ฉันท์สินี เมืองหนู 63070501221

INC491 Homework 4

Stock Trend Prediction using Recurrent Neural Network (RNN)

Description Task 1:

- Use RNN model to predict the trend of a stock price on the next day.
- Use data of 10 previous days to predict the trend of the price on the 11th day. (Is it up/ down/neutral to the close price of the 10th day?)
- Dataset: PTT
  - PTT dataset is a real stock price of "PTT" from year 2011 to 2019.
  - PTT dataset consists of 6 columns → Date and 5 features (OHLCV)
- To do lists:
  - OHLCV will be used as 5 features of the input and use 10 timesteps → input shape is [None, 10, 5].
  - Must normalize both the prices (OHLC) and the volume (V).
  - Must generate a label of 3 classes and drop the rows that has no label → Up, Down, Neutral
  - Implement an LSTM model and train it.
  - Divide the data into 2 parts;
    - Year 2011-2016 for training set
    - Year 2017-2019 for testing set
  - Must use a sliding window to arrange the data in batch.
  - Apply a strategy "hold-one-day when the trend is up" to the test set (2017-2019).
    - Calculate the total gain using this strategy and compare it with the buy-and-hold strategy.
    - The buy-and-hold strategy buys stock on 1/4/2017 at 37.4 and sell the stock on 12/30/2019 at 44 Baht. Thus, it accrues a profit of 6.6 baht or 17.65% over 3 years.
  - What to submit
    - Diagram of network and parameter used.
    - Printed source code
    - Graph result
    - The profit when apply "hold-one-day when the trend is up" strategy.

โหลดข้อมูลและตรวจสอบข้อมูลเบื้องต้น

```
df = pd.read_csv('ptt.csv', parse_dates = ['Date'])
df
```

	Date	Open	High	Low	Close	Volume
0	2011-01-04	32.50	33.30	32.30	33.00	105964752
1	2011-01-05	33.00	33.20	32.80	33.20	60864168
2	2011-01-06	33.30	33.40	32.90	33.20	39651640
3	2011-01-07	33.10	33.10	32.20	32.20	55886640
4	2011-01-10	32.10	32.20	31.60	31.80	95325912
...	...	...	...	...	...	...
2194	2019-12-24	45.00	45.00	44.00	44.25	32913200
2195	2019-12-25	44.25	44.25	43.75	44.25	11687500
2196	2019-12-26	44.25	44.50	44.25	44.50	11117700
2197	2019-12-27	44.50	44.75	43.50	44.25	55385800
2198	2019-12-30	44.25	44.75	44.00	44.00	33688500

2199 rows × 6 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2199 entries, 0 to 2198
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Date    2199 non-null   datetime64[ns]
 1   Open    2199 non-null   float64       
 2   High    2199 non-null   float64       
 3   Low     2199 non-null   float64       
 4   Close   2199 non-null   float64       
 5   Volume  2199 non-null   int64         
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 103.2 KB
```

ข้อมูลที่ได้จาก df.info() จะเห็นได้ว่า column Date มีชนิดข้อมูลเป็น datetime แต่เนื่องจากข้อมูลสำหรับที่จะนำไปเทรน RNN/LSTM model โดยทั่วไปจะสนใจส่วนที่เป็น value (ไม่เอาข้อมูลเวลา) จึงต้องทำการ set คอลัมน์ Date เป็น index

