Forecasting Ethereum (ETH-USD)

Price using LSTM

จัดทำโดย

1. นายปริญญา อบอุ่น รหัสนิสิต 6610502145
2. นายจารุกิตติ์ ผลวัฒนานุวงศ์ รหัสนิสิต 6610505306

เสนอ

ผศ.ดร.ภารุจ รัตนวรพันธุ์

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา

การเรียนรู้เชิงลึก (Deep Learning) 01204466-65

ภาคเรียนที่ 1 ปีการศึกษา 2568

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตบางเขน

สารบัญ

1. หัวข้อนี้น่าสนใจอย่างไร ทำไมถึงเลือกหัวข้อนี้

ความน่าสนใจเนื่องจากตลาดสกุลเงินดิจิทัล (Cryptocurrency) โดยเฉพาะ Ethereum (ETH) ซึ่งมีมูลค่าตลาดมาก มีความผันผวนสูงและได้รับความสนใจจากนักลงทุนทั่วโลก การพยากรณ์ราคาของสินทรัพย์ที่มีความผันผวนสูงเช่นนี้ถือเป็นปัญหาด้านอนุกรมเวลา (Time-Series) ทางการเงินที่มีความท้าทาย

1. **ความท้าทายเชิงเทคนิค:** ราคาของ ETH มีความสัมพันธ์ที่ซับซ้อน (Non-linear) กับข้อมูลในอดีตและปัจจัยภายนอกหลายอย่าง ซึ่งท้าทายความสามารถของแบบจำลองในการจับรูปแบบเหล่านี้
2. **การประยุกต์ใช้จริง:** แบบจำลองที่สามารถพยากรณ์ทิศทางหรือราคาของ ETH ได้อย่างแม่นยำ สามารถนำไปประยุกต์ใช้ในการวางแผนการลงทุน การบริหารความเสี่ยง หรือการสร้างกลยุทธ์การซื้อขายอัตโนมัติได้
3. **ความเหมาะสม Deep Learning:** ปัญหาแสดงให้เห็นถึงประสิทธิภาพของโมเดล Deep Learning ประเภท Recurrent Neural Networks (RNNs) และ LSTM ในการจัดการกับข้อมูลที่มีลำดับและความสัมพันธ์ระยะยาว (Long-term dependencies)

2. ทำไมหัวข้อนี้จึงต้องใช้ Deep Learning

- ในการจัดการกับธรรมชาติของข้อมูลอนุกรมเวลา การเรียนรู้ความจำระยะยาว และการวิเคราะห์ความสัมพันธ์ที่ซับซ้อนหลายมิติ ทำให้ LSTM เป็นเครื่องมือที่มีประสิทธิภาพสูงและเหมาะสมสำหรับการพยายามแก้ปัญหาที่มีความท้าทายสูงอย่างการพยากรณ์ราคา Cryptocurrency

ข้อเด่น:

- การจัดการข้อมูลอนุกรมเวลา: LSTM ถูกออกแบบมาโดยเฉพาะเพื่อจัดการกับข้อมูลที่มีลำดับ (Sequential Data)
- การจดจำระยะยาว (Long-Term Dependencies): หัวใจของ LSTM คือ Cell State และระบบ Gate(Input, Forget, Output) ที่ช่วยให้โมเดลสามารถเรียนรู้ได้ว่าควรจดจำข้อมูลใดจากใน

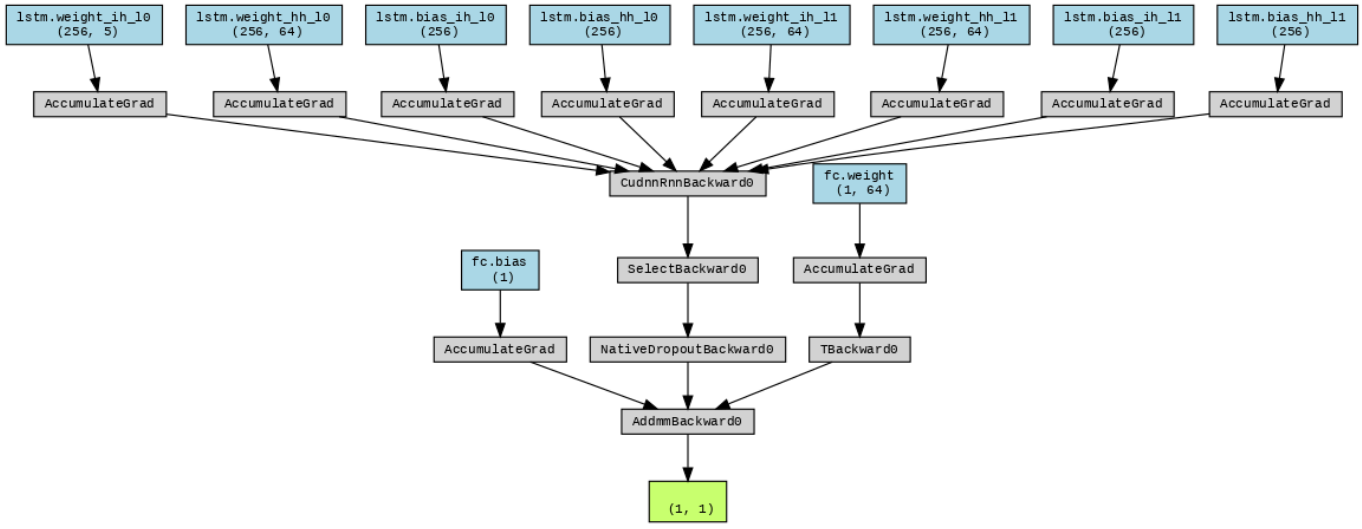
อดีต (เช่น แนวโน้มราคาเมื่อ 60 วันที่แล้ว) และควรลืมข้อมูลใด (เช่น ความผันผวนเล็กน้อยที่ไม่สำคัญ) ซึ่งจำเป็นอย่างยิ่งในการวิเคราะห์ตลาดการเงิน

