ระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น

Intelligence System for Stock Prediction and Stock Investments

ก้องภูมิ อัฐมโนลาภ

KONGPHUM ARTHAMANOLAP

พีรไชย อรุณสุริยศักดิ์

PEERACHAI ARUNSURIYASAK

ภัทราพร บุญมี

PHATTRAPORN BOONME

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงงาน 2

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2563

ระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น

Intelligence System for Stock Prediction and Stock Investments

โดย

ก้องภูมิ อัฐมโนลาภ

พีรไชย อรุณสุริยศักดิ์

ภัทราพร บุญมี

อาจารย์ที่ปรึกษา

รศ.ดร.ภัทรพงษ์ ผาสุกกิจ

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงงาน 2

ภาควิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2563

ใบรับรองวิชาโครงงาน 2

รายงานวิชาโครงงาน 2 ปีการศึกษา 2563

ภาควิชา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น

Intelligence System for Stock Prediction and Stock Investments

ผู้จัดทำ นายก้องภูมิ อัฐมโนลาภ รหัสประจำตัว 60010044

นายพีรไชย อรุณสุริยศักดิ์ รหัสประจำตัว 60010729

นางสาวภัทราพร บุญมี รหัสประจำตัว 60010773

รายงานนี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

รศ.ดร.ภัทรพงษ์ ผาสุกกิจ

อาจารย์ที่ปรึกษา

หัวข้อโครงงาน ระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น
นักศึกษา นายก้องภูมิ อัฐมโนลาภ รหัสประจำตัว 60010044
นายพีรไชย อรุณสุริยศักดิ์ รหัสประจำตัว 60010729
นางสาวภัทราพร บุญมี รหัสประจำตัว 60010773
ปริญญา วิศวกรรมศาสตรบัณฑิต
ภาควิชา วิศวกรรมอิเล็กทรอนิกส์
ปีการศึกษา 2563
อาจารย์ที่ปรึกษาโครงการ รศ.ดร.ภัทรพงษ์ ผาสุขกิจ

บทคัดย่อ

หุ้นเป็นสินทรัพย์ที่ให้ผลตอบแทนที่มีมูลค่าที่สูงเมื่อเทียบกับเงินที่ลงทุนไป แต่อย่างไรก็ตามมูลค่าของหุ้นนั้นมีการผันผวนที่สูงซึ่งเสี่ยงต่อการลงทุน เนื่องจากหุ้นนั้นมีตัวแปรที่ควบคุมมูลค่าที่หลากหลายและมีจำนวนมากจึงต้องใช้ประสบการณ์ของผู้ลงทุนในการลงทุนอย่างมาก ด้วยสาเหตุนี้จึงเป็นที่มาของโครงงานนี้ที่จะสร้างระบบการลงทุนในหุ้นแบบอัตโนมัติ ซึ่งใช้หลักการของโมเดลสำหรับการทำนายผลราคาหุ้นในอนาคตโดยใช้โมเดลความจำระยะสั้นระยะยาวเป็นโมเดลพื้นฐานและหลักการของโมเดลการเรียนรู้แบบเสริมกำลังสำหรับการทำนายพฤติกรรมการลงทุนในหุ้น โดยจะนำระบบที่ได้ไปทำการลงทุนในหุ้นแบบจำลองผ่านโปรแกรม Alpaca เพื่อลงทุนในหุ้นของบริษัท Apple เป็นสกุลเงินดอลลาร์สหรัฐ

Project Title	Intelligence System for Stock Prediction and Stock Investments
Student	Mr.Kongpoom Arthamanolap Student ID 60010044
	Mr.Peerachai Arunsuriyasak Student ID 60010729
	Ms.Phattraporn Boonme Student ID 60010773
Degree	Bachelor of Engineering
Program	Electronics Engineering
Year	2020
Project Advisor	Assoc. Prof. Dr. Pattarapong Phasukkit

ABSTRACT

Stocks are assets that provide a high return on investment. However, stocks are highly fluctuating, which is risky for investment. Since stocks have many variables that control a variety of values and therefore require a lot of investor experience.

For this reason, it is the origin of this project to create an automated stock investment system based on the stock prediction model, using the Long-Short Term Memory Neural Network, and the principles of a reinforcement learning model for the investment action prediction. This system will be using the Alpaca API to doing paper trade for AAPL Stock in USD.

กิตติกรรมประกาศ

โครงงานระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้นสำเร็จได้ในครั้งนี้ต้อขอขอบคุณ อาจารย์ที่ปรึกษา และพี่ปริญญาโท ภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ ที่ช่วยให้คำแนะนำการทำโครงงาน การใช้โปรแกรม และแก้ไขข้อผิดพลาดต่าง ๆ เกี่ยวกับโครงงาน และขอขอบคุณห้องปฏิบัติการ BurnLab ที่เอื้อเฟื้อสถานที่ในการทำโครงงาน ทำให้โครงงานนี้ออกมาสำเร็จตามต้องการ

สุดท้ายนี้ขอขอบคุณเพื่อนๆในภาควิชาอิเล็กทรอนิกส์ และห้องปฏิบัติการ BurnLab ที่ให้ความช่วยเหลือทำให้โครงงานสำเร็จและหวังว่าโครงงานเล่มนี้จะเป็นประโยชน์ต่อผู้อ่านไม่มากก็น้อย หาผิดพลาดประการใดต้องขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

สารบัญ

หน้า

สารบัญรูป

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1

บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

เนื่องด้วยภาควิชาวิศวกรรมอิเล็กทรอนิกส์ได้มีนโยบายให้นักศึกษาจัดทำโครงงานสิ่งประดิษฐ์ในวิชา Project 2 ซึ่งคณะผู้จัดทำได้คิดค้นทำระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น โดยนำเอาความรู้จากภาคการเรียนมาประยุกต์ใช้ในทางปฏิบัติ เพื่อศึกษาการสร้างระบบการทำนายหุ้นและการตัดสินใจลงทุนในหุ้นด้วยปัญญาประดิษฐ์

1.2 วัตถุประสงค์ของโครงงาน

- ได้ออกแบบประยุกต์ใช้การสร้างระบบการลงทุนในหุ้นแบบอัตโนมัติด้วยปัญญาประดิษฐ์
- นำความรู้จากการสอนภาคทฤษฎีมาใช้ในการทำทางปฏิบัติในการทำโครงงาน
- เป็นการฝึกการทำงานภาคปฏิบัติในการทำโครงงานสิ่งประดิษฐ์เพื่อการทำงานในอนาคต

1.3 สมมุติฐานการศึกษา

สามารถสร้างโมเดลปัญญาประดิษฐ์ในการทำนายราคาปิดของหุ้นในอนาคตจากค่าหุ้นในอดีต และทำนายพฤติกรรมการลงทุนในหุ้นให้กับระบบการลงทุนในหุ้นแบบอัตโนมัติได้อย่างมีประสิทธิภาพและสามารถทำการลงทุนในหุ้นจากระบบจำลอง (Paper Trade) ได้

1.4 ขอบเขตในการจัดทำโครงงาน

- ระบบสามารถทำการทำนายราคาปิดหุ้นในอนาคตได้
- ระบบสามารถทำนายพฤติกรรมการลงทุนในหุ้นได้
- ระบบสามารถทำการลงทุนในหุ้นแบบจำลองบนโปรแกรม Alpaca ได้

1.5 ประโยชน์ที่คาดว่าจะได้รับ

- ได้ความรู้ด้านการทำโมเดลปัญญาประดิษฐ์สำหรับการทำนาย และโมเดลสำหรับการเรียนรู้แบบเสริมกำลัง
- ระบบนี้จะช่วยให้ผู้ลงทุนในหุ้นได้มีการวางแผนการลงหุ้นได้ดีขึ้น จากการทำนายราคาหุ้นในอนาคตก่อนที่จะลงทุน และ พฤติกรรมการลงทุนในหุ้นที่ทำนายจากโมเดลการเรียนรู้แบบเสริมกำลัง
- คณะผู้จัดทำจะสามารถนำความรู้ในการทำโครงงานประดิษฐ์ไปประยุกต์ใช้ได้ในอนาคตได้

1.6 แผนงานการทำงานของโครงงาน

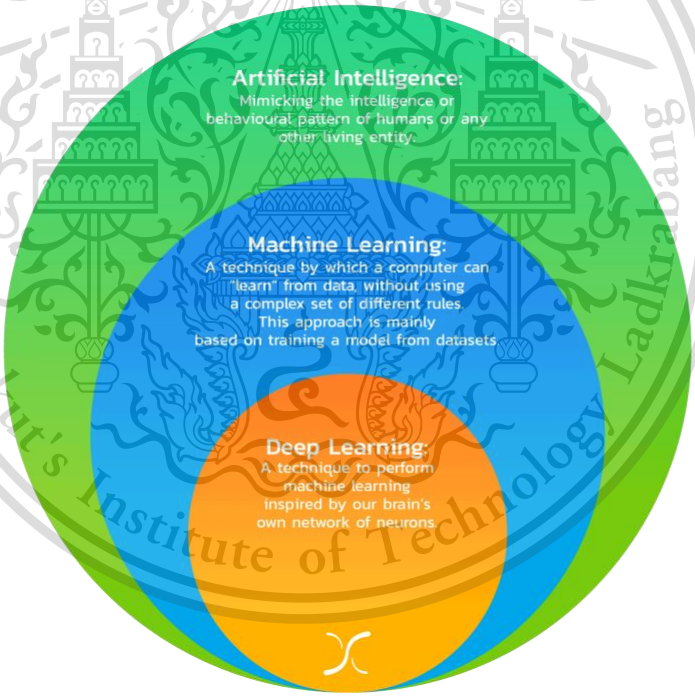
รายละเอียด	มกราคม		กุมภาพันธ์		มีนาคม		เมษายน	
	1-2	3-4	1-2	3-4	1-2	3-4	1-2	3-4
ศึกษาการทำโมเดล Reinforcement Learning								
ศึกษาการอัพเดตโมเดลสำหรับการเรียนรู้จากข้อมูลใหม่ๆแบบ Realtime								
พัฒนาโมเดลสำหรับการทำนายหุ้นให้มีผลที่แม่นยำมากขึ้น								
พัฒนาโมเดลการแบ่งแยกความรู้สึกทางภาษาให้มีผลแม่นยำมากขึ้น								
ออกแบบโมเดล Reinforcement learning สำหรับการลงทุนในหุ้น								
ทดสอบโมเดล Reinforcement Learning จากข้อมูลเก่า								
สร้างระบบโมเดล Reinforcement Learning เข้ากับการอัพเดตข้อมูลเพื่อพัฒนาโมเดลแบบ Realtime								
ปรับจูนโมเดลให้มีความแม่นยำและคงที่ให้มากขึ้น								

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 Deep Learning

Deep Learning คือวิธีการเรียนรู้แบบอัตโนมัติด้วยการ เลียนแบบการทำงานของโครงข่ายประสาทของมนุษย์ (Neurons) โดยนำระบบโครงข่ายประสาท (Neural Network) มาซ้อนกันหลายชั้น (Layer) และทำการเรียนรู้ข้อมูลตัวอย่าง ซึ่งข้อมูล ดังกล่าวจะถูกนำไปใช้ในการตรวจจับรูปแบบ (Pattern) หรือจัด หมวดหมู่ข้อมูล (Classify the Data) โดย Deep Learning เป็นซับเซ็ตของการเรียนรู้ของเครื่องจักร (Machine Learning) ซึ่งคือการรับข้อมูลจำนวนมหาศาล เพื่อจดจำความแตกต่างหรือลักษณะเด่นและทำการแบ่งข้อมูลออกเป็นกลุ่ม เช่น หากมีปีกก็แยกไปกลุ่มนก ไม่มีปีกแต่มีสี่ขาก็แยกไปกลุ่มสุนัข ตามจุดเด่นที่เห็นได้ชัด เป็นต้น ยิ่งเรียนรู้มากก็จะยิ่งแยกแยะจุดเด่นดังกล่าวได้ดีขึ้น