```
df.set_index('Date', inplace=True)
```

```
df
```

Date	Open	High	Low	Close	Volume
2011-01-04	32.50	33.30	32.30	33.00	105964752
2011-01-05	33.00	33.20	32.80	33.20	60864168
2011-01-06	33.30	33.40	32.90	33.20	39651640
2011-01-07	33.10	33.10	32.20	32.20	55886640
2011-01-10	32.10	32.20	31.60	31.80	95325912
...	...	...	...	...	...
2019-12-24	45.00	45.00	44.00	44.25	32913200
2019-12-25	44.25	44.25	43.75	44.25	11687500
2019-12-26	44.25	44.50	44.25	44.50	11117700
2019-12-27	44.50	44.75	43.50	44.25	55385800
2019-12-30	44.25	44.75	44.00	44.00	33688500

2199 rows × 5 columns

จากการตรวจสอบข้อมูลก็จะพบว่าข้อมูลประกอบข้อมูลราคาหุ้นและข้อมูล volume ซึ่งมีทั้งหมด 5 columns คือ

- Open: ราคาของหุ้น ณ เวลาเริ่มเปิดทำการซื้อขาย (เวลาเปิดตลาด)
- High/Low: ราคาของหุ้นสูงสุด ต่ำสุด ที่ซื้อขายกันในวันนั้น
- Close: ราคาของหุ้น ณ เวลาปิดทำการซื้อขาย (เวลาปิดตลาด)
- Volume: ปริมาณหรือจำนวนหุ้นที่มีการซื้อขายในวันนั้น

```
df.describe()
```

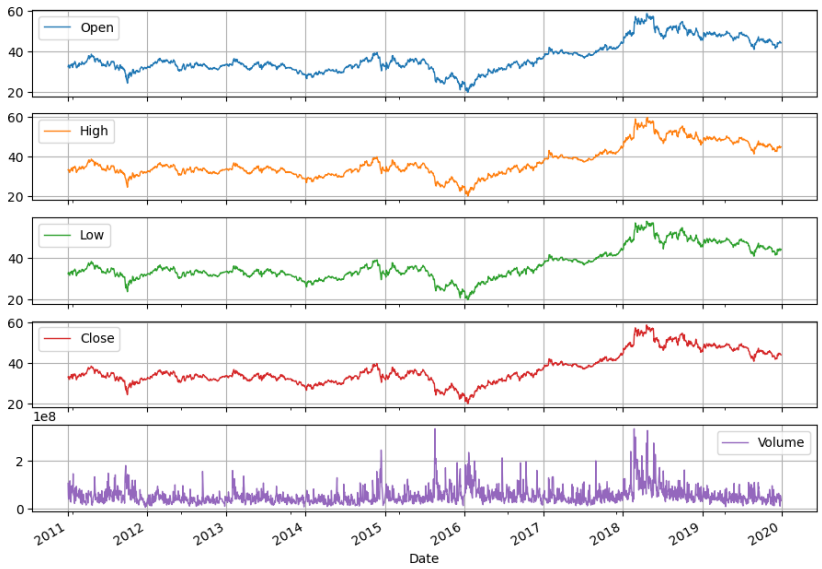
	Open	High	Low	Close	Volume
count	2199.000000	2199.000000	2199.000000	2199.000000	2.199000e+03
mean	36.756276	37.081219	36.405093	36.730696	5.546045e+07
std	7.651295	7.706596	7.596040	7.645925	3.650298e+07
min	19.900000	20.200000	19.700000	19.800000	6.309250e+06
25%	31.800000	32.100000	31.600000	31.800000	3.302700e+07
50%	34.300000	34.600000	34.000000	34.200000	4.593573e+07
75%	41.200000	41.600000	41.000000	41.200000	6.538856e+07
max	58.600000	59.500000	57.600000	58.800000	3.327409e+08

ข้อมูลเบื้องต้น

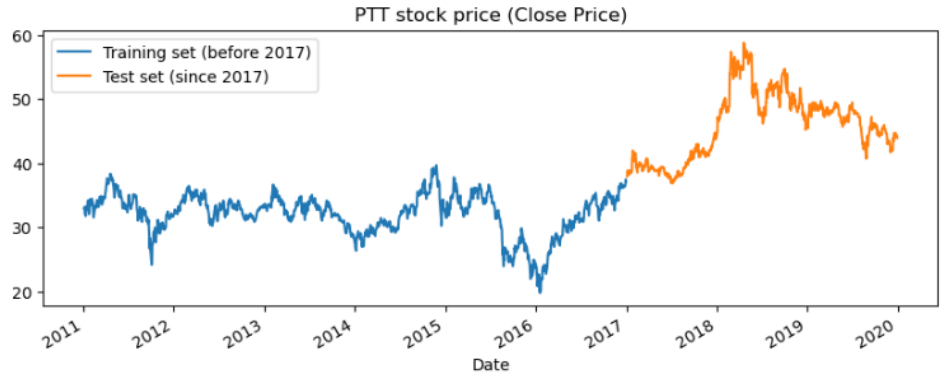
- จำนวน sample ทั้งหมด = 2199 (2199 rows) และมี 5 features (OHLCV)
- ไม่มี missing value
- จากการตรวจสอบข้อมูลจำเป็นต้องมีการ Normalize ข้อมูลราคาและข้อมูล volumes ปรับช่วงของข้อมูลให้อยู่ในช่วงเดียวกันก่อนนำข้อมูลไปใช้ เพราะข้อมูลราคาและข้อมูล volume มีช่วงค่าที่แตกต่างกันมาก

Visualization