- ความสัมพันธ์แบบ Non-linear: สามารถเรียนรู้ความสัมพันธ์ที่ซับซ้อนและไม่เป็นเส้นตรงระหว่างตัวแปรต่างๆ (OHLCV) ได้อย่างมีประสิทธิภาพ
- Multivariate Analysis: สามารถรับ Input ได้หลายมิติ (5 features) เพื่อพยากรณ์ Output (1 feature)

ข้อด้อย:

- Black Box: ตีความได้ยากว่าทำไมโมเดลจึงตัดสินใจพยากรณ์ค่านั้นๆ ออกมา
- ทรัพยากรในการ Train: ใช้ทรัพยากรในการประมวลผลสูง (ต้องการ GPU) และใช้เวลา Train นานกว่า เมื่อเทียบกับโมเดล ARIMA หรือ Random Forest
- การปรับจูน: มี Hyperparameters จำนวนมากที่ต้องปรับจูน (เช่น จำนวนชั้น, จำนวน hidden units, learning rate) ซึ่งต้องใช้เวลาและความเข้าใจในโมเดลเพื่อปรับจูน

3. สถาปัตยกรรม LSTM model



ภาพที่ 1: Model LSTM Architecture

โครงสร้างโดยรวม :

1. **Input Layer:**
    - รับข้อมูลในรูปแบบ [Batch Size, Sequence Length, Input Dimension]

- Sequence Length = 60 (ใช้ข้อมูล 60 วันย้อนหลังในการพยากรณ์)
- Input Dimension = 5 (จำนวน Features: 'Open', 'High', 'Low', 'Close', 'Volume')

2. **Stacked LSTM Layers:**
   - ใช้ nn.LSTM แบบ 2 ชั้น (Stacked LSTM) ซึ่งหมายความว่า Output จาก LSTM ชั้นแรกจะถูกป้อนเป็น Input ให้กับ LSTM ชั้นที่สอง
   - n_layers = 2
   - hidden_dimension = 64: แต่ละเซลล์ LSTM ในแต่ละชั้นมี Hidden State และ Cell State ขนาด 64
   - dropout = 0.3: มีการใช้ Dropout 30% ระหว่างชั้น LSTM ทั้งสอง (หาก n_layers > 1) เพื่อป้องกัน Overfitting
   - batch_first = True: ระบุว่ามิติแรกของ Input คือ Batch Size
3. **การเชื่อมต่อภายใน LSTM Cell:**
   - ในแต่ละ LSTM (โหนด) จะมีการคำนวณที่ซับซ้อนโดยใช้ Gate 3 ตัว คือ Input Gate, Forget Gate, และ Output Gate
   - Activation Functions: ภายในเซลล์ LSTM จะใช้ฟังก์ชัน Sigmoid (สำหรับ Gate ต่างๆ เพื่อให้ค่าอยู่ระหว่าง 0-1) และฟังก์ชัน Tanh (สำหรับปรับค่า Cell State และ Hidden State ให้อยู่ระหว่าง -1 ถึง 1)
4. **Dropout Layer:**
   - โมเดลนี้ใช้ nn.Dropout ที่มี p = 0.3
   - Dropout นี้จะถูกใช้กับ Hidden State สุดท้ายของชั้น LSTM สุดท้าย (hidden[-1]) ก่อนที่จะส่งเข้าไปยัง Fully Connected Layer
5. **Output (Fully Connected) Layer:**
   - ใช้ nn.Linear
   - in_features = 64: รับ Input จาก Hidden State ของ LSTM
   - out_features = 1: พยากรณ์ค่าเดียว คือ ราคา 'Close'
   - Activation Function (Final): ไม่มี (หรือเรียกว่า Linear Activation) ซึ่งเป็นมาตรฐานสำหรับงาน Regression ที่ต้องการพยากรณ์ค่าตัวเลขที่ต่อเนื่อง (ไม่ได้จำกัดช่วง 0-1 หรือ 0-inf)

