจัยที่ศึกษาการค้นหาตัวยา (Drug Discovery) อีกด้วย

[2]https://www.rdkit.org

13.2 ไลบรารี่สำหรับคำนวณลักษณะเฉพาะเชิงอิเล็กทรอนิกส์

ลักษณะเฉพาะเชิงอิเล็กทรอนิกส์ (Molecular Electronic Feature) เช่น Electron Density, Electrostatic Map, และ Frontier Molecular Orbitals (FMOs) มักจะถูกนำมาใช้ในการสร้างชุดข้อมูลเพื่อฝึกสอนโมเดลเพราะว่าคุณสมบัติเหล่านี้เป็นคุณสมบัติเชิงอิเล็กทรอนิกส์ที่มักจะถูกคำนวณออกมาโดยปกติอยู่แล้วด้วยโปรแกรมทางเคมีควอนตัมเชิงคำนวณ (Computational Quantum Chemistry Software)[1] อย่างไรก็ตามลักษณะเฉพาะเหล่านี้อาจจะยังไม่สามารถที่จะอธิบายคุณลักษณะเชิงอิเล็กทรอนิกส์บางอย่างที่ซับซ้อนของโมเลกุลได้ดีเท่าที่ควร เช่น คุณสมบัติที่เกี่ยวข้องกับควอนตัม ดังนั้นจึงมีลักษณะเฉพาะอื่น ๆ ที่ถูกพัฒนาขึ้นมาโดยรวมข้อมูลเชิงควอนตัมของเข้าไปด้วยเพื่อให้มีความถูกต้องมากขึ้น ซึ่งการที่จะคำนวณลักษณะเฉพาะเหล่านี้ก็มีความซับซ้อนอยู่มิใช่น้อย ดังนั้นจึงได้มีผู้พัฒนาไลบรารี่ที่สามารถคำนวณพารามิเตอร์เหล่านี้ให้เราได้

ตาราง 13.1 รายชื่อและเว็บไซต์ของไลบรารี่ Dataset และ Feature สำหรับเคมีควอนตัม

ไลบรารี่	เว็บไซต์	อ้างอิง
AFLOWLIB	https://aflow.org/aflow-ml	Curtarolo และคณะ[309]
Materials Project	https://materialsproject.org	Jain และคณะ[310]
OQMD	https://oqmd.org	Kirklin และคณะ[311]
libAtoms	https://github.com/libAtoms	Bartók และคณะ[312]
Matminer	https://github.com/markovmodel/deeptime	Ward และคณะ[313]
Chemical VAE	https://ccs-psi.org/node/22	Gómez-Bombarelli และคณะ[182]
JARVIS-DFT	https://github.com/usnistgov/jarvis	Choudhary และคณะ[314]
DScribe	https://github.com/SINGROUP/dscribe	Himanen และคณะ[315]
NOMAD	https://analytics-toolkit.nomad-coe.eu	Draxl และ Scheffler[316]
OMDB	https://omdb.mathub.io/dataset	Olsthoorn และคณะ[317]
Khazana	https://khazana.gatech.edu	Chapman และคณะ[318]
MolDis	https://moldis.tifrh.res.in/index.html	Kayastha และ Ramakrishnan[319]

ตารางที่ 13.1 แสดงไลบรารี่หรือโปรแกรมแพคเกจที่มีชุดข้อมูลให้เรานำมาใช้ได้ รวมถึงไลบรารี่ที่สามารถคำนวณ Feature ของอะตอมและโมเลกุลได้ด้วย

13.2.1 DScribe

DScribe เป็นไลบรารี่ที่สามารถคำนวณลักษณะเฉพาะเชิงอิเล็กทรอนิกส์ของโมเลกุลที่ซับซ้อนได้เยอะมาก[315] สำหรับเวอร์ชันปัจจุบันสามารถคำนวณลักษณะเฉพาะดังต่อไปนี้[2]

[1] https://en.wikipedia.org/wiki/List_of_quantum_chemistry_and_solid-state_physics_software

[2] * สามารถคำนวณ Derivative ได้

- Coulomb Matrix*
- Sine Matrix
- Ewald Sum Matrix
- Atom-centered Symmetry Functions
- Smooth Overlap of Atomic Positions*
- Many-body Tensor Representation
- Local Many-body Tensor Representation
- Valle-Oganov Descriptor

คู่มือการใช้งานอ่านได้ที่ https://singroup.github.io/dscribe/latest/index.html

ตัวอย่างโค้ดของการใช้ DScribe ในการสร้าง Coulomb Matrix (CM)

```
from dscribe.descriptors import CoulombMatrix

atomic_numbers = [1, 8]
rcut = 6.0
nmax = 8
lmax = 6

# Setting up the CM descriptor
cm = CoulombMatrix(
    n_atoms_max=6,
)
```

ตัวอย่างโค้ดของการใช้ DScribe ในการสร้าง SOAP kernel

```
from ase.build import molecule

# Molecule created as an ASE.Atoms
water = molecule("H2O")

# Create SOAP output for the system
soap_water = soap.create(water, positions=[0])

# Create output for multiple system
samples = [molecule("H2O"), molecule("NO2"), molecule("CO2")]
positions = [[0], [1, 2], [1, 2]]
# Serial
```

```
coulomb_matrices = soap.create(samples, positions)
# Parallel
coulomb_matrices = soap.create(samples, positions, n_jobs=2)
```

จากโค้ดด้านบนนั้นสิ่งที่เราจะได้ออกมาก็คือเมทริกซ์ที่แต่ละแถวนั้นจะบ่งบอกถึง Local Environment ของแต่ละอะตอมในโมเลกุล โดยความยาวของ Feature Vector ของ SOAP นั้นจะขึ้นอยู่กับจำนวนของ Species ที่เรากำหนดและขึ้นอยู่กับค่า n_{max} และ l_{max} ด้วย โดยผู้อ่านสามารถลองเปลี่ยนค่าของ n_{max} และ l_{max} เพื่อดูว่ามีผลต่อการเปลี่ยนค่าของ Feature Vector อย่างไร

นอกจากนี้เรายังสามารถเรียกใช้ Method `derivatives` ของ Attribute `soap` เพื่อคำนวณ Derivative ของ SOAP ได้ด้วย ดังนี้

```
derivatives, descriptors = soap.derivatives(
    traj,
    positions=[[[0, 0, 0]]] * len(r),
    method="analytical"
)
```

13.3 ไลบรารี่สำหรับสร้างโมเดล

ตาราง 13.2 รายชื่อและเว็บไซต์ของไลบรารี่โมเดล ML สำหรับเคมีควอนตัม

ไลบรารี่/โมเดล	เว็บไซต์	อ้างอิง
AMP	https://amp.readthedocs.io/en/latest https://singroup.github.io/dscribe	Khorshidi and Peterson[320]
GAP	https://github.com/libAtoms/QUIP	Bartók และ Csányi[28]
SNAP	https://lammps.sandia.gov/doc/pair_snap.html https://github.com/materialsvirtuallab/snap	Thompson และคณะ[164]
AENET	http://ann.atomistic.net	Artrith และ Urban[321]
AGNI	https://lammps.sandia.gov/doc/pair_agni.html	Huan และคณะ[322]
PROPhet	https://github.com/biklooost/PROPhet	Kolb และคณะ[323]
TensorMol	https://github.com/jparkhill/TensorMol	Yao และคณะ[192]
ANI	https://github.com/isayev/ASE_ANI	Smith และคณะ[169]
COMP6	https://github.com/isayev/COMP6	Smith และคณะ[170]
DeePMD-kit95	https://github.com/deepmodeling/deepmd-kit	Wang และคณะ[303]
VAMPnet	https://github.com/markovmodel/deeptime	Mardt และคณะ[324]
CGCNN	https://github.com/txie-93/cgcnn	Xie และ Grossman[325]
ElemNet	https://github.com/dipendra009/ElemNet	Jha และคณะ[326]
OQMD-SC	https://github.com/dipendra009/ElemNet	Jha และคณะ[327]

13.3.1 SchNetPack

SchNetPack เป็นชุดโปรแกรมสำหรับการฝึกสอนโมเดล Neural Network สำหรับการทำนายคุณสมบัติเชิงอะตอม[328] ถูกเขียนโดยภาษา Python 100% และใช้ไลบรารี PyTorch เป็น Backend สำหรับการสร้างโมเดล SchNet ซึ่งเป็นตัวโมเดลหลักของ SchNetPack ผู้อ่านสามารถดาวน์โหลดซอร์สโค้ดของ SchNetPack และศึกษาวิธีการติดตั้งและใช้งานได้ที่เว็บไซต์ https://schnetpack.readthedocs.io

Feature หลัก ๆ ของ SchNetPack มีดังนี้

- รองรับการสร้างโมเดล SchNet ซึ่งถูกพัฒนาโดยใช้อัลกอริทึม Convolutional Neural Network (CNN) สำหรับโมเลกุลโดยเฉพาะ[143,144]
- รองรับการสร้างโมเดล PaiNN ซึ่งเป็น Equivariant Message-Passing สำหรับโมเลกุลเช่นเดียวกัน[291]
- สามารถทำนายค่าเอาต์พุตได้หลากหลาย เช่น Dipole Moments, Polarizability, Stress และคุณสมบัติอื่น ๆ ของโมเลกุล
- มีโมดูลสำหรับ Electrostatics และ Ewald Summation
- รองรับการเพิ่มความเร็วการคำนวณและฝึกสอนโมเดลด้วย GPU

13.3.2 sGDML

sGDML เป็นไลบรารี่ที่รับความนิยมในการสร้างโมเดลของโมเลกุลโดยการใช้ค่าพลังงานและแรงในการฝึกสอนโมเดล[329] คู่มือการใช้งานอ่านได้ที่ http://quantum-machine.org/gdml/doc

เราสามารถติดตั้ง sGDML โดยใช้ Python Package Manager เช่น PIP ได้ด้วยคำสั่งต่อไปนี้[1]

```
pip install sgdml
```

ตรวจสอบว่า sGDML ถูกติดตั้งและพร้อมใช้งานหรือไม่

```
(sgdml) rangsiman@linux:~$ which sgdml
/home/rangsiman/sgdml/bin/sgdml

(sgdml) rangsiman@linux:~$ sgdml
usage: sgdml [-h] [--version]
       {all,create,train,validate,select,test,show,reset} ...
```

[1]เพื่อป้องกันปัญหา Conflict ระหว่างไลบรารี่ใน Python Environment ผู้เขียนได้สร้าง Environment แยกขึ้นมาสำหรับ sGDML โดยเฉพาะซึ่งสามารถใช้ `venv` หรือ `conda` ในการสร้าง Environment ได้ครับ

```
sgdml: error: the following arguments are required: command
```

ตัวอย่างการใช้งาน Force Field ที่ฝึกสอนด้วย sGDML

เมื่อเราฝึกสอนโมเดลด้วย sGDML แล้วสิ่งที่เราจะได้ออกมาก็คือโมเดล Force Field ที่เราสามารถนำไปใช้ในในการจำลองโมเลกุล เช่น การจำลองพลวัตโมเลกุลหรือ Molecular Dynamics (MD) ได้ซึ่ง Force Field ก็เปรียบเสมือนเป็นสิ่งที่คำนวณพลังงานและแรงของโมเลกุลให้กับ MD เพื่อนำไปใช้ในการขยับหรือเปลี่ยนตำแหน่งของโมเลกุลในเฟรม (Frame) ต่อ ๆ ไปในการจำลอง โดยผู้อ่านสามารถดาวน์โหลดไฟล์โมเดล Force Field ที่ผ่านการฝึกสอนมาแล้วหรือ Pre-trained Model (`m_ethanol.npz`) ได้โดยใช้คำสั่ง

```
(sgdml) rangsiman@linux:~$ sgdml-get model
```

ตัวอย่างด้านล่างคือโค้ดที่ใช้ Force Field ที่ถูกฝึกสอนด้วย sGDML กับไลบรารี่ ASE ในการจำลองโมเลกุล Ethanol

```
from sgdml.intf.ase_calc import SGDMLCalculator

from ase.io import read
from ase.optimize import QuasiNewton
from ase.md.velocitydistribution import
    (MaxwellBoltzmannDistribution, Stationary, ZeroRotation)
from ase.md.verlet import VelocityVerlet
from ase import units

model_path = 'm_ethanol.npz'
calc = SGDMLCalculator(model_path)

mol = read('ethanol.xyz')
mol.set_calculator(calc)

# do a quick geometry relaxation
qn = QuasiNewton(mol)
qn.run(1e-4, 100)

# set the momenta corresponding to T=300K
MaxwellBoltzmannDistribution(mol, 300 * units.kB)
Stationary(mol) # zero linear momentum
ZeroRotation(mol) # zero angular momentum

# run MD with constant energy using the velocity verlet algorithm
dyn = VelocityVerlet(mol, 0.2 * units.fs, trajectory='md.traj')
    # 0.2 fs time step.

```

```
# function to print the potential, kinetic and total energy
def printenergy(a):
    epot = a.get_potential_energy() / len(a)
    ekin = a.get_kinetic_energy() / len(a)
    print('Energy per atom: Epot = %.3feV  Ekin = %.3feV
    (T=%3.0fK)  '
          'Etot = %.3feV' % (epot, ekin, ekin / (1.5 *
    units.kB), epot + ekin))

# now run the MD simulation
printenergy(mol)
for i in range(10000):
    dyn.run(10)
    printenergy(mol)
```

โดยผู้อ่านสามารถใช้พิกัดตำแหน่งเริ่มต้นของโมเลกุล Ethanol ดังต่อไปนี้ได้ (`ethanol.xyz`)

```
9

C     0.423139    0.365862    0.380202
C    -0.565742    0.883093   -0.769835
O     0.153111   -1.105280    0.316890
H     1.470913    0.558882    0.104617
H     0.067699    0.805186    1.280056
H    -0.194544    0.628689   -1.745578
H    -1.498634    0.279985   -0.631160
H    -0.685436    1.905989   -0.598842
H     0.107894   -1.518377    1.203705
```

13.3.3 PiNN

PiNN[330] เป็น Python Library ที่รวบรวมโมเดลและวิธีในการทำนายคุณสมบัติต่าง ๆ ของโมเลกุลเข้าไปด้วยกัน โดยในเวอร์ชันปัจจุบันนั้นมี Neural Network ที่ถูกติดตั้งไว้ใน PiNN แล้ว 2 โมเดลคือ PiNet และ Behler-Parrinello Neural Network

ผู้อ่านสามารถติดตั้งไลบรารี่ PiNN ได้โดยใช้คำสั่งต่อไปนี้

```
pip install git+https://github.com/Teoroo-CMC/PiNN
```

หรือ

```
git clone https://github.com/Teoroo-CMC/PiNN.git
cd PiNN && pip install -e .
```

รายละเอียดเพิ่มเติมดูได้ที่เว็บไซต์ https://github.com/Teoroo-CMC/PiNN และศึกษาคู่มือการใช้งานได้ที่ https://teoroo-cmc.github.io/PiNN

13.3.4 TorchANI

TorchANI[331] เป็น Neural Network ที่ใช้สำหรับการทำนายศักย์ (Potential) ของโมเลกุลซึ่งถูกพัฒนาโดยใช้ PyTorch Framework เป็นหลัก

การติดตั้ง TorchANI สามารถทำได้โดยต้องติดตั้ง PyTorch ก่อนแล้ว ดังนี้

```
# PyTorch
pip install numpy
DL_PT="https://download.pytorch.org/whl/nightly/cu100/torch_nightly.html"
pip install --pre torch torchvision -f $DOWNLOAD_PYTORCH

# TorchANI
pip install torchani
```

รายละเอียดเพิ่มเติมดูได้ที่เว็บไซต์ https://github.com/aiqm/torchani และศึกษาคู่มือการใช้งานได้ที่ https://aiqm.github.io/torchani

ภาคผนวก

ภาคผนวก A

พีชคณิตเชิงเส้น

1 สเกลาร์, เวกเตอร์, และเมทริกซ์

ปริมาณทางกายภาพสามารถแบ่งออกเป็น 3 ปริมาณ ได้แก่ สเกลาร์ (Scalar), เวกเตอร์ (Vector), และเมทริกซ์ (Matrix) โดยปริมาณแต่ละตัวที่ใช้ในคณิตศาสตร์มีความหมายง่าย ๆ ดังนี้

สเกลาร์ คือปริมาณที่มีเพียงขนาดอย่างเดียว โดยสามารถบอกแต่ขนาดอย่างเดียวก็ได้ความหมายสมบูรณ์ ไม่ต้องบอกทิศทาง กล่าวคือ เป็นแค่เพียงตัวเลขเดี่ยว ๆ เท่านั้น

เวกเตอร์ คือปริมาณที่ใช้ดำเนินการบนปริภูมิเวกเตอร์ (Vector Space) ซึ่งจะมีความหมายค่อนข้างกว้าง แต่จะมีนิยามคล้าย ๆ กับเวกเตอร์ในทางฟิสิกส์ กล่าวคือ เวกเตอร์จะมีทั้งขนาดและองค์ประกอบบ่งบอกทิศทาง ในส่วนของการเขียนโปรแกรมนั้นเวกเตอร์คือ Array ขนาด 1 มิติ

เครื่องหมายที่ใช้แทนเวกเตอร์มีดังต่อไปนี้

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \tag{A.1}$$

เมทริกซ์ คือปริมาณที่เกิดจากเวกเตอร์มากกว่า 1 เวกเตอร์มารวมกัน โดยจะมีองค์ประกอบเป็นจำนวนแถวและจำนวนหลัก ถ้าหากเมทริกซ์มีเพียงแค่แถวเดียว หรือหลักเดียว เราจะกล่าวได้ว่านั่นคือเวกเตอร์นั่นเอง ในส่วนของการเขียนโปรแกรมนั้นเมทริกซ์คือ Array ขนาด 2 มิติ

ตัวอย่างของเมทริกซ์มีดังต่อไปนี้

$$a = \begin{bmatrix} a^2 & 2a & 8 \\ 18 & 7a-4 & 10 \end{bmatrix} \tag{A.2}$$

$$b = \begin{bmatrix} a^2 & 2a & 8 \\ 18 & 7a-4 & 10 \end{bmatrix} \tag{A.3}$$

2 ประเภทของเมทริกซ์

เมทริกซ์แบบพิเศษมีด้วยกันหลากหลายแบบด้วยกัน โดยเมทริกซ์แบบพิเศษที่มักเจอมีดังต่อไปนี้

Zero Matrix หรือ Null Matrix เมทริกซ์ที่มีสมาชิกทุกตัวเป็น 0 หมด มีนิยามดังต่อไปนี้

$$0_{K_{m,n}} = \begin{bmatrix} 0_K & 0_K & \cdots & 0_K \\ 0_K & 0_K & \cdots & 0_K \\ \vdots & \vdots & \ddots & \vdots \\ 0_K & 0_K & \cdots & 0_K \end{bmatrix}_{m \times n} \tag{A.4}$$

ตัวอย่างของ Zero Matrix มีดังนี้

$$0_{1,1} = \begin{bmatrix} 0 \end{bmatrix} \tag{A.5}$$

$$0_{2,2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{A.6}$$

$$0_{2,3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{A.7}$$

Identity Matrix หรือ Unit Matrix ใช้สัญลักษณ์ I หรือ I_n เมทริกซ์ที่สมาชิกในแนวทแยง (Diagonal Elements) มีค่ากับเท่า 1 ทุกตัวและสมาชิกที่ไม่ได้อยู่ในแนวทแยง (Off-diagonal Elements) เป็น 0 ทั้งหมด ถ้าในกรณีที่สมาชิกในแนวทแยงอย่างน้อยหนึ่งตัวที่ไม่มีค่าเท่ากับ 1 แต่สมาชิกนอกแนวทแยงยังเป็น 0 อยู่ เราจะเรียกเมทริกซ์นั้นว่า เมทริกซ์แนวทแยง (Diagonal Matrix)

ตัวอย่างของ Identity Matrix ตามขนาด มีดังนี้

$$I_1 = \begin{bmatrix} 1 \end{bmatrix}, \tag{A.8}$$

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{A.9}$$

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{A.10}$$

$$\ldots, \tag{A.11}$$

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{A.12}$$

3 การดำเนินการของเมทริกซ์

Transpose ถ้าหากเรามีรูปแบบซึ่งจริง ๆ ก็คือเมทริกซ์ขนาด 2 มิติแล้วเราทำการคูณด้วยเมทริกซ์การหมุน (Rotation Matrix) สิ่งที่เราจะได้คือเราจะได้รูปภาพที่ถูกหมุนไป โดยการที่เรากระทำการหมุนเมทริกซ์นั้นเราเรียกว่า Transpose อธิบายง่าย ๆ คือการ Tranpose เมทริกซ์นั้นก็คือการสลับแถวกับหลักของเมทริกซ์ หรือทำการหมุนสมาชิกที่ไม่ใช่แถวทแยง (Off-diagonal) รอบ ๆ แนวทแยงนั่นเอง

การบวกและการลบ เมทริกซ์สองเมทริกซ์ที่มีขนาดเท่ากัน (จำนวนแถวและจำนวนหลักเท่ากัน) สามารถบวกและลบกันได้ โดยให้ทำการบวกหรือลบสมาชิกที่มีดัชนีตรงกันได้โดยตรงเลย

การคูณด้วยสเกลาร์ การคูณเมทริกซ์ด้วยปริมาณสเกลาร์สามารถทำได้ง่าย ๆ โดยให้คูณสมาชิกทุกตัวของเมทริกซ์ด้วยตัวเลขตัวนั้น