รูปที่ 2.1 ภาพรวมของ AI, ML, DL

2.1.1 ชนิดของ Deep Learning

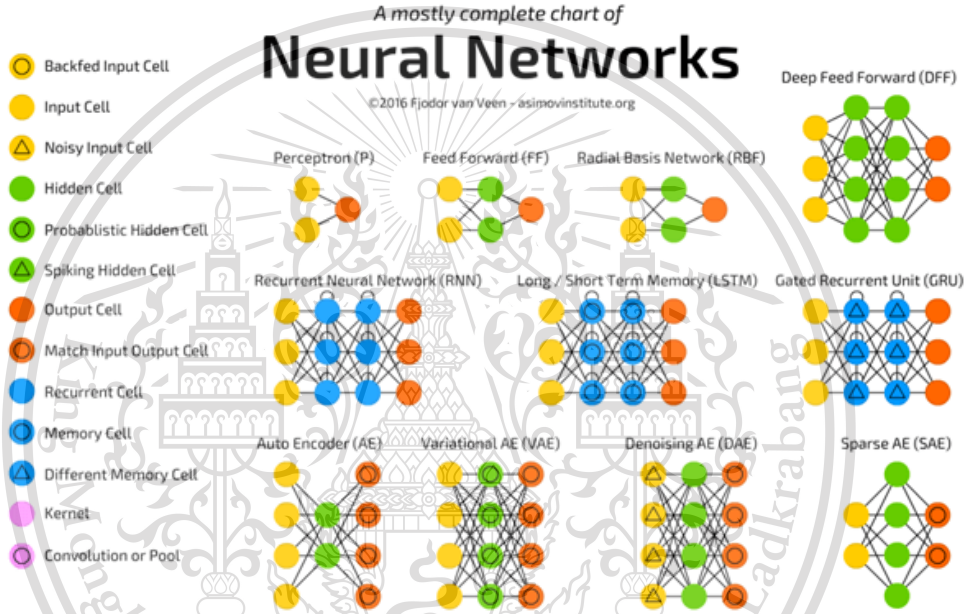
- โครงข่ายประสาทแบบป้อนไปข้างหน้า (Feed-forward neural networks)

Feed-forward neural networks ถือเป็นโมเดลที่มีโครงสร้างที่เรียบง่ายที่สุด เพราะว่าการดำเนินการของข้อมูลจะเป็นไปในทิศทางเดียว ก็คือ รับข้อมูลจาก input layer แล้วส่งไปต่อไปยัง

hidden layer เลื่อยๆ จนกระทั้งถึง output layer ก็จะหยุด (สังเกตุได้ว่าจะไม่มีวงวน(loop) เกิดขึ้นเลย)

- โครงข่ายแบบวนซ้ำ (Recurrent neural networks : RNN)

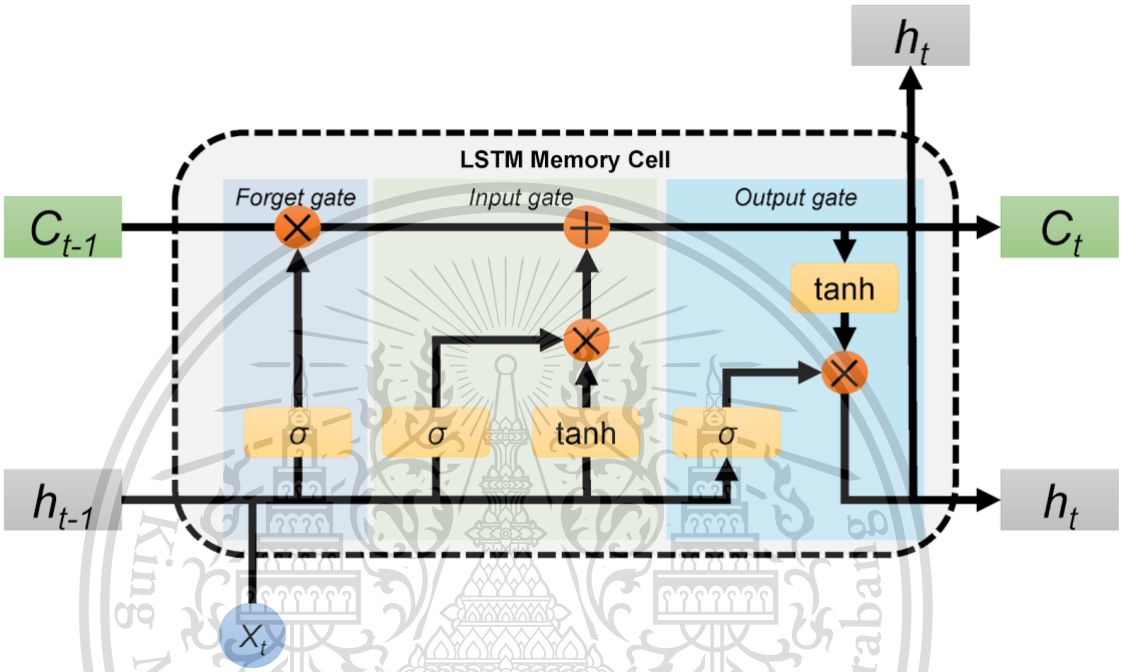
Recurrent neural networks คือ neural networks หลายเลเยอร์ที่สามารถเก็บ(store) ข้อมูล(information) ไว้ที่ node จึงทำให้มันสามารถรับข้อมูลเป็นแบบลำดับ (data sequences) และให้ผลลัพธ์ออกเป็นลำดับของข้อมูลได้ อธิบายอย่างง่ายๆ RNN ก็คือ neural network เชื่อมต่อกันหลายๆอันและยังสามารถต่อกันเป็นวงวน(loop) ได้นั่นเองเพราะฉะนั้น RNN จึงเหมาะสมในการประมวลผลข้อมูลที่เป็นลำดับอย่างมาก



รูปที่ 2.2 รูปตัวอย่างของโมเดล Deep Learning ชนิดต่าง ๆ

2.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) คือ วิธีการทาง machine learning ชนิดหนึ่งที่ใช้กับข้อมูลที่มีความเป็นลำดับ เช่น เสียง, วิดีโอ และหุ้น ซึ่งถือว่าเป็นรูปแบบหนึ่งของ Recurrent Neural Network (RNN) แบบหนึ่ง แต่ LSTM เป็น RNN ที่มีรายละเอียดเพิ่มขึ้นมา เพื่อแก้ไขปัญหาบางอย่างของ RNN



รูปที่ 2.3 long Short-Term Memory (LSTM) architectures

2.2.1 การทำงานของ Long Short-Term Memory (LSTM)

จากรูป 2.3 ตัวแปรใน LSTM มี Cell หรือ ตัวเก็บ state ของ memory cell ใน LSTM, Gate เป็นตัวควบคุมการไหลของข้อมูล ซึ่งก็คือ ค่า analog ที่คอยควบคุมว่าเมื่อไหร่ควร read, write หรือ forget ซึ่งเหมือนกับประตูที่ดูว่า เมื่อไหร่ควรเปิดให้ข้อมูลไหลเข้า, ไหลออก หรือหายไป (forget)

Forget เหมือนการล้าง cell state เดิมออกไป คือ เตรียมเคลียร์พื้นที่รับข้อมูลใหม่ แต่ตัวที่จะตัดสินใจว่าจะลบหรือไม่ เป็นหน้าที่ของ forget gate (อย่างที่กล่าวข้างต้นว่า gate จะเป็นตัวควบคุมการไหลของข้อมูลข้างใน) ซึ่งเป็นเหมือนตัวตัดสิน ถ้า forget gate ให้ค่าเป็น 0 ก็จะลบ cell state เดิมออกไป แต่ถ้า forget gate ให้ค่าเป็น 1 ก็จะยังเก็บ cell state นี้ต่อไป การสร้าง forget gate นี้ จะต้องดู input data ที่เข้ามา ประกอบกับ hidden state ก่อนหน้าประกอบการตัดสินใจ โดยจะใช้ sigmoid function เป็นตัวตัดสิน ดังสมการ

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

Write คือ เมื่อมี input data ใหม่เข้ามาต้องตัดสินใจว่า 1) อัพเดท cell state ด้วย input data ใหม่ดีหรือไม่ ในกรณีนี้จะควบคุมโดยสิ่งที่เรียกว่า input gate ตรงนี้โดยใช้ sigmoid function เป็นตัวตัดสินว่าจะอนุญาตให้อัพเดทหรือไม่ แน่นอนว่าการคำนวณนี้ใช้ค่า input data ที่เข้ามา กับ hidden state ก่อนหน้านั้น

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

และ 2) ถ้าอัพเดทจะอัพเดทด้วยค่าอะไร ในกรณีที่สอง แล้วถ้าอัพเดทจริงๆ จะอัพเดทด้วยค่าอะไร ซึ่งรอบนี้จะใช้สิ่งที่เรียกว่า Input modulation gate เป็นตัวจัดการ โดยสมการก็จะคล้ายกับ input gate แต่ว่าจะใช้เป็น tanh function แทน ซึ่งค่าที่ได้นั้น จะมองว่าเป็น cell state candidate ก็ได้

$$g_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

อัพเดท cell state ตอนนี้เมื่อได้ข้อมูลจาก forget gate, input gate และ input modulation gate แล้ว ซึ่งเพียงพอต่อการอัพเดท cell state ทีนี้ก็รวมทุกสิ่งทุกอย่างเข้าด้วยกัน จะได้ดังสมการ

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

จากส่วนแรกของสมการ จะเห็นว่า ถ้า forget gate บอกว่าให้ลบ cell state เดิมทิ้ง (f_t มีค่าเป็น 0) เราก็จะไม่เอาค่า c_t- 1 มาประกอบการอัพเดท cell state เลย แต่ถ้า f_t มีค่าเป็น 1 เราก็จะยังคงค่า c_t- 1 เอาไว้ประกอบการพิจารณาการอัพเดทอยู่ ในส่วนหลังของสมการจะเป็นส่วนของการอัพเดท cell state จากข้อมูลใหม่ ที่มีค่าที่จะอัพเดทเตรียมรอไว้แล้วจาก input modulation gate หรือ g_t ทีนี้ก็จะมาดูว่า ค่า g_t ที่เตรียมไว้จะได้ใช้หรือไม่ได้ใช้ ซึ่งจะใช้ output จาก input gate หรือ i_t มาตัดสิน ถ้า i_t เป็น 1 ก็จะใช้ค่า g_t อัพเดท แต่ถ้า i_t เป็น 0 ก็จะไม่ใช้ g_t ที่ dimension นั้นอัพเดทจากค่าทั้งหมดที่ได้มา

การ Read จาก RNN ตัวเดิม สิ่งที่ต้องผลิตออกไปก็คือ hidden state ณ เวลาที่ t หรือ ht ออกไป ซึ่งเมื่อตอนที่เวลา t+1 (เหมือนการตั้ง permission ว่าจะสามารถมา read ได้หรือไม่) หรือว่าจะเก็บเอาไว้ ไม่ส่งค่า ht ต่อ ตรงนี้ก็จะมี output gate มาช่วยตัดสินใจ โดยจะใช้สูตรเดิมกับ forget gate และ input gate อยู่ ก็คือ ใช้ sigmoid function กับค่า hidden state ตัวก่อนหน้า กับ input data ที่เข้ามาในตอนนั้น

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

LSTM นี้จะได้เอาค่า h_t นี้ไปคำนวณด้วย ดังนั้น read คือ การที่จะอนุญาตให้คนข้างนอกมา read ตัว h_t ได้หรือไม่ จะได้ output เป็นค่า h_t สำหรับ sequence ถัดไป ดังสมการ

$$h_t = o_t \odot tanh(c_t)$$

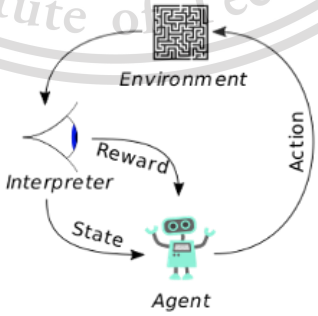
จะเห็นว่า ถ้า output gate ให้ o_t มีค่าเป็น 0 แล้ว ค่าของ h_t ก็จะมีค่าเป็น 0 คือ ไม่ส่งค่าใดๆ ออกไป ในขณะเดียวกัน ถ้า o_t มีค่าเป็น 1 ก็จะคำนวณค่า h_t และส่งออกไปข้างนอก หรืออนุญาตให้สามารถดูค่า h_t ได้

2.3 Reinforcement learning

การเรียนรู้แบบเสริมกำลัง(Reinforcement Learning) เป็นการเรียนรู้ของเครื่อง(Machine Learning) แบบหนึ่ง ที่มีหลักการทำงานเสมือนกับการที่มนุษย์เรียนรู้บางสิ่งบางอย่างด้วยการลองผิดลองถูก และมีการเรียนรู้เกิดขึ้นระหว่างทางว่าการกระทำไหนดีหรือไม่ดี ซึ่ง Reinforcement Learning ประกอบด้วยองค์ประกอบหลัก ดังต่อไปนี้

- Agent: ผู้กระทำ Action
- Action (a): การกระทำของ Agent ที่ส่งผลบางอย่างต่อ Environment
- Environment (e): ระบบที่ Agent ต้องมีปฏิสัมพันธ์ด้วย
- State (s): สถานการณ์ของ Environment ที่ทาง Agent สามารถรับรู้ได้
- Policy (π): หลักการที่ Agent ใช้ในการตัดสินใจเลือก Action หลังจากประเมินสถานการณ์แล้ว
- Reward (R): ตัวประเมินผลลัพธ์ที่เกิดจากการกระทำของ Agent เช่น คะแนน กำไรที่ได้รับ หรือ ผลแพ้ชนะเป็นต้น