4. อธิบายโค้ด PyTorch

โค้ดใน Jupyter Notebook นี้ใช้ PyTorch ในการสร้าง Train และ Evaluate โมเดล สามารถแบ่งการทำงานได้ดังนี้:

ส่วนที่ 1: การนำเข้า Libraries และตั้งค่า

- นำเข้า Libraries ที่จำเป็น เช่น `torch`, `torch.nn`, `yfinance`, `pandas`, `sklearn.preprocessing`
- ตั้งค่า `device` ให้เป็น 'cuda' (GPU) หากมีการเชื่อมต่อ GPU ไว้ เพื่อเร่งความเร็วในการ Train

ส่วนที่ 2: การดึงและจัดรูปแบบข้อมูล

- ใช้ `yfinance.download()` เพื่อดึงข้อมูล "ETH-USD" แบบรายวัน ("1d") ย้อนหลังสูงสุด ("max")
- ทำการ Flatten คอลัมน์ที่ซ้อนกัน (Multi-level columns) ที่ได้จาก `yfinance` ให้เป็น DataFrame ที่อ่านง่าย (Date, Open, High, Low, Close, Volume)

ส่วนที่ 3: การเตรียมข้อมูล

- **เลือก Features และ Target:** กำหนด Features (X) เป็น 5 คอลัมน์ (OHLCV) และ Target (y) เป็น 'Close'
- **การ Scaling:**
    - ใช้ `MinMaxScaler` จาก `sklearn` เพื่อปรับสเกลข้อมูลทั้งหมดให้อยู่ในช่วง 0 ถึง 1
    - มีการป้องกัน Data Leakage โดยการ `fit` ตัว Scaler (ทั้ง `scaler_X` และ `scaler_y`) ด้วยข้อมูลส่วน Training เท่านั้น (`data[:fit_end]`)
- **การสร้าง Sequences:**

  - ใช้ Sliding Window approach โดยวนลูปสร้างคู่ข้อมูล (X, y):
    - X[i] = ข้อมูล 60 วันติดต่อกัน (วันที่ i ถึง i+59) → shape (60, 5)
    - y[i] = ราคา Close ของวันที่ i+60 → shape (1,)

- **การแบ่งข้อมูล:**
    - แบ่งข้อมูลทั้งหมดเป็น 80% (Train) และ 20% (Test) ตามลำดับเวลา
- **การแปลงเป็น Tensors:**
    - ใช้ `torch.tensor()` แปลง `numpy arrays` ให้เป็น PyTorch Tensors และส่งไปที่ `device` (GPU)

ส่วนที่ 4: การสร้างโมเดล

- สร้างคลาส `LSTM_model` ที่สืบทอดจาก `nn.Module`
- `__init__`: กำหนดสถาปัตยกรรมตามที่อธิบายในข้อ 4 (nn.LSTM, nn.Linear, nn.Dropout)
- `forward`: กำหนดขั้นตอนการไหลของข้อมูล (Forward Pass) โดยดึง Hidden State สุดท้าย (`hidden[-1]`) มาใช้ในการพยากรณ์

ส่วนที่ 5: การตั้งค่าการ Train

- **Hyperparameters:**
    - `input_dimension=5`, `hidden_dimension=64`, `n_layers=2`, `max_epoch=750`, `lr=0.00001`, `batch_size=256`
- **สร้างชุดข้อมูลสำหรับ Train/Validate:**
    - มีการแบ่งข้อมูล `X_train` (80% แรกของทั้งหมด) ออกเป็นชุด Train ใหม่ (`X_tr`, `y_tr` - 80% ของ 80%) และชุด Validation (`X_val`, `y_val` - 20% ของ 80%)
    - ชุด `X_test` (20% สุดท้ายของทั้งหมด) จะถูกเก็บไว้ใช้ประเมินผลในตอนท้ายเท่านั้น
- **DataLoader:** สร้าง `DataLoader` สำหรับชุด Train และ Validation (ตั้งค่า `shuffle=False` เพราะเป็นข้อมูล Time-Series)
- **Loss & Optimizer:** กำหนด `torch.nn.MSELoss` เป็น Loss Function และ `torch.optim.Adam` เป็น Optimizer