การคูณเมทริกซ์ด้วยเมทริกซ์แบบจุด (Dot Product) สมมติว่าเรามีเมทริกซ์ A กับเมทริกซ์ B การที่เมทริกซ์สองตัวนี้จะคูณกันได้นั้นจะต้องไม่ขัดกับเงื่อนไขดังต่อไปนี้ “สมาชิกของผลคูณของเมทริกซ์ในแถวที่ i หลักที่ j จะเกิดสมาชิกในแถวที่ i ของเมทริกซ์ที่อยู่หน้า คูณกับสมาชิกในหลักที่ j ของเมทริกซ์หลักเป็นคู่ ๆ แล้วนำมาบวกกัน”

การคูณเมทริกซ์ด้วยเมทริกซ์แบบอาดามาร์ (Hadamard Product) เป็นการคูณสมาชิกของเมทริกซ์ที่มีขนาดมิติเท่ากันและนำสมาชิกที่มีตำแหน่งตรงกันมาคูณกันโดยตรง

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \odot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 \cdot b_1 & a_2 \cdot b_2 \\ a_3 \cdot b_3 & a_4 \cdot b_4 \end{bmatrix} \tag{A.13}$$

4 เทนเซอร์

บางครั้งเราจำเป็นที่จะต้องจัดการข้อมูลที่มีจำนวนของมิติที่มากกว่า 2 มิติ นั่นคือเราไม่สามารถใช้เวกเตอร์หรือเมทริกซ์ได้อีกต่อไป โดยเราจะต้องใช้เทนเซอร์ (Tensor) แทน เพราะว่าเทนเซอร์คือ Array ที่มีจำนวนมิติ n มิติ สรุปง่าย ๆ คือเวกเตอร์นั้นคือเทนเซอร์ 1 มิติ และเมทริกซ์คือเทนเซอร์ 2 มิติ แล้วถ้าเป็น 3 มิติล่ะ เราจะเรียกว่าเป็นคิวบ์ (Cube) และเทนเซอร์ 4 มิติ เราก็จะเรียกว่าเป็นเวกเตอร์ของคิวบ์ และ 5 มิติก็จะเป็นเมทริกซ์ของคิวบ์นั่นเอง

ภาคผนวก B

ไลบรารี่การเรียนรู้ของเครื่อง

1 ไลบรารี่สำหรับการเรียนรู้ของเครื่องแบบทั่วไป

ในปัจจุบันได้มีการพัฒนาเครื่องมือหรือชุดโปรแกรม (Framework) สำหรับงานทางด้านสถิติ วิทยาศาสตร์ข้อมูล และปัญญาประดิษฐ์เป็นจำนวนมาก และแน่นอนว่าชุดเครื่องมือเหล่านี้ก็มีไลบรารี่สำหรับการเขียนโปรแกรมด้วยภาษาคอมพิวเตอร์ เช่น Python ซึ่งได้รับความนิยมเป็นอันดับนั่นก็เพราะตัวภาษามีไวยากรณ์ (Syntax) ที่เรียนรู้ได้ง่ายไม่ซับซ้อน จึงทำให้ผู้ที่หัดเขียนโปรแกรมเบื้องต้นเลือกใช้ภาษา Python ในการฝึกฝน

ไลบรารี่ที่ได้รับความนิยมในการเขียนโปรแกรมสำหรับการสร้างโมเดล การทำนายหรือพยากรณ์คำตอบและการวิเคราะห์ข้อมูล รวมไปถึงการนำเสนอข้อมูลในรูปแบบของกราฟและตารางนั้นมีหลายตัวด้วยกัน โดยผู้เขียนหยิบเลือกมาเฉพาะไลบรารี่ที่หลายคนเลือกใช้

NumPy มีชื่อเต็มคือ *Numerical Python* ชุดเครื่องมือสำหรับการทำงานเกี่ยวกับ N-dimensional Array (Numpy Array) ซึ่งเป็นโครงสร้างข้อมูลที่มีความสำคัญและถูกใช้งานบ่อยมาก นั่นก็เพราะว่า Array นั้นคือโครงสร้างหลักที่เราสามารถนำมาใช้ในการสร้างโครงสร้างข้อมูลประเภทต่าง ๆ เช่น เวกเตอร์ เมทริกซ์ รวมไปถึงเทนเซอร์ ซึ่งเราจะนำมาใช้ในการเก็บข้อมูลที่มีขนาดหลายมิติ เช่น Feature โดยสามารถเก็บให้อยู่ในรูปของเวกเตอร์แบบ 1 มิติหรือจะเป็นเมทริกซ์แบบ 2 มิติก็ได้ นอกจากนี้ NumPy ยังเป็นไลบรารี่พื้นฐานของไลบรารี่อื่น ๆ อีกมายมาก ไม่ว่าจะเป็น SciPy, Scikit-Learn และ SymPy เป็นต้น

SciPy มีชื่อเต็มคือ *Scientific Python* เป็นไลบรารี่ที่ถูกพัฒนาต่อยอดมาจาก NumPy โดยใช้ Array เป็นโครงสร้างข้อมูล โดย SciPy จะมีโมดูล (Module) ที่จะเน้นไปทางการคำนวณทางด้านคณิตศาสตร์และวิทยาศาสตร์ มีฟังก์ชันพื้นฐานให้เราเลือกใช้มากมาย เช่น พีชคณิตเชิงเส้น (Linear Algebra), สถิติและการวิเคราะห์ และการแก้สมการเชิงอนุพันธ์สามัญ (Ordinary Differential Equation)

Scikit-Learn ไลบรารี่ที่ถูกนำมาใช้ในการสร้างโมเดล ML แบบที่เป็น Regression และ Classification มากที่สุดตัวหนึ่ง โดยสามารถสร้างโมเดลได้ทั้งแบบ Supervised ML และ Unsupervised ML โดยมีโมเดลให้เราเลือกใช้งานหลายประเภท เช่น Linear Regression, Logistic Regression, Ridge Regression, Support Vector Machines, Random Forests และ Nearest Neighbors นอกจากนี้ Scikit-Learn ยังมีฟังก์ชันในการจัดการข้อมูล จัดแบ่งข้อมูลและวัดผลโมเดลด้วย อย่างไรก็ตาม Scikit-Learn ไม่ได้ถูกออกแบบมาเพื่อสร้างโมเดล Neural Network

Pandas ไลบรารี่ยอดฮิตที่หลายคนเลือกใช้เพื่อนำมาจัดการและวิเคราะห์ข้อมูลในรูปแบบของตาราง (คล้าย ๆ Excel) โดยมีการจัดเก็บโครงสร้างของข้อมูลในรูปแบบ DataFrame ซึ่งเปรียบเสมือกเป็นตารางสำหรับข้อมูล 2 มิติ โดย Pandas มีฟังก์ชันที่อำนวยความสะอวดในการจัดการข้อมูลในตาราง (Cell) และยังทำงานร่วมกับ NumPy ได้อีกด้วย

Matplotlib ไลบรารี่สำหรับพล็อตกราฟแบบต่าง ๆ ทั้ง 2 มิติและ 3 มิติ ซึ่งช่วยให้เราวิเคราะห์ข้อมูลได้ง่ายขึ้น โดยผู้ใช้งานสามารถนำเข้าข้อมูลประเภทแบบ List, Array หรือ DataFrame เช่น กราฟเส้น (Line Plots), กราฟแท่ง (Bar Charts), ฮิสโตแกรม (Histograms), แผนภูมิวงกลม (Pie Charts) และแผนภูมิกระจาย (Scatter Plots) รวมไปถึงพื้นผิว (Surface Plot)

Anaconda ชุดซอฟต์แวร์ที่ช่วยให้เราสามารถติดตั้งและจัดการไลบรารี่สำหรับ Python ที่กล่าวมาข้างต้นได้หมดทุกตัว โดยมีข้อดีคือเราสามารถสร้าง Environment หลาย ๆ อันสำหรับแต่ละโปรเจกต์ได้ และใน Environment นั้น ๆ เราก็สามารถติดตั้งไลบรารี่ด้วยเวอร์ชันที่เราต้องการได้ ช่วยให้แก้ปัญหาการขัดแย้งกันระหว่างเวอร์ชันที่ต่างกันของไลบรารี่ตัวเดียวกันได้

2 ไลบรารี่สำหรับการเรียนรู้เชิงลึก

ในการสร้างโมเดล Neural Network ที่มีความซับซ้อนนั้นไลบรารี่ที่กล่าวมาก่อนหน้านี้นั้นไม่สามารถทำได้ ดังนั้นเราจะต้องใช้ไลบรารี่ที่ถูกออกแบบมาโดยเฉพาะสำหรับการสร้างโมเดลและฝึกสอนโมเดล ซึ่งในปัจจุบันมีไลบรารี่ 2 ตัวที่ได้รับความนิยมสำหรับ Neural Network ดังนี้

TensorFlow ไลบรารี่ที่พัฒนาโดย Google สามารถใช้งานได้ฟรีและยังเป็นไลบรารี่แบบ Open-source เหมาะสำหรับการคำนวณเชิงตัวเลขที่รวดเร็ว เราสามารถใช้ TensorFlow สร้างโมเดล Neural Network และปรับแต่งโมเดลได้ตามต้องการเนื่องจากว่าตัว Framework นั้นยืดหยุ่นมาก นอกจากนี้ TensorFlow ยังรองรับการฝึกสอนโมเดลบนระบบคลัสเตอร์แบบ Distributed Memory และรองรับการใช้ GPU หลายตัว รวมไปถึงสามารถใช้งานร่วมกับ Tensor Processing Unit (TPU) ซึ่งถูกพัฒนาโดย Google เช่นเดียวกันได้อีกด้วย

PyTorch ไลบรารีที่พัฒนาขึ้นมาเพื่อให้ผู้ใช้งานสามารถเขียนโปรแกรม Neural Network ได้อย่างง่าย โดยในช่วงหลังได้รับการสนับสนุนโดย Meta (Facebook) โดยจุดประสงค์หลักของ PyTorch ก็คือพัฒนาเพื่อใช้กับคอมพิวเตอร์วิทัศน์ (Computer Vision) และการประมวลผลภาษาธรรมชาติ (Natural Language Processing) จุดเด่นของ PyTorch ก็คือสามารถใช้ GPU ในการเพิ่มความเร็วในการฝึกสอนได้อย่างมีประสิทธิภาพ ถ้าจะบอกว่า PyTorch เป็นคู่แข่งสำคัญของ TensorFlow ก็ไม่ผิดนัก

ไลบรารี่ทั้งคู่สามารถทำงานได้อย่างมีประสิทธิภาพพอ ๆ กัน[1] ซึ่งทั้งสองตัวก็มีทั้งข้อดีและข้อเสียต่างกัน ดังนั้นการเลือกใช้ไลบรารี่จึงขึ้นอยู่กับความถนัดและความชอบของแต่ละคน นอกจากนี้ไลบรารี่ทั้งคู่ยังรองรับ GPU ในการฝึกสอนโมเดลได้อย่างมีประสิทธิภาพอีกด้วย

3 การติดตั้งไลบรารี่

การติดตั้งไลบรารี่หรือชุดโปรแกรมสำหรับใช้ในการเขียนโปรแกรมด้วยภาษา Python นั้นสามารถทำได้ง่าย ๆ ผ่านตัวติดตั้งชุดโปรแกรมสองตัวซึ่งได้รับความนิยมมากที่สุด ดังนี้

pip เป็นตัวจัดการชุดชุดโปรแกรมของ Python ซึ่งเป็นตัวติดตั้งแบบมาตรฐาน โดยจะทำการเชื่อมต่อไปยัง Repository ของชุดโปรแกรมที่ชื่อว่า Python Package Index (PyPI) โดยฐานข้อมูลของ PyPI นั้นเข้าไปดูได้ที่ https://pypi.org

conda เป็นตัวจัดการชุดคำสั่งและดูระบบต่าง ๆ ของเครื่อง ซึ่ง Repository ที่ conda ใช้นั้นจะแยกออกมาจาก PyPI โดย conda มีความจุดเด่นในเรื่องของการจัดการ Dependencies ต่าง ๆ ของชุดโปรแกรมที่ถูกติดตั้งมาจากทั้งภายนอกและภายใน นอกจากภาษา Python แล้ว conda ยังรอบรับการจัดการของภาษาอื่น ๆ ได้อีกด้วย เช่น R, Ruby, Lua, Scala, Java, JavaScript, C/C++, และ Fortran ซึ่ง conda เป็นหนึ่งในผลิตภัณฑ์ของ Anaconda[2] โดยสามารถเข้าไปดูฐานข้อมูลไลบรารี่ของ Python ที่สามารถติดตั้งและจัดการผ่าน conda ได้ที่ https://anaconda.org

จากประสบการณ์ส่วนตัวของผู้เขียนนั้น pip จะได้เปรียบในเรื่องของการเวอร์ชันของชุดโปรแกรมที่จะได้รับการอัพเดทอยู่ตลอดเวลา นั่นก็เพราะว่า PyPI เป็น Repository ที่นักพัฒนาชุดโปรแกรมของ Python ส่วนใหญ่นั้นจะเลือกใช้ในการจัดเก็บชุดโปรแกรมของตนเอง แต่จุดอ่อนอย่างหนึ่งของ pip ก็คือการตรวจสอบความขัดแย้ง (Conflict) ระหว่างเวอร์ชันและ Dependencies ของโปรแกรมหลาย ๆ โปรแกรมในกรณีที่ต้องใช้ร่วมกัน ซึ่งตรงจุดนี้เองที่ conda สามารถทำได้ดีกว่า pip กล่าวคือ ในการติดตั้งชุดโปรแกรมนั้น conda จะทำการตรวจสอบก่อนว่าโปรแกรมนั้น ๆ จะมีปัญหาที่ซ้อนทับกับโปรแกรมอื่น ๆ หรือไม่ เช่น อาจจะใช้

[1] จริง ๆ แล้วมีบทความที่เปรียบเทียบประสิทธิภาพระหว่าง TensorFlow และ PyTorch แต่โดยส่วนตัวแล้วผู้เขียนคิดว่าความสามารถก็ไม่ได้ต่างกันมาก สิ่งที่มีผลจริง ๆ สำหรับการใช้ Neural Network คือการเลือกใช้โมเดลมากกว่า

[2] บริษัทที่ให้คำแนะนำและบริการเทคโลยีเกี่ยวกับการจัดการชุดโปรแกรมภายในองค์กร

โปรแกรมอีกตัวหนึ่งร่วมกันแต่ว่าใช้เวอร์ชันที่ต่างกัน ซึ่ง conda ก็จะหลีกเลี่ยงปัญหานี้ นอกจากนี้ยังมีการจัดการแบ่งสิ่งแวดล้อม (Environment) แยกสำหรับแต่ละโปรเจ็คเพื่อป้องกันปัญหาดังกล่าวได้อีกด้วย

ตัวอย่างด้านล่างคือการติดตั้ง TensorFlow บนระบบปฏิบัติการ Linux หรือ Windows Subsystem for Linux 2 (WSL2) ณ วันที่ผู้เขียนกำลังเขียนบทนี้ TensorFlow เวอร์ชันเสถียร (Stable Version) คือ 2.10.0 ซึ่งรอบรับ Python 3.7-3.10[1]

3.1 ติดตั้ง TensorFlow ด้วย pip สำหรับกรณีที่รองรับ CPU อย่างเดียว

```
python3 -m pip install tensorflow
```

หลังจากนั้นสามารถตรวจสอบการติดตั้งและเรียกใช้งาน TensorFlow ได้ตามปกติโดยรันคำสั่งต่อไปนี้

```
python3 -c "import tensorflow as tf;
   print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

3.2 ติดตั้ง TensorFlow ด้วย conda สำหรับกรณีที่รองรับ GPU

สำหรับกรณีที่ต้องการติดตั้ง TensorFlow ที่รองรับการทำงานร่วมกับ GPU ด้วยนั้น ผู้เขียนแนะนำให้ติดตั้งด้วย conda เพราะว่าสามารถติดตั้ง Driver ของการ์ดจอและ CUDA Toolkit ได้โดยอัตโนมัติ ซึ่งจะแตกต่างจากกรณีที่ติดตั้งด้วย pip ซึ่งเราจะต้องทำการติดตั้ง Driver ของ GPU และ CUDA toolkit เอง

```
conda update --all -y
conda install tensorflow-gpu
```

หลังจากนั้นสามารถตรวจสอบการติดตั้งและยืนยันว่า TensorFlow สามารถตรวจพบ GPU ของเครื่องได้โดยรันคำสั่งต่อไปนี้

```
python3 -c "import tensorflow as tf;
   print(tf.config.list_physical_devices('GPU'))"
```

ในกรณีข้างบนที่ใช้ conda ในการตั้งติด TensorFlow นั้นตัว conda จะทำการค้นหาแพคเกจของ TensorFlow ในแชนแนลหลักที่เป็นค่าเริ่มต้นก่อน นั่นคือแชนแนล anaconda โดยจะทำการติดตั้ง TensorFlow เวอร์ชันล่าสุดที่แชนแนลนี้มี ซึ่งเวอร์ชันล่าสุดที่แชนแนลนี้มีก็ไม่ใช่เวอร์ชันล่าสุดที่มีใน PyPI ดังนั้นถ้าหากเราต้องการที่จะติดตั้ง TensorFlow เวอร์ชันล่าสุดที่เทียบเท่ากับการติดตั้งด้วย pip เราจะต้องเปลี่ยนแชนแนลที่จะให้ conda นั้นไปทำการค้นหาแพคเกจ ซึ่งสามารถทำได้โดยใช้คำสั่งต่อไปนี้ซึ่งผมกำหนดให้แชนแนลเป็น `conda-forge`

[1]รายละเอียดของ Configuration ของ TensorFlow สามารถดูได้บนเว็บไซต์หลัก