มีตัวละครหลัก ๆ 2 ตัวคือ Agent, Environment ทั้ง 2 ตัวจะปฏิสัมพันธ์กันโดย Agent จะทำการสังเกตผลจากการตอบกลับ(Feed Back) ที่มาจาก Environment เพื่อที่ Agent จะออก Action ไปที่ Environment ได้เหมาะสม เราคาดหวังว่า Agent จะพัฒนาทุก ๆ ครั้งที่มันออก Action เพื่อให้ Agent ออก Action ได้ดีที่สุดและบรรลุเป้าหมายที่วางไว้



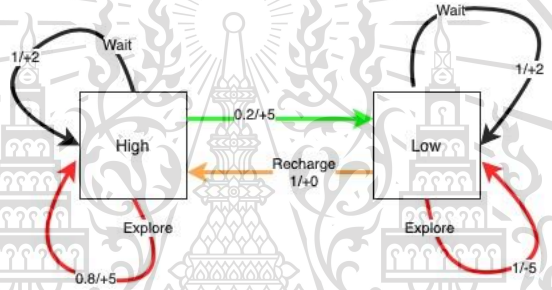
รูปที่ 2.4 หลักการทำงานของ Reinforcement Learning

2.3.1 Markov Decision Process (MDP)

การที่เราจะนำการเรียนรู้ในชีวิตจริงเวลาจะตัดสินใจในการเลือกที่จะทำการกระทำอะไรบางอย่างมาสอนให้คอมพิวเตอร์เข้าใจและสามารถเรียนรู้สิ่งต่าง ๆ ได้นั้นจำเป็นที่จะต้องใช้โมเดลทางคณิตศาสตร์ที่ชื่อว่า Markov Decision Process(MDP)

Markov Decision Process ถูกใช้สำหรับการเลือกตัดสินใจของ Agent เนื่องจาก Environment มีความไม่แน่นอน ทำให้ผลลัพธ์ที่ได้ในบางครั้งไม่แน่นอน เช่น การที่ Agent เลือกทำ Action เดิม สำหรับสถานการณ์ (State) เดิม อาจจะไม่ได้ผลลัพธ์แบบเดิมเสมอไป

ในโมเดล Reinforcement Learning ส่วนใหญ่มักจะนำ Markov Decision Process มาใช้ โดยจะมีลักษณะเป็นกราฟที่ชี้ระหว่างความสัมพันธ์ของแต่ละ State, Action และ Reward ในรูปที่ 2.2 จะแสดงตัวอย่างของรูปกราฟ MDP ของรถเก็บขยะอัตโนมัติโดยจะมี State เป็นไปได้ 2 แบบคือ รถเก็บขยะมีแบตเตอรี่สูง (High) และแบตเตอรี่ต่ำ (Low) มี Action ทั้งหมด 3 แบบคือ ค้นหาขยะ (Explore), หยุดรถ (Wait) และทำการชาร์จแบตเตอรี่ของรถ (Recharge)



รูปที่ 2.5 กราฟ Markov Decision Process ของรถเก็บขยะแบบอัตโนมัติ

โดยค่าที่อยู่ระหว่างเส้นกราฟของ Action จะมี 2 ค่าคือค่าความน่าจะเป็นที่ออก Action นั้น ๆ แล้วจะมีโอกาสไป State ถัดไปที่ State ไหน และอีกค่าหนึ่งก็คือค่า Reward ของการกระทำ Action นั้น ๆ

2.3.2 Optimal Policy (π^*)

การที่เราจะได้โมเดล Reinforcement ที่ดีเราก็ควรทำให้ Agent เรามีหลักการในการตัดสินใจเลือก Action ในระบบ Environment (Policy) ได้ดีโดยเราจะทำการปรับค่า Policy จากการเรียนรู้ไปเรื่อย ๆ จนได้ค่า Policy ที่เหมาะสมที่สุดหรือเรียกได้ว่า Optimal Policy (π^*) ซึ่งจะทำให้ได้มาซึ่ง Reward มากที่สุด สามารถหา Optimal Policy ได้จากสมการดังต่อไปนี้

$$\pi^* = \text{argmax}_\pi R_t$$

โดยในสมการก็หมายถึง Optimal Policy จะมาจากรูปแบบ Action ที่ทำให้เกิด Reward รวม (R_t) มากที่สุดโดย R_t จะหาได้จากสมการดังต่อไปนี้

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$$

ค่า $\boldsymbol{\gamma}$ (Discounted Return) จะมีค่าอยู่ในช่วง 0 ถึง 1 โดยจะเป็นค่าที่ทำให้โมเดลของเราบ่งบอกว่าจะสนใจ Reward ใกล้ ๆ หรือ ไกล ๆ มากกว่า โดยถ้ามีค่าใกล้ 0 จะทำให้สนใจ Reward ที่จะเกิดใกล้ ๆ มากกว่า แต่ถ้ามีค่าใกล้ 1 จะสนใจ Reward ที่ออกไปไกลมากขึ้น

2.3.3 Explore – Exploit Dilemma

การนำ Reinforcement Learning มาใช้จริง ในการจะเลือก Action จะเกิดจาก Optimal Policy ที่ State เมื่อเวลา t ตามสมการดังต่อไปนี้

$$a_t = \pi^*(s_t)$$

แต่ในการที่จะเรียนรู้ให้ได้ Optimal Policy ในการเลือก Action จะใช้หลักการของ Explore – Exploit Dilemma มาใช้ในการเลือก Action

Explore (Random Action): เป็นการให้ Agent เลือก Action ขึ้นมาแบบสุ่มโดยไม่คำนึงถึง Reward ที่จะได้รับ ซึ่งจะทำให้โมเดลได้เรียนรู้สิ่งใหม่มากขึ้นจากการกระทำใหม่ ๆ

Exploit (Greedy Action): เป็นการให้ Agent เลือก Action ที่ดีที่สุด โดยคาดหวังว่าการออก Action นั้น ๆ จะได้ Reward รวม (R_t) ที่ดีที่สุด

โดย Explore – Exploit Dilemma จะเป็นปัญหาที่กล่าวถึงการเลือกระหว่าง Explore และ Exploit ถ้าเราให้โมเดลเลือกที่จะ Exploit มากเกินไป Agent ก็จะไม่ได้รับประสบการณ์ที่เพียงพอที่จะเลือก Action ที่ดีที่สุดได้ แต่ถ้าเลือกที่จะทำ Explore มากเกินไปก็จะทำให้โมเดลไม่ได้ทำการ Exploit เพื่อให้ได้ Reward รวม ที่ดีที่สุดได้ จึงต้องมีการเลือกวิธีที่เหมาะสมในการที่จะให้ Agent ตอบปัญหาเกี่ยวกับ Explore – Exploit Dilemma

เมื่อโมเดลของเรามี Expected Cumulative Reward (Reward รวมที่คาดหวัง) ตรงกับ Actual Reward (Reward รวมครั้งนั้น ๆ ที่โมเดลเรียนรู้มาถึง) สำหรับทุกความเป็นไปได้ นั่นก็แสดงว่าโมเดลของเราได้ทำการ Explore และ Exploit Environment นั้น ๆ จนครบทุกความเป็นไปได้แล้ว และได้มาซึ่ง Optimal Policy

2.3.4 Epsilon greedy

Epsilon Greedy เป็นวิธีหนึ่งในการช่วยให้โมเดลจัดการกับปัญหา Explore – Exploit Dilemma โดยจะให้โอกาสในการเลือกที่จะ Explore หรือ Exploit เปลี่ยนไปตามค่า Epsilon ($\boldsymbol{\varepsilon}$) โดยค่า Epsilon จะมีค่าอยู่ในช่วงตั้งแต่ 0 ถึง 1 โดยจะเป็นค่าความน่าจะเป็นที่โมเดลจะเลือก Action ด้วยการ Explore = $\boldsymbol{\varepsilon}$ และความน่าจะเป็นในการที่จะ Exploit = 1 – $\boldsymbol{\varepsilon}$ มักจะใช้ค่า Epsilon ที่เปลี่ยนไปตามเวลาโดยค่าเริ่มต้นที่ 1 เพื่อให้โมเดลมีการ Explore ในช่วงแรกให้มากที่สุด และ ค่อย ๆ ลดค่า Epsilon ลงไปเรื่อย ๆ เพื่อให้มีการ Exploit มากขึ้น วิธีการนี้จะทำให้โมเดลมีการเลือก Action จากการ Explore และ Exploit ในสัดส่วนที่เหมาะสม

2.4 Deep Q-Network

Deep Q-Network (DQN) เป็นโมเดลการเรียนรู้แบบเสริมกำลัง (Reinforcement Learning) แบบหนึ่งซึ่งใช้หลักการของ Q – Learning และ การเรียนรู้แบบเชิงลึก (Deep Learning) เข้าด้วยกันโดยนำการเรียนรู้แบบเชิงลึกมาใช้ในการประมาณค่าบนตาราง Q ขึ้นมา

2.4.1 Q – Learning

Q – Learning เป็นโมเดลการเรียนรู้แบบเสริมกำลังที่จะใช้ค่า quality value (Q value) ในการหา π^* จากสมการดังต่อไปนี้

$$\pi^* = \operatorname{argmax}_a Q(s, a)$$

ค่า Q value จะถูกคำนวณจากสมการ Bellman Equation เป็นสมการที่จะคำนวณหาค่า Q valueของ Action จากค่า Reward ในการทำ Action นั้น และค่า Q value มากสุดของ State ที่ Action นั้นนำไปถึงตามสมการดังต่อไปนี้

$$Q(s, a) = r + \gamma\,(\max Q(\acute{s}, \acute{a}))$$

โดยค่า Q value จะเป็นค่าที่อยู่ในทุก Action ของทุก State โดยอาจจะแสดงผลได้เป็นตาราง และจะอัพเดตค่าไปเรื่อย ๆจากการ Explore และ Exploit ของโมเดลเพื่อให้การเลือก Action มาจากค่า Q value ที่มากที่สุดใน State นั้น ๆ ซึ่งจะทำให้ Reward ของโมเดลได้ไปถึง Expected Cumulative Reward

รูปที่ 2.6 ตัวอย่างตาราง Q value ของโมเดล Q learning

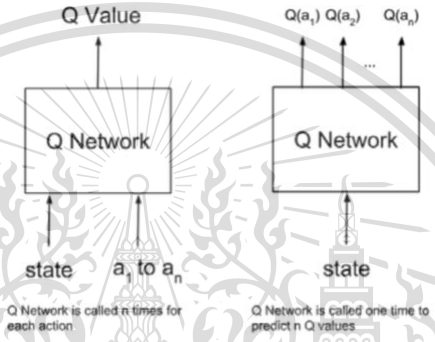
2.4.2 หลักการของ Deep Q-Network

ในการใช้โมเดลการเรียนรู้แบบเสริมกำลัง (Reinforcement Learning) ด้วยโมเดล Q – Learning จะสามารถแก้ปัญหาสำหรับโจทย์ที่มี State ที่ไม่เยอะมาก เช่น การออกแบบให้หุ่นยนต์เดินในเขาวงกตจำลอง เป็นต้น แต่ถ้าหากปัญหาที่ต้องการจะใช้ Reinforcement Learning มาแก้มี State และ Action ที่เยอะมาก หรือเป็นปัญหาที่เกิดขึ้นไม่รู้จบ เช่น การออกแบบให้คอมพิวเตอร์สามารถเล่นเกมได้ หรือ การเล่นหุ้น จะทำให้ต้องออกแบบตาราง Q ตามจำนวน State ที่เป็นไปได้ทั้งหมด ซึ่งมีจำนวนมาก ดังนั้นจึงมีการใช้หลักการของ Deep

Learning มาช่วยในการแก้ปัญหาสำหรับการคิดค่า Q value ของ State โดยจะนำโมเดล Deep Learning มาทำนายค่า Q value บนตาราง Q แทนโดยเรียกหลักการนี้ว่า Deep Q-Network

Deep Q-Network มีการแบ่งออกได้เป็น 2 ประเภทตามข้อมูลขาเข้าและข้อมูลขาออกของโมเดล Deep Learning โดยจะมีชื่อว่า Q Network ดังต่อไปนี้

- ข้อมูลขาเข้าเป็น State และ Action ใด Action หนึ่ง และจะทำนายค่า Q – value ของ Action นั้นออกมา
- ข้อมูลขาเข้าเป็น State และทำนายค่า Q – value ของ Action ทั้งหมดออกมา



รูปที่ 2.7 สถาปัตยกรรมของโมเดลสำหรับทำนายค่า Q – value ของ Deep Q-Network

โดยการหา Policy ของ Deep Q-Network ตามหลักการของ Epsilon Greedy ได้ดังต่อไปนี้

$$\pi = \begin{cases} sample(a) & Random < \epsilon \\ argmaxQ(s,a) & otherwise \end{cases}$$