ส่วนที่ 6: การ Train โมเดล

- วนลูปตามจำนวน `max_epoch` (750)
- **Training Phase:**
    - ตั้งค่า `model.train()`
    - คำนวณ `y_pred`, คำนวณ `loss_value`

    - `otm.zero_grad()`, `loss_value.backward()`, `otm.step()` เพื่อทำการ Backpropagation
- **Validation Phase:**
    - ตั้งค่า `model.eval()` และใช้ `torch.no_grad()`
    - คำนวณ `val_loss` จากชุดข้อมูล Validation
- **Save Best Model:** เปรียบเทียบ `val_loss` กับ `best_val_loss` และบันทึกโมเดลที่ดีที่สุดลงไฟล์ `best_lstm_model.pth`
- สุดท้าย โหลดโมเดลที่ดีที่สุดกลับมาใช้งาน

ส่วนที่ 7: การประเมินผล

- โหลดโมเดลที่ดีที่สุด (`best_lstm_model.pth`)
- ตั้งค่า `model.eval()` และใช้ `torch.no_grad()`
- ทำการพยากรณ์ข้อมูลด้วย `X_test`
- **Inverse Transform:** ใช้ `scaler_y.inverse_transform()` แปลงค่าที่พยากรณ์ได้ (ซึ่งอยู่ในสเกล 0-1) กลับไปเป็นค่าเงินดอลลาร์ (USD) จริง
- คำนวณค่า Metrics (RMSE, MAE, MAPE) และแสดงผลลัพธ์ รวมถึงสร้างกราฟเปรียบเทียบ

5. หลักอธิบายวิธีการ Train โมเดล LSTM

```
# Training Loop
best_val_loss = float('inf')
train_losses = []
val_losses = []

for epoch in range(max_epoch):
    # ========== Training ==========
    model.train()
    train_loss = 0

    for X_batch, y_batch in train_loader:
        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)

        # Forward pass
        y_pred = model(X_batch)
```

```
            loss_value = loss_fn(y_pred, y_batch)

            # Backward pass
            otm.zero_grad()
            loss_value.backward()
            otm.step()

            train_loss += loss_value.item()

        train_loss /= len(train_loader)
        train_losses.append(train_loss)

        # ========== Validation ==========
        model.eval()
        val_loss = 0

        with torch.no_grad():
            for X_batch, y_batch in val_loader:
                X_batch = X_batch.to(device)
                y_batch = y_batch.to(device)
                y_pred = model(X_batch)
                loss_value = loss_fn(y_pred, y_batch)
                val_loss += loss_value.item()

        val_loss /= len(val_loader)
        val_losses.append(val_loss)

        # Save best model
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), 'best_lstm_model.pth')

        # Print progress
        if (epoch + 1) % 5 == 0:
            print(f'Epoch [{epoch+1}/{max_epoch}] - Train Loss:
{train_loss:.6f}, Val Loss: {val_loss:.6f}')

print(f'\nTraining completed! Best Val Loss: {best_val_loss:.6f}')

# Load best model
```

```
model.load_state_dict(torch.load('best_lstm_model.pth'))
```

คำอธิบายโค้ด

1. การวนลูป Training (Training Loop):
   - โมเดลทำการเทรนทั้งหมด 750 epochs
   - แต่ละ epoch แบ่งเป็น 2 ขั้นตอน: Training Phase และ Validation Phase
2. Training Phase:
   - ตั้งโมเดลเป็นโหมด training (model.train())
   - วนลูปผ่านข้อมูลใน train_loader ทีละ batch (batch size = 256)
   - ในแต่ละ batch:
     - Forward Pass: ป้อนข้อมูล X_batch เข้าโมเดล → ได้ `y_pred`
     - คำนวณ Loss: เปรียบเทียบ `y_pred` กับ `y_batch` ด้วย MSE Loss
     - Backward Pass:
       - otm.zero_grad() ล้าง gradients เก่า
       - loss_value.backward() คำนวณ gradients
       - otm.step() อัปเดต weights ของโมเดล
   - คำนวณ Train Loss เฉลี่ย ของ epoch นั้น
3. Validation Phase:
   - ตั้งโมเดลเป็นโหมด evaluation (model.eval())
   - ใช้ torch.no_grad() เพื่อไม่คำนวณ gradients (ประหยัด memory)
   - วนลูปผ่านข้อมูลใน val_loader และคำนวณ Validation Loss
   - ไม่มีการ Backward Pass เพราะเป็นการประเมินผลเท่านั้น
4. Model Checkpointing (Early Stopping):
   - เปรียบเทียบ val_loss ของ epoch ปัจจุบันกับ best_val_loss
   - ถ้า val_loss ต่ำกว่า → บันทึกโมเดลลงไฟล์ best_lstm_model.pth
5. การติดตามผล:
   - พิมพ์ค่า Train Loss และ Val Loss ทุกๆ 5 epochs
   - เก็บค่า loss ทุก epoch ไว้ใน list เพื่อพล็อตกราฟ