```
conda install --channel conda-forge tensorflow-gpu
```

ภาคผนวก C

เทคนิคการเขียนโมเดล TensorFlow

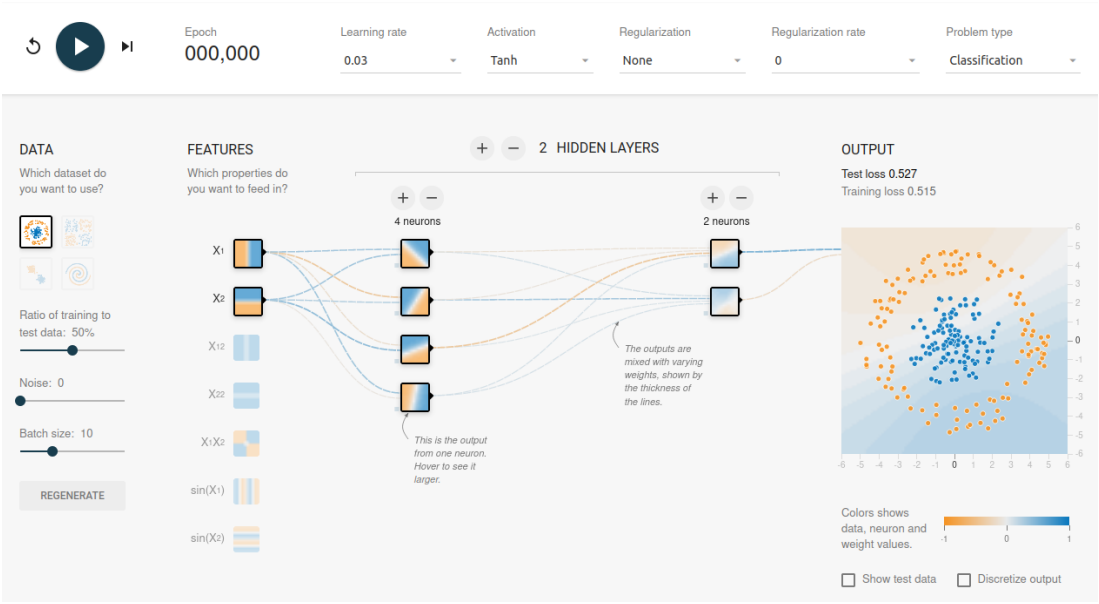
TensorFlow ถือได้ว่าเป็นไลบรารี่ Neural Network ที่ได้รับความนิยมมากที่สุดในโลกก็ว่าได้ ด้วยฟังก์ชันและโมเดลที่มีให้เลือกใช้งานได้หลากหลาย ทำให้ TensorFlow ถูกนำมาใช้งานในการสร้างและฝึกสอนโมเดล Neural Network ในงานประเภทต่าง ๆ

1 TensorFlow Playground

TensorFlow Playground คือเว็บไซต์ที่ทาง TensorFlow พัฒนาขึ้นมาเพื่อให้เราเรียนรู้เกี่ยวกับ Neural Network ซึ่งจะมีคอนโซลและเครื่องมือให้เราได้ออกแบบและฝึกฝนโมเดล Neural Network ที่มีขนาดเล็ก ไม่ซับซ้อนมาก แต่ทำงานได้จริง ซึ่งสามารถช่วยให้เราเห็นภาพว่าเกิดอะไรขึ้นบ้างในระหว่างที่มีการฝึกสอนโมเดล ข้อดีของ TensorFlow Playground ก็คือใช้งานได้สะดวกมาก ๆ เพราะสามารถใช้งานผ่านเว็บไซต์ได้เลยและไม่ต้องติดตั้งโปรแกรมอื่น ๆ ให้ยุ่งยาก โดยเข้าใช้งานได้ที่ http://playground.tensorflow.org

ภาพที่ C.1 แสดงส่วนควบคุมและแสดงผลของการฝึกสอนโมเดล โดยในส่วนด้านบนนั้นจะเป็นคอนโซลในการควบคุมการฝึกสอนโมเดลซึ่งเราสามารถกดเริ่มการฝึกสอนโมเดลได้ทันที และสามารถกดหยุดชั่วคราวได้ด้วย นอกจากนี้เรายังสามารถกำหนด Hyperparameters เช่น Learning Rate, Activation Function, Regularization, Regularization Rate รวมไปถึงประเภทของโจทย์ปัญหาได้ด้วย

ในส่วนทางด้านซ้ายนั้นก็คือการเลือกชุดข้อมูลและการจัดการกับชุดข้อมูล เช่น เราสามารถกำหนดสัดส่วนของการแบ่งชุดข้อมูลได้ โดยค่าเริ่มต้นคือ 50% และยังกำหนด Noise ได้เช่นกัน สำหรับบริเวณตรงกลางนั้นจะเป็นการดีไซน์หรือออกแบบ Nueral Network ที่ต้องการฝึกสอนนั่นเอง เราสามารถเพิ่มหรือลดจำนวนของ Hidden Layer ได้และกำหนดจำนวนของ Node ในแต่ละชั้นได้ด้วยเช่นกัน และทางด้านขวาก็คือการแสดงผลการ Fit ของโมเดลกับข้อมูลซึ่งในกรณีของโจทย์แบบ Classification นั้นจะใช้สีในการแบ่งหรือเป็นเกณฑ์ในการบอกถึงความสามารถในการ Classify ของโมเดล นอกจากนี้ยังมีบอกค่า Test Loss และ



ภาพ C.1 Interface ของ TensorFlow Playground แสดงส่วนควบคุมการตั้งค่า Hyperparameters และการฝึกฝนโมเดล

Train Loss อีกด้วย

สำหรับคำแนะนำในการใช้งานเพื่อศึกษา Neural Network นั้น ผู้เขียนขอแนะนำให้ผู้อ่านได้ลองเล่น TensorFlow Playground และทำตามดังต่อไปนี้

1. เริ่มต้นจากโจทย์ที่ง่ายที่สุดก่อนคือ Classification ของข้อมูลแบบแยก 2 กอง โดยลองลบ Hidden Layer ออกให้หมดและฝึกสอนโมเดลตั้งแต่ไม่มี Hidden Layer แล้วทำการเพิ่มจำนวนของ Hidden Layer ขึ้นเรื่อย ๆ แล้วสังเกตประสิทธิภาพในการ Classify ของโมเดล

2. เมื่อได้ผลลัพธ์การ Classification เป็นที่น่าพอใจแล้วจึงเพิ่มระดับความยากไปที่โจทย์ที่ยากปานกลาง ได้แก่ การแยกข้อมูลแบบโดนัทล้อมรอบ แล้วตามด้วยการแยกแบบแยกทแยง โดยทำการเพิ่มจำนวน Layer และ Neuron ในแต่ละ Layer

3. หลังจากที่เราศึกษาถึงผลของจำนวน Layer และ Neuron แล้ว ให้ลองปรับเปลี่ยน Hyperparameter อื่น ๆ เช่น เพิ่ม Feature แล้วให้สังเกตและเปรียบเทียบเอาต์พุต เช่น Loss และจำนวน Epoch ที่ใช้ฝึกสอนโมเดล

4. ศึกษาทำความเข้าใจ ความหมายของ ส่วนประกอบต่าง ๆ ของ Neural Network เราอาจจะเพิ่มความท้าทาย เช่น อาจจะจัดแข่งกับเพื่อนโดยกำหนดข้อจำกัดว่า ฝึกสอนโมเดลได้ไม่เกินกี่ Epoch, กำหนด Test Loss ห้ามเกิน 0.05, ให้ใช้ได้มากที่สุด 4 Layer หรือห้ามใช้ ReLU สำหรับ Activation Function

2 การเขียน TensorFlow เบื้องต้น

ตัวอย่างด้านล่างคือการสร้างและฝึกสอนโมเดล Neural Network ด้วย Keras ซึ่งเป็น API ของ TensorFlow จะเห็นได้ว่าเราสามารถเขียนโค้ด Python โดยเริ่มจากการนำเข้าชุดข้อมูลตัวอย่างซึ่งเป็น Mnist มีการแปลงข้อมูล ตามด้วยการสร้าง Neural Network โดยใช้วิธี Sequential กำหนดจำนวนชั้น ประเภทของแต่ละชั้น และมีพารามิเตอร์ที่เราสามารถกำหนดได้ เช่น จำนวน Node หรือ Neuron ของแต่ละชั้น และ Activation Function หลังจากนั้นก็ทำการ Compile โมเดลซึ่งเราสามารถกำหนด Optimizer และ Loss Function ได้ด้วย เมื่อทำการสร้างโมเดลเสร็จแล้ว ก็จะต่อด้วยการฝึกสอนหรือเทรน (Train) โมเดลตามจำนวนรอบ (Epoch) และขั้นตอนสุดท้ายคือการทดสอบโมเดลโดยการทำนายและประเมินผล

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

จากโค้ดด้านบนจะเห็นได้เลยว่าจริง ๆ แล้วการเขียนโค้ด ML โดยเฉพาะ Neural Network ไม่ได้ยากเลย เพราะว่าในปัจจุบันเรามีเครื่องมือที่เป็น Framework ต่าง ๆ ที่ถูกพัฒนาขึ้นมาเพื่อช่วยอำนวยความสะดวกให้เรากับในการเขียนโค้ด เพียงแค่ไม่กี่สิบบรรทัดเราก็สามารถสร้างโมเดลได้แล้ว เมื่อเราเข้าใจพื้นฐานในการสร้างโมเดลแล้ว ถ้าหากเราต้องการที่จะต่อยอดโดยการปรับแต่งโมเดลเพื่อให้สามารถจัดการกับงานที่ซับซ้อนขึ้นก็ไม่ใช่เรื่องยาก

3 การปรับแต่ง Loss Function

```
import tensorflow as tf
import tensorflow.keras.backend as kb
import numpy as np

def custom_loss(y_actual, y_pred):
    custom_loss =
   tf.experimental.numpy.log10(kb.sum(kb.abs(y_actual - y_pred))
   / y_actual.shape[0])
    return custom_loss


x = np.random.randint(1, 4, size=(1000,))
x = np.asarray(x).T

y = x**2
y = np.asarray(y).T
x = x.astype(np.float32)
y = y.astype(np.float32)

keras_model = tf.keras.Sequential(
    [
        tf.keras.layers.Dense(32, activation=tf.nn.relu,
   input_shape=[1]),
        tf.keras.layers.Dense(32, activation=tf.nn.relu),
        tf.keras.layers.Dense(1),
    ]
)

optimizer = tf.keras.optimizers.RMSprop(0.001)
keras_model.compile(loss=custom_loss, optimizer=optimizer)
keras_model.fit(x, y, batch_size=20, epochs=50)
```

4 การฝึกสอนโมเดลด้วยการประมวลผลแบบขนานบน GPU หลายตัว

```
import tensorflow as tf

# Use all avialable GPUs
mirrored_strategy = tf.distribute.MirroredStrategy()
# Specify which GPU to be used
```

```
6  mirrored_strategy =
       tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
7
8  with mirrored_strategy.scope():
9      model = tf.keras.Sequential([tf.keras.layers.Dense(1,
       input_shape=(1,))])
10
11 model.compile(loss="mse", optimizer="sgd")
12
13 dataset = tf.data.Dataset.from_tensors(([1.0],
       [1.0])).repeat(100).batch(10)
14 model.fit(dataset, epochs=2)
15 model.evaluate(dataset)
```

ตัวอย่างด้านบนเป็นการใช้ `MirroredStrategy` ซึ่งเป็นวิธีหนึ่งของการทำ Distributed Training ของ TensorFlow โดยเราทำการสร้าง Scope ขึ้นมาก่อน โดยในโค้ดด้านบนเราสร้าง `mirrored_strategy` ซึ่งเราสามารถเลือก GPU ที่ต้องการใช้ในการฝึกสอนโมเดลได้ด้วย หลังจากนั้นก็ใช้ฟังก์ชัน `with` สำหรับทำการกำหนด Scope และภายในฟังก์ชันนี้เราก็ทำการสร้าง Neural Network ขึ้นมา หลังจากนั้นก็ทำการคอมไพล์โมเดลซึ่งสามารถกำหนดให้อยู่นอก Scope ได้แล้วก็ทำการฝึกสอนโมเดลโดยใช้วิธี `fit` ตามปกติ

ภาคผนวก D

โปรแกรมทางด้านเคมีควอนตัม

1 Gaussian

โปรแกรม Gaussian เป็นโปรแกรมที่เรียกได้ว่าเป็นตำนานของโปรแกรมทางด้านเคมีควอนตัม นั่นก็เพราะว่า Gaussian ได้ถูกพัฒนามาอย่างยาวนาน ซึ่งถือว่าเป็นโปรแกรมแรกของงานวิจัยสายนี้เลยก็ว่าได้ โดย Gaussian ได้ถูกพัฒนาขึ้นในกลุ่มวิจัยของศาสตราจารย์ John A. Pople ในช่วงปี ค.ศ. 1970 และมีการพัฒนาต่อเรื่อยมาจนถึงปัจจุบัน โดยเวอร์ชันล่าสุดของ Gaussian (ณ วันที่ผู้เขียนเขียนหนังสือเล่มนี้) คือ เวอร์ชัน 16[332]

คุณสมบัติหรือ Feature ของโปรแกรม Gaussian นั้นคือสามารถคำนวณคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลขนาดเล็ก (ไม่เกิน 50 อะตอม) ขนาดกลาง (50 - 120 อะตอม) และขนาดใหญ่ (มากกว่า 120 อะตอม) ได้อย่างแม่นยำ[1] โดยจุดเด่นของ Gaussian ก็คือการคำนวณคุณสมบัติเชิงอิเล็กทรอนิกส์ของโมเลกุลด้วยวิธี DFT และด้วยอัลกอริทึมของตัวโปรแกรมนั้น ทำให้ Gaussian ได้รับการยอมรับว่าเป็นหนึ่งในโปรแกรมที่ให้ผลการคำนวณที่ถูกต้องและน่าเชื่อถือ และสามารถนำไปเปรียบเทียบกับผลการทดลองได้ สำหรับ Gaussian นั้นรองรับการคำนวณแบบวิธี OpenMP นั่นคือสามารถทำการประมวลผลแบบขนานได้

[1] ความแม่นยำและความถูกต้องของผลการคำนวณอ้างอิงตามประสบการณ์ของผู้เขียน โดยมีปัจจัยที่ส่งผลต่อค่าความถูกต้อง เช่น วิธีที่ใช้ในการคำนวณและ Basis Set

โดยใช้หน่วยประมวลผล CPU หลายตัวพร้อม ๆ กันได้ และนอกจากนี้แล้วในเวอร์ชัน 16 ตัวโปรแกรมยังรองรับกราฟฟิคการ์ด GPU สำหรับการคำนวณโดยใช้วิธี DFT อีกด้วย

ตัวอย่างไฟล์อินพุตของโปรแกรม Gaussian สำหรับการคำนวณพลังงานของโมเลกุล Formaldehyde ด้วยวิธี Hartree-Fock

```
%chk=formaldehyde.chk
%mem=128MB
#P HF/6-31G(d) scf=tight

HF/6-31G(d) sp formaldehyde

0 1
C1
O2  1  r2
H3  1  r3  2  a3
H4  1  r4  2  a4  3  d4

r2=1.20
r3=1.0
r4=1.0
a3=120.
a4=120.
d4=180.
```

รายละเอียดเพิ่มเติมเกี่ยวกับโปรแกรม Gaussian ดูได้ที่เว็บไซต์ https://gaussian.com

2 ORCA

โปรแกรม ORCA เป็นอีกหนึ่งโปรแกรมทางเคมีควอนตัมที่มีประสิทธิภาพและความสามารถในการคำนวณสูง[333,334] สามารถคำนวณได้หลายวิธี โดยสามารถคำนวณ DFT และวิธี Semi-empirical ได้ รวม

ไปถึงวิธี Post Hartree-Fock อื่น ๆ ด้วย โดย ORCA ถูกใช้อย่างแพร่หลายในงานวิจัยทางด้านเคมีอินทรีย์และเคมีอนินทรีย์ โดยเฉพาะการศึกษาสารประกอบเชิงซ้อนของโลหะทรานซิชัน (Transition Metal Complex) ซึ่งเป็นโมเลกุลที่มีขนาดใหญ่และมีความซับซ้อนในเชิงของโครงสร้างอิเล็กทรอนิกส์มากกว่าโมเลกุลอินทรีย์ขนาดเล็ก และ ORCA ยังมีความโดดเด่นในด้านของความแม่นยำและความเร็วในการคำนวณเกี่ยวกับคุณสมบัติเชิงสเปกตรัมของโมเลกุล

โปรแกรม ORCA ถูกพัฒนาในกลุ่มวิจัยของศาสตราจารย์ Frank Neese โดยสามารถดาวน์โหลดตัวโปรแกรม (เฉพาะไฟล์ Binary ที่ถูกคอมไพล์แล้ว) มาใช้ได้ฟรีสำหรับวัตถุประสงค์ด้านการศึกษาและการทำงานวิจัย

ตัวอย่างไฟล์อินพุตของโปรแกรม ORCA สำหรับการคำนวณพลังงานของโมเลกุลน้ำด้วยวิธี Hartree-Fock

```
!HF DEF2-SVP

%SCF
   MAXITER 500
END

* xyz 0 1
O   0.0000   0.0000   0.0626
H  -0.7920   0.0000  -0.4973
H   0.7920   0.0000  -0.4973
*
```

รายละเอียดเพิ่มเติมเกี่ยวกับโปรแกรม ORCA ดูได้ที่เว็บไซต์ https://orcaforum.kofo.mpg.de/app.php/portal หรือดูคู่มือและแบบฝึกหัดสอนการใช้โปรแกรมได้ที่เว็บไซต์ https://www.orcasoftware.de/tutorials_orca/index.html

3 NWChem

โปรแกรม NWChem เป็นโปรแกรมการคำนวณทางเคมีควอนตัมและพลศาสตร์เชิงโมเลกุล (Molecular Dynamics)[335] พัฒนาโดยสถาบัน Pacific Northwest National Laboratory (PNNL) ในช่วงปี ค.ศ. 1990 โดยรองรับการคำนวณด้วยวิธี DFT และ Post Hartree-Fock เช่น Möllor-Plesset (MP), Configuration Interaction (CI), Coupled Cluster (CC) และ Multiconfiguration SCF (MCSCF)

NWChem นั้นถูกพัฒนาเพื่อให้สามารถประมวลผลบน Supercomputer ที่มีประสิทธิภาพสูงได้ NWChem ถูกเขียนขึ้นโดยใช้ภาษา Fortran 77 และใช้ไลบรารี่ทางด้านพีชคณิตสำหรับการประมวลผล เช่น BLAS, LAPACK, และ ScaLAPACK และสามารถประมวลผลแบบขนาดได้โดยใช้วิธี Message-Passing Interface (MPI) นอกจากนี้ NWChem ยังรองรับการประมวลผลด้วย GPU สำหรับการคำนวณด้วยวิธี Coupled Cluster ซึ่งถือว่าเป็นจุดเด่นของ NWChem เลยก็ว่าได้ โปรแกรม NWChem เป็นแบบ Open-source มีการพัฒนาอย่างต่อเนื่องเรื่อยมาจนถึงปัจจุบัน ซึ่งนักวิจัย นักศึกษา และคนทั่วไปสามารถร่วมพัฒนาและใช้งานโปรแกรมได้ฟรี

สำหรับผู้อ่านที่ใช้งานระบบปฏิบัติการ Debian หรือ Ubuntu นั้นก็สามารถติดตั้งโปรแกรม NWChem ได้อย่างง่ายดายด้วยคำสั่ง

```
sudo apt-get install nwchem
```

ตัวอย่างไฟล์อินพุตของโปรแกรม NWChem สำหรับการคำนวณพลังงานของโมเลกุลน้ำด้วยวิธี Hartree-Fock

```
start h2o
title "Water in 6-31g basis set"

geometry units au
  O      0.00000000    0.00000000    0.00000000
  H      0.00000000    1.43042809   -1.10715266
  H      0.00000000   -1.43042809   -1.10715266
end

basis
  H library 6-31g
  O library 6-31g
end

task scf
```

รายละเอียดเพิ่มเติมเกี่ยวกับโปรแกรม NWChem ดูได้ที่เว็บไซต์ https://nwchemgit.github.io

4 PySCF

โปรแกรม PySCF เป็นโปรแกรมสำหรับการคำนวณทางเคมีควอนตัมซึ่งถูกเขียนขึ้นมาโดยใช้ภาษา Python โดยทีมนักวิจัยจาก California Institute of Technology[336] จุดเด่นของโปรแกรม PySCF ก็คือมีขนาดที่เล็ก (Lightweight) และมีโมดูลในการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์ที่หลากหลาย สามารถพัฒนาต่อได้ง่ายไม่ซับซ้อนเพราะว่าใช้ภาษา Python ในการเขียน และใช้งานได้ง่าย เหมาะกับผู้ที่ต้องการเขียนหรือแก้ไขโค้ดโปรแกรมทางเคมีควอนตัม นอกจากนี้ผู้ใช้งานสามารถใช้ PySCF ในการคำนวณคุณสมบัติของโมเลกุล ปรับแต่ง Hamiltonians ของ Wavefunction โดยใช้ทฤษฎีสนามเฉลี่ย (Mean-Field Theory)

โค้ดเกือบทั้งหมดของโปรแกรม PySCF ถูกเขียนด้วยภาษา Python แต่ว่าโค้ดของโปรแกรมบางส่วนที่เกี่ยวข้องกับการคำนวณที่ซับซ้อนนั้นถูกเขียนด้วยภาษา C ซึ่งช่วยลดระยะเวลาในการคำนวณและทำให้ PySCF นั้นมีประสิทธิภาพเทียบเท่าได้กับโปรแกรมคำนวณทางเคมีควอนตัมโปรแกรมอื่น ๆ ที่ช่วยด้วยภาษาระดับล่าง เช่น C หรือ Fortran นอกเหนือจากโมดูลหลักที่ใช้ในการคำนวณโครงสร้างเชิงอิเล็กทรอนิกส์

การติดตั้งโปรแกรม PySCF นั้นก็สามารถทำได้ง่ายมาก เพียงแค่ผู้ใช้งานมีโปรแกรม Python เวอร์ชัน 3 ก็สามารถติดตั้งได้โดยใช้คำสั่งต่อไปนี้

```
pip install pyscf[geomopt]
```

ซึ่งจะโมดูล Geometry Optimization ของ PySCF ด้วย ถ้าหากไม่ต้องการติดตั้งโมดูลนี้ก็ให้ใช้แค่ `pyscf`

```
from pyscf import gto, scf

mol = gto.Mole()
mol.verbose = 5
# mol.output = 'out_h2o'
mol.atom = '''
O        0.000000    0.000000    0.117790
H        0.000000    0.755453   -0.471161
H        0.000000   -0.755453   -0.471161'''
mol.basis = 'ccpvdz'
mol.symmetry = 1
mol.build()

mf = scf.RHF(mol)
mf.kernel()
```

รายละเอียดเพิ่มเติมเกี่ยวกับโปรแกรม PySCF ดูได้ที่เว็บไซต์ https://pyscf.org

ภาคผนวก E

ออร์บิทัลของอะตอมไฮโดรเจน

1 ฟังก์ชันคลื่นของอะตอมไฮโดรเจน

ในหัวข้อนี้ผู้อ่านจะได้เรียนรู้วิธีการเขียนโค้ดด้วยภาษา Python สำหรับการพล็อตรูปร่างของ Wavefunction หรือออร์บิทัลของอะตอมไฮโดรเจน ก่อนอื่นนั้นเราจะต้องมาทำความเข้าใจกับสมการ Wavefunction ของอะตอมไฮโดรเจนที่มี 1 อิเล็กตรอนกันก่อน โดยมีสมการดังต่อไปนี้

$$\psi_{nlm}(r,\theta,\phi) = R_{nl}(r)Y_{lm}(\theta,\phi) \tag{E.1}$$

ซึ่งเป็นผลคูณระหว่างส่วนที่เป็นเชิงรัศมี (Radial Part) หรือ $R_{nl}(r)$ และส่วนที่เป็นเชิงมุม (Angular Part) หรือ $Y_{lm}(\theta,\phi$ ตามลำดับ โดยมีสมการที่เป็นฟังก์ชันของเลขควอนตัมดังต่อไปนี้

$$R_{nl}(r) = \sqrt{\left(\frac{2}{na_0}\right)^3 \frac{(n-l-1)!}{2n(n+l)!}} e^{-r/na_0} \left(\frac{2r}{na_0}\right)^l \cdot L^{2l+1}_{n-l-1}\left(\frac{2r}{na_0}\right) \tag{E.2}$$

และ

$$Y_{lm}(\theta,\phi) = \Theta_{lm}(\theta)\Phi_m(\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_{lm}(cos\theta) \cdot e^{im\phi} \tag{E.3}$$

โดยในขั้นตอนการพล็อต Wavefunction นั้นเราจะทำการสร้างฟังก์ชันสำหรับ $R_{nl}(r)$ และ $Y_{lm}(\theta,\phi$ ตามลำดับ แล้วหลังจากนั้นก็นำมาร่วมกันเพื่อพล็อตเป็นออร์บิทัลในปริภูมิ 3 มิติต่อไป

หมายเหตุ

1. เราพิจารณาส่วนจริง (Real Part) ของ Psherical Harmonics ของ Wavefunction เท่านั้น โดยเราจะไม่พิจารณาส่วนจินตภาพ (Imaginary Part)
2. เราจะใช้ Atomic Units สำหรับปริมาณดังต่อไปนี้เพื่อความสะดวกในการเขียนโค้ด $a_0 = 1, \hbar = 1, m_e = 1, e = 1$

2 การเขียนโค้ดสำหรับพล็อตออร์บิทัล

ผู้เขียนแนะนำให้ผู้อ่านใช้ IPython-based Platform สำหรับเขียนโค้ด เช่น Jupyter Notebook

- **เตรียมไลบรารี่**