```
df.plot(lw=1.0, subplots=True, grid=True, figsize=(11,8))
```




```
df.Close[:'2016'].plot(figsize=(10, 3.5), legend=True)
df.Close['2017':].plot(figsize=(10, 3.5), legend=True)
plt.legend(['Training set (before 2017)', 'Test set (since 2017)'])
plt.title('PTT stock price (Close Price)')
plt.show()
```



จากการ Visualization ข้อมูลราคาปิดเบื้องต้น จะพบว่าราคาปิดของ Training set มีค่าโดยประมาณไม่เกิน 40 แต่ Test set มีค่าเกิน 40 ดังนั้น หากหลังจากนี้โมเดลไม่สามารถทำนายราคาปิดของ Test set ในปี 2017 เป็นต้นไปได้อย่างแม่นยำ ส่วนตัวคิดว่าอาจเป็นเรื่องปกติ นี่อาจเป็นหนึ่งเหตุผลที่ทำให้ไม่สามารถทำนายได้อย่างแม่นยำ เพราะว่าโมเดลไม่เคยได้เรียนรู้ลักษณะข้อมูลที่มีราคาปิดสูงกว่า 40 เลย จึงอาจลดความสามารถในการทำนายได้ อาจไม่สามารถทำนายการเปลี่ยนแปลงได้อย่างแม่นยำถ้าหากราคาสูงกว่านี้ ส่งผลให้หลังจาก Train แล้ว ข้อมูลชุด Test ไม่ฟิตกับข้อมูลชุด Train

เตรียมข้อมูล

1. ทำการ Normalize โดยใช้ standard scale

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
features = ['Open', 'High', 'Low', 'Close', 'Volume']
data = df.loc[:,features].values
```

```
data_sc = sc.fit_transform(data)
```

```
data_scaler = pd.DataFrame(data_sc, columns=[f'sc_{c}' for c in features])
data_scaler # ข้อมูลที่ถูก Normalize แล้ว
```

	sc_Open	sc_High	sc_Low	sc_Close	sc_Volume
0	-0.556408	-0.490759	-0.540548	-0.488044	1.383881
1	-0.491045	-0.503738	-0.474710	-0.461880	0.148069
2	-0.451827	-0.477780	-0.461542	-0.461880	-0.433181
3	-0.477972	-0.516716	-0.553716	-0.592698	0.011678
4	-0.608699	-0.633526	-0.632723	-0.645026	1.092364
...	...	...	...	...	...
2194	1.077674	1.027767	1.000078	0.983663	-0.617823
2195	0.979629	0.930425	0.967159	0.983663	-1.199434
2196	0.979629	0.962872	1.032998	1.016368	-1.215047
2197	1.012310	0.995320	0.934239	0.983663	-0.002045
2198	0.979629	0.995320	1.000078	0.950959	-0.596579

2199 rows × 5 columns

2. นำ Dataset ไปสร้างเป็น feature และ label ในรูปของ matrix X, y โดยนำค่า step ถัดไป ไปเป็นข้อมูล label (y)

เนื่องจากข้อมูลตอนนี้มี 2199 rows และ 5 columns สำหรับเป็น features นั่นคือ ยังไม่มี label (y) ในที่นี้เราจะนำข้อมูลที่มีไปสร้างเป็น y ซึ่งขึ้นอยู่กับว่าเราจะให้ model เรียนรู้ทีละกี่ step กรณีนี้ใช้ 10 steps (timesteps=10) นั่นคือการนำค่าใน step ต่อไปมาเป็นค่า y ของลำดับปัจจุบัน นอกจากนี้ y ที่สร้างขึ้นจะเป็น label ที่ได้จากการเทียบราคาปิดจากเงื่อนไขโจทย์