6. การโหลดโมเดลที่ดีที่สุด:
   - หลังจากเทรนครบ 750 epochs => โหลดโมเดลที่มี Val Loss ต่ำที่สุดกลับมาใช้งาน
   - โมเดลนี้จะถูกใช้ในการทดสอบกับ Test Set

6. ลิงค์ Github

1. https://github.com/parinya-ao/project_deep_learning
2. https://github.com/Jarukit-Jack/project_deep_learningvv

7. Data Engineering จัดการกับ Dataset

Dataset และแหล่งที่มา:

- ข้อมูล: ข้อมูลอนุกรมเวลาราคาของ Ethereum (ETH-USD) แบบรายวัน
- แหล่งที่มา: Yahoo Finance(secondary source)
- วิธีการดึงข้อมูล: ใช้ Library `yfinance` ของ Python
- Features (Input): 5 ตัวแปร ได้แก่ 'Open', 'High', 'Low', 'Close', 'Volume'
- Target (Output): 1 ตัวแปร ได้แก่ 'Close' (ราคาปิด)
- Sequence Length: 60 วัน (ใช้ข้อมูล 60 วันย้อนหลังในการพยากรณ์ราคาปิดของวันถัดไป)

ข้อมูลเพิ่มเติม:

- ช่วงเวลาของข้อมูล: [ระบุวันที่เริ่มต้น - วันที่สิ้นสุด] เช่น "2017-11-09 ถึง 2025-10-25"
- จำนวนข้อมูลทั้งหมด: 2,897 วัน (ประมาณ 7.9 ปี)

การแบ่งข้อมูล (Data Split): ข้อมูลทั้งหมด 2,897 วัน (ณ วันที่รันโค้ด) ถูกแบ่งตามลำดับเวลาเพื่อป้องกัน Data Leakage ดังนี้:

1. **Training Set (80%):** 2,269 วันแรก (ใช้สำหรับ `fit` Scaler และ Train/Validate โมเดล)
2. **Test Set (20%):** 568 วันสุดท้าย (ใช้สำหรับประเมินผลโมเดลขั้นสุดท้าย)

จากนั้น ชุด Training Set (2,269 วัน) ถูกแบ่งย่อยอีกเป็น:

- **Training (ย่อย):** 1,815 วัน (80% ของชุด Train)
- **Validation:** 454 วัน (20% ของชุด Train)

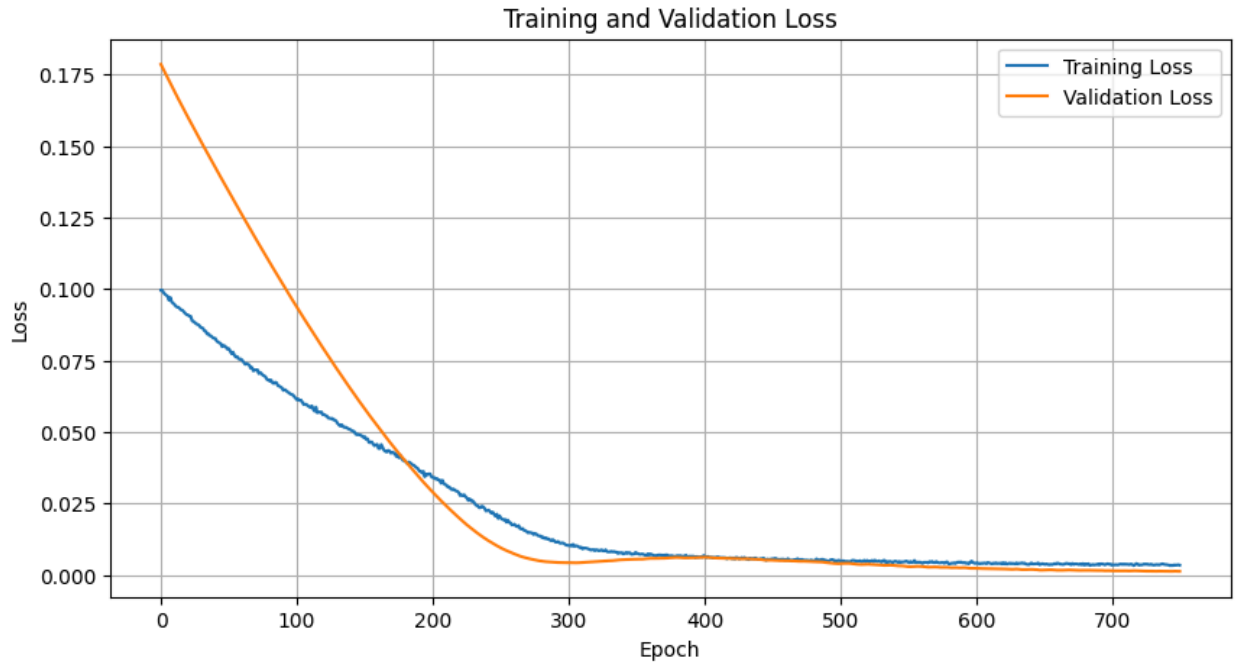
กระบวนการ Train:

- **Loss Function:** ใช้ **Mean Squared Error (MSE)** (`torch.nn.MSELoss`) ซึ่งเป็น Loss Function มาตรฐานสำหรับงาน Regression โดยจะคำนวณค่าเฉลี่ยของความคลาดเคลื่อนกำลังสองระหว่างค่าจริงและค่าที่พยากรณ์ได้
- **Optimizer:** ใช้ **Adam** (`torch.optim.Adam`) ซึ่งเป็น Optimizer ที่ปรับ Learning Rate อัตโนมัติและมีประสิทธิภาพสูง ตั้งค่า Learning Rate เริ่มต้นที่ `0.00001`
- **Epochs:** Train ทั้งหมด `750` epochs
- **Batch Size:** `256`
- **การป้องกัน Overfitting:** ใช้วิธี **Early Stopping (แบบ Implicit)** โดยการติดตาม Validation Loss ในทุก Epoch และบันทึกเฉพาะโมเดล (Checkpoint) ที่มี Validation Loss ต่ำที่สุด สุดท้ายจึงโหลดโมเดลที่ดีที่สุดนั้นมาใช้งาน

8. อธิบายการประเมิน (Evaluate) Model

การประเมินโมเดลแบ่งเป็น 2 ส่วน คือ การประเมินระหว่างการ Train (Validation Loss) และ การประเมินขั้นสุดท้าย (Test Metrics)

1. ค่า Loss จากการ Train:



ภาพที่ 2: Training and Validation Loss

- **Training Loss (เส้นสีน้ำเงิน):** ลดลงอย่างต่อเนื่องตลอด 750 epochs ซึ่งหมายความว่าโมเดลสามารถเรียนรู้รูปแบบจากข้อมูลที่ใช้ Train ได้ดี
- **Validation Loss (เส้นสีส้ม):** ลดลงในทิศทางเดียวกับ Training Loss และเริ่มคงที่ (Flatline) ในช่วงประมาณ 600-750 epochs
- กราฟนี้แสดงให้เห็นว่าโมเดลเรียนรู้ได้ดี

2. Metric ที่ใช้ประเมิน (บน Test Set): โมเดลที่ดีที่สุด (จาก Validation) ถูกนำมาทดสอบกับ Test Set (568 วัน) ซึ่งเป็นข้อมูลที่โมเดลไม่เคยเห็นมาก่อน ผลลัพธ์ มีดังนี้:

- ปัญหา **Regression Metric** ที่เหมาะสม: RMSE, MAE, MAPE, R^2 เพิ่ม R^2 Score (Coefficient of Determination)
- **RMSE** (Root Mean Squared Error): 286.01
    - Metric นี้จะลงโทษความผิดพลาด (การทายผิดมากๆ) มากกว่า MAE
- **MAE** (Mean Absolute Error): 217.23
    - โดยเฉลี่ยแล้ว โมเดลพยากรณ์ราคาผิดพลาดไปประมาณ 217.23 USD
- **MAPE** (Mean Absolute Percentage Error): 7.40%
    - โดยเฉลี่ยแล้ว โมเดลพยากรณ์ราคาคลาดเคลื่อนไป 7.40% จากราคาจริง

2. Metric ที่ใช้ประเมิน (บน Test Set): โมเดลที่ดีที่สุด (จาก Validation) ถูกนำมาทดสอบกับ Test Set (568 วัน) ซึ่งเป็นข้อมูลที่โมเดลไม่เคยเห็นมาก่อน ผลลัพธ์ มีดังนี้

2.1 Metrics สำหรับวัดความคลาดเคลื่อน (Error Metrics):

1. **RMSE (Root Mean Squared Error): 286.01 USD**
   การตีความ*:* โดยเฉลี่ยแล้ว การพยากรณ์คลาดเคลื่อนประมาณ 286 USD(ตลอดระยะเวลาการเทรน ข้อมูลในเวลา 7.9 ปี)ในบริบทที่ราคา ETH มักอยู่ระหว่าง 2,000–4,000 USD ความผิดพลาด ~286 USD ประมาณ 7–14% ของราคา
2. **MAE (Mean Absolute Error): 217.23 USD**