```
%matplotlib inline
import matplotlib.pyplot as plt

from matplotlib import cm, colors
from mpl_toolkits.mplot3d import Axes3D

import numpy as np
import scipy.integrate as integrate

# Increase resolution for retina display
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('retina')

# Load interactive widgets
import ipywidgets as widgets
import ipyvolume as ipv
```

ถ้าหากผู้อ่านยังไม่ได้ติดตั้งไลบรารี่สามารถติดตั้งได้โดยใช้คำสั่ง

```
pip install LIBRARY_NAME
```

- **พล็อต Radial Part ของ Wavefunction**

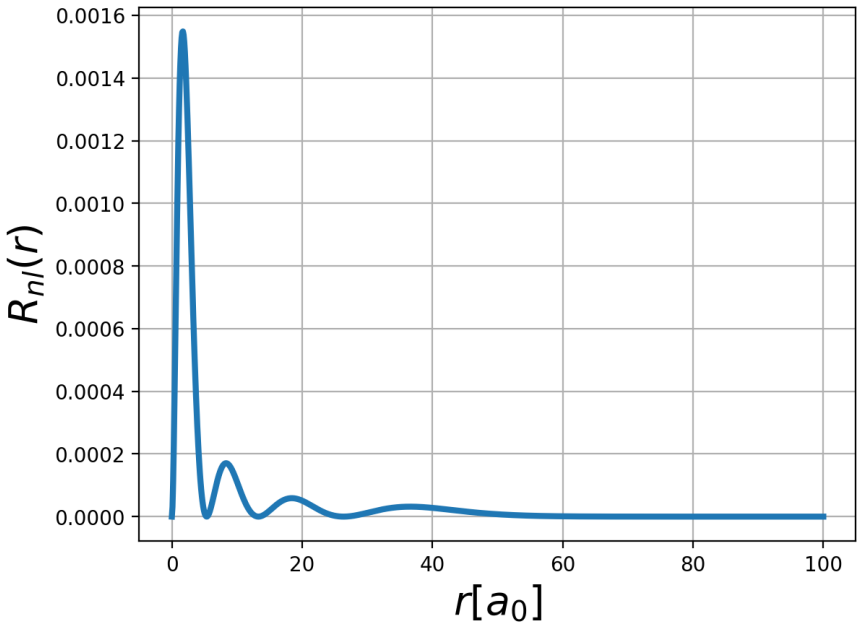
เริ่มต้นด้วยการสร้างฟังก์ชันสำหรับ Radial Part ของ Wavefunction ซึ่งเราจะใช้พหุนามลาแกร์ (Laguerre Polynomials) โดยกำหนดให้ $n = 0$ และ $l == $ เป็นค่าเริ่มต้น

```
def psi_R(r, n=1, l=0):
    coeff = np.sqrt((2.0/n)**3 * spe.factorial(n-l-1)
   /(2.0*n*spe.factorial(n+l)))
```

```
    laguerre = spe.assoc_laguerre(2.0*r/n,n-l-1,2*l+1)

    return coeff * np.exp(-r/n) * (2.0*r/n)**l * laguerre
```

ทำการกำหนด Grid Space สำหรับการพล็อตค่าของ Radial Part

```
r = np.linspace(0, 100, 1000)
R = psi_R(r, n=5, l=1)

plt.plot(r, R**2, lw=3)
plt.xlabel('$r [a_0]$',fontsize=20)
plt.ylabel('$R_{nl}(r)$', fontsize=20)
plt.grid('True')
```

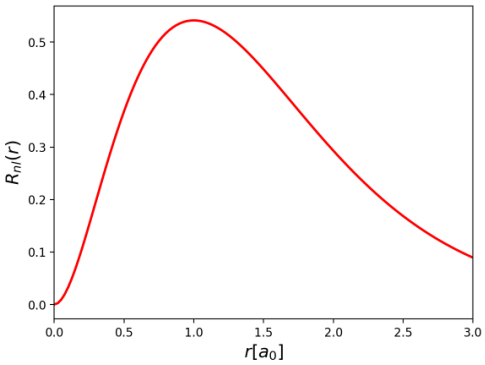


ภาพ E.1 Radial Part สำหรับ $n = 5$ และ $l = 0$

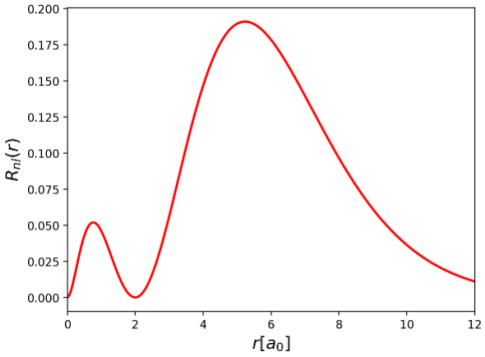
โดยจะได้พล็อตตามภาพที่ E.1 ขั้นตอนต่อไปคือทำการเพิ่ม Interactive Widget สำหรับพล็อตที่สามารถปรับค่าเลขควอนตัมได้

```
nmax=10

@widgets.interact(n = np.arange(1, nmax, 1), l = np.arange(0,
    nmax-1, 1))

def plot_radial(n=1, l=0):
    r = np.linspace(0,250,10000)
    psi2 = psi_R(r,n,l)**2 * (r**2)
```

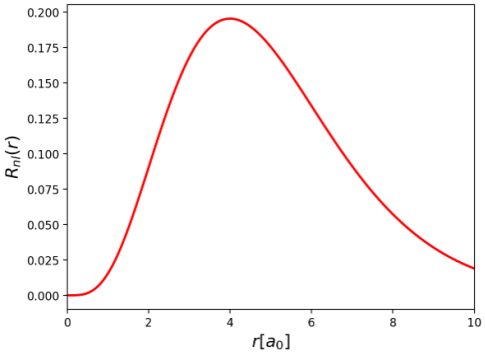
```
    plt.plot(r, psi2, lw=2, color='red')

    # Styling the plot
    plt.xlabel('$r [a_0]$')
    plt.ylabel('$R_{nl}(r)$')
    rmax = n**2*(1+0.5*(1-l*(l+1)/n**2))
    plt.xlim([0, 2*rmax])
```



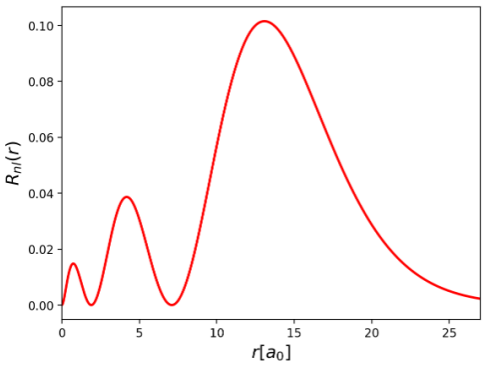
(a) $n = 1$ และ $l = 0$



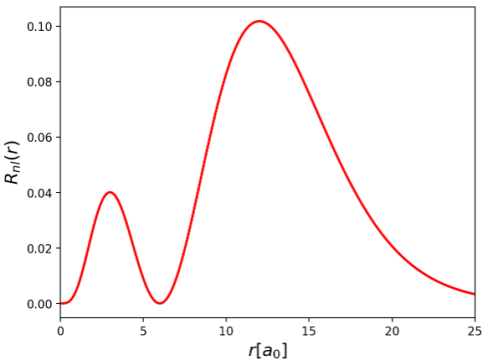
(b) $n = 2$ และ $l = 0$



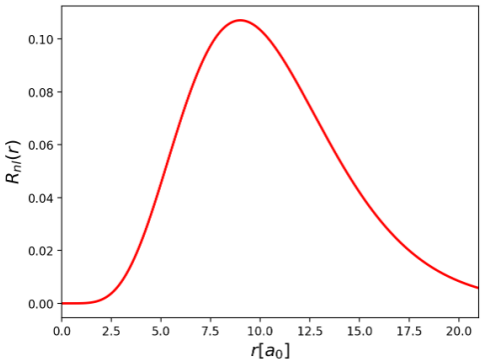
(c) $n = 2$ และ $l = 1$



(d) $n = 3$ และ $l = 0$



(e) $n = 3$ และ $l = 1$



(f) $n = 3$ และ $l = 2$

ภาพ E.2 Radial Part ของ Wavefunction ของอะตอมไฮโดรเจน

- **พล็อต Angular Part ของ Wavefunction**

สร้างฟังก์ชันสำหรับสร้าง Angular Part (สนใจเฉพาะส่วนจริงเท่านั้น) เช่นเดียวกันกับ Radial Part

```
1 def psi_ang(phi,theta, l=0, m=0):
```

```
    sphHarm = spe.sph_harm(m,l,phi,theta)

    return sphHarm.real
```

ทำการสร้าง Grid Space สำหรับการพล็อต Spherical Harmonics แบบ 3 มิติ

```
phi, theta = np.linspace(0, np.pi, 100), np.linspace(0, 2*np.pi,
    100)
phi, theta = np.meshgrid(phi, theta)
Ylm = psi_ang(theta,phi,l=2,m=0)
```

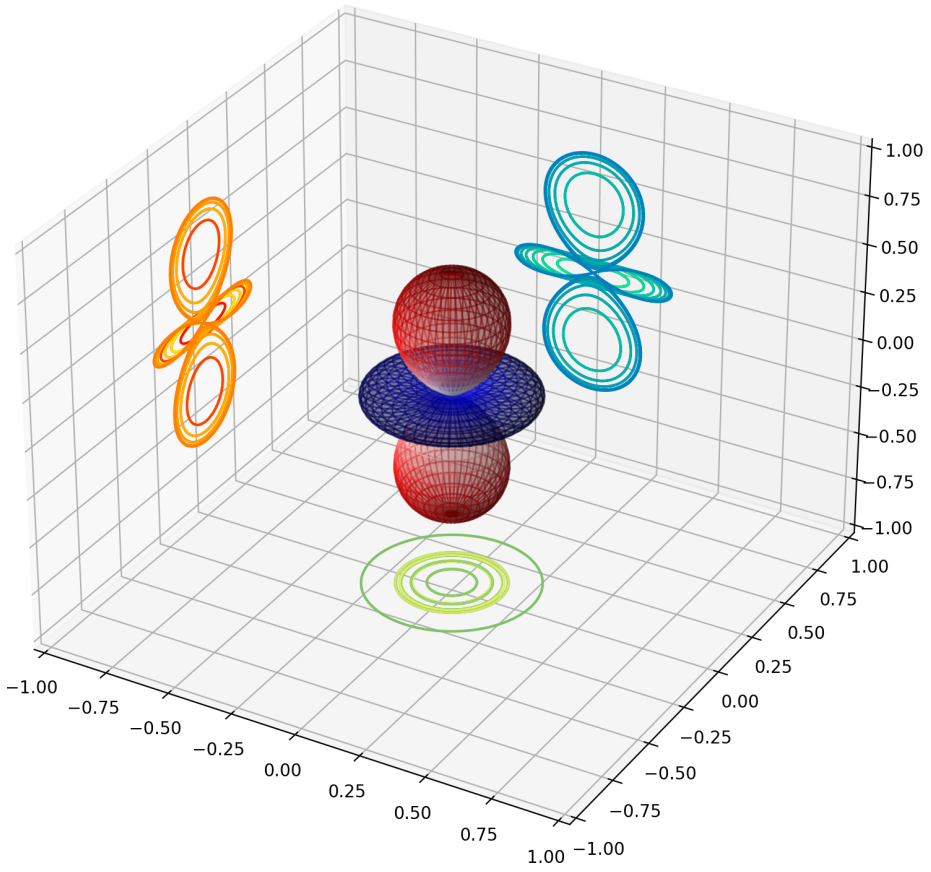
ทำการกำหนดค่าเริ่มต้น Domain สำหรับการพล็อตออร์บิทัล นั่นคือ x กับ y แล้วทำการคำนวณค่ามุม Angular Part z สำหรับแต่ละจุดบน Grid

```
x = np.sin(phi) * np.cos(theta) * abs(Ylm)
y = np.sin(phi) * np.sin(theta) * abs(Ylm)
z = np.cos(phi) * abs(Ylm)
```

ทำการกำหนดตั้งค่าสำหรับการพล็อตแบบ 3 มิติ โดยกำหนดให้ `projection='3d'` สำหรับ Matplotlib

```
# Set up the 3D Canvas
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

# Normalize color bar to [0,1] scale
fcolors = (Ylm - Ylm.min())/(Ylm.max() - Ylm.min())

# Make 3D plot of real part of spherical harmonic
ax.plot_surface(x, y, z, facecolors=cm.seismic(fcolors),
    alpha=0.3)

# Project 3D plot onto 2D planes
cset = ax.contour(x, y, z,20, zdir='z',offset = -1,
    cmap='summer')
cset = ax.contour(x, y, z,20, zdir='y',offset =  1,
    cmap='winter' )
cset = ax.contour(x, y, z,20, zdir='x',offset = -1,
    cmap='autumn')

# Set axes limit to keep aspect ratio 1:1:1
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)
```

```
19  ax.set_zlim(-1, 1)
```



ภาพ E.3 Angular Part สำหรับ $l = 2$ และ $m = 0$

- **รวม Radial Part และ Angular Part สำหรับพล็อตออร์บิทัล**

สร้างฟังก์ชันสำหรับคำนวณ Wavefunction ของไฮโดรเจนโดยรับค่าอินพุตดังต่อไปนี้

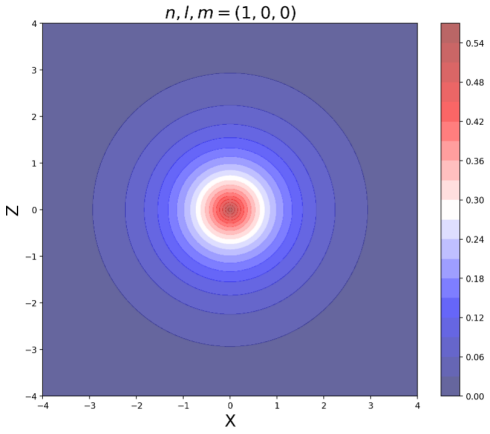
- r: Radial Coordinate (r)
- theta: Polar Coordinate (θ)
- phi: Azimuthal Coordinate (ϕ)
- n: Principle Quantum Number (n)
- l: Angular Momentum Quantum Number (l)
- m: Magnetic Quantum Number (m)

แล้วทำการคืนค่าออกมาเป็น Wavefunction ซึ่งได้จากผลคูณระหว่าง ψ_R และ ψ_{ang}

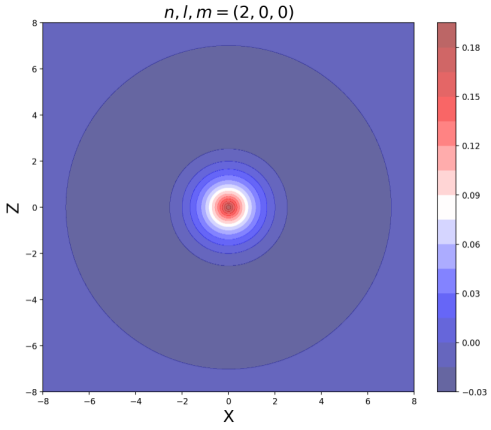
```
def HFunc(r, theta, phi, n, l, m):
    # Hydrogen wavefunction // a_0 = 1

    return psi_R(r, n, l) * psi_ang(phi, theta, l,m)
```

กำหนดค่าเริ่มต้นของเลขควอนตัมหลักและเลขควอนตัมเชิงมุม เช่น กำหนดค่าสูงสุดคือ 10 ซึ่งเมื่อทำการพล็อตแล้วเราสามารถเปลี่ยนค่า n, l, และ m ได้ตามต้องการเพราะว่าเราใช้ Interactive Plot

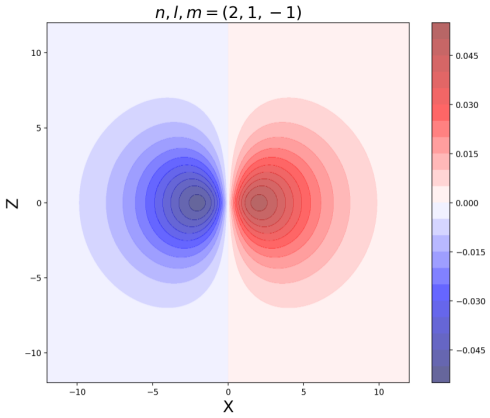
```
nmax = 10
lmax = nmax-1

@widgets.interact(n=np.arange(1,nmax,1), l =
   np.arange(0,nmax-1,1), m=np.arange(-lmax,lmax+1,1))

def psi_xz_plot(n=1, l=0, m=0):
    plt.figure(figsize=(10, 8))
    limit = 4*(n+l)
    x_1d = np.linspace(-limit, limit, 500)
    z_1d = np.linspace(-limit, limit, 500)
    x,z = np.meshgrid(x_1d, z_1d)
    y   = 0

    r = np.sqrt(x**2 + y**2 + z**2)
    theta = np.arctan2(np.sqrt(x**2+y**2), z)
    phi = np.arctan2(y, x)
    psi_nlm = HFunc(r,theta,phi,n,l,m)

    # Try cmap = inferno, rainbow, autumn, summer
    # plt.pcolormesh(x, z, psi_nlm, cmap='inferno')
    plt.contourf(x, z, psi_nlm, 20, cmap='seismic', alpha=0.6) #
   Classic orbitals
    plt.colorbar()
    plt.title(f"$n, l, m={n,l,m}$", fontsize=20)
    plt.xlabel('X', fontsize=20)
    plt.ylabel('Z', fontsize=20)
```



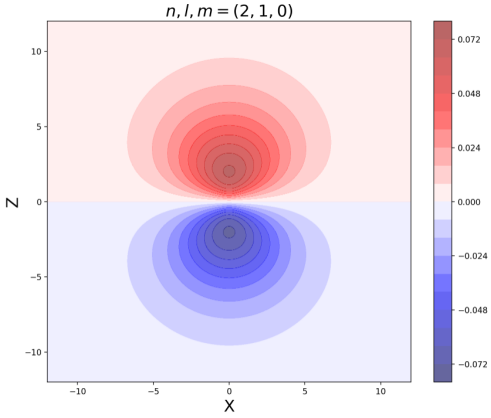
(a) $n = 1$, $l = 0$, และ $m = 0$



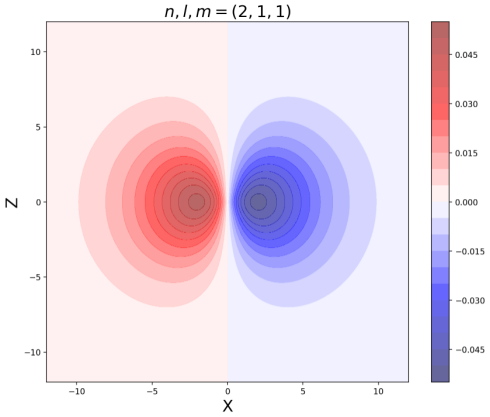
(b) $n = 2$, $l = 0$, และ $m = 0$



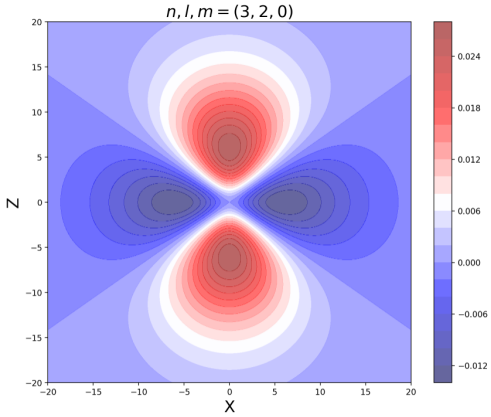
(c) $n = 2$, $l = 1$, และ $m = -1$



(d) $n = 2$, $l = 1$, และ $m = 0$



(e) $n = 2$, $l = 1$, และ $m = 1$



(f) $n = 3$, $l = 2$, และ $m = 0$

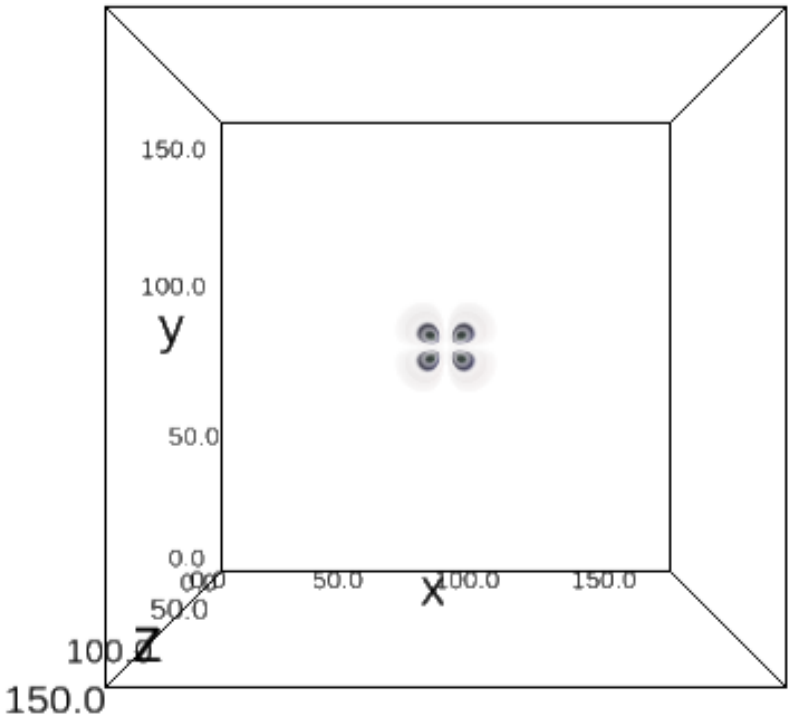
ภาพ E.4 Wavefunction ของอะตอมไฮโดรเจน ซึ่งรวม Radial และ Angular Part เข้าด้วยกัน

คำนวณค่าสำหรับการพล็อตออร์บิทัลแบบสมบูรณ์แบบ Interactive