ในตัวโมเดล *Q Network* สำหรับทำนายค่า *Q value* จะมีการใช้ฟังก์ชันการสูญเสีย*(loss function)* สำหรับการเรียนรู้ของโมเดลตามหลักการของการหาค่า *Q value* ของ *Q-Learning* และ ค่าคลาดเคลื่อนกำลังสองเฉลี่ย (MSE) โดยจะมาจากสมการดังต่อไปนี้

$$\mathcal{L} = (r + \gamma\,(\mathrm{maxQ}(\acute{s}, \acute{a}) - Q(s,a))^2$$

แต่ปัญหาที่พบหากใช้โมเดล Q Network เพียงตัวเดียวสำหรับทำนายค่า Q value ของทั้ง Q(s,a) (ค่า Q value จาก State ปัจจุบัน) และค่า Q(s',a') (ค่า Q value ของ State ถัดไป) จะพบว่ามีค่าความถูกต้องที่น้อยเนื่องมาจาก 2 สาเหตุคือข้อมูลแต่ละ State มีค่าความต่างของความสัมพันธ์ที่สูง (high correlation) และค่าคำตอบที่ไม่อยู่นิ่ง (non-stationary target) ดังนั้นแล้วจึงใช้หลักการการเล่นซ้ำของประสบการณ์ (experience replay) มาช่วยโดยในระหว่างการจำลองของโมเดลจะทำ

การเก็บบัฟเฟอร์ ของ State (s), Action (a), Reward (r), Next State (s') เป็นค่า experience และเมื่อจบถึงการจำลอง 1 รอบ (episode) จะสุ่มค่าใน experience ออกมาจำนวนหนึ่งเพื่อให้โมเดลเรียนรู้เพื่อปรับปรุงความสามารถของโมเดล และจะแยก Q Network เป็น 2 โมเดลเป็นโมเดลที่สำหรับทำนาย Q(s,a) (Q Network) และ Q(s',a') (target Q Network) โดยโมเดล target Q Network จะทำการสำเนาน้ำหนักของโมเดล Q Network ทุก ๆ รอบที่มีการทำ experience replay ถึงจำนวนรอบหนึ่งและ ภาพรวมลำดับขั้นของระบบของการเรียนรู้ Deep Q-Network จะอธิบายได้ดังต่อไปนี้

- วนซ้ำตามจำนวน episodes ครั้ง 1, …., M:
- กำหนดค่าเริ่มต้นของ State s ขึ้นมา
- วนซ้ำตามจำนวนขั้นตอน (step) ใน 1 episode:
- เลือก Action จากหลักการของ Explore – Exploit Dilemma
- ให้ Agent ทำ Action ที่เลือกมา เก็บค่า Reward และเก็บค่า State ถัดไปจากการทำ Action (Next State)
- เก็บค่า State, Action, Reward, Next State เข้า experience
- อัพเดตค่า State = Next State
- เมื่อบัฟเฟอร์ experience มีจำนวน >= ค่าที่กำหนดไว้
- $Q_{max} = \begin{cases} r_{j+1} & if\ episode\ terminate\ at\ j+1 \\ r_{j+1} + \gamma \max Q_{target}(s_{j+1}, a_{j+1}; \theta^{-}) & otherwise \end{cases}$
- ทำการปรับปรุงโมเดล Q Network จากฟังก์ชันการสูญเสีย (loss function) = $(Q_{max} - \mathrm{Q}(s_j, a_j; \theta))^2$
- เมื่อทำการทำการ experience replay ถึงจำนวนรอบที่กำหนดไว้
- $Q_{target} = Q_{Network}$ หรือจะบอกได้อีกอย่างว่า $\theta^{-} = \theta$

2.5 API (Application Programming Interface)

API คือ คำสั่ง (Code) ที่อนุญาตให้ software program สามารถสื่อสารระหว่างกันได้ API เป็นช่องทางสำหรับขอใช้บริการคำสั่ง จาก operation system (OS) หรือ application อื่น ๆ ซึ่งใช้งานโดยติดตั้ง function และเรียกใช้งานตาม document ที่เขียนไว้

2.6 Alpaca

Alpaca เป็นโบรกเกอร์ซึ่งมีสำนักงานใหญ่อยู่ใน Silicon Valley ซึ่งสร้าง API การซื้อขายหุ้นโดยไม่มีค่าคอมมิชชั่น (บริการนายหน้าจัดทำโดย Alpaca Securities LLC, สมาชิก FINRA / SIPC) Alpaca มีทีมงานของบุคคลที่มีภูมิหลังที่หลากหลายซึ่งมีความเชี่ยวชาญด้านการเงินและเทคโนโลยีอย่างลึกซึ้ง ซึ่งได้รับการสนับสนุนจากนักลงทุนชั้นนำในอุตสาหกรรมทั่วโลก

รูปที่ 2.8 Alpaca API

2.6.1 การซื้อขายแบบจำลอง (Paper Trading)

เป็นการซื้อขายจำลองสภาพแวดล้อมแบบเรียลไทม์ที่สามารถทดสอบโค้ดได้ สามารถรีเซ็ตและทดสอบอัลกอริทึมได้มากเท่าที่ต้องการโดยใช้ข้อมูลตลาดแบบเรียลไทม์ฟรี ทุกอย่างในฝั่งโบรกเกอร์ทำงานในลักษณะเดียวกับบัญชีจริงยกเว้นคำสั่งซื้อจะไม่ถูกส่งไปยังการแลกเปลี่ยนจริง แต่ระบบจะจำลองการกรอกคำสั่งซื้อโดยอิงตามคำพูดแบบเรียลไทม์ เมื่อเรียกใช้อัลกอริทึมกับตลาดจริงมีหลายสิ่งที่อาจเกิดขึ้นได้ซึ่งอาจไม่เห็นในการทดสอบย้อนหลัง คำสั่งซื้ออาจไม่เต็มไปด้วยราคาอาจพุ่งสูงขึ้นหรือเครือข่ายอาจถูกตัดการเชื่อมต่อและอาจจำเป็นต้องลองอีกครั้ง ในระหว่างขั้นตอนการพัฒนาซอฟต์แวร์สิ่งสำคัญคือต้องทดสอบอัลกอริทึมเพื่อตรวจจับสิ่งเหล่านี้ล่วงหน้า

การเปรียบเทียบกับการซื้อขายปกติ การซื้อขายแบบจำลองเป็นเพียงการจำลองเท่านั้น เป็นการประมาณที่ดีสำหรับสิ่งที่คาดหวังในการซื้อขายจริง แต่ไม่ใช่สิ่งทดแทนสำหรับการซื้อขายจริงและประสิทธิภาพอาจแตกต่างกัน โดยเฉพาะอย่างยิ่ง Paper Trading ไม่ได้คำนึงถึง ผลกระทบทางการตลาดของคำสั่งซื้อ, การรั่วไหลของข้อมูลคำสั่งซื้อ, การเลื่อนหลุดของราคาเนื่องจากเวลาในการตอบสนอง, ตำแหน่งคิวคำสั่งซื้อ (สำหรับคำสั่ง limit ที่ไม่สามารถทำการตลาดได้), การปรับราคา, ค่าธรรมเนียมการกำกับดูแล, เงินปันผล

บทที่ 3

วิธีการดำเนินงานโครงการ

3.1 ข้อมูลที่นำมาใช้

การทดลองสร้างโมเดลทำนายตลาดหุ้นและโมเดลลงทุนในหุ้นอัตโนมัตินั้นได้อาศัยข้อมูลจากเว็บไซต์ Yahoo ซึ่งเป็นเว็บไซต์ที่รวบรวมข้อมูลการซื้อขายแลกเปลี่ยนหุ้นในสกุลเงินดอลล่าร์ (USD) โดยในการทดลองนี้จะใช้การซื้อขายแลกเปลี่ยนหุ้นของบริษัท Apple ซึ่งข้อมูลที่ใช้เป็นการซื้อขายตั้งแต่ วันที่ 1 มกราคม 2005 ถึง วันที่ 20 เมษายน 2021 รวมทั้งหมดเป็นระยะเวลาประมาณ 15 ปี โดยเก็บข้อมูลการซื้อขายราคาเป็นรายวัน ในรูปแบบของไฟล์ CSV

3.2 การเรียนรู้ของโมเดลทำนายผลราคาหุ้นในอนาคต

3.2.1 การเตรียมข้อมูลเพื่อสร้างโมเดลทำนายหุ้น

การเลือกคุณลักษณะของราคาหุ้นจากเว็บไซต์ Yahoo จากข้อมูลในข้อ 3.1 มีโครงสร้างของข้อมูลที่นำมา ดังนี้

- Close ราคาปิด
- Open ราคาเปิด
- High ราคาสูงสุด
- Low ราคาต่ำสุด
- Volume ปริมาณการซื้อขาย
- Adjust price ราคาปิดรายวันที่มีการปรับผลของเงินปันผล

จากข้อมูลข้างต้นการทดลองนี้จะใช้ข้อมูลในส่วนของราคาปิด(Close) ในการสร้างโมเดลทำนายราคาหุ้นจากนั้นทำการแบ่งข้อมูลออกเป็นชุดสำหรับการฝึกฝนโมเดลและชุดข้อมูลสำหรับการทดสอบประเมินผลโมเดล

การสร้างโมเดล deep learning เราได้จัดชุดข้อมูลให้ข้อมูลฝั่งเข้าโมเดล (input) เป็น data ราคาปิดของหุ้นตามในข้อ 3.2 ทั้งหมด 60 วันก่อนหน้า เพื่อให้โมเดลสามารถทำนายชุดข้อมูลขาออก (output) ที่ทำนายเป็นราคาปิด (close) ของหุ้น 1 วันต่อมา จากนั้นเราแปลงข้อมูลให้ลดความซับซ้อนลง(Normalization) โดยใช้ Minmax Scaler ดังสมการดังนี้

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

จากสมการเราจะนำชุดข้อมูลที่ได้จากการทำ Minmax Scaler มากำหนด Feature Range ใหม่ให้มีค่าอยู่ในช่วง -1 ถึง 1 เพื่อให้เหมาะสมกับโมเดล Long – Short Term Memory มีสมการดังนี้

$$X_{scaled[-1,1]} = X_{scaled} * (F_{max} - F_{min}) + F_{min}$$

กระบวนการทั้งหมดสามารถเขียนโปรแกรมโดยการเขียนโปรแกรมด้วยภาษา python ในรูป 3.1

```
[ ] dataset_train = pd.read_csv('AAPL.csv')
    training_set = dataset_train.iloc[:, 4].values
    news_dataset = pd.read_csv('bert_result.csv')
    training_news_set = news_dataset.iloc[:, 1:4].values
```

```
dataset_train.head()
```

	Date	High	Low	Open	Close	Volume	Adj Close
0	2005-01-03	1.162679	1.117857	1.156786	1.130179	6.919920e+08	0.974950
1	2005-01-04	1.169107	1.124464	1.139107	1.141786	1.096810e+09	0.984962
2	2005-01-05	1.165179	1.143750	1.151071	1.151786	6.804336e+08	0.993589
3	2005-01-06	1.159107	1.130893	1.154821	1.152679	7.055552e+08	0.994359
4	2005-01-07	1.243393	1.156250	1.160714	1.236607	2.227450e+09	1.066760

```
[ ] sc = MinMaxScaler(feature_range=(-1,1))
    training_set_scaled = sc.fit_transform(training_set.reshape(-1, 1))
```

```
[ ] X_train = []
    y_train = []
    X0_train = []
    for i in range(60, 2035):
        X_train.append(training_set_scaled[i-60:i, 0])
        y_train.append(training_set_scaled[i, 0])
        X0_train.append(training_news_set[i])

    X_train, y_train,  X0_train = np.array(X_train), np.array(y_train), np.array(X0_train)

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

รูปที่ 3.1 ตัวอย่างโปรแกรมที่ถูกเขียนโดยภาษา Python ในส่วนขั้นตอนการเตรียมข้อมูล

3.2.2 การสร้างโมเดล

ในส่วนนี้ได้สร้างโมเดลที่เป็น LSTM โดยมีข้อมูลจากราคาปิดหุ้นทั้งหมด 60 วัน โดยมี LSTM ทั้งหมด 4 ชั้น โดยในแต่ละชั้นมี LSTM ทั้งหมด 50 ยูนิตและแต่ละชั้นจะถูกคั่นด้วยการทำ Regularization แบบ Dropout โดยมี Drop Rate ในทุกๆชั้นเท่ากับ 0.2 ส่วนสุดท้ายจะเป็นชั้น Fully Connected หรือ Dense Layer 1 ยูนิต สำหรับการทำนายผลของราคาปิดหุ้นในวันถัดไป โดยทั้งหมดได้ทำการเขียนขึ้นมาด้วยภาษา python โดยมีโครงสร้างของโมเดล ทำนายดังรูปที่ 3.7

```
model = Sequential()

model.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units=1))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 60, 50)            10400
_________________________________________________________________
dropout_8 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
lstm_4 (LSTM)                (None, 60, 50)            20200
_________________________________________________________________
dropout_9 (Dropout)          (None, 60, 50)            0
_________________________________________________________________
lstm_5 (LSTM)                (None, 60, 50)            20200
_________________________________________________________________
dropout_10 (Dropout)         (None, 60, 50)            0
_________________________________________________________________
lstm_6 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_11 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