   การตีความ*:* โดยเฉลี่ยแล้ว โมเดลพยากรณ์ผิดพลาด 217.23 USD สังเกตว่า *MAE vs RMSE:* RMSE (286) > MAE (217) แสดงว่ามี outliers บางวัน (ที่พยากรณ์ผิดมาก) ซึ่ง RMSE จะให้น้ำหนักมากกว่า

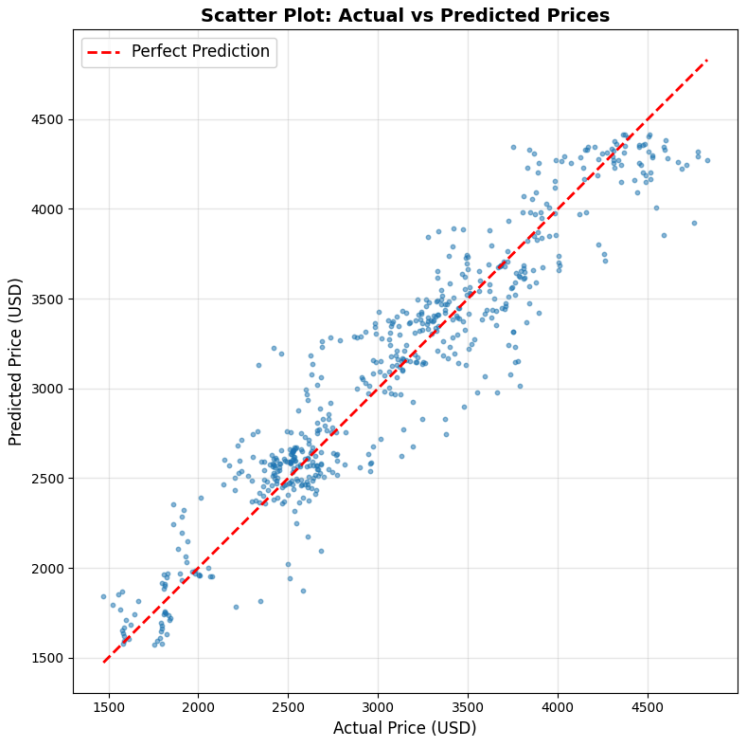
3. **MedAE (Median Absolute Error): 165.71 USD**

การตีความ*:* 50% ของการพยากรณ์มีความผิดพลาดน้อยกว่า 165.71 USD บอกว่า MedAE (165.71) < MAE (217.23) มี outliers บางวันที่ดึงค่า MAE ให้สูงขึ้น

4. **MAPE (Mean Absolute Percentage Error): 7.40%**

การประเมิน: MAPE < 10% ถือว่าดีมากสำหรับการพยากรณ์ราคา Cryptocurrency ที่มีความผันผวนสูง