```
# Variables to adjust
maxi = 60
resolution = 160

base = np.linspace(-maxi, maxi, resolution)[:, np.newaxis,
   np.newaxis]
x2 = np.tile(base, (1, resolution, resolution))
y2 = np.swapaxes(x2, 0, 1)
z2 = np.swapaxes(x2, 0, 2)

total = np.concatenate((x2[np.newaxis,:], y2[np.newaxis,:],
   z2[np.newaxis,:]), axis=0)

r2 = np.linalg.norm(total, axis=0)
# Alternative theta calculation
# theta3 = np.abs(np.arctan2(np.linalg.norm(total[:2], axis=0),
   -total[2]))

np.seterr(all='ignore')
phi2 = np.arctan(np.divide(total[2], np.linalg.norm(total[:2],
   axis=0))) + np.pi/2
theta2 = np.arctan2(total[1], total[0])
```

ทำการพล็อต Wavefunction แบบสมบูรณ์

```
ipv.figure()
psi = HFunc(r2,theta2,phi2,2,1,1)
ipv.volshow(r2**2 * np.sin(phi2)*psi**2)
ipv.show()
```



ภาพ E.5 ออร์บิทัลของอะตอมไฮโดรเจนในปริภูมิ 3 มิติ สำหรับ $n = 2, l = 1$ และ $m = 1$

เรายังสามารถตรวจสอบขนาดของ Array ของข้อมูลที่ใช้ในการพล็อต Wavefunction ได้ ดังนี้