ในที่นี้เราได้สร้างฟังกชัน convertToMatrix ขึ้นมาเพื่อให้ง่ายในการทำขั้นตอนต่อไป โดยที่ลักษณะของการทำงานของฟังก์ชัน convertToMatrix มีดังนี้

- วนลูปผ่านข้อมูล data โดยเริ่มต้นจาก index 0 ไปจนถึง (len(data) - timesteps) ซึ่งหมายถึงจำนวน sequence ที่จะถูกสร้างขึ้น และสำหรับแต่ละ sequence จะทำการเลือก features ในช่วง 10 วันก่อนหน้า (ตาม timesteps) เพื่อเป็น input feature ในการทำนาย
- สร้าง label ของทิศทางของราคาในวันถัดไป (up, down, neutral) โดยเทียบราคาปิดของวันที่ 11 (วันถัดไป) กับวันที่ 10 (วันปัจจุบัน) แล้วแปลง label ให้อยู่ในรูปแบบ one-hot encoding โดยใช้ to_categorical จาก Keras
- สะสม sequence และ label ที่ได้เข้า list X และ y ตามลำดับ หลังจากนั้นก็ทำการแปลง X ให้อยู่ในรูปแบบของ NumPy array แล้วส่งคืน X และ y เป็นผลลัพธ์ของการแปลงข้อมูล

```python
from keras.utils import to_categorical

def convertToMatrix(data, timesteps=10):
    num_classes = 3  # จำนวน classes (Up, Down, Neutral)
    X = []
    y = []

    # วนลูปเพื่อสร้าง sequences และ labels
    for i in range(len(data) - timesteps):
        # เลือก features ของ 10 วันก่อนหน้า
        selected_features = data[i:i + timesteps, :]
        X.append(selected_features)

        # สร้าง label เป็น array 3 ตัวที่เริ่มต้นทั้งหมดเป็น 0
        label = np.zeros(num_classes)

        # ราคาปิดของวันที่ 11 และ 10
        close_t_plus_1 = data[i + timesteps, -2]
        close_t = data[i + timesteps - 1, -2]

        # กำหนด label ตามเงื่อนไข Up, Down, หรือ Neutral
        if close_t_plus_1 > close_t + 0.01 * close_t:
            label = 2  # Up
        elif close_t_plus_1 < close_t - 0.01 * close_t:
            label = 0  # Down
        else:
            label = 1  # Neutral

        y.append(label)

    # แปลง labels เป็นรูปแบบ one-hot encoding
    X = np.array(X)
    y = to_categorical(np.array(y), num_classes=num_classes)

    return X, y
```

3. แบ่งข้อมูลเป็น train, test set และใช้ฟังก์ชัน convertToMatrix ในการเปลี่ยนข้อมูลเป็น matrix

- แบ่งข้อมูลเป็น train set และ test set

```python
# ตรวจสอบขนาดข้อมูลทั้งหมด และนับจำนวนข้อมูลที่จะใช้สำหรับ train

print('Samples:', df.shape)
print('Train set:', df[:'2016'].shape) # ตั้งแต่เริ่มต้นจนถึงปลายปี 2016
```

```
Samples: (2199, 5)
Train set: (1466, 5)
```

```python
n_train = 1466 # จำนวน samples ที่จะนำไป train
```

```python
data_sc.shape
```

```
(2199, 5)
```

```python
# นำข้อมูลที่ผ่าน Normalize แล้ว ไปแบ่งเป็น train, test
train, test = data_sc[0:n_train], data_sc[n_train:]
print('Train set:', train.shape)
print('Test set:', test.shape)
```

```
Train set: (1466, 5)
Test set: (733, 5)
```