2.2 Metrics สำหรับวัดความสัมพันธ์ (Goodness-of-Fit Metrics)



ภาพที่ 3: Predicted price vs Actual price

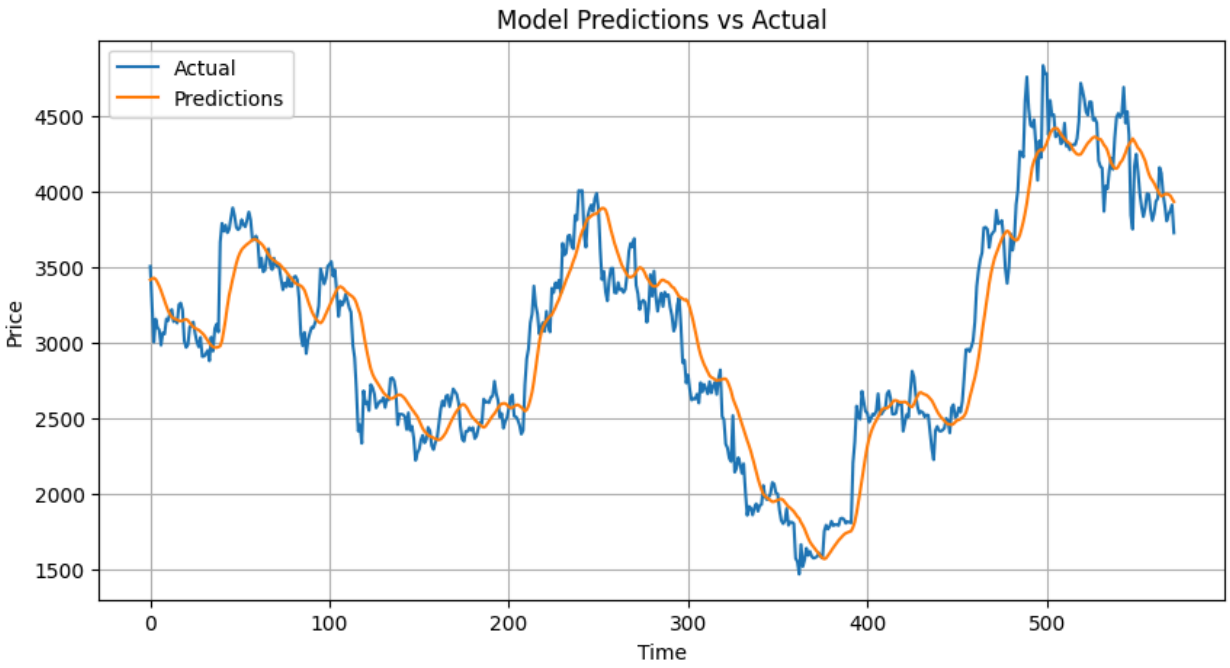
1. R^2 (R-squared): 0.8790

โมเดลอธิบายความแปรปรวนของราคา ETH ได้ 87.90% โมเดลจับรูปแบบได้ดีแต่สามารถพัฒนาต่อเพราะมี 12.1% ของความแปรปรวนที่โมเดลไม่สามารถอธิบายได้

2. Correlation (Pearson's r): 0.938

การตีความ: ค่าที่พยากรณ์และค่าจริงมีความสัมพันธ์กันสูงมาก (93.85%) แสดงว่าโมเดลสามารถติดตาม "ทิศทาง" และ "ขนาด" ของการเคลื่อนไหวราคาได้ดี

3. การประเมินเชิงคุณภาพ (Visual Evaluation):



ภาพที่ 4: Model Prediction vs Actual Prediction

- **กราฟ Full Test Set และ Zoom-in:** แสดงให้เห็นว่าเส้นสีส้ม (Predicted) สามารถเคลื่อนไหวตาม "ทิศทาง" และ "แนวโน้ม" ทั่วไปของเส้นสีน้ำเงิน (Actual) ได้ค่อนข้างดี
- **ข้อจำกัด:** โมเดลมีแนวโน้มตามหลังราคาจริงเล็กน้อย และไม่สามารถพยากรณ์ "จุดสูงสุด/ต่ำสุด" ที่มีความผันผวนรุนแรง (Spikes) ได้แม่นยำ 100% ซึ่งเป็นลักษณะทั่วไปของโมเดล Time-Series ที่เรียนรู้จากข้อมูลในอดีต
- **Scatter Plot:** จุดต่างๆ กระจายตัวอยู่ใกล้เคียงกับเส้นสีแดง (Perfect Prediction) แสดงว่าค่าที่พยากรณ์ได้มีความสัมพันธ์ที่ดีกับค่าจริง
- **Error Distribution:** กราฟ Error ส่วนใหญ่กระจุกตัวอยู่ที่ 0 แต่มี "หางยาว" (Fat Tails) ซึ่งยืนยันว่าโมเดลทำงานได้ดีในสภาวะปกติ แต่ยังคงมีความคลาดเคลื่อนสูงในวันที่ตลาดผันผวนรุนแรง

9. อธิบายบทความอ้างอิงและงานที่เกี่ยวข้อง

1. โค้ดอ้างอิง (Reference Code):

- สถาปัตยกรรมของโมเดล `LSTM_model` ในโปรเจกต์นี้ ได้รับแรงบันดาลใจและอ้างอิงจาก Repository บน Github:
  - Vahdat, N. (2020). *ETH_Price_Prediction*. Github. https://github.com/NimaVahdat/ETH_Price_Prediction/blob/main/Networks/networks.py

2. บทความวิชาการที่เกี่ยวข้อง (Related Foundational Work):

- งานวิจัยพื้นฐานที่นำเสนอแนวคิดของ Long Short-Term Memory (LSTM) ซึ่งเป็นหัวใจหลักของโปรเจกต์นี้:
  - Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

---

10. ส่วนแบ่งงาน

1. นายปริญญา อบอุ่น (50%):
   - การ load และ preprocess dataset (15%)
   - การออกแบบและสร้าง LSTM model architecture (20%)
   - การประเมินผลและวิเคราะห์ metrics (10%)
   - การสร้างกราฟ visualization (5%)
2. นายจารุกิตติ์ ผลวัฒนานุวงศ์ (50%):
   - การ tuning hyperparameters (15%)
   - การเขียนโค้ด training loop และ validation (20%)
   - การเขียนรายงานและจัดรูปแบบเอกสาร (10%)
   - การทดลองและเปรียบเทียบผลลัพธ์ (5%)