```
psi.shape
# Output
(160, 160, 160)
```

จากตัวอย่างโค้ดข้างต้นจะพบว่าจริง ๆ แล้วการแสดง Wavefunction หรือออร์บิทัลที่แสดงความน่าจะเป็นหรือโอกาสที่จะพบอิเล็กตรอนนั้นไม่ได้ซับซ้อน ถ้าหากเรามีฟังก์ชันคณิตศาสตร์ที่ตรงไปตรงมา เราก็สามารถเขียนโค้ดตามฟังก์ชันดังกล่าวได้เลย

บรรณานุกรม

[1] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (July 1959), pp. 210–229 (cit. on p. 3).

[2] A. M. Turing. “I.—COMPUTING MACHINERY AND INTELLIGENCE”. In: *Mind* LIX.236 (Oct. 1950), pp. 433–460 (cit. on p. 4).

[3] John Jumper et al. “Highly Accurate Protein Structure Prediction with AlphaFold”. In: *Nature* 596.7873 (Aug. 2021), pp. 583–589 (cit. on p. 4).

[4] Ashish Vaswani et al. *Attention Is All You Need*. Dec. 2017. arXiv: 1706.03762 [cs] (cit. on p. 6).

[5] Volker Strassen. “Gaussian Elimination Is Not Optimal”. In: *Numerische Mathematik* 13.4 (Aug. 1969), pp. 354–356 (cit. on p. 6).

[6] Hugh M. Cartwright, ed. *Machine Learning in Chemistry: The Impact of Artificial Intelligence*. 1st edition. Croydon, UK: Royal Society of Chemistry, July 2020 (cit. on p. 8).

[7] *Machine Learning in Chemistry*. https://pubs.acs.org/doi/book/10.1021/acs.infocus.7e4001 (cit. on p. 8).

[8] W. Kohn, A. D. Becke, and R. G. Parr. “Density Functional Theory of Electronic Structure”. In: *The Journal of Physical Chemistry* 100.31 (Jan. 1996), pp. 12974–12980 (cit. on p. 8).

[9] Martin Korth. “Density Functional Theory: Not Quite the Right Answer for the Right Reason Yet”. In: *Angewandte Chemie International Edition* 56.20 (2017), pp. 5396–5398 (cit. on p. 9).

[10] Benjamin G. Janesko. “Replacing Hybrid Density Functional Theory: Motivation and Recent Advances”. In: *Chemical Society Reviews* 50.15 (Aug. 2021), pp. 8470–8495 (cit. on p. 9).

[11] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362 (cit. on p. 9).

[12] F. Pedregosa et al. “Scikit-Learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 9).

[13] Martín Abadi et al. *TensorFlow: Large-scale Machine Learning on Heterogeneous Systems*. 2015 (cit. on p. 9).

[14] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035 (cit. on p. 9).

[15] Mike Innes. “Flux: Elegant Machine Learning with Julia”. In: *Journal of Open Source Software* 3.25 (May 2018), p. 602 (cit. on p. 9).

[16] MATLAB. *Version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010 (cit. on p. 9).

[17] Roman M. Balabin and Ekaterina I. Lomakina. “Neural Network Approach to Quantum-Chemistry Data: Accurate Prediction of Density Functional Theory Energies”. In: *The Journal of Chemical Physics* 131.7 (Aug. 2009), p. 074104 (cit. on p. 9).

[18] Matthias Rupp. “Machine Learning for Quantum Mechanics in a Nutshell”. In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1058–1073 (cit. on p. 11).

[19] *Computational Complexity of Machine Learning Algorithms*. https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/ (cit. on p. 11).

[20] Zachary J. Baum et al. “Artificial Intelligence in Chemistry: Current Trends and Future Directions”. In: *Journal of Chemical Information and Modeling* 61.7 (July 2021), pp. 3197–3212 (cit. on p. 12).

[21] Max Mowbray et al. “Industrial Data Science – a Review of Machine Learning Applications for Chemical and Process Industries”. In: *Reaction Chemistry & Engineering* 7.7 (June 2022), pp. 1471–1509 (cit. on p. 12).

[22] Raquel Rodríguez-Pérez, Filip Miljković, and Jürgen Bajorath. “Machine Learning in Chemoinformatics and Medicinal Chemistry”. In: *Annual Review of Biomedical Data Science* 5.1 (2022), pp. 43–65 (cit. on p. 12).

[23] S. Wold et al. “The Collinearity Problem in Linear Regression. The Partial Least Squares (PLS) Approach to Generalized Inverses”. In: *SIAM Journal on Scientific and Statistical Computing* 5.3 (Sept. 1984), pp. 735–743 (cit. on pp. 31, 256).

[24] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Ed. by Francis Bach. Adaptive Computation and Machine Learning Series. Cambridge, MA, USA: MIT Press, Nov. 2005 (cit. on p. 31).

[25] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32 (cit. on pp. 32, 256).

[26] J. R. Quinlan. “Induction of Decision Trees”. In: *Machine Learning* 1.1 (Mar. 1986), pp. 81–106 (cit. on p. 32).

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 34).

[28] Albert P. Bartók et al. “Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons”. In: *Physical Review Letters* 104.13 (Apr. 2010), p. 136403 (cit. on pp. 46, 235, 282).

[29] Albert P. Bartók and Gábor Csányi. “Gaussian Approximation Potentials: A Brief Tutorial Introduction”. In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1051–1057 (cit. on p. 46).

[30] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, June 2011, pp. 315–323 (cit. on p. 69).

[31] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Feb. 2015. arXiv: 1502.01852 [cs] (cit. on p. 70).

[32] Hugh R. Wilson and Jack D. Cowan. "Excitatory and Inhibitory Interactions in Localized Populations of Model Neurons". In: *Biophysical Journal* 12.1 (Jan. 1972), pp. 1–24 (cit. on p. 71).

[33] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. Illustrated edition. Cambridge, Massachusetts: The MIT Press, Mar. 2019 (cit. on p. 83).

[34] Oludare Isaac Abiodun et al. "State-of-the-Art in Artificial Neural Network Applications: A Survey". In: *Heliyon* 4.11 (Nov. 2018) (cit. on p. 84).

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on p. 84).

[36] Herbert Jaeger and Harald Haas. "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication". In: *Science* 304.5667 (Apr. 2004), pp. 78–80 (cit. on p. 84).

[37] Laith Alzubaidi et al. "Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions". In: *Journal of Big Data* 8.1 (Mar. 2021), p. 53 (cit. on p. 84).

[38] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. "Deep Learning for AI". In: *Communications of the ACM* 64.7 (June 2021), pp. 58–65 (cit. on p. 103).

[39] Robert Reuven Sokal and Charles Duncan Michener. *A Statistical Method for Evaluating Systematic Relationships*. University of Kansas, 1958 (cit. on p. 112).

[40] J. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics* 5.1 (Jan. 1967), pp. 281–298 (cit. on p. 112).

[41] Gale Young and A. S. Householder. "Discussion of a Set of Points in Terms of Their Mutual Distances". In: *Psychometrika* 3.1 (Mar. 1938), pp. 19–22 (cit. on p. 116).

[42] Warren S. Torgerson. "Multidimensional Scaling: I. Theory and Method". In: *Psychometrika* 17.4 (Dec. 1952), pp. 401–419 (cit. on p. 116).

[43] Aldo Glielmo et al. "Unsupervised Learning Methods for Molecular Simulation Data". In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 9722–9758 (cit. on pp. 116, 258).

[44] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 2000), pp. 2319–2323 (cit. on p. 116).

[45] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". In: *Neural Computation* 10.5 (July 1998), pp. 1299–1319 (cit. on p. 118).

[46] R. R. Coifman et al. "Geometric Diffusions as a Tool for Harmonic Analysis and Structure Definition of Data: Diffusion Maps". In: *Proceedings of the National Academy of Sciences* 102.21 (May 2005), pp. 7426–7431 (cit. on p. 119).

[47] Ronald R. Coifman and Stéphane Lafon. "Diffusion Maps". In: *Applied and Computational Harmonic Analysis*. Special Issue: Diffusion Maps and Wavelets 21.1 (July 2006), pp. 5–30 (cit. on p. 119).

[48] Z. Trstanova, B. Leimkuhler, and T. Lelièvre. “Local and Global Perspectives on Diffusion Maps in the Analysis of Molecular Systems”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 476.2233 (Jan. 2020), p. 20190036 (cit. on p. 120).

[49] Boaz Nadler et al. “Diffusion Maps, Spectral Clustering and Reaction Coordinates of Dynamical Systems”. In: *Applied and Computational Harmonic Analysis*. Special Issue: Diffusion Maps and Wavelets 21.1 (July 2006), pp. 113–127 (cit. on p. 120).

[50] Mark A. Kramer. “Nonlinear Principal Component Analysis Using Autoassociative Neural Networks”. In: *AIChE Journal* 37.2 (1991), pp. 233–243 (cit. on p. 120).

[51] D. Ballard. “Modular Learning in Neural Networks”. In: *AAAI*. 1987 (cit. on p. 120).

[52] Wolfgang P. Schleich et al. “Schrödinger Equation Revisited”. In: *Proceedings of the National Academy of Sciences* 110.14 (Apr. 2013), pp. 5374–5379 (cit. on p. 127).

[53] E. Schrödinger. “Quantisierung Als Eigenwertproblem”. In: *Annalen der Physik* 384.4 (1926), pp. 361–376 (cit. on p. 127).

[54] E. Schrödinger. “Quantisierung Als Eigenwertproblem”. In: *Annalen der Physik* 386.18 (1926), pp. 109–139 (cit. on p. 127).

[55] Attila Szabo and Neil S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Reprint Edition. Mineola, N.Y: Dover Publications, July 1996 (cit. on pp. 129, 155, 172, 174).

[56] Christopher J. Cramer. *Essentials of Computational Chemistry: Theories and Models*. 2nd edition. Chichester, West Sussex, England ; Hoboken, NJ: Wiley, Oct. 2004 (cit. on pp. 129, 135).

[57] Frank Jensen. *Introduction to Computational Chemistry*. 3rd edition. Chichester, UK ; Hoboken, NJ: Wiley, Feb. 2017 (cit. on pp. 129, 133, 138, 161).

[58] Peter W. Atkins and Ronald S. Friedman. *Molecular Quantum Mechanics*. 5th edition. Oxford ; New York: Oxford University Press, Dec. 2010 (cit. on pp. 130, 136, 156).

[59] Richard A. Friesner. “Ab Initio Quantum Chemistry: Methodology and Applications”. In: *Proceedings of the National Academy of Sciences* 102.19 (May 2005), pp. 6648–6653 (cit. on p. 133).

[60] Trygve Helgaker, Poul Jorgensen, and Jeppe Olsen. *Molecular Electronic-Structure Theory*. John Wiley & Sons, Aug. 2014 (cit. on p. 133).

[61] Ireneusz Grabowski et al. “Comparing Ab Initio Density-Functional and Wave Function Theories: The Impact of Correlation on the Electronic Density and the Role of the Correlation Potential”. In: *The Journal of Chemical Physics* 135.11 (Sept. 2011), p. 114111 (cit. on p. 135).

[62] Walter Thiel. “Semiempirical Quantum–Chemical Methods”. In: *WIREs Computational Molecular Science* 4.2 (2014), pp. 145–157 (cit. on p. 135).

[63] Anders S. Christensen et al. “Semiempirical Quantum Mechanical Methods for Noncovalent Interactions for Chemical and Biochemical Applications”. In: *Chemical Reviews* 116.9 (May 2016), pp. 5301–5337 (cit. on p. 135).

[64] Kristian K□iž and Jan □ezáč. “Benchmarking of Semiempirical Quantum-Mechanical Methods on Systems Relevant to Computer-Aided Drug Design”. In: *Journal of Chemical Information and Modeling* 60.3 (Mar. 2020), pp. 1453–1460 (cit. on p. 135).

[65] Max Wolfsberg and Lindsay Helmholz. “The Spectra and Electronic Structure of the Tetrahedral Ions MnO4-, CrO4–, and ClO4-”. In: *The Journal of Chemical Physics* 20.5 (May 1952), pp. 837–843 (cit. on p. 138).

[66] J. Harris. “Simplified Method for Calculating the Energy of Weakly Interacting Fragments”. In: *Physical Review B* 31.4 (Feb. 1985), pp. 1770–1779 (cit. on p. 138).

[67] P. Hohenberg and W. Kohn. “Inhomogeneous Electron Gas”. In: *Physical Review* 136.3B (Nov. 1964), B864–B871 (cit. on p. 147).

[68] Roberto Peverati and Donald G. Truhlar. “Quest for a Universal Density Functional: The Accuracy of Density Functionals across a Broad Spectrum of Databases in Chemistry and Physics”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372.2011 (Mar. 2014), p. 20120476 (cit. on p. 149).

[69] Matthias Ernzerhof and Gustavo E. Scuseria. “Assessment of the Perdew–Burke–Ernzerhof Exchange-Correlation Functional”. In: *The Journal of Chemical Physics* 110.11 (Mar. 1999), pp. 5029–5036 (cit. on p. 149).

[70] Pratibha Dev, Saurabh Agrawal, and Niall J. English. “Determining the Appropriate Exchange-Correlation Functional for Time-Dependent Density Functional Theory Studies of Charge-Transfer Excitations in Organic Dyes”. In: *The Journal of Chemical Physics* 136.22 (June 2012), p. 224301 (cit. on p. 149).

[71] Roberto Peverati and Donald G. Truhlar. “Exchange–Correlation Functional with Good Accuracy for Both Structural and Energetic Properties While Depending Only on the Density and Its Gradient”. In: *Journal of Chemical Theory and Computation* 8.7 (July 2012), pp. 2310–2319 (cit. on p. 149).

[72] Wenjing Zhang, Donald G. Truhlar, and Mingsheng Tang. “Tests of Exchange-Correlation Functional Approximations Against Reliable Experimental Data for Average Bond Energies of 3d Transition Metal Compounds”. In: *Journal of Chemical Theory and Computation* 9.9 (Sept. 2013), pp. 3965–3977 (cit. on p. 149).

[73] Bikash Kanungo, Paul M. Zimmerman, and Vikram Gavini. “Exact Exchange-Correlation Potentials from Ground-State Electron Densities”. In: *Nature Communications* 10.1 (Oct. 2019), p. 4497 (cit. on p. 149).

[74] Jeong-Hwan Han and Takuji Oda. “Performance of Exchange-Correlation Functionals in Density Functional Theory Calculations for Liquid Metal: A Benchmark Test for Sodium”. In: *The Journal of Chemical Physics* 148.14 (Apr. 2018), p. 144501 (cit. on p. 149).

[75] S. Sharma et al. “Source-Free Exchange-Correlation Magnetic Fields in Density Functional Theory”. In: *Journal of Chemical Theory and Computation* 14.3 (Mar. 2018), pp. 1247–1253 (cit. on p. 149).

[76] Pedro Borlido et al. “Large-Scale Benchmark of Exchange–Correlation Functionals for the Determination of Electronic Band Gaps of Solids”. In: *Journal of Chemical Theory and Computation* 15.9 (Sept. 2019), pp. 5069–5079 (cit. on p. 149).

[77] Eduardo Fabiano et al. “Investigation of the Exchange-Correlation Potentials of Functionals Based on the Adiabatic Connection Interpolation”. In: *Journal of Chemical Theory and Computation* 15.2 (Feb. 2019), pp. 1006–1015 (cit. on p. 149).

[78] Tom Cardeynaels et al. “Finding the Optimal Exchange–Correlation Functional to Describe the Excited State Properties of Push–Pull Organic Dyes Designed for Thermally Activated Delayed Fluorescence”. In: *Physical Chemistry Chemical Physics* 22.28 (July 2020), pp. 16387–16399 (cit. on p. 149).

[79] Marcelo T. de Oliveira et al. “Do Double-Hybrid Exchange–Correlation Functionals Provide Accurate Chemical Shifts? A Benchmark Assessment for Proton NMR”. In: *Journal of Chemical Theory and Computation* 17.11 (Nov. 2021), pp. 6876–6885 (cit. on p. 149).

[80] Zhandos Moldabekov et al. “Benchmarking Exchange-Correlation Functionals in the Spin-Polarized Inhomogeneous Electron Gas under Warm Dense Conditions”. In: *Physical Review B* 105.3 (Jan. 2022), p. 035134 (cit. on p. 149).

[81] Vincent L. Lignères and Emily A. Carter. “An Introduction to Orbital-Free Density Functional Theory”. In: *Handbook of Materials Modeling: Methods*. Ed. by Sidney Yip. Dordrecht: Springer Netherlands, 2005, pp. 137–148 (cit. on p. 149).

[82] W. Kohn and L. J. Sham. “Self-Consistent Equations Including Exchange and Correlation Effects”. In: *Physical Review* 140.4A (Nov. 1965), A1133–A1138 (cit. on p. 150).

[83] Richard M. Martin. *Electronic Structure: Basic Theory and Practical Methods*. Second. Cambridge: Cambridge University Press, 2020 (cit. on pp. 150, 161).

[84] J. C. Slater. “The Theory of Complex Spectra”. In: *Physical Review* 34.10 (Nov. 1929), pp. 1293–1322 (cit. on p. 155).

[85] Eberhard Engel and Reiner M. Dreizler. *Density Functional Theory: An Advanced Course*. Theoretical and Mathematical Physics. Berlin, Heidelberg: Springer, 2011 (cit. on p. 161).

[86] John P. Perdew and Karla Schmidt. “Jacob’s Ladder of Density Functional Approximations for the Exchange-Correlation Energy”. In: *AIP Conference Proceedings* 577.1 (July 2001), pp. 1–20 (cit. on p. 169).

[87] John P. Perdew, Kieron Burke, and Matthias Ernzerhof. “Generalized Gradient Approximation Made Simple”. In: *Physical Review Letters* 77.18 (Oct. 1996), pp. 3865–3868 (cit. on p. 171).

[88] Per-Olov Löwdin. “On the Non-Orthogonality Problem Connected with the Use of Atomic Wave Functions in the Theory of Molecules and Crystals”. In: *The Journal of Chemical Physics* 18.3 (Mar. 1950), pp. 365–375 (cit. on p. 175).

[89] U. Chandra Singh and Peter A. Kollman. “An Approach to Computing Electrostatic Charges for Molecules”. In: *Journal of Computational Chemistry* 5.2 (1984), pp. 129–145 (cit. on p. 176).

[90] Lisa Emily Chirlian and Michelle Miller Francl. “Atomic Charges Derived from Electrostatic Potentials: A Detailed Study”. In: *Journal of Computational Chemistry* 8.6 (1987), pp. 894–905 (cit. on p. 176).

[91] Curt M. Breneman and Kenneth B. Wiberg. “Determining Atom-Centered Monopoles from Molecular Electrostatic Potentials. The Need for High Sampling Density in Formamide Conformational Analysis”. In: *Journal of Computational Chemistry* 11.3 (1990), pp. 361–373 (cit. on p. 176).

[92] Wendy D. Cornell et al. “Application of RESP Charges to Calculate Conformational Energies, Hydrogen Bond Energies, and Free Energies of Solvation”. In: *Journal of the American Chemical Society* 115.21 (Oct. 1993), pp. 9620–9631 (cit. on p. 176).

[93] F. L. Hirshfeld. “Bonded-Atom Fragments for Describing Molecular Charge Densities”. In: *Theoretica chimica acta* 44.2 (June 1977), pp. 129–138 (cit. on p. 177).

[94] R. F. W. Bader. “Atoms in Molecules”. In: *Accounts of Chemical Research* 18.1 (Jan. 1985), pp. 9–15 (cit. on p. 177).

[95] Richard F. W. Bader. “A Quantum Theory of Molecular Structure and Its Applications”. In: *Chemical Reviews* 91.5 (July 1991), pp. 893–928 (cit. on p. 177).

[96] Willis B. Person and James H. Newton. “Dipole Moment Derivatives and Infrared Intensities. I. Polar Tensors”. In: *The Journal of Chemical Physics* 61.3 (Aug. 1974), pp. 1040–1049 (cit. on p. 177).

[97] Alberto Milani and Chiara Castiglioni. “Atomic Charges from Atomic Polar Tensors: A Comparison of Methods”. In: *Journal of Molecular Structure: THEOCHEM* 955.1 (Sept. 2010), pp. 158–164 (cit. on p. 177).

[98] Martin Thomas et al. “Computing Vibrational Spectra from Ab Initio Molecular Dynamics”. In: *Physical Chemistry Chemical Physics* 15.18 (Apr. 2013), pp. 6608–6622 (cit. on pp. 183, 184).

[99] L. Jensen et al. “Theory and Method for Calculating Resonance Raman Scattering from Resonance Polarizability Derivatives”. In: *The Journal of Chemical Physics* 123.17 (Nov. 2005), p. 174110 (cit. on p. 184).

[100] Johann Mattiat and Sandra Luber. “Time Domain Simulation of (Resonance) Raman Spectra of Liquids in the Short Time Approximation”. In: *Journal of Chemical Theory and Computation* 17.1 (Jan. 2021), pp. 344–356 (cit. on p. 184).

[101] Kevin Yang et al. “Analyzing Learned Molecular Representations for Property Prediction”. In: *Journal of Chemical Information and Modeling* 59.8 (Aug. 2019), pp. 3370–3388 (cit. on p. 189).

[102] Tomaž Stepišnik et al. “A Comprehensive Comparison of Molecular Feature Representations for Use in Predictive Modeling”. In: *Computers in Biology and Medicine* 130 (Mar. 2021), p. 104197 (cit. on p. 190).

[103] Felix Musil et al. “Physics-Inspired Structural Representations for Molecules and Materials”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 9759–9815 (cit. on pp. 192, 194, 205, 207, 259).

[104] John A. Keith et al. “Combining Machine Learning and Computational Chemistry for Predictive Insights Into Chemical Systems”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 9816–9872 (cit. on pp. 194, 259).

[105] Matthias Rupp et al. “Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning”. In: *Physical Review Letters* 108.5 (Jan. 2012), p. 058301 (cit. on pp. 195, 219, 237).

[106] Felix Faber et al. “Crystal Structure Representations for Machine Learning Models of Formation Energies”. In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1094–1101 (cit. on pp. 196, 197).

[107] Bastiaan J. Braams and Joel M. Bowman. “Permutationally Invariant Potential Energy Surfaces in High Dimensionality”. In: *International Reviews in Physical Chemistry* 28.4 (Oct. 2009), pp. 577–606 (cit. on p. 196).

[108] Katja Hansen et al. “Assessment and Validation of Machine Learning Methods for Predicting Molecular Atomization Energies”. In: *Journal of Chemical Theory and Computation* 9.8 (Aug. 2013), pp. 3404–3419 (cit. on pp. 196, 237).

[109] Grégoire A. Gallet and Fabio Pietrucci. “Structural Cluster Analysis of Chemical Reactions in Solution”. In: *The Journal of Chemical Physics* 139.7 (Aug. 2013), p. 074101 (cit. on p. 196).

[110] Christopher R. Collins et al. “Constant Size Descriptors for Accurate Machine Learning Models of Molecular Properties”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241718 (cit. on p. 196).

[111] Katja Hansen et al. “Machine Learning Predictions of Molecular Properties: Accurate Many-Body Potentials and Nonlocality in Chemical Space”. In: *The Journal of Physical Chemistry Letters* 6.12 (June 2015), pp. 2326–2331 (cit. on pp. 199, 237).

[112] *QML: A Python Toolkit for Quantum Machine Learning — QML 0.4.0 Documentation*. https://www.qmlcode.org/ (cit. on p. 199).

[113] Weitao Yang. “Direct Calculation of Electron Density in Density-Functional Theory”. In: *Physical Review Letters* 66.11 (Mar. 1991), pp. 1438–1441 (cit. on p. 200).

[114] Giulia Galli and Michele Parrinello. “Large Scale Electronic Structure Calculations”. In: *Physical Review Letters* 69.24 (Dec. 1992), pp. 3547–3550 (cit. on p. 200).

[115] Stefan Goedecker. “Linear Scaling Electronic Structure Methods”. In: *Reviews of Modern Physics* 71.4 (July 1999), pp. 1085–1123 (cit. on p. 200).

[116] E. Prodan and W. Kohn. “Nearsightedness of Electronic Matter”. In: *Proceedings of the National Academy of Sciences* 102.33 (Aug. 2005), pp. 11635–11638 (cit. on p. 200).

[117] Michele Ceriotti, Michael J. Willatt, and Gábor Csányi. “Machine Learning of Atomic-Scale Properties Based on Physical Principles”. In: *Handbook of Materials Modeling : Methods: Theory and Modeling*. Ed. by Wanda Andreoni and Sidney Yip. Cham: Springer International Publishing, 2018, pp. 1–27 (cit. on pp. 200, 224, 236).

[118] Albert P. Bartók, Risi Kondor, and Gábor Csányi. “On Representing Chemical Environments”. In: *Physical Review B* 87.18 (May 2013), p. 184115 (cit. on pp. 200, 202).

[119] Sandip De et al. “Comparing Molecules and Solids across Structural and Alchemical Space”. In: *Physical Chemistry Chemical Physics* 18.20 (May 2016), pp. 13754–13769 (cit. on pp. 200, 201, 237).

[120] Jörg Behler. “Atom-Centered Symmetry Functions for Constructing High-Dimensional Neural Network Potentials”. In: *The Journal of Chemical Physics* 134.7 (Feb. 2011), p. 074106 (cit. on pp. 203, 255).

[121] Felix A. Faber et al. “Alchemical and Structural Distribution Based Representation for Universal Quantum Machine Learning”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241717 (cit. on pp. 205, 207).

[122] Bing Huang and O. Anatole von Lilienfeld. “Quantum Machine Learning Using Atom-in-Molecule-Based Fragments Selected on the Fly”. In: *Nature Chemistry* 12.10 (Oct. 2020), pp. 945–951 (cit. on pp. 205, 237).