จากเดิมข้อมูลมีขนาดมิติข้อมูลคือ (2199, 5) นั่นคือ มีจำนวน sample ทั้งหมดคือ 2199 samples ในที่นี้เราต้องทำการแบ่งเป็นข้อมูลที่ผ่านการ Normalize แล้วมาเป็นข้อมูลสำหรับ train และ test โดยที่เราจะใช้ข้อมูลปี 2011-2016 สำหรับ training set และใช้ข้อมูลปี 2017-2019 สำหรับ testing set

เนื่องจาก dataset ptt ข้อมูลที่มีจะเป็นการเรียงตามวันที่อยู่แล้ว ขั้นตอนแรกจึงทำการตรวจสอบว่าตั้งแต่ข้อมูลแรก (ปี 2011) ถึงข้อมูลปลายปี 2016 มีจำนวน sample เท่าไหร่ ก็จะพบว่าจำนวน sample ตั้งแต่ 2011-2016 มี 1466 samples ซึ่งเราจะนำไปใช้สำหรับ train และข้อมูลหลังจากนี้ที่เหลือ (733 samples) ก็จะใช้สำหรับ test

- **ใช้ฟังก์ชัน convertToMatrix ในการเปลี่ยนข้อมูลเป็น matrix (X, y)**

```
timesteps = 10

# Training set (before conversion)
print('train/test set (before conversion):', train.shape, test.shape)

# เรียกใช้ฟังก์ชัน convertToMatrix เพื่อแปลงข้อมูลเป็น X, y
X_train, y_train = convertToMatrix(train, timesteps)
X_test, y_test = convertToMatrix(test, timesteps)

# Training set (after conversion)
print('train/test set (after conversion):', X_train.shape, X_test.shape)
```

```
train/test set (before conversion): (1466, 5) (733, 5)
train/test set (after conversion): (1456, 10, 5) (723, 10, 5)
```

```
# ขนาดและมิติข้อมูลเมื่อแบ่งเรียบร้อยแล้ว
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (1456, 10, 5)
y_train shape: (1456, 3)
X_test shape: (723, 10, 5)
y_test shape: (723, 3)
```

ก่อนหน้านี้ที่แบ่งข้อมูลเป็น train และ test จะได้ขนาด sample train, test เป็น 1466 และ 733 ตามลำดับ แต่หลังจากที่นำข้อมูล train, test ไปแปลงเป็นข้อมูล matrix จะมีขนาดเป็น 1456 และ 723 ตามลำดับ เนื่องจากมีการดึงค่าออกไปใช้เป็นค่า y หลังจากนั้นเราได้ทำการแบ่งเป็น X train, y train, X test, y test ก็จะได้มิติข้อมูล ดังนี้

- Train (1466, 5) → X_train (1456, 10, 5) และ y_train (1456, 3)
- Test (733, 5) → X_test (723, 10, 5) และ y_test (723, 3)

สร้างและเทรนโมเดล

- ใช้ optimizer adam with learning rate = 0.03
- เทรนด้วยจำนวน epoch = 200
- validation data = X_test, y_test
- ไม่กำหนดอื่น ๆ เพิ่มเติม ; batch_size, drop out เพราะจากการทดลอง ไม่กำหนดเพิ่มจากนี้ได้ผลลัพธ์ที่ดีกว่า

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
from keras.optimizers import Adam
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(128, return_sequences=True),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.LSTM(32, return_sequences=False),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(3, activation=None),
    tf.keras.layers.Softmax()
])
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.03)
loss = tf.keras.losses.CategoricalCrossentropy()
```