รูปที่ 3.2 โครงสร้างโมเดลของการทำนายหุ้น

3.2.3 การฝึกฝนให้โมเดล (Model Training)

ได้ใช้การ Optimization Algorithm และ Loss function แบบ Adam และ mean squared error และใช้การทำซ้ำ (Iteration method) ทั้งหมด 1000 รอบมาช่วยให้โมเดลมีการเรียนรู้ที่ดีขึ้นโดยสามารถเขียนเป็นโปรแกรมโดยใช้ภาษา Python ได้ดังรูป 3.8

```
model.compile(optimizer='adam',loss='mean_squared_error')

model.fit(X_train,
          y_train,
          epochs=1000,
          batch_size=32,
          callbacks=[keras.callbacks.EarlyStopping(monitor="loss",
                                                   patience=5,
                                                   mode="min")])
```

รูปที่ 3.3 ตัวอย่างโปรแกรมที่ถูกเขียนโดยภาษา Python ในส่วนขั้นตอนการฝึกฝนให้โมเดล

3.2.4 การวัดผลโมเดล (Model Evaluating)

ในส่วนของการวัดผลนั้นได้มีการเลือกใช้ Mean Absolute Percentage Error (MAPE) มาใช้ในการวัดผลดังสมการ

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|,$$

3.3 การเรียนรู้ของระบบสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้น

3.3.1 การเตรียมข้อมูลก่อนเข้าโมเดล Reinforcement Learning

การที่จะทำนายราคาหุ้นแบบอัตโนมัตินั้น ได้อาศัยหลักการทำงานของโมเดลการเรียนรู้แบบเสริมกำลัง (Reinforcement Learning) โดยจะนำข้อมูลราคาหุ้น close price ในอดีตที่จัดเก็บจากเว็บไซต์ Yahoo เป็นจำนวน 4028 ข้อมูล จัดเก็บเป็นข้อมูลแบบ csv และแบ่งข้อมูลเป็น 2 ชุดโดยเป็นข้อมูลสำหรับเรียนรู้ (train data) จำนวน 3222 ข้อมูล และข้อมูลสำหรับการวัดผล (test data) จำนวน 806 ข้อมูล โดยเป็นอัตราส่วน 80% ต่อ 20% (train data : test data)

```python
import pandas as pd
from pandas_datareader import data as pdr

def APPL_stock_dataset(start_date, tf):
    """
    Creates the dataset containing all stock prices
    :returns: APPL.csv

    * tf(time frame) = 'd' (Day), 'wk' (Week), and 'mo' (Month).
    """
    stock_data = pdr.get_data_yahoo('AAPL', start=start_date, interval=tf)
    stock_data.to_csv('AAPLday_01.csv')

APPL_stock_dataset('2005-01-01', 'd')

def Prepare_Data(data, train_test_ratio):
    #Transform data to datetime
    data['Date'] = pd.to_datetime(data['Date'])
    #Split train/test
    split_ratio = int(np.rint(train_test_ratio * len(data)))
    train = data[:split_ratio]
    test = data[split_ratio:]
    return train, test

data_dir = 'AAPLday_01.csv'
data = pd.read_csv(data_dir)
train_ratio = 0.8
train, test = Prepare_Data(data, train_ratio)
```

รูปที่ 3.4 ตัวอย่างโปรแกรมสำหรับจัดเตรียมข้อมูลก่อนเข้าโมเดลการเรียนรู้แบบเสริมกำลัง

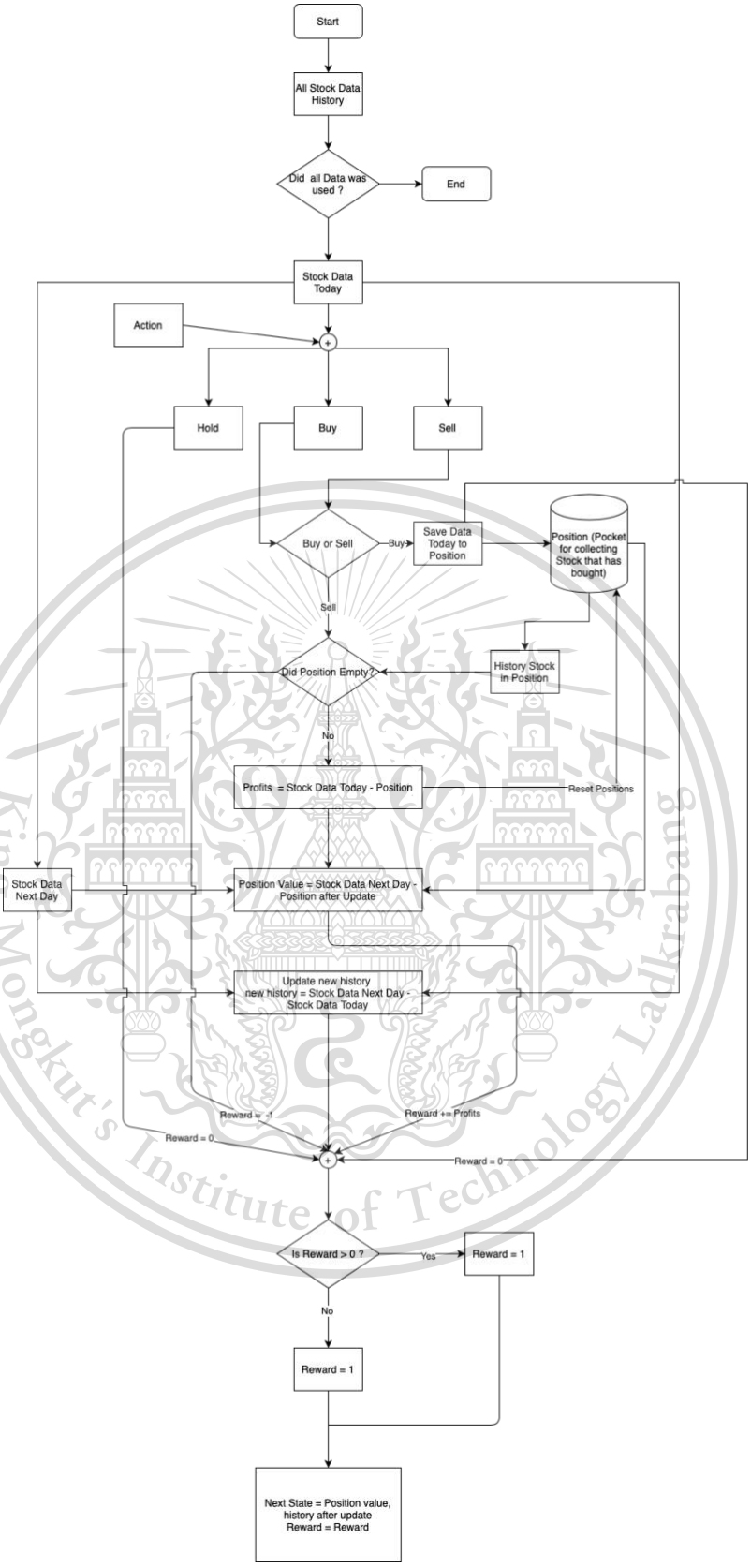
3.3.2 การออกแบบ Environment ของโมเดล Reinforcement Learning

ในการที่จะให้โมเดล Reinforcement Learning เรียนรู้การลงทุนในหุ้น จำเป็นที่จะต้องออกแบบ Environment สำหรับให้ Agent เรียนรู้ ให้เหมาะสำหรับการจะลงทุนในหุ้นได้ ซึ่งจะเป็น Environment ที่เมื่อใส่ State ณ ปัจจุบันและ Action ที่ได้จากตัว Agent เข้าไปจะให้ค่า State ถัดไป, Reward และสถานะว่าถึงตอนจบรอบจำลองแล้วรึยัง(Done Episode) ซึ่งใน State ที่เราออกแบบจะจำลองค่าที่ชื่อว่า Position value ซึ่งเป็นค่าหุ้นที่เราถืออยู่ลบกับราคาหุ้นในวันถัดไป หรือจะพูดได้ว่าเป็นค่าแนวโน้มของหุ้นที่เราถืออยู่เทียบกับราคาหุ้นที่จะเกิดในวันถัดไป รวมเข้ากับ

ค่าที่ชื่อ history เป็นค่าของประวัติของ ราคาหุ้นถัดไปลบกับราคาหุ้นวันนี้ เป็นชุดทั้งหมด 90 วัน ย้อนหลังจากวันที่สนใจ และจะให้จบรอบจำลองเมื่อใช้ข้อมูลสำหรับการเรียนรู้ครบทั้งหมด

```python
class Environment1:

    def __init__(self, data, history_t=90):
        self.data = data
        self.history_t = history_t
        self.reset()

    def reset(self):
        self.t = 0
        self.done = False
        self.profits = 0
        self.positions = []
        self.position_value = 0
        self.history = [0 for _ in range(self.history_t)]
        return [self.position_value] + self.history # obs

    def step(self, act):
        reward = 0
        # act = 0: stay, 1: buy, 2: sell
        if act == 1:
            # positions = all stock that we buy stack
            self.positions.append(self.data.iloc[self.t, :]['Close'])
        elif act == 2: # sell
            # just sell!!!
            if len(self.positions) == 0:
                reward = -1
            # just hold or buy!!!, reset [positions] , get profits, get reward,
            else:
                profits = 0
                for p in self.positions:
                    #sell Close at t's day - Close that we buy (unawhile)
                    profits += (self.data.iloc[self.t, :]['Close'] - p)
                reward += profits
                self.profits += profits
                self.positions = []

        # set next time
        self.t += 1

        self.position_value = 0
        #time + stock buy
        for p in self.positions:
            self.position_value += (self.data.iloc[self.t, :]['Close'] - p)

        #pop first element
        # history of +, close price
        self.history.pop(0)
        #time + 1day
        self.history.append(self.data.iloc[self.t, :]['Close'] - self.data.iloc[(self.t-1), :])
        if (self.t==len(self.data)-1):
            self.done=True
        # clipping reward
        if reward > 0:
            reward = 1
        elif reward < 0:
            reward = -1
        return [self.position_value] + self.history, reward, self.done # obs, reward, done
```

รูปที่ 3.5 ตัวอย่างโปรแกรมการออกแบบ Environment สำหรับโมเดล Reinforcement Learning



รูปที่ 3.6 แผนภาพของการทำงาน Environment

3.3.3 การออกแบบ Agent ของ Reinforcement Learning

การที่เราจะได้มาซึ่ง Action ซึ่งเป็นคำตอบสำหรับนำไปใช้ต่อในการสร้างระบบการลงทุนในหุ้นแบบอัตโนมัติ ว่าเราจะใช้การกระทำใดในการลงทุนในหุ้นจะต้องออกแบบ Agent เพื่อสำหรับกำหนดการออก Action และปรับปรุงโมเดล Q Network เพื่อให้เรียนรู้การทำนายผลค่า Q value ที่จะได้มา

โดยในการทดลองได้กำหนด Action มีทั้งหมด 3 แบบคือสั่งให้ระบบ ซื้อ (BUY), รอ (HOLD) และขาย (SELL)ในการที่จะได้ซึ่ง Action ในระหว่างการเรียนรู้ของโมเดลได้ใช้หลักการของ Epsilon Greedy สำหรับการตอบปัญหา Explore – Exploit Dilemma โดยเริ่มจากกำหนดค่า Epsilon ที่รอบจำลอง (Episode) แรกไว้ที่ 1 และจะค่อย ๆ ลดทอน (decay) ลงเรื่อย ๆ จนมีค่าน้อยที่สุดที่ 0.1

```
self.epsilon = 1.0
self.epsilon_min = 0.1
self.epsilon_decay = self.epsilon_min / self.epsilon
self.epsilon_decay = self.epsilon_decay ** \
                     (1. / float(episodes))
```

รูปที่ 3.7 รูปตัวอย่างโปรแกรมสำหรับการใช้ Epsilon Greedy ในการสร้าง Agent

ซึ่งในโมเดล Q Network ได้ใช้โมเดล Multi Layer Perceptron (MLP) เป็นโมเดลที่มี Neural Network หลายชั้น โดย Q Network โมเดลนี้จะมีทั้งหมด 2 โมเดลที่มีลักษณะสถาปัตยกรรมเหมือนกันคือ Q model และ Target Q model ซึ่งใน Q model จะใช้สำหรับทำนายค่า Q value จาก State ปัจจุบันส่วน Target Q model จะทำนายค่า Q value จาก State ถัดไปที่ Action ที่ทำนายจาก Q model พาไปถึง โดยข้อมูลขาเข้าของโมเดลจะเป็น State ซึ่งมีขนาด (None, 91) ซึ่งประกอบไปด้วยค่า Position value 1 ค่า และ history จำนวน 90 วันย้อนหลัง ซึ่ง State ทั้งหมดจะได้มาจาก Environment และข้อมูลขาออกจะเป็น Q value ทั้งหมด 3 ค่าซึ่งแสดงถึงค่า Q value ของการซื้อ, รอ และขาย

```
def build_model(self, n_inputs, n_outputs):
    """Q Network is 256-256-256 MLP

    Arguments:
        n_inputs (int): input dim
        n_outputs (int): output dim

    Return:
        q_model (Model): DQN
    """
    inputs = Input(shape=(n_inputs, ), name='state')
    x = Dense(128, activation='relu')(inputs)
    x = Dense(256, activation='relu')(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(n_outputs,
              activation='linear',
              name='action')(x)
    q_model = Model(inputs, x)
    q_model.summary()
    return q_model
```