[123] Kijeong Kwac and Minhaeng Cho. “Differential Evolution Algorithm Approach for Describing Vibrational Solvatochromism”. In: *The Journal of Chemical Physics* 151.13 (Oct. 2019), p. 134112 (cit. on p. 205).

[124] M. Gastegger et al. “wACSF—Weighted Atom-Centered Symmetry Functions as Descriptors in Machine Learning Potentials”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241709 (cit. on pp. 206, 238).

[125] Martin P. Bircher, Andreas Singraber, and Christoph Dellago. “Improved Description of Atomic Environments Using Low-Cost Polynomial Functions with Compact Support”. In: *Machine Learning: Science and Technology* 2.3 (June 2021), p. 035026 (cit. on p. 206).

[126] Kangyu Zhang, Lichang Yin, and Gang Liu. “Physically Inspired Atom-Centered Symmetry Functions for the Construction of High Dimensional Neural Network Potential Energy Surfaces”. In: *Computational Materials Science* 186 (Jan. 2021), p. 110071 (cit. on p. 206).

[127] Marco Eckhoff and Jörg Behler. “High-Dimensional Neural Network Potentials for Magnetic Systems Using Spin-Dependent Atom-Centered Symmetry Functions”. In: *npj Computational Materials* 7.1 (Oct. 2021), pp. 1–11 (cit. on p. 206).

[128] Kijeong Kwac, Holly Freedman, and Minhaeng Cho. “Machine Learning Approach for Describing Water OH Stretch Vibrations”. In: *Journal of Chemical Theory and Computation* 17.10 (Oct. 2021), pp. 6353–6365 (cit. on p. 206).

[129] Bing Huang and O. Anatole von Lilienfeld. “Communication: Understanding Molecular Representations in Machine Learning: The Role of Uniqueness and Target Similarity”. In: *The Journal of Chemical Physics* 145.16 (Oct. 2016), p. 161102 (cit. on pp. 207, 237, 238).

[130] Felix A. Faber et al. “Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error”. In: *Journal of Chemical Theory and Computation* 13.11 (Nov. 2017), pp. 5255–5264 (cit. on pp. 207, 237, 238).

[131] Haoyan Huo and Matthias Rupp. “Unified Representation of Molecules and Crystals for Machine Learning”. In: *Machine Learning: Science and Technology* (2022) (cit. on pp. 207, 237).

[132] Marcel F. Langer, Alex Goeßmann, and Matthias Rupp. “Representations of Molecules and Materials for Interpolation of Quantum-Mechanical Simulations via Machine Learning”. In: *npj Computational Materials* 8.1 (Mar. 2022), pp. 1–14 (cit. on p. 207).

[133] Ralf Drautz. “Atomic Cluster Expansion for Accurate and Transferable Interatomic Potentials”. In: *Physical Review B* 99.1 (Jan. 2019), p. 014104 (cit. on p. 207).

[134] Dávid Péter Kovács et al. “Linear Atomic Cluster Expansion Force Fields for Organic Molecules: Beyond RMSE”. In: *Journal of Chemical Theory and Computation* 17.12 (Dec. 2021), pp. 7696–7711 (cit. on p. 207).

[135] Jigyasa Nigam, Sergey Pozdnyakov, and Michele Ceriotti. “Recursive Evaluation and Iterative Contraction of N-body Equivariant Features”. In: *The Journal of Chemical Physics* 153.12 (Sept. 2020), p. 121101 (cit. on p. 207).

[136] Peter Kirkpatrick and Clare Ellis. “Chemical Space”. In: *Nature* 432.7019 (Dec. 2004), pp. 823–823 (cit. on p. 212).

[137] Daniel Probst and Jean-Louis Reymond. “Visualization of Very Large High-Dimensional Data Sets as Minimum Spanning Trees”. In: *Journal of Cheminformatics* 12.1 (Feb. 2020), p. 12 (cit. on p. 214).

[138] Lars Ruddigkeit et al. “Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17”. In: *Journal of Chemical Information and Modeling* 52.11 (Nov. 2012), pp. 2864–2875 (cit. on pp. 216, 219).

[139] Raghunathan Ramakrishnan et al. “Quantum Chemistry Structures and Properties of 134 Kilo Molecules”. In: *Scientific Data* 1.1 (Aug. 2014), p. 140022 (cit. on pp. 216, 219).

[140] Lorenz C. Blum and Jean-Louis Reymond. “970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13”. In: *Journal of the American Chemical Society* 131.25 (July 2009), pp. 8732–8733 (cit. on p. 219).

[141] Grégoire Montavon et al. “Machine Learning of Molecular Electronic Properties in Chemical Compound Space”. In: *New Journal of Physics* 15.9 (Sept. 2013), p. 095003 (cit. on pp. 219, 237).

[142] Raghunathan Ramakrishnan et al. “Electronic Spectra from TDDFT and Machine Learning in Chemical Space”. In: *The Journal of Chemical Physics* 143.8 (Aug. 2015), p. 084111 (cit. on p. 219).

[143] Kristof Schütt et al. “SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 219, 263, 265, 283).

[144] Kristof T. Schütt et al. “Quantum-Chemical Insights from Deep Tensor Neural Networks”. In: *Nature Communications* 8.1 (Jan. 2017), p. 13890 (cit. on pp. 219, 262, 283).

[145] Justin S. Smith, Olexandr Isayev, and Adrian E. Roitberg. “ANI-1, A Data Set of 20 Million Calculated off-Equilibrium Conformations for Organic Molecules”. In: *Scientific Data* 4.1 (Dec. 2017), p. 170193 (cit. on p. 219).

[146] David Mendez et al. “ChEMBL: Towards Direct Deposition of Bioassay Data”. In: *Nucleic Acids Research* 47.D1 (Jan. 2019), pp. D930–D940 (cit. on p. 219).

[147] Kuzma Khrabrov et al. “nablaDFT: Large-Scale Conformational Energy and Hamiltonian Prediction Benchmark and Dataset”. In: *Physical Chemistry Chemical Physics* 24.42 (Nov. 2022), pp. 25853–25863 (cit. on p. 220).

[148] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data Using T-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605 (cit. on p. 221).

[149] Anna C. Belkina et al. “Automated Optimized Parameters for T-distributed Stochastic Neighbor Embedding Improve Visualization and Analysis of Large Datasets”. In: *Nature Communications* 10.1 (Nov. 2019), p. 5415 (cit. on p. 221).

[150] Geoffrey E Hinton and Sam Roweis. “Stochastic Neighbor Embedding”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Becker, S. Thrun, and K. Obermayer. Vol. 15. MIT Press, 2002 (cit. on p. 221).

[151] Julia Westermayr et al. “Perspective on Integrating Machine Learning into Computational Chemistry and Materials Science”. In: *The Journal of Chemical Physics* 154.23 (June 2021), p. 230903 (cit. on pp. 224, 258).

[152] Jörg Behler. “Perspective: Machine Learning Potentials for Atomistic Simulations”. In: *The Journal of Chemical Physics* 145.17 (Nov. 2016), p. 170901 (cit. on p. 233).

[153] V. Botu et al. “Machine Learning Force Fields: Construction, Validation, and Outlook”. In: *The Journal of Physical Chemistry C* 121.1 (Jan. 2017), pp. 511–522 (cit. on p. 233).

[154] Felix Brockherde et al. “Bypassing the Kohn-Sham Equations with Machine Learning”. In: *Nature Communications* 8.1 (Oct. 2017), p. 872 (cit. on p. 233).

[155] Volker L. Deringer, Miguel A. Caro, and Gábor Csányi. “Machine Learning Interatomic Potentials as Emerging Tools for Materials Science”. In: *Advanced Materials* 31.46 (2019), p. 1902765 (cit. on pp. 233, 235).

[156] Bohayra Mortazavi et al. “Machine-Learning Interatomic Potentials Enable First-Principles Multiscale Modeling of Lattice Thermal Conductivity in Graphene/Borophene Heterostructures”. In: *Materials Horizons* 7.9 (Sept. 2020), pp. 2359–2367 (cit. on p. 233).

[157] Yunxing Zuo et al. “Performance and Cost Assessment of Machine Learning Interatomic Potentials”. In: *The Journal of Physical Chemistry A* 124.4 (Jan. 2020), pp. 731–745 (cit. on p. 233).

[158] Gurjot Dhaliwal, Prasanth B. Nair, and Chandra Veer Singh. “Machine Learned Interatomic Potentials Using Random Features”. In: *npj Computational Materials* 8.1 (Jan. 2022), pp. 1–10 (cit. on p. 233).

[159] Frank Noé et al. “Machine Learning for Molecular Simulation”. In: *Annual Review of Physical Chemistry* 71.1 (2020), pp. 361–390 (cit. on pp. 233, 258).

[160] Ganesh Sivaraman et al. “Machine-Learned Interatomic Potentials by Active Learning: Amorphous and Liquid Hafnium Dioxide”. In: *npj Computational Materials* 6.1 (July 2020), pp. 1–8 (cit. on p. 233).

[161] Christoph Schran et al. “Machine Learning Potentials for Complex Aqueous Systems Made Simple”. In: *Proceedings of the National Academy of Sciences* 118.38 (Sept. 2021), e2110077118 (cit. on p. 233).

[162] Chenghan Li and Gregory A. Voth. “Using Machine Learning to Greatly Accelerate Path Integral Ab Initio Molecular Dynamics”. In: *Journal of Chemical Theory and Computation* 18.2 (Feb. 2022), pp. 599–604 (cit. on p. 233).

[163] Alexander V. Shapeev. “Moment Tensor Potentials: A Class of Systematically Improvable Interatomic Potentials”. In: *Multiscale Modeling & Simulation* 14.3 (Jan. 2016), pp. 1153–1173 (cit. on p. 236).

[164] A. P. Thompson et al. “Spectral Neighbor Analysis Method for Automated Generation of Quantum-Accurate Interatomic Potentials”. In: *Journal of Computational Physics* 285 (Mar. 2015), pp. 316–330 (cit. on pp. 236, 282).

[165] Zhi Deng et al. “An Electrostatic Spectral Neighbor Analysis Potential for Lithium Nitride”. In: *npj Computational Materials* 5.1 (July 2019), pp. 1–8 (cit. on p. 236).

[166] M. A. Cusentino, M. A. Wood, and A. P. Thompson. “Explicit Multielement Extension of the Spectral Neighbor Analysis Potential for Chemically Complex Systems”. In: *The Journal of Physical Chemistry A* 124.26 (July 2020), pp. 5456–5464 (cit. on p. 236).

[167] M. Domina, M. Cobelli, and S. Sanvito. “Spectral Neighbor Representation for Vector Fields: Machine Learning Potentials Including Spin”. In: *Physical Review B* 105.21 (June 2022), p. 214439 (cit. on p. 236).

[168] Jörg Behler and Michele Parrinello. “Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces”. In: *Physical Review Letters* 98.14 (Apr. 2007), p. 146401 (cit. on pp. 236, 254, 255).

[169] J. S. Smith, O. Isayev, and A. E. Roitberg. “ANI-1: An Extensible Neural Network Potential with DFT Accuracy at Force Field Computational Cost”. In: *Chemical Science* 8.4 (Mar. 2017), pp. 3192–3203 (cit. on pp. 236, 238, 260, 282).

[170] Justin S. Smith et al. “Less Is More: Sampling Chemical Space with Active Learning”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241733 (cit. on pp. 236, 238, 282).

[171] Justin S. Smith et al. “Approaching Coupled Cluster Accuracy with a General-Purpose Neural Network Potential through Transfer Learning”. In: *Nature Communications* 10.1 (July 2019), p. 2903 (cit. on pp. 236, 238).

[172] Christian Devereux et al. “Extending the Applicability of the ANI Deep Learning Molecular Potential to Sulfur and Halogens”. In: *Journal of Chemical Theory and Computation* 16.7 (July 2020), pp. 4192–4202 (cit. on p. 236).

[173] Grégoire Montavon et al. “Learning Invariant Representations of Molecules for Atomization Energy Prediction”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012 (cit. on p. 237).

[174] Albert P. Bartók et al. “Machine Learning Unifies the Modeling of Materials and Molecules”. In: *Science Advances* 3.12 (Dec. 2017), e1701816 (cit. on p. 237).

[175] Michael J. Willatt, Félix Musil, and Michele Ceriotti. “Feature Optimization for Atomistic Machine Learning Yields a Data-Driven Construction of the Periodic Table of the Elements”. In: *Physical Chemistry Chemical Physics* 20.47 (Dec. 2018), pp. 29661–29668 (cit. on p. 237).

[176] Anders S. Christensen et al. “FCHL Revisited: Faster and More Accurate Quantum Machine Learning”. In: *The Journal of Chemical Physics* 152.4 (Jan. 2020), p. 044107 (cit. on p. 237).

[177] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, July 2017, pp. 1263–1272 (cit. on pp. 238, 268).

[178] Fang Hou et al. “Comparison Study on the Prediction of Multiple Molecular Properties by Various Neural Networks”. In: *The Journal of Physical Chemistry A* 122.46 (Nov. 2018), pp. 9128–9134 (cit. on p. 238).

[179] K. T. Schütt et al. “SchNet – A Deep Learning Architecture for Molecules and Materials”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241722 (cit. on pp. 238, 263, 265).

[180] Nicholas Lubbers, Justin S. Smith, and Kipton Barros. “Hierarchical Modeling of Molecular Energies Using a Deep Neural Network”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241715 (cit. on pp. 238, 255).

[181] Oliver T. Unke and Markus Meuwly. “A Reactive, Scalable, and Transferable Model for Molecular Energies from a Neural Network Approach Based on Local Information”. In: *The Journal of Chemical Physics* 148.24 (June 2018), p. 241708 (cit. on p. 238).

[182] Rafael Gómez-Bombarelli et al. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: *ACS Central Science* 4.2 (Feb. 2018), pp. 268–276 (cit. on pp. 238, 280).

[183] Oliver T. Unke and Markus Meuwly. “PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges”. In: *Journal of Chemical Theory and Computation* 15.6 (June 2019), pp. 3678–3693 (cit. on pp. 238, 252, 257).

[184] K. T. Schütt et al. “Unifying Machine Learning and Quantum Chemistry with a Deep Neural Network for Molecular Wavefunctions”. In: *Nature Communications* 10.1 (Nov. 2019), p. 5024 (cit. on pp. 238, 264).

[185] Florbela Pereira et al. “Machine Learning Methods to Predict Density Functional Theory B3LYP Energies of HOMO and LUMO Orbitals”. In: *Journal of Chemical Information and Modeling* 57.1 (Jan. 2017), pp. 11–21 (cit. on p. 238).

[186] Annika Stuke et al. “Chemical Diversity in Molecular Orbital Energy Predictions with Kernel Ridge Regression”. In: *The Journal of Chemical Physics* 150.20 (May 2019), p. 204121 (cit. on p. 238).

[187] Alexander M. Chang, Jessica G. Freeze, and Victor S. Batista. “Hammett Neural Networks: Prediction of Frontier Orbital Energies of Tungsten–Benzylidyne Photoredox Complexes”. In: *Chemical Science* 10.28 (July 2019), pp. 6844–6854 (cit. on p. 238).

[188] Obaidur Rahaman and Alessio Gagliardi. “Deep Learning Total Energies and Orbital Energies of Large Organic Molecules Using Hybridization of Molecular Fingerprints”. In: *Journal of Chemical Information and Modeling* 60.12 (Dec. 2020), pp. 5971–5983 (cit. on p. 238).

[189] Gareth John Moore, Olivier Bardagot, and Natalie Banerji. “Deep Transfer Learning: A Fast and Accurate Tool to Predict the Energy Levels of Donor Molecules for Organic Photovoltaics”. In: *Advanced Theory and Simulations* 5.5 (2022), p. 2100511 (cit. on p. 238).

[190] Zong-Rong Ye et al. “Assessment of Predicting Frontier Orbital Energies for Small Organic Molecules Using Knowledge-Based and Structural Information”. In: *ACS Engineering Au* 2.4 (Aug. 2022), pp. 360–368 (cit. on p. 238).

[191] Zak E. Hughes et al. “Description of Potential Energy Surfaces of Molecules Using FFLUX Machine Learning Models”. In: *Journal of Chemical Theory and Computation* 15.1 (Jan. 2019), pp. 116–126 (cit. on p. 246).

[192] Kun Yao et al. “The TensorMol-0.1 Model Chemistry: A Neural Network Augmented with Long-Range Physics”. In: *Chemical Science* 9.8 (Feb. 2018), pp. 2261–2269 (cit. on pp. 246, 282).

[193] James L. McDonagh et al. “Machine Learning of Dynamic Electron Correlation Energies from Topological Atoms”. In: *Journal of Chemical Theory and Computation* 14.1 (Jan. 2018), pp. 216–224 (cit. on p. 247).

[194] Takuro Nudejima et al. “Machine-Learned Electron Correlation Model Based on Correlation Energy Density at Complete Basis Set Limit”. In: *The Journal of Chemical Physics* 151.2 (July 2019), p. 024104 (cit. on p. 247).

[195] Sebastian Dick and Marivi Fernandez-Serra. “Machine Learning Accurate Exchange and Correlation Functionals of the Electronic Density”. In: *Nature Communications* 11.1 (July 2020), p. 3509 (cit. on pp. 247, 248).

[196] Ruocheng Han, Mauricio Rodríguez-Mayorga, and Sandra Luber. “A Machine Learning Approach for MP2 Correlation Energies and Its Application to Organic Compounds”. In: *Journal of Chemical Theory and Computation* 17.2 (Feb. 2021), pp. 777–790 (cit. on p. 247).

[197] Etienne Cuierrier, Pierre-Olivier Roy, and Matthias Ernzerhof. “Constructing and Representing Exchange–Correlation Holes through Artificial Neural Networks”. In: *The Journal of Chemical Physics* 155.17 (Nov. 2021), p. 174121 (cit. on p. 247).

[198] Etienne Cuierrier et al. “The Fourth-Order Expansion of the Exchange Hole and Neural Networks to Construct Exchange–Correlation Functionals”. In: *The Journal of Chemical Physics* 157.17 (Nov. 2022), p. 171103 (cit. on p. 247).

[199] James Kirkpatrick et al. “Pushing the Frontiers of Density Functionals by Solving the Fractional Electron Problem”. In: *Science* 374.6573 (Dec. 2021), pp. 1385–1389 (cit. on p. 247).

[200] Yi Zhou et al. “Toward the Exact Exchange–Correlation Potential: A Three-Dimensional Convolutional Neural Network Construct”. In: *The Journal of Physical Chemistry Letters* 10.22 (Nov. 2019), pp. 7264–7269 (cit. on p. 250).

[201] Qin Wu and Weitao Yang. “A Direct Optimization Method for Calculating Density Functionals and Exchange–Correlation Potentials from Electron Densities”. In: *The Journal of Chemical Physics* 118.6 (Feb. 2003), pp. 2498–2509 (cit. on p. 250).

[202] Toby Lewis-Atwell, Piers A. Townsend, and Matthew N. Grayson. “Machine Learning Activation Energies of Chemical Reactions”. In: *WIREs Computational Molecular Science* 12.4 (2022), e1593 (cit. on p. 251).

[203] Colin A. Grambow, Lagnajit Pattanaik, and William H. Green. “Deep Learning of Activation Energies”. In: *The Journal of Physical Chemistry Letters* 11.8 (Apr. 2020), pp. 2992–2997 (cit. on p. 251).

[204] Xin Li et al. “Predicting Regioselectivity in Radical C-H Functionalization of Heterocycles through Machine Learning”. In: *Angewandte Chemie* 132.32 (2020), pp. 13355–13361 (cit. on p. 251).

[205] Brajesh K. Rai and Gregory A. Bakken. “Fast and Accurate Generation of Ab Initio Quality Atomic Charges Using Nonparametric Statistical Regression”. In: *Journal of Computational Chemistry* 34.19 (2013), pp. 1661–1671 (cit. on p. 252).

[206] Patrick Bleiziffer, Kay Schaller, and Sereina Riniker. “Machine Learning of Partial Charges Derived from High-Quality Quantum-Mechanical Calculations”. In: *Journal of Chemical Information and Modeling* 58.3 (Mar. 2018), pp. 579–590 (cit. on p. 252).

[207] Raymond E. Carhart, Dennis H. Smith, and R. Venkataraghavan. “Atom Pairs as Molecular Features in Structure-Activity Studies: Definition and Applications”. In: *Journal of Chemical Information and Computer Sciences* 25.2 (May 1985), pp. 64–73 (cit. on p. 252).

[208] Benjamin Nebgen et al. “Transferable Dynamic Molecular Charge Assignment Using Deep Neural Networks”. In: *Journal of Chemical Theory and Computation* 14.9 (Sept. 2018), pp. 4687–4698 (cit. on pp. 252, 255).

[209] Srinivasu Kancharlapalli et al. “Fast and Accurate Machine Learning Strategy for Calculating Partial Atomic Charges in Metal–Organic Frameworks”. In: *Journal of Chemical Theory and Computation* 17.5 (May 2021), pp. 3052–3064 (cit. on p. 252).

[210] Koichiro Kato et al. “High-Precision Atomic Charge Prediction for Protein Systems Using Fragment Molecular Orbital Calculation and Machine Learning”. In: *Journal of Chemical Information and Modeling* 60.7 (July 2020), pp. 3361–3368 (cit. on p. 252).

[211] Jike Wang et al. “DeepChargePredictor: A Web Server for Predicting QM-based Atomic Charges via State-of-the-Art Machine-Learning Algorithms”. In: *Bioinformatics* 37.22 (Nov. 2021), pp. 4255–4257 (cit. on p. 252).

[212] Lisanne Knijff and Chao Zhang. “Machine Learning Inference of Molecular Dipole Moment in Liquid Water”. In: *Machine Learning: Science and Technology* 2.3 (July 2021), 03LT03 (cit. on p. 253).

[213] Xiangyue Liu, Gerard Meijer, and Jesús Pérez-Ríos. “A Data-Driven Approach to Determine Dipole Moments of Diatomic Molecules”. In: *Physical Chemistry Chemical Physics* 22.42 (2020), pp. 24191–24200 (cit. on p. 253).

[214] Florbela Pereira and João Aires-de-Sousa. “Machine Learning for the Prediction of Molecular Dipole Moments Obtained by Density Functional Theory”. In: *Journal of Cheminformatics* 10.1 (Aug. 2018), p. 43 (cit. on p. 253).

[215] Carsten G. Staacke et al. “Kernel Charge Equilibration: Efficient and Accurate Prediction of Molecular Dipole Moments wi th a Machine-Learning Enhanced Electron Density Model”. In: *Machine Learning: Science and Technology* 3.1 (Mar. 2022), p. 015032 (cit. on p. 253).

[216] Jiace Sun, Lixue Cheng, and Thomas F. Miller. “Molecular Dipole Moment Learning via Rotationally Equivariant Derivative Kernels in Molecular-Orbital-Based Machine Learning”. In: *The Journal of Chemical Physics* 157.10 (Sept. 2022), p. 104109 (cit. on p. 253).

[217] Max Veit et al. “Predicting Molecular Dipole Moments by Combining Atomic Partial Charges and Atomic Dipoles”. In: *The Journal of Chemical Physics* 153.2 (July 2020), p. 024113 (cit. on p. 253).

[218] Minh Nguyen Vo et al. “Method for Predicting Dipole Moments of Complex Molecules for Use in Thermophysical Property Estimation”. In: *Industrial & Engineering Chemistry Research* 58.41 (Oct. 2019), pp. 19263–19270 (cit. on p. 253).

[219] Simon Axelrod, Eugene Shakhnovich, and Rafael Gómez-Bombarelli. “Excited State Non-Adiabatic Dynamics of Large Photoswitchable Molecules Using a Chemically Transferable Machine Learning Potential”. In: *Nature Communications* 13.1 (June 2022), p. 3440 (cit. on p. 253).

[220] Michael Gastegger, Jörg Behler, and Philipp Marquetand. “Machine Learning Molecular Dynamics for the Simulation of Infrared Spectra”. In: *Chemical Science* 8.10 (2017), pp. 6924–6935 (cit. on pp. 254, 255).

[221] Richard Beckmann et al. “Infrared Spectra at Coupled Cluster Accuracy from Neural Network Representations”. In: *Journal of Chemical Theory and Computation* (Aug. 2022) (cit. on p. 254).

[222] Jörg Behler. “Neural Network Potential-Energy Surfaces in Chemistry: A Tool for Large-Scale Simulations”. In: *Physical Chemistry Chemical Physics* 13.40 (Oct. 2011), pp. 17930–17955 (cit. on p. 254).

[223] Jörg Behler. “Constructing High-Dimensional Neural Network Potentials: A Tutorial Review”. In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1032–1050 (cit. on p. 254).

[224] Jan H. Schuur, Paul Selzer, and Johann Gasteiger. “The Coding of the Three-Dimensional Structure of Molecules by Molecular Transforms and Its Application to Structure-Spectra Correlations and Studies of Biological Activity”. In: *Journal of Chemical Information and Computer Sciences* 36.2 (Jan. 1996), pp. 334–344 (cit. on p. 255).

[225] Jan Schuur and Johann Gasteiger. “Infrared Spectra Simulation of Substituted Benzene Derivatives on the Basis of a 3D Structure Representation”. In: *Analytical Chemistry* 69.13 (July 1997), pp. 2398–2405 (cit. on p. 255).

[226] Robert Hecht-Nielsen. “Counterpropagation Networks”. In: *Applied Optics* 26.23 (Dec. 1987), pp. 4979–4984 (cit. on p. 255).

[227] Paul Selzer et al. “Rapid Access to Infrared Reference Spectra of Arbitrary Organic Compounds: Scope and Limitations of an Approach to the Simulation of Infrared Spectra by Neural Networks”. In: *Chemistry – A European Journal* 6.5 (2000), pp. 920–927 (cit. on p. 255).

[228] Thomas Kostka, Paul Selzer, and Johann Gasteiger. “A Combined Application of Reaction Prediction and Infrared Spectra Simulation for the Identification of Degradation Products of S-Triazine Herbicides”. In: *Chemistry – A European Journal* 7.10 (2001), pp. 2254–2260 (cit. on p. 255).