รูปที่ 3.8 รูปตัวอย่างสถาปัตยกรรมของโมเดล Q Network

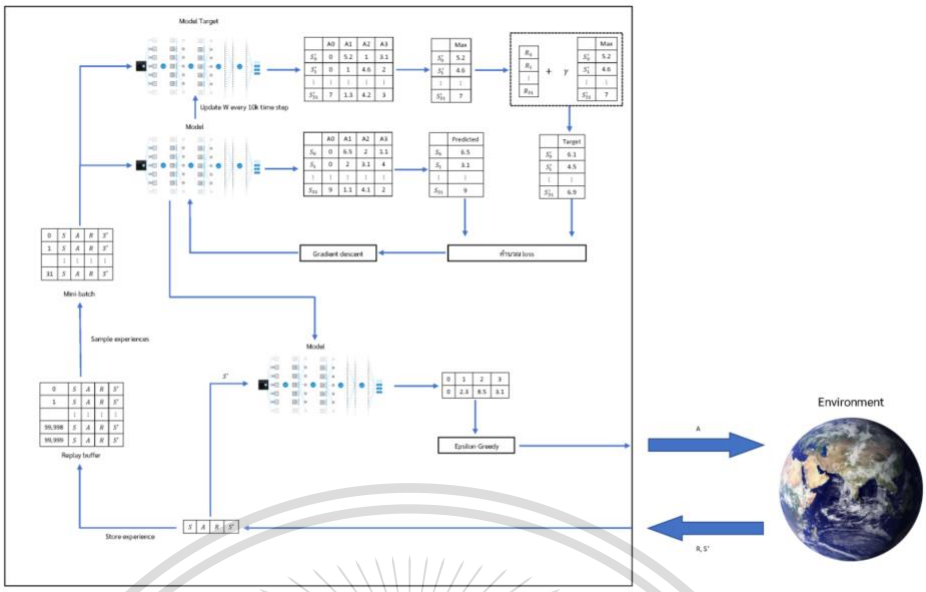
3.3.4 การเรียนรู้ของโมเดล Deep Q-Network

ในการที่จะเรียนรู้โมเดล Deep Q-Network จะต้องกำหนดค่าพารามิเตอร์ต่าง ๆ เพื่อให้ใช้สำหรับการเรียนรู้ โดยกำหนดให้เรียนรู้ทั้งหมด 1000 รอบการเรียนรู้จำลอง (episode) ,ขนาดของบัฟเฟอร์สำหรับการนำไปเรียนรู้ (experience buffer) = 150, จะอัพเดตน้ำหนักของโมเดล Target Q Network ให้เท่ากับโมเดล Q Network เมื่อมีการทำ experience replay ครบ 10 ครั้ง และจะให้ตัดจบการเรียนรู้เมื่อมีค่า Reward เฉลี่ยใน 100 รอบจำลองเกินจำนวน 1000 โดยทั้งหมดเราจะเขียนเป็นการวนซ้ำของจำนวนรอบการจำลอง โดยอาศัยชุดโปรแกรมจาก Environment และ Agent ที่เขียนขึ้นมาโดยโปรแกรมทั้งหมดจะเขียนจากภาษา Python และใช้ Keras เป็นแบ็กเอ็นด์ โดยหลังจากทำการเรียนรู้เสร็จจะจัดเก็บน้ำหนักของโมเดล Q Network เป็นไฟล์ .h5 ไปใช้ต่อในระบบของการเทรดหุ้นแบบอัตโนมัติ

รูปที่ 3.9 ตัวอย่างไฟล์ที่จัดเก็บของโมเดล Q Network เป็นไฟล์ .h5

```
# Q-Learning sampling and fitting
for episode in range(episode_count):
    print('Episode :', episode)
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    done = False
    total_reward = 0
    while not done:

        action = agent.act(state)
        next_state, reward, done = env.step(action)

        next_state = np.reshape(next_state, [1, state_size])
        # store every experience unit in replay buffer
        agent.remember(state, action, reward, next_state, done)
        state = next_state
        total_reward += reward

    # call experience relay
    if len(agent.memory) >= batch_size:
        agent.replay(batch_size)

    scores.append(total_reward)
    mean_score = np.mean(scores)
    if mean_score >= win_reward \
            and episode >= win_trials:
        print("Solved in episode %d: \
                Mean survival = %0.2lf in %d episodes"
              % (episode, mean_score, win_trials))
        print("Epsilon: ", agent.epsilon)
        agent.save_weights()
        break
    if (episode + 1) % reported_episode == 0:
        print("Episode %d: Mean survival = \
                %0.2lf in %d episodes" %
              ((episode + 1), mean_score, reported_episode))
    agent.save_weights()
    dict_epsilon = {'epsilon' : agent.epsilon}
    with open(agent.model_path + 'epsilon.json', 'w') as fp:
      json.dump(dict_epsilon, fp)
agent.save_weights()
dict_epsilon = {'epsilon' : agent.epsilon}
with open(agent.model_path + 'epsilon.json', 'w') as fp:
    json.dump(dict_epsilon, fp)
```

รูปที่ 3.10 ตัวอย่างโปรแกรมของการเรียนรู้ของ Reinforcement Learning



รูปที่ 3.11 แผนภาพการทำงานทั้งหมดของการเรียนรู้ Deep Q-Network

3.4 การสร้างระบบลงทุนในหุ้นอัตโนมัติ

จากการสร้างและเรียนรู้โมเดลทำนายหุ้นและโมเดลสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้น เราจึงนำโมเดลทั้ง 2 โมเดลไปใช้กับการลงทุนในหุ้นแบบจำลอง (Paper trade) เพื่อบรรลุจุดประสงค์ของโครงงานในการสร้างระบบลงทุนในหุ้นอัตโนมัติ

เราจึงได้ออกแบบโปรแกรมสำหรับการทำนายพฤติกรรมของการลงทุนในหุ้นแบบอัตโนมัติจากราคาหุ้นในตลาดการซื้อขาย โดยข้อมูลขาเข้าของโมเดลสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นจำเป็นต้องมีราคาปิดของหุ้นในวันถัดไป ดังนั้นจึงต้องใช้คำตอบจากการทำนายของโมเดลทำนายหุ้นเพื่อเป็นข้อมูลขาเข้า

3.4.1 โปรแกรมสำหรับทำนายราคาหุ้นในวันถัดไปจากราคาหุ้นในตลาด

จากโมเดลทำนายหุ้นที่ได้ถูกเรียนรู้มาแล้วจะต้องนำไปติดตั้งเข้ากับระบบการลงทุนในหุ้นแบบอัตโนมัติโดยเราได้ทำการสร้างโมดูลที่ทำนายราคาหุ้นของวันถัดไปจากราคาหุ้นในตลาดและจะนำผลลัพธ์ที่ได้ไปเข้าเป็นข้อมูลขาเข้า Environment ของโมเดล Reinforcement เป็นค่าหุ้นในวันถัดไปสำหรับการคำนวณค่า Position Value และ History ในการคำนวณ State ถัดไปของโมเดล โดยโมดูลนี้ได้ออกแบบด้วยภาษา Python

```python
#60 days before to predict next day
def get_predict(x, model, norm_func):
  x_pre = np.array(x)
  x_test = norm_func.fit_transform(x_pre.reshape(-1, 1))
  x_test = np.reshape(x_test, (1, 60, 1))
  predict = model.predict(x_test)
  predict_nextday = norm_func.inverse_transform(predict)
  return predict_nextday
```

รูปที่ 3.12 ตัวอย่างโปรแกรมการทำนายราคาหุ้นในวันถัดไปในโมดูล

3.4.2 โปรแกรมสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นจากราคาหุ้นในตลาด

หลังจากที่เราได้ราคาในวัดถัดไปจากโมเดลทำนายราคาหุ้นในวันถัดไปเราจะนำราคาในวันถัดไปมาเข้า Environment ของ Reinforcement Learning ซึ่งจะเป็น Environment ที่จะให้ค่าเฉพาะ State ถัดไปเพื่อที่จะสร้างโมดูลสำหรับการเลือกพฤติกรรมจากลงทุนในหุ้นในตลาด

```python
class predict_rl():
    def __init__(self, model_path, history_t = 90):
        self.input_size = history_t + 1
        self.model_path = model_path
        # buy, hold, sell
        self.output_size = 3

    def _load_model(self):
        inputs = Input(shape=(self.input_size, ), name='state')
        x = Dense(128, activation='relu')(inputs)
        x = Dense(256, activation='relu')(x)
        x = Dense(256, activation='relu')(x)
        x = Dense(256, activation='relu')(x)
        x = Dense(self.output_size,
                  activation='linear',
                  name='action')(x)
        model = Model(inputs, x)
        model.load_weights(self.model_path)
        return model
    def get_action(self, env, flag_start, old_state_path):
        if flag_start:
            raw_state = np.load(old_state_path)

        else:
            raw_state = env.reset()
        state = np.reshape(raw_state, [1, self.input_size])
        model = self._load_model()
        act = model.predict(state)
        pact = np.argmax(act[0])
        next_state = env.step(pact, raw_state)
        np.save(old_state_path, next_state)
        # print(pact, next_state)
        return pact, next_state
```

รูปที่ 3.13 ตัวอย่างโปรแกรมการเลือกพฤติกรรมในหุ้นในโมดูล

```python
class Environment:

    def __init__(self, data_today,data_tmrw, positions_path, history_t=90):
        self.data = data_today
        self.data_tmrw = data_tmrw
        self.history_t = history_t
        self.positions_path = positions_path
        self.reset()


    def reset(self):
        self.done = False
        self.profits = 0
        self.positions = []
        self.position_value = 0
        self.history = [0 for _ in range(self.history_t)]
        return [self.position_value] + self.history # obs

    def step(self, act, state):
        reward = 0
        if not os.path.exists('positions.npy'):
            np.save('positions.npy',self.positions)
        if os.path.exists('old_state.npy'):
            self.history = list(state[1:])
        self.positions = np.load(self.positions_path)
        # act = 0: stay, 1: buy, 2: sell
        if act == 1:
            list(self.positions).append(self.data.iloc[0,5])
        elif act == 2: # sell
            if len(self.positions) == 0:
                reward = -1
            else:
                profits = 0
                for p in self.positions:
                    profits += (self.data.iloc[0,5] - p)
                reward += profits
                self.profits += profits
                self.positions = []


        self.position_value = 0

        for p in self.positions:
            self.position_value += (self.data_tmrw - p)

        self.history.pop(0)

        self.history.append(self.data_tmrw- self.data.iloc[0,5])

        np.save(self.positions_path,self.positions)

        return [self.position_value] + self.history
```

รูปที่ 3.14 ตัวอย่างโปรแกรมการออกแบบ Environment สำหรับการทำนายพฤติกรรม

3.4.3 โปรแกรมสำหรับปรับปรุงโมเดล

เนื่องจากเราใช้โมดูลนี้สำหรับราคาหุ้นจริงในตลาดซึ่งราคาหุ้นจะมีการแปรผันไปในตามแต่ละช่วงเวลา ซึ่งในอนาคตโมเดลทำนายหุ้นและโมเดลสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นอาจทำให้การตัดสินใจและการทำนายผิดพลาดจากราคาที่แปรผันไป

ดังนั้นเราจึงออกแบบโมดูลสำหรับการปรับปรุงโมเดลเมื่อได้ทำการซื้อขายไปถึงช่วงระยะเวลาหนึ่งซึ่งเรากำหนดให้มีการปรับปรุงทุกๆการซื้อขาย 30 ครั้ง โดยการเก็บหน่วยความจำจำลอง (Memory logs) ซึ่งจะทำการบันทึกการซื้อขาย เพื่อตรวจสอบจำนวนครั้งในการซื้อขายแล้ว

จึงปรับปรุงโมเดลทำนายหุ้นเป็นจำนวน 10 Epochs และสำหรับโมเดลสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นจำนวน 1 Episode จากข้อมูลราคาหุ้นที่ยังไม่ได้เรียนรู้

```python
def train_predict_model(self, model_path, X_train, y_train,
                        epoch=10,
                        batch_size = 32,
                        optimizer = 'adam',
                        loss = 'mean_squared_error'):
    model = keras.models.load_model(model_path)
    model.fit(X_train, y_train,
              epochs=10,
              batch_size=batch_size,
              callbacks=[keras.callbacks.EarlyStopping(monitor="loss",
                                                       patience=5,
                                                       mode="min")])
    model.save(model_path)

def train_rl_model(self, historical_data, memory_logs, weights_file):
    datatmrw = 119.5
    retrain = rl_update.retrain_dqn(historical_data, memory_logs)
    input_size = 90 + 1
    output_size = 3

    agent = rl_update.DQNAgent(input_size, output_size, weights_file)
    env = rl_update.Environment_training(historical_data, datatmrw)
    retrain.train(env, agent)


def update(self):
    config, sc = self.load_config()
    historical_data = config['historical_data']
    memory_logs = config['memory_logs']
    X_train, y_train = self.preprocess(sc, historical_data)
    self.train_predict_model(config['model_path'], X_train, y_train)
    self.train_rl_model(historical_data, memory_logs, config['rl_model_path'])
```