[229] Nihat Yildiz, Mehmet Karabacak, and Mustafa Kurt. “Neural Network Consistent Empirical Physical Formula Construction for DFT Based Nonlinear Vibrational Spectra Intensities of N-(2-Methylphenyl) and N-(3-Methylphenyl) Methanesulfonamides”. In: *Journal of Molecular Structure*. STRUCTURAL APPLICATIONS OF TERAHERTZ SPECTROSCOPY 1006.1 (Dec. 2011), pp. 642–649 (cit. on p. 255).

[230] Nihat Yildiz et al. “Neural Network Consistent Empirical Physical Formula Construction for Density Functional Theory Based Nonlinear Vibrational Absorbance and Intensity of 6-Choloronicotinic Acid Molecule”. In: *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy* 90 (May 2012), pp. 55–62 (cit. on p. 255).

[231] Michael Gastegger and Philipp Marquetand. “High-Dimensional Neural Network Potentials for Organic Reactions and an Improved Training Algorithm”. In: *Journal of Chemical Theory and Computation* 11.5 (May 2015), pp. 2187–2198 (cit. on p. 255).

[232] Andrew E. Sifain et al. “Discovering a Transferable Charge Assignment Model Using Machine Learning”. In: *The Journal of Physical Chemistry Letters* 9.16 (Aug. 2018), pp. 4495–4501 (cit. on p. 255).

[233] T. Visser, H. J. Luinge, and J. H. van der Maas. “Recognition of Visual Characteristics of Infrared Spectra by Artificial Neural Networks and Partial Least Squares Regression”. In: *Analytica Chimica Acta* 296.2 (Oct. 1994), pp. 141–154 (cit. on p. 256).

[234] H. J. Luinge, J. H. van der Maas, and T. Visser. “Partial Least Squares Regression as a Multivariate Tool for the Interpretation of Infrared Spectra”. In: *Chemometrics and Intelligent Laboratory Systems* 28.1 (Apr. 1995), pp. 129–138 (cit. on p. 256).

[235] Arthur H. Carrieri and Pascal I. Lim. “Neural Network Pattern Recognition of Thermal-Signature Spectra for Chemical Defense”. In: *Applied Optics* 34.15 (May 1995), pp. 2623–2635 (cit. on p. 256).

[236] Weiqiang Fu and W. Scott Hopkins. “Applying Machine Learning to Vibrational Spectroscopy”. In: *The Journal of Physical Chemistry A* 122.1 (Jan. 2018), pp. 167–171 (cit. on p. 256).

[237] David J. Wales and Jonathan P. K. Doye. “Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms”. In: *The Journal of Physical Chemistry A* 101.28 (July 1997), pp. 5111–5116 (cit. on p. 256).

[238] Jonathan A. Fine et al. “Spectral Deep Learning for Prediction and Prospective Validation of Functional Groups”. In: *Chemical Science* 11.18 (May 2020), pp. 4618–4630 (cit. on p. 256).

[239] Hao Ren et al. “A Machine Learning Vibrational Spectroscopy Protocol for Spectrum Prediction and Spectrum-Based Structure Recognition”. In: *Fundamental Research* 1.4 (July 2021), pp. 488–494 (cit. on p. 256).

[240] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on p. 256).

[241] Joseph R. Cendagorta et al. “Comparison of the Performance of Machine Learning Models in Representing High-Dimensional Free Energy Surfaces and Generating Observables”. In: *The Journal of Physical Chemistry B* 124.18 (May 2020), pp. 3647–3660 (cit. on p. 257).

[242] Christoph Scherer et al. “Kernel-Based Machine Learning for Efficient Simulations of Molecular Liquids”. In: *Journal of Chemical Theory and Computation* 16.5 (May 2020), pp. 3194–3204 (cit. on p. 257).

[243] Oliver T. Unke et al. “Machine Learning Force Fields”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10142–10186 (cit. on pp. 257–259).

[244] Maksim Kulichenko et al. “The Rise of Neural Networks for Materials and Chemical Dynamics”. In: *The Journal of Physical Chemistry Letters* 12.26 (July 2021), pp. 6227–6243 (cit. on p. 257).

[245] Jörg Behler. “First Principles Neural Network Potentials for Reactive Simulations of Large Molecular and Condensed Systems”. In: *Angewandte Chemie International Edition* 56.42 (2017), pp. 12828–12840 (cit. on p. 258).

[246] Bryan R. Goldsmith et al. “Machine Learning for Heterogeneous Catalyst Design and Discovery”. In: *AIChE Journal* 64.7 (2018), pp. 2311–2323 (cit. on p. 258).

[247] Giuseppe Carleo et al. “Machine Learning and the Physical Sciences”. In: *Reviews of Modern Physics* 91.4 (Dec. 2019), p. 045002 (cit. on p. 258).

[248] Xin Yang et al. “Concepts of Artificial Intelligence for Computer-Assisted Drug Discovery”. In: *Chemical Reviews* 119.18 (Sept. 2019), pp. 10520–10594 (cit. on p. 258).

[249] Daniel C. Elton et al. “Deep Learning for Molecular Design—a Review of the State of the Art”. In: *Molecular Systems Design & Engineering* 4.4 (Aug. 2019), pp. 828–849 (cit. on p. 258).

[250] Gabriel R. Schleder et al. “From DFT to Machine Learning: Recent Approaches to Materials Science–a Review”. In: *Journal of Physics: Materials* 2.3 (May 2019), p. 032001 (cit. on p. 258).

[251] Michele Ceriotti. “Unsupervised Machine Learning in Atomistic Simulations, between Predictions and Understanding”. In: *The Journal of Chemical Physics* 150.15 (Apr. 2019), p. 150901 (cit. on p. 258).

[252] Pavlo O. Dral. “Quantum Chemistry in the Age of Machine Learning”. In: *The Journal of Physical Chemistry Letters* 11.6 (Mar. 2020), pp. 2336–2347 (cit. on p. 258).

[253] O. Anatole von Lilienfeld, Klaus-Robert Müller, and Alexandre Tkatchenko. “Exploring Chemical Compound Space with Quantum-Based Machine Learning”. In: *Nature Reviews Chemistry* 4.7 (July 2020), pp. 347–358 (cit. on p. 258).

[254] Tim Mueller, Alberto Hernandez, and Chuhong Wang. “Machine Learning for Interatomic Potential Models”. In: *The Journal of Chemical Physics* 152.5 (Feb. 2020), p. 050902 (cit. on p. 258).

[255] Sergei Manzhos and Tucker Jr. Carrington. “Neural Network Potential Energy Surfaces for Small Molecules and Reactions”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10187–10217 (cit. on pp. 258, 259).

[256] Paraskevi Gkeka et al. “Machine Learning Force Fields and Coarse-Grained Variables in Molecular Dynamics: Application to Materials and Biological Systems”. In: *Journal of Chemical Theory and Computation* 16.8 (Aug. 2020), pp. 4757–4775 (cit. on p. 258).

[257] Takashi Toyao et al. “Machine Learning for Catalysis Informatics: Recent Applications and Prospects”. In: *ACS Catalysis* 10.3 (Feb. 2020), pp. 2260–2297 (cit. on p. 258).

[258] Sergei Manzhos. “Machine Learning for the Solution of the Schrödinger Equation”. In: *Machine Learning: Science and Technology* 1.1 (Apr. 2020), p. 013002 (cit. on p. 258).

[259] Julia Westermayr and Philipp Marquetand. “Machine Learning for Electronically Excited States of Molecules”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 9873–9926 (cit. on pp. 258, 259).

[260] Jörg Behler. “Four Generations of High-Dimensional Neural Network Potentials”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10037–10072 (cit. on pp. 258, 259).

[261] Peikun Zheng et al. “Artificial Intelligence-Enhanced Quantum Chemical Method with Broad Applicability”. In: *Nature Communications* 12.1 (Dec. 2021), p. 7022 (cit. on p. 258).

[262] Manas Sajjan et al. “Quantum Machine Learning for Chemistry and Physics”. In: *Chemical Society Reviews* 51.15 (Aug. 2022), pp. 6475–6573 (cit. on p. 258).

[263] Megan A. Lim et al. “Exploring Deep Learning of Quantum Chemical Properties for Absorption, Distribution, Metabolism, and Excretion Predictions”. In: *Journal of Chemical Information and Modeling* (June 2022) (cit. on p. 258).

[264] Shampa Raghunathan and U. Deva Priyakumar. “Molecular Representations for Machine Learning Applications in Chemistry”. In: *International Journal of Quantum Chemistry* 122.7 (2022), e26870 (cit. on p. 258).

[265] Thijs Stuyver and Connor W. Coley. “Quantum Chemistry-Augmented Neural Networks for Reactivity Prediction: Performance, Generalizability, and Explainability”. In: *The Journal of Chemical Physics* 156.8 (Feb. 2022), p. 084104 (cit. on p. 258).

[266] Mohammadamin Tavakoli et al. “Quantum Mechanics and Machine Learning Synergies: Graph Attention Neural Networks to Predict Chemical Reactivity”. In: *Journal of Chemical Information and Modeling* 62.9 (May 2022), pp. 2121–2132 (cit. on p. 258).

[267] Ruocheng Han, Rangsiman Ketkaew, and Sandra Luber. “A Concise Review on Recent Developments of Machine Learning for the Prediction of Vibrational Spectra”. In: *The Journal of Physical Chemistry A* 126.6 (Feb. 2022), pp. 801–812 (cit. on p. 258).

[268] Puck van Gerwen et al. “Physics-Based Representations for Machine Learning Properties of Chemical Reactions”. In: *Machine Learning: Science and Technology* 3.4 (Oct. 2022), p. 045005 (cit. on p. 258).

[269] Zhuoran Qiao et al. “Informing Geometric Deep Learning with Electronic Interactions to Accelerate Quantum Chemistry”. In: *Proceedings of the National Academy of Sciences* 119.31 (Aug. 2022), e2205221119 (cit. on p. 258).

[270] H. J. Kulik et al. “Roadmap on Machine Learning in Electronic Structure”. In: *Electronic Structure* 4.2 (June 2022), p. 023004 (cit. on p. 258).

[271] Bing Huang and O. Anatole von Lilienfeld. “Ab Initio Machine Learning in Chemical Compound Space”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10001–10036 (cit. on p. 259).

[272] Volker L. Deringer et al. “Gaussian Process Regression for Materials and Molecules”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10073–10141 (cit. on p. 259).

[273] Markus Meuwly. “Machine Learning for Chemical Reactions”. In: *Chemical Reviews* 121.16 (Aug. 2021), pp. 10218–10239 (cit. on p. 259).

[274] Justin S. Smith et al. “The ANI-1ccx and ANI-1x Data Sets, Coupled-Cluster and Density Functional Theory Properties for Molecules”. In: *Scientific Data* 7.1 (May 2020), p. 134 (cit. on p. 260).

[275] Julia Westermayr, Michael Gastegger, and Philipp Marquetand. “Combining SchNet and SHARC: The SchNarc Machine Learning Approach for Excited-State Dynamics”. In: *The Journal of Physical Chemistry Letters* 11.10 (May 2020), pp. 3828–3834 (cit. on p. 265).

[276] Martin Richter et al. “SHARC: Ab Initio Molecular Dynamics with Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings”. In: *Journal of Chemical Theory and Computation* 7.5 (May 2011), pp. 1253–1258 (cit. on p. 265).

[277] Sebastian Mai, Philipp Marquetand, and Leticia González. “Nonadiabatic Dynamics: The SHARC Approach”. In: *WIREs Computational Molecular Science* 8.6 (2018), e1370 (cit. on p. 265).

[278] Stefan Chmiela et al. “Machine Learning of Accurate Energy-Conserving Molecular Force Fields”. In: *Science Advances* 3.5 (May 2017), e1603015 (cit. on p. 265).

[279] Stefan Chmiela et al. “Towards Exact Molecular Dynamics Simulations with Machine-Learned Force Fields”. In: *Nature Communications* 9.1 (Sept. 2018), p. 3887 (cit. on p. 265).

[280] Huziel E. Sauceda et al. “Molecular Force Fields with Gradient-Domain Machine Learning (GDML): Comparison and Synergies with Classical Force Fields”. In: *The Journal of Chemical Physics* 153.12 (Sept. 2020), p. 124109 (cit. on p. 265).

[281] Stefan Chmiela et al. “Accurate Molecular Dynamics Enabled by Efficient Physically Constrained Machine Learning Approaches”. In: *Machine Learning Meets Quantum Physics*. Ed. by Kristof T. Schütt et al. Lecture Notes in Physics. Cham: Springer International Publishing, 2020, pp. 129–154 (cit. on p. 265).

[282] Stefan Chmiela et al. *Accurate Global Machine Learning Force Fields for Molecules with Hundreds of Atoms*. Oct. 2022. arXiv: 2209.14865 [physics] (cit. on p. 265).

[283] LiHong Hu et al. “Combined First-Principles Calculation and Neural-Network Correction Approach for Heat of Formation”. In: *The Journal of Chemical Physics* 119.22 (Dec. 2003), pp. 11501–11507 (cit. on p. 266).

[284] Jianming Wu and Xin Xu. “The X1 Method for Accurate and Efficient Prediction of Heats of Formation”. In: *The Journal of Chemical Physics* 127.21 (Dec. 2007), p. 214105 (cit. on p. 266).

[285] Roman M. Balabin and Ekaterina I. Lomakina. “Neural Network Approach to Quantum-Chemistry Data: Accurate Prediction of Density Functional Theory Energies”. In: *The Journal of Chemical Physics* 131.7 (Aug. 2009), p. 074104 (cit. on p. 266).

[286] Raghunathan Ramakrishnan et al. “Big Data Meets Quantum Chemistry Approximations: The Δ-Machine Learning Approach”. In: *Journal of Chemical Theory and Computation* 11.5 (May 2015), pp. 2087–2096 (cit. on p. 266).

[287] Marcel Ruth, Dennis Gerbig, and Peter R. Schreiner. “Machine Learning of Coupled Cluster (T)-Energy Corrections via Delta (Δ)-Learning”. In: *Journal of Chemical Theory and Computation* 18.8 (Aug. 2022), pp. 4846–4855 (cit. on p. 267).

[288] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80 (cit. on p. 267).

[289] Jie Zhou et al. “Graph Neural Networks: A Review of Methods and Applications”. In: *AI Open* 1 (Jan. 2020), pp. 57–81 (cit. on p. 267).

[290] Chi Chen et al. “Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals”. In: *Chemistry of Materials* 31.9 (May 2019), pp. 3564–3572 (cit. on p. 268).

[291] Kristof T. Schütt, Oliver T. Unke, and Michael Gastegger. *Equivariant Message Passing for the Prediction of Tensorial Properties and Molecular Spectra*. June 2021. arXiv: 2102.03150 `[physics]` (cit. on pp. 272, 283).

[292] Matteo Aldeghi and Connor W. Coley. “A Graph Representation of Molecular Ensembles for Polymer Property Prediction”. In: *Chemical Science* 13.35 (2022), pp. 10486–10498 (cit. on p. 272).

[293] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014 (cit. on p. 273).

[294] Oleksii Prykhodko et al. “A de Novo Molecular Generation Method Using Latent Vector Based Generative Adversarial Network”. In: *Journal of Cheminformatics* 11.1 (Dec. 2019), p. 74 (cit. on p. 274).

[295] Young Jae Lee, Hyungu Kahng, and Seoung Bum Kim. “Generative Adversarial Networks for De Novo Molecular Design”. In: *Molecular Informatics* 40.10 (2021), p. 2100045 (cit. on p. 274).

[296] Andrew E. Blanchard, Christopher Stanley, and Debsindhu Bhowmik. “Using GANs with Adaptive Training Data to Search for New Molecules”. In: *Journal of Cheminformatics* 13.1 (Feb. 2021), p. 14 (cit. on p. 274).

[297] Sungwon Kim et al. “Generative Adversarial Networks for Crystal Structure Prediction”. In: *ACS Central Science* 6.8 (Aug. 2020), pp. 1412–1420 (cit. on p. 274).

[298] Yabo Dan et al. “Generative Adversarial Networks (GAN) Based Efficient Sampling of Chemical Composition Space for Inverse Design of Inorganic Materials”. In: *npj Computational Materials* 6.1 (June 2020), pp. 1–7 (cit. on p. 274).

[299] Ziareena A. Al-Mualem and Carlos R. Baiz. “Generative Adversarial Neural Networks for Denoising Coherent Multidimensional Spectra”. In: *The Journal of Physical Chemistry A* 126.23 (June 2022), pp. 3816–3825 (cit. on p. 274).

[300] Małgorzata Z. Makoś et al. “Generative Adversarial Networks for Transition State Geometry Prediction”. In: *The Journal of Chemical Physics* 155.2 (July 2021), p. 024116 (cit. on p. 274).

[301] Ashish Vaswani et al. *Attention Is All You Need*. Dec. 2017. arXiv: 1706.03762 [cs] (cit. on p. 274).

[302] Łukasz Maziarka et al. *Molecule Attention Transformer*. Feb. 2020. arXiv: 2002.08264 [physics, stat] (cit. on p. 277).

[303] Han Wang et al. "DeePMD-kit: A Deep Learning Package for Many-Body Potential Energy Representation and Molecular Dynamics". In: *Computer Physics Communications* 228 (July 2018), pp. 178–184 (cit. on pp. 277, 282).

[304] Brandon Anderson, Truong-Son Hy, and Risi Kondor. *Cormorant: Covariant Molecular Neural Networks*. Nov. 2019. arXiv: 1906.04015 [physics, stat] (cit. on p. 277).

[305] V. Zaverkin and J. Kästner. "Gaussian Moments as Physically Inspired Molecular Descriptors for Accurate and Scalable Machine Learning Potentials". In: *Journal of Chemical Theory and Computation* 16.8 (Aug. 2020), pp. 5410–5421 (cit. on p. 277).

[306] Michael Gastegger, Kristof T. Schütt, and Klaus-Robert Müller. "Machine Learning of Solvent Effects on Molecular Spectra and Reactions". In: *Chemical Science* 12.34 (Sept. 2021), pp. 11473–11483 (cit. on p. 277).

[307] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. *Directional Message Passing for Molecular Graphs*. Apr. 2022. arXiv: 2003.03123 [physics, stat] (cit. on p. 277).

[308] *RDKit*. https://www.rdkit.org/ (cit. on p. 279).

[309] Stefano Curtarolo et al. "AFLOW: An Automatic Framework for High-Throughput Materials Discovery". In: *Computational Materials Science* 58 (June 2012), pp. 218–226 (cit. on p. 280).

[310] Anubhav Jain et al. "Commentary: The Materials Project: A Materials Genome Approach to Accelerating Materials Innovation". In: *APL Materials* 1.1 (July 2013), p. 011002 (cit. on p. 280).

[311] Scott Kirklin et al. "The Open Quantum Materials Database (OQMD): Assessing the Accuracy of DFT Formation Energies". In: *npj Computational Materials* 1.1 (Dec. 2015), pp. 1–15 (cit. on p. 280).

[312] Albert P. Bartók et al. "Machine Learning a General-Purpose Interatomic Potential for Silicon". In: *Physical Review X* 8.4 (Dec. 2018), p. 041048 (cit. on p. 280).

[313] Logan Ward et al. "Matminer: An Open Source Toolkit for Materials Data Mining". In: *Computational Materials Science* 152 (Sept. 2018), pp. 60–69 (cit. on p. 280).

[314] Kamal Choudhary, Brian DeCost, and Francesca Tavazza. "Machine Learning with Force-Field-Inspired Descriptors for Materials: Fast Screening and Mapping Energy Landscape". In: *Physical Review Materials* 2.8 (Aug. 2018), p. 083801 (cit. on p. 280).

[315] Lauri Himanen et al. "DScribe: Library of Descriptors for Machine Learning in Materials Science". In: *Computer Physics Communications* 247 (Feb. 2020), p. 106949 (cit. on p. 280).

[316] Claudia Draxl and Matthias Scheffler. "The NOMAD Laboratory: From Data Sharing to Artificial Intelligence". In: *Journal of Physics: Materials* 2.3 (May 2019), p. 036001 (cit. on p. 280).

[317] Bart Olsthoorn et al. "Band Gap Prediction for Large Organic Crystal Structures with Machine Learning". In: *Advanced Quantum Technologies* 2.7-8 (2019), p. 1900023 (cit. on p. 280).

[318] J. Chapman, R. Batra, and R. Ramprasad. "Machine Learning Models for the Prediction of Energy, Forces, and Stresses for Platinum". In: *Computational Materials Science* 174 (Mar. 2020), p. 109483 (cit. on p. 280).

[319] Prakriti Kayastha and Raghunathan Ramakrishnan. “High-Throughput Design of Peierls and Charge Density Wave Phases in Q1D Organometallic Materials”. In: *The Journal of Chemical Physics* 154.6 (Feb. 2021), p. 061102 (cit. on p. 280).

[320] Alireza Khorshidi and Andrew A. Peterson. “Amp: A Modular Approach to Machine Learning in Atomistic Simulations”. In: *Computer Physics Communications* 207 (Oct. 2016), pp. 310–324 (cit. on p. 282).

[321] Nongnuch Artrith and Alexander Urban. “An Implementation of Artificial Neural-Network Potentials for Atomistic Materials Simulations: Performance for TiO2”. In: *Computational Materials Science* 114 (Mar. 2016), pp. 135–150 (cit. on p. 282).

[322] Tran Doan Huan et al. “A Universal Strategy for the Creation of Machine Learning-Based Atomistic Force Fields”. In: *npj Computational Materials* 3.1 (Sept. 2017), pp. 1–8 (cit. on p. 282).

[323] Brian Kolb, Levi C. Lentz, and Alexie M. Kolpak. “Discovering Charge Density Functionals and Structure-Property Relationships with PROPhet: A General Framework for Coupling Machine Learning and First-Principles Methods”. In: *Scientific Reports* 7.1 (Apr. 2017), p. 1192 (cit. on p. 282).

[324] Andreas Mardt et al. “VAMPnets for Deep Learning of Molecular Kinetics”. In: *Nature Communications* 9.1 (Jan. 2018), p. 5 (cit. on p. 282).

[325] Tian Xie and Jeffrey C. Grossman. “Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties”. In: *Physical Review Letters* 120.14 (Apr. 2018), p. 145301 (cit. on p. 282).

[326] Dipendra Jha et al. “ElemNet: Deep Learning the Chemistry of Materials From Only Elemental Composition”. In: *Scientific Reports* 8.1 (Dec. 2018), p. 17593 (cit. on p. 282).

[327] Dipendra Jha et al. “Enhancing Materials Property Prediction by Leveraging Computational and Experimental Data Using Deep Transfer Learning”. In: *Nature Communications* 10.1 (Nov. 2019), p. 5316 (cit. on p. 282).

[328] K. T. Schütt et al. “SchNetPack: A Deep Learning Toolbox For Atomistic Systems”. In: *Journal of Chemical Theory and Computation* 15.1 (Jan. 2019), pp. 448–455 (cit. on p. 283).

[329] Stefan Chmiela et al. “sGDML: Constructing Accurate and Data Efficient Molecular Force Fields Using Machine Learning”. In: *Computer Physics Communications* 240 (July 2019), pp. 38–45 (cit. on p. 283).

[330] Yunqi Shao et al. “PiNN: A Python Library for Building Atomic Neural Networks of Molecules and Materials”. In: *Journal of Chemical Information and Modeling* 60.3 (Mar. 2020), pp. 1184–1193 (cit. on p. 285).

[331] Xiang Gao et al. “TorchANI: A Free and Open Source PyTorch-Based Deep Learning Implementation of the ANI Neural Network Potentials”. In: *Journal of Chemical Information and Modeling* 60.7 (July 2020), pp. 3408–3415 (cit. on p. 286).

[332] M. J. Frisch et al. *Gaussian□16 Revision C.01*. 2016 (cit. on p. 302).

[333] Frank Neese. “The ORCA Program System”. In: *WIREs Computational Molecular Science* 2.1 (2012), pp. 73–78 (cit. on p. 303).

[334] Frank Neese. “Software Update: The ORCA Program System, Version 4.0”. In: *WIREs Computational Molecular Science* 8.1 (2018), e1327 (cit. on p. 303).

[335] E. Aprà et al. “NWChem: Past, Present, and Future”. In: *The Journal of Chemical Physics* 152.18 (May 2020), p. 184102 (cit. on p. 304).

[336] Qiming Sun et al. “PySCF: The Python-based Simulations of Chemistry Framework”. In: *WIREs Computational Molecular Science* 8.1 (2018), e1340 (cit. on p. 306).

ดรรชนีภาษาไทย

ดรรชนีภาษาอังกฤษ

ประวัติผู้เขียน

รังสิมันต์ เกษแก้ว สำเร็จการศึกษาปริญญาตรี (พ.ศ. 2559) และปริญญาโท (พ.ศ. 2562) สาขาเคมี จากภาควิชาเคมี คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ ปัจจุบันกำลังศึกษาปริญญาเอกสาขาเคมีทฤษฎีที่ภาควิชาเคมี มหาวิทยาลัยแห่งซูริค ประเทศสวิตเซอร์แลนด์ หัวข้องานวิจัยที่สนใจ ได้แก่ เคมีควอนตัม เมตาไดนามิกส์ การถ่ายโอนอิเล็กตรอน ปัญญาประดิษฐ์ และการพัฒนาซอฟต์แวร์เคมีเชิงคำนวณ

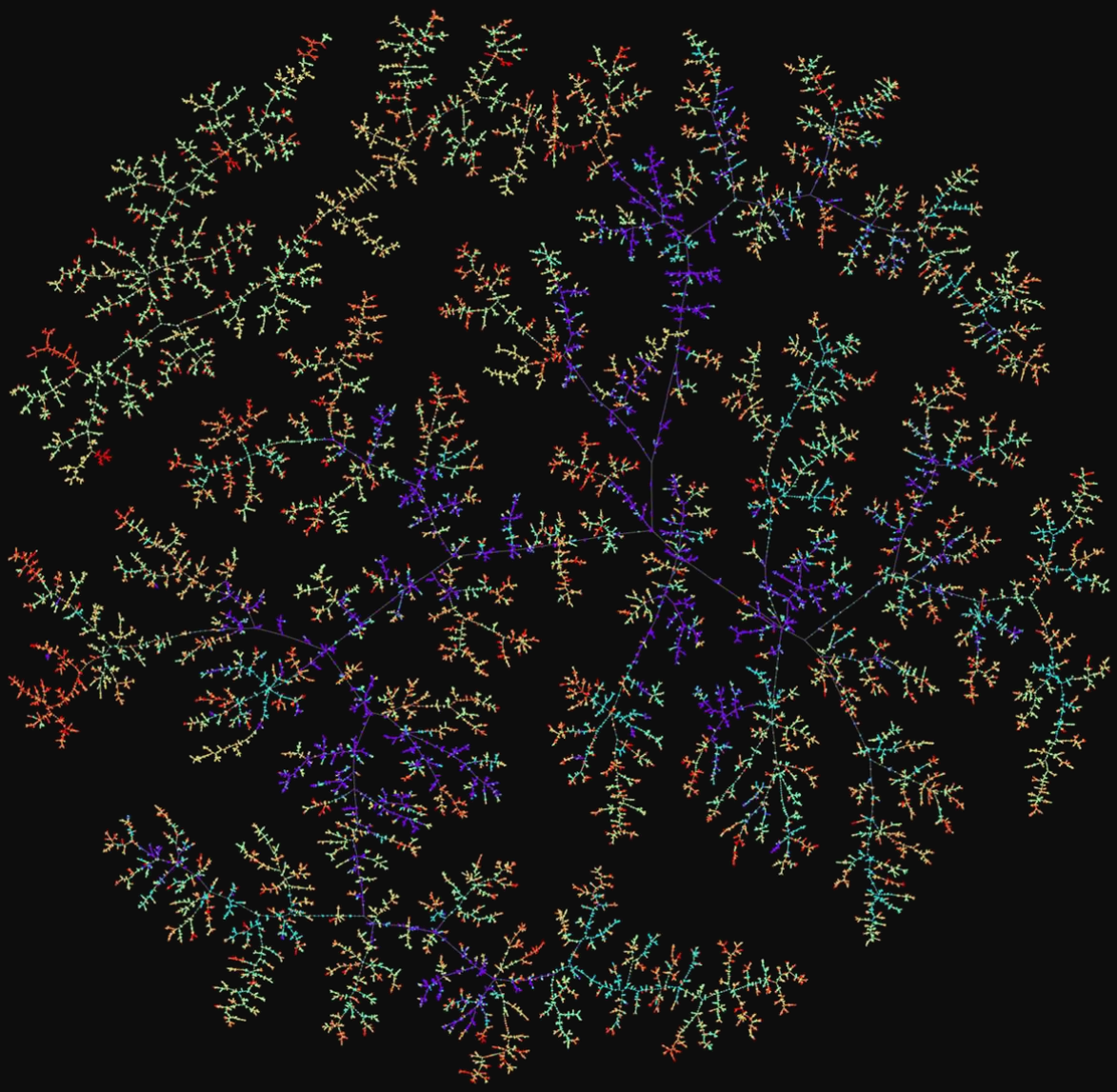
ประสบการณ์การทำงาน

- พ.ศ. 2563 ที่ปรึกษาบริษัท New Equilibrium Biosciences, Boston, MA
- พ.ศ. 2564 ที่ปรึกษาบริษัท ติ๊งกิ้ง แมชชีนส์ จำกัด (Thinking Machines)
- พ.ศ. 2564 คณะกรรมการจัดการแข่งขันปัญญาประดิษฐ์สำหรับเคมีแห่งประเทศไทย (TMLCC)
- พ.ศ. 2564 คณะกรรมการจัดงาน PyCon Thailand และ PyCon APAC 2021
- พ.ศ. 2565 นักเขียนบทความบริษัท คลาวด์ เอชเอ็ม จำกัด (Cloud HM)

ผู้ก่อตั้งเพจและกลุ่ม Facebook

- วิทย์ตามิน
- Computational Chemistry and Machine Learning Thailand
- Thai Computational Science Students

ดูบทความและผลงานเพิ่มเติมของผู้เขียนได้ที่ https://rangsimanketkaew.github.io



การเรียนรู้ของเครื่องสำหรับเคมีควอนตัม

Machine Learning for Quantum Chemistry