รูปที่ 3.15 ตัวอย่างโปรแกรมสำหรับการปรับปรุงโมเดล

```python
def update_memory_logs(mem_path, today_price):
    if os.path.exists(mem_path) == True:
        print('Memory logs are already exists')
        mem_logs = pd.read_csv(mem_path)
        logs = today_price
        logs.columns = [''] * len(logs.columns)
        logs.columns = ["Date", "Open", "High", "Low", "Close", "Volume"]
        mem = pd.concat([mem_logs, logs])
        mem.to_csv(mem_path)
        print('Momory logs has updated')
    elif os.path.exists(mem_path) == False:
        print('Memory logs are not already exists.')
        mem_logs = pd.DataFrame(columns=["Date", "Open", "High", "Low", "Close", "Volume"])
        print('Memory logs are created.')
        mem_logs.to_csv(mem_path)
        logs = today_price
        logs.columns = [''] * len(logs.columns)
        logs.columns = ["Date", "Open", "High", "Low", "Close", "Volume"]
        mem = pd.concat([mem_logs, logs])
        mem.to_csv(mem_path)
        print('Momory logs has updated.')

    return mem
```

รูปที่ 3.16 ตัวอย่างโปรแกรมสำหรับการเก็บ Memory logs

3.4.4 โปรแกรมแสดงผลลัพธ์ของระบบลงทุนในหุ้นแบบอัตโนมัติ

จากโมดูลสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นจากราคาหุ้นในตลาดจะได้ผลลัพธ์เป็น Action สำหรับการซื้อขายหุ้นเพื่อที่จะนำไปจำลองการลงทุนในหุ้นแบบจำลอง

โดยโมดูลนี้จะเป็นโมดูลที่ประกอบการทำงานของ โปรแกรมสำหรับทำนายราคาหุ้นในวันถัดไปจากราคาหุ้นในตลาด, โปรแกรมสำหรับการเลือกพฤติกรรมในการลงทุนในหุ้นจากราคาหุ้นในตลาด และ โปรแกรมสำหรับปรับปรุงโมเดลเข้าด้วยกัน

```python
def main_action(config_path, close_price, data_tdy):
    config, sc = load_config(config_path)
    model = keras.models.load_model(config["model_path"])
    nextday_close_price = get_predict(close_price, model, sc)
    data_tmrw = nextday_close_price[0][0]
    predict = predict_rl(config["rl_model_path"])
    env = Environment(data_tdy,data_tmrw,config['positions'])
    flag_start = os.path.exists(config["old_state"])
    action, next_state = predict.get_action(env, flag_start, config["old_state"])
    mem_logs = update_memory_logs(config['memory_logs'], data_tdy)

    if len(mem_logs) >= 30:
        APPL_stock_dataset('2005-01-01', 'd', config['historical_data'])
        print('loading dataset successful.')
        up = update_model.update_model(config_path)
        up.update()
        print('Update model successful')
        os.remove(config['memory_logs'])

    return action
```

รูปที่ 3.17 ตัวอย่างโปรแกรมแสดงผลลัพธ์ของระบบลงทุนในหุ้นแบบอัตโนมัติ

3.4.5 โปรแกรมสำหรับส่งผลการตัดสินใจสำหรับการลงทุนไปยัง Alpaca

หลังจากที่ได้ผลลัพธ์จากโมเดล reinforcement learning โดย action 0 = รอ, 1 = ซื้อ, 2 = ขาย จากนั้นตรวจสอบเวลาว่าตลาดหุ้นเปิดหรือปิดแล้วทำการซื้อขายหุ้นตาม action ที่ได้

```python
# authentication and connection details
api_key = '[redacted]'
api_secret = '[redacted]'
base_url = 'https://paper-api.alpaca.markets'

api = tradeapi.REST(api_key, api_secret, base_url, api_version='v2')
asset = api.get_asset('AAPL')
clock = api.get_clock()
```

รูปที่ 3.18 การส่ง request ไปยัง Alpaca โดยใช้ API

```python
def trade_asset(action_rl):
    act = {0:"hold", 1:"buy", 2:"sell"}
    if asset.tradable == True:
        if clock.is_open == True and asset.tradable == True:
            if action_rl == 1 or action_rl == 2:
                api.submit_order(symbol=asset.symbol,
                                 qty=100,
                                 side=act[action_rl],
                                 time_in_force='cls',
                                 type='market')
                print('Order {} 100 shares of {} is submitted.'.format(act[action_rl], asset.symbol))
            else:
                print('Action for today is {}'.format(act[action_rl]))
        else:
            time_dif = clock.next_open - clock.timestamp
            day_diff = time_dif.components.days
            hours_diff = time_dif.components.hours
            min_diff = time_dif.components.minutes
            print('The market is closed.')
            print('And open in %d day %d hours %d minutes.' %(day_diff, hours_diff, min_diff))

    else :
        print('Can not trade AAPL.')

out = main_action('config.json', past60days_close_price, today_price)
trade_asset(out)
```

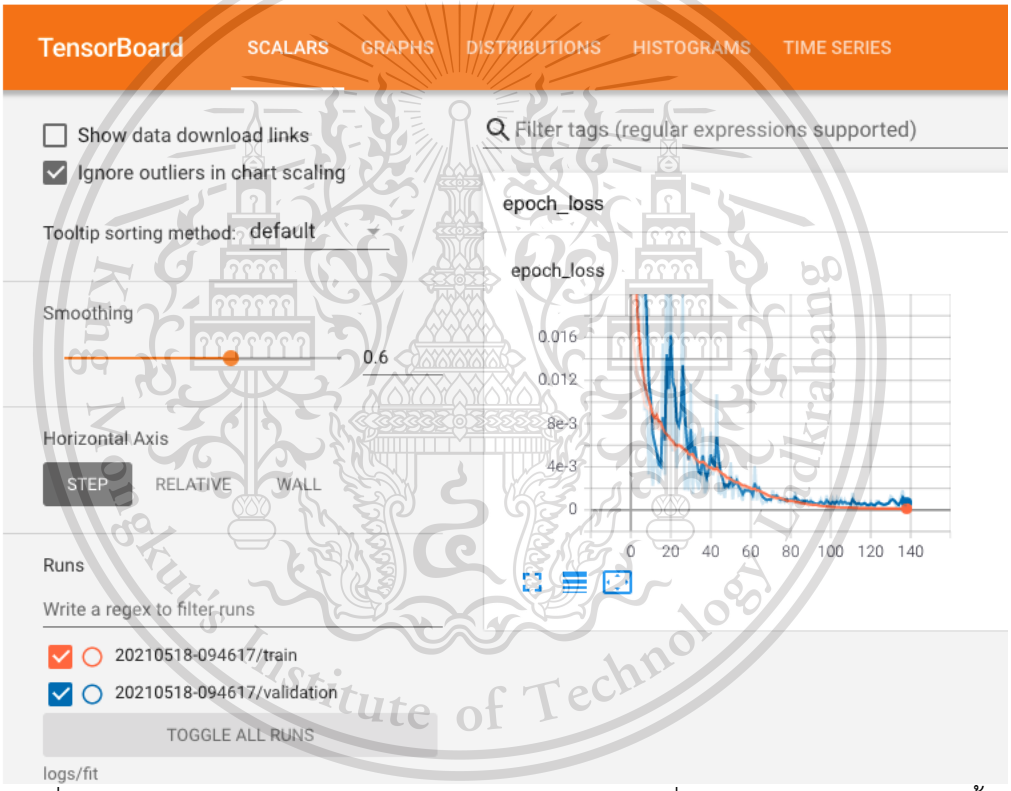
รูปที่ 3.19 ฟังก์ชันสำหรับส่งคำสั่งการซื้อขายไปยัง Alpaca

บทที่ 4

ผลการทดลอง

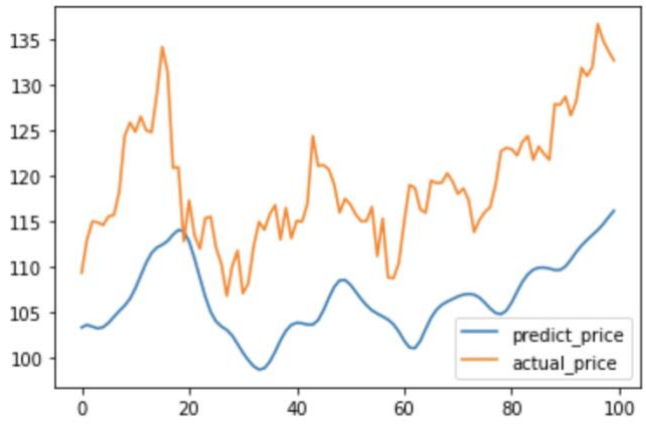
4.1 ผลการเรียนรู้ของโมเดลสำหรับทำนายผลราคาหุ้น

หลังจากการฝึกฝนโมเดลสำหรับการทำนายผลราคาหุ้นด้วย LSTM Neural Network ได้กำหนดพารามิเตอร์ต่าง ๆ ของการเรียนรู้คือ ทำการเรียนรู้เป็นจำนวน 1000 epochs และได้ใส่ค่าให้โมเดลหยุดการเรียนรู้เมื่อค่าสูญเสียของข้อมูลทดสอบ (Validation Loss) ไม่เปลี่ยนแปลงเกิน 0.1 เป็นจำนวน 5 epoch (Early Stopping) จะได้กราฟที่พล็อตระหว่างค่าความคลาดเคลื่อนของโมเดล (Loss) และจำนวนรอบการทำซ้ำ (epochs) จากโมดูล Tensorboard ซึ่งเป็นเครื่องมือที่จะจัดเก็บค่าประวัติการเรียนรู้เพื่อนำมาวิเคราะห์ และปรับปรุงโมเดลให้ได้ผลลัพธ์ตามที่ต้องการ ดังรูปที่ 4.1



รูปที่ 4.1 กราฟของการเทรนโมเดลระหว่างค่าความคลาดเคลื่อนและจำนวนรอบการทำซ้ำ

จากการทดลองผลของโมเดล เราได้พล็อตกราฟเปรียบเทียบผลการทำนายราคาหุ้น และราคาจริงในชุดข้อมูลสำหรับทดสอบทั้งหมดโดยเป็นราคาปิดหุ้นหลังจากวันที่ 17 เดือนกรกฎาคม ปี 2020 ถึงวันที่ 25 เดือนกันยายน 2020 เป็นจำนวน 50 วัน ตามดังรูปที่ 4.2



รูปที่ 4.2 รูปเปรียบเทียบราคาปิดหุ้นที่ทำนายกับราคาปิดหุ้นจริงในแต่ละวัน

```
[18] predict = model.predict(X_test)

     predict_price = sc.inverse_transform(predict.reshape(-1, 1))
     actual_price = sc.inverse_transform(y_test.reshape(-1, 1))

[20] import matplotlib.pyplot as plt
     plt.plot(np.arange(0,50), predict_price)
     plt.plot(np.arange(0,50), actual_price)

     plt.legend(["predict_price", "actual_price"])
```

รูปที่ 4.3 ตัวอย่างโปรแกรมสำหรับการวัดผลโมเดลทำนายหุ้น

4.2 ผลการเรียนรู้ของโมเดลสำหรับเลือกพฤติกรรมการลงทุนในหุ้น

หลังจากทำการเรียนรู้โมเดล Deep Q-Network เป็นจำนวนทั้งหมด 1000 episodes และจัดเก็บน้ำหนักของโมเดล Deep Q-Network เป็นไฟล์ .h5 ได้นำข้อมูลที่แบ่งไว้สำหรับการวัดผล (test data) จำนวน 806 ข้อมูล เข้าตัวโปรแกรมทดสอบโมเดลที่เขียนขึ้น โดยเป็นโปรแกรมที่เขียนจากภาษา Python และจำลองให้เหมือนตอนเรียนรู้แต่เลือกที่จะ Exploit อย่างเดียว หรือก็คือเลือก Action ที่ได้ตามค่า Q value มากสุดที่ได้จาก State ที่ใส่เข้าไปทดสอบในโมเดล และทำการจัดเก็บค่ากำไรทั้งหมด (Profits) เพื่อวัดผลว่าได้กำไรเท่าไหร่ ซึ่งค่ากำไรจะแปรผันตาม Reward ที่ได้จากการขายหุ้น

```
import tensorflow as tf
from tf.keras.models import Sequential
from tf.keras.layers import Dense, Activation
test_env = Environment1(test)
state = test_env.reset()
test_acts = []
test_rewards = []


input_size=test_env.history_t+1
output_size=3
inputs = Input(shape=(input_size, ), name='state')
x = Dense(128, activation='relu')(inputs)
x = Dense(256, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(output_size,
          activation='linear',
          name='action')(x)
q_model = Model(inputs, x)
q_model.summary()
## model_batch_150, testrl is very good
q_model.load_weights('./models/model_final_500/trading_dqn.h5')
for _ in range(len(test_env.data)-1):


    state = np.reshape(state, [1, input_size])
    acc = q_model.predict(state)
    pact = np.argmax(acc[0])
    test_acts.append(pact)

    obs, reward, done = test_env.step(pact)
    test_rewards.append(reward)

    state = obs

test_profits = test_env.profits
```

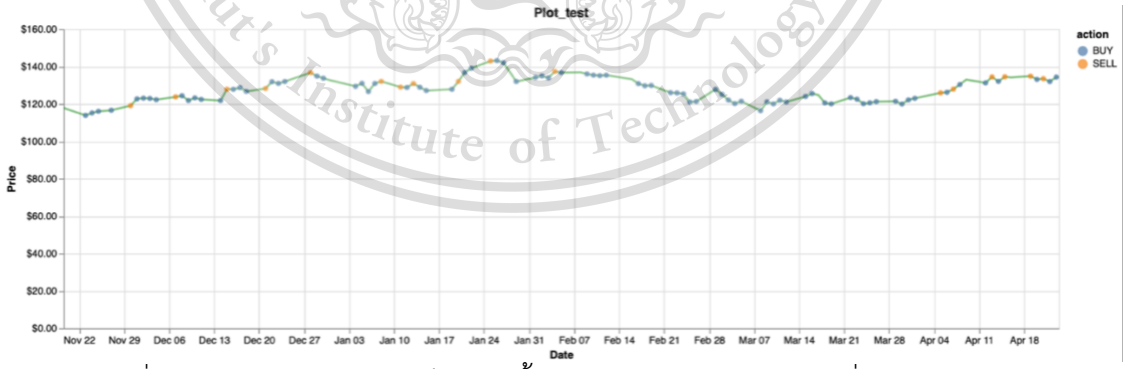
รูปที่ 4.4 รูปตัวอย่างโปรแกรมของการวัดผลโมเดล Deep Q-Network

ซึ่งจากการทดสอบได้ผล Profits จากราคาหุ้นทั้งหมด 806 ข้อมูลโดยเป็นข้อมูลตั้งแต่วันที่ 6 เดือนกุมภาพันธ์ 2018 ถึงวันที่ 20 เดือนเมษายน 2021ที่นำมาตัดสินใจว่าจะซื้อ, รอ หรือขาย ได้รวมทั้งหมด 626.468 USD ซึ่งในการซื้อหรือขายจำลองว่าซื้อจำนวน 1 เหรียญตามราคาค่า Close Price และขายหุ้นตามราคา Close Price

```python
import tensorflow as tf

test_env = Environment1(test)
state = test_env.reset()
test_acts = []
test_rewards = []


input_size=test_env.history_t+1
output_size=3
inputs = Input(shape=(input_size, ), name='state')
x = Dense(128, activation='relu')(inputs)
x = Dense(256, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dense(output_size,
          activation='linear',
          name='action')(x)
q_model = Model(inputs, x)
q_model.summary()
## model_batch_150, testrl is very good
q_model.load_weights('./models/model_final_500/trading_dqn.h5')
for _ in range(len(test_env.data)-1):

    state = np.reshape(state, [1, input_size])
    acc = q_model.predict(state)
    pact = np.argmax(acc[0])
    test_acts.append(pact)

    obs, reward, done = test_env.step(pact)
    test_rewards.append(reward)

    state = obs

test_profits = test_env.profits

print('Profits from test_data:',test_profits)
```

```
Profits from test_data: 626.4675025939941
```

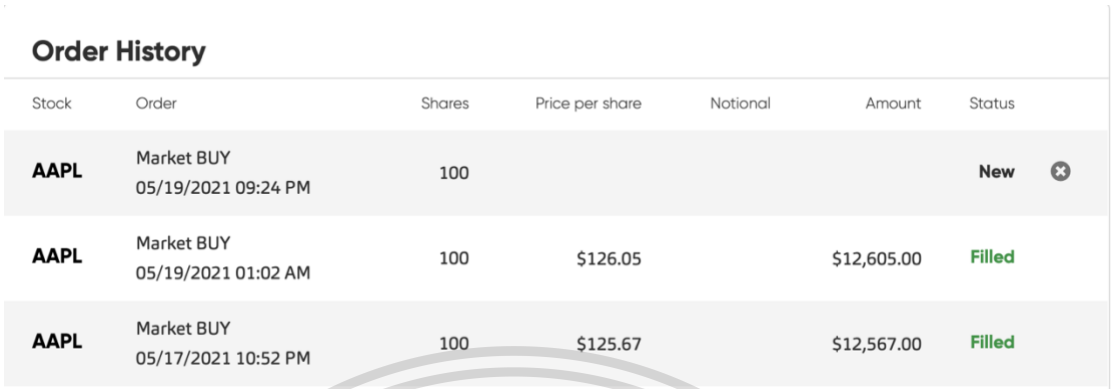
รูปที่ 4.5 รูปการวัดผลค่ากำไรที่ได้จากข้อมูลวัดผล



รูปที่ 4.6 กราฟตัวอย่างการพล็อตการซื้อ, รอ และขายของชุดข้อมูลที่ผ่านการวัดผล

4.3 ผลลัพธ์จากการส่งคำขอซื้อขายไปยัง Alpaca

หลังจากส่งคำสั่งซื้อขายหุ้นที่ได้จากโมเดล Reinforcement learning ไปยัง Alpaca สามารถตรวจสอบคำสั่งซื้อขายหุ้นที่ได้ส่งไปได้ทางเว็บไซต์ Alpaca



Order History

Stock	Order	Shares	Price per share	Notional	Amount	Status
AAPL	Market BUY 05/19/2021 09:24 PM	100				**New**
AAPL	Market BUY 05/19/2021 01:02 AM	100	$126.05		$12,605.00	**Filled**
AAPL	Market BUY 05/17/2021 10:52 PM	100	$125.67		$12,567.00	**Filled**

รูปที่ 4.6 คำสั่งซื้อขายที่ถูกส่งไปยังเว็บไซต์ Alpaca

บทที่ 5

สรุปผลการทดลอง

5.1 สรุปผลการทดลอง

จากการทดลองสร้างระบบอัจฉริยะเพื่อการทำนายหุ้นและลงทุนในหุ้น โดยใช้หลักการของโมเดลความจำระยะสั้นระยะยาว (LSTM Neural Network) เพื่อทำนายราคาหุ้นในอนาคต และหลักการของโมเดลการเรียนรู้แบบเสริมกำลัง (Reinforcement Learning) ด้วยโมเดล Deep Q-Network เพื่อทำนายพฤติกรรมการลงทุนในหุ้นของระบบด้วยการซื้อ, รอ หรือขายหุ้น และนำไปจำลองการซื้อขายหุ้นบนโปรแกรมการลงทุนในหุ้นจำลอง (Paper Trade) ชื่อ Alpaca ได้พบว่าในการนำโมเดลทั้ง 2 นำมาใช้ร่วมกันในการนำค่าทำนายราคาหุ้นในวันถัดไปใช้เป็นข้อมูลขาเข้าร่วมกับข้อมูลราคาหุ้นในวันนั้นให้กับโมเดล Deep Q-Network มีการใช้ร่วมกันได้โดยสามารถสร้างเป็นโปรแกรมสำหรับส่งผลทำนายพฤติกรรมที่เลือกว่าจะ ซื้อ, รอ หรือขายหุ้น ไปจำลองการซื้อขายหุ้นบน Alpaca ได้ และในประสิทธิภาพของโมเดล ในด้านของโมเดลสำหรับทำนายราคาหุ้นในอนาคตสามารถทำนายราคาหุ้นได้ใกล้เคียงกับราคาหุ้นในอนาคตจริง โดยมีแนวโน้มของการขึ้นลงของราคาหุ้นใกล้เคียงกัน และในโมเดลสำหรับทำนายพฤติกรรมการลงทุนในหุ้นได้ทดลองกับข้อมูลราคาหุ้นในอดีตและได้กำไรมากถึง 626.468 USD ในข้อมูลขาเข้าที่เป็นราคาหุ้นทั้งหมด 806 ข้อมูลหรือจำลองว่าทำการลงทุนกับหุ้น 3 ปีตั้งแต่ปี 2018 ถึง 2021 โดยถ้านำระบบทั้งหมดนี้ไปใช้งานจริง สามารถนำไปใช้เป็นส่วนหนึ่งในการตัดสินใจก่อนจะลงทุนในหุ้นที่สนใจ และนำไปใช้สำหรับการเรียนรู้สำหรับการลงทุนในตลาดหุ้นสำหรับผู้ที่ศึกษาเกี่ยวกับการลงทุนในหุ้นได้

5.2 วิจารณ์ผลการทดลอง

จากการทดลองได้พบว่าถ้าหากอยากให้ระบบมีความแม่นยำและมีประสิทธิภาพในการตัดสินใจลงทุนในหุ้นมากขึ้น ควรที่จะปรับและเลือกโมเดลที่นำมาใช้ให้ดีขึ้น เช่น หากโมเดลที่ทำนายราคาหุ้นในอนาคตมีการทำนายราคาหุ้นที่ผิดพลาด ก็จะมีผลต่อไปที่โมเดลสำหรับทำนายพฤติกรรมของการลงทุนได้ เพราะค่าที่ทำนายราคาหุ้นออกมาจะเป็นข้อมูลขาเข้าให้กับ Environment ของโมเดลสำหรับทำนายพฤติกรรม ดังนั้นจึงมีทางปรับปรุงได้หลายทาง เช่น การทำให้โมเดลทำนายราคาหุ้นในอนาคตมีความผิดพลาดน้อยที่สุดเพราะจะเห็นได้ว่าโมเดลในการทดลองยังไม่ได้ค่าที่ตรงหรือเท่ากับราคาหุ้นในอนาคตจริง ๆ แต่ได้ค่าที่ใกล้เคียงกับแนวโน้มของราคาในอนาคตเท่านั้น หรือจะปรับการคิด State จาก Environment ในโมเดลเรียนรู้แบบเสริมกำลังให้มีการคิด State ที่ดีขึ้น และแสดงถึงสถานะของการลงทุนในหุ้น ณ ตอนนั้นได้ดียิ่งขึ้นกว่าเดิม หรืออาจจะใช้โมเดลเรียนรู้แบบเสริมกำลังชนิดอื่นที่มีประสิทธิภาพมากขึ้น ทั้งนี้ในการนำระบบในการทดลองนี้ไปใช้ ควรจะออกแบบให้เหมาะสมกับหุ้นตัวที่เราสนใจ เช่นในการทดลองสนใจหุ้นของ AAPL ถ้าหากมีการใช้กับหุ้นตัวอื่นก็

ควรเปลี่ยนข้อมูล และ เปลี่ยนการออกแบบโมเดลให้เหมาะสมด้วย นอกจากนั้นควรที่จะศึกษาการลงทุนให้ดีก่อนนำระบบนี้ไปใช้ และนำโมเดลไปทดสอบในระบบจำลองการเทรดจำลองให้ระบบมีความแม่นยำและมีประสิทธิภาพก่อนนำไปใช้จริงเพื่อความปลอดภัยของทรัพย์สิน

เอกสารอ้างอิง

[1] Chukamphaeng, N. (2020, August 20). มาทำความรู้จักกับ Reinforcement Learning แบบเบาๆกันเถอะ. Medium. https://medium.com/@nutorbitx/มาทำความรู้จักกับ-reinforcement-learning-แบบเบาๆกันเถอะ-d36e71237b8

[2] S. (2019, March 4). shivamakhauri04/TradingBot. GitHub. https://github.com/shivamakhauri04/TradingBot

[3] Atienza, R. (2020). Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition. Packt Publishing.

[4] magnetolabs. (2021, March 16). สร้าง AI ให้รู้ลึก รู้จริง! ทำความรู้จักว่า Deep Learning คืออะไร. PTT ExpresSo. https://blog.pttexpresso.com/get-to-know-deep-learning/

[5] Linn Software, Inc. Normalized Price. https://www.linnsoft.com/techind/normalized-price

[6] Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM--a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv preprint arXiv:1909.09586.

[7] Phi, M. (2020, June 28). Illustrated Guide to LSTM's and GRU's: A step by step explanation. Medium. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[8] Brownlee, J. (2020, August 28). How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting. Machine Learning Mastery. https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/

[9] Eckhardt, K. (2018, November 29). Choosing the right Hyperparameters for a simple LSTM using Keras. Medium. https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046

[10] Loukas, S. (2020, July 31). Time-Series Forecasting: Predicting Stock Prices Using An LSTM Model. Medium. https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f