```
model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='log')
```

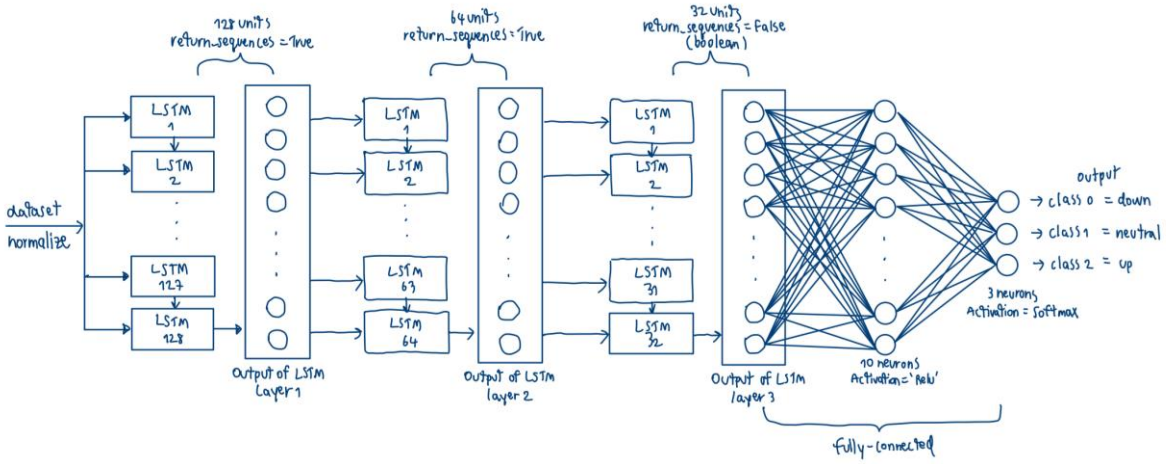
```
history = model.fit(X_train, y_train,
                    epochs=200,
                    validation_data=(X_test, y_test),
                    callbacks=[tensorboard_callback])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 128)           68608

 lstm_1 (LSTM)               (None, 10, 64)            49408

 lstm_2 (LSTM)               (None, 32)                12416

 dense (Dense)               (None, 10)                330

 dense_1 (Dense)             (None, 3)                 33

 softmax (Softmax)           (None, 3)                 0

=================================================================
Total params: 130,795
Trainable params: 130,795
Non-trainable params: 0
_________________________________________________________________
```

Diagram of network:

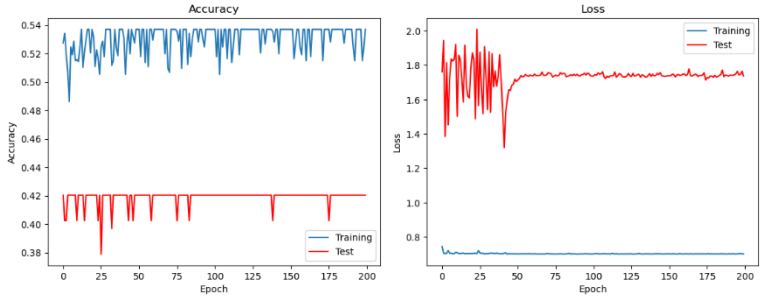
- LSTM จำนวน 3 ชั้น ต่อด้วย fully connected 1 ชั้น หลังจากนั้นต่อด้วย Output layer (dense) ขนาด 3 Node



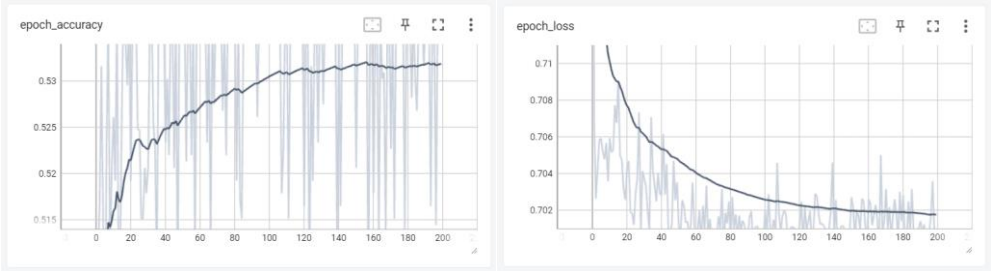
ประเมินโมเดล

```
# ประเมินโมเดล
loss_test, acc_test = model.evaluate(X_test, y_test, verbose=0)
loss_train, acc_train = model.evaluate(X_train, y_train, verbose=0)

print(f'Accuracy (train set): {acc_train:.3f}')
print(f'Accuracy (test set): {acc_test:.3f}')
print(f'Loss (train set): {loss_train:.3f}')
print(f'Loss (test set): {loss_test:.3f}')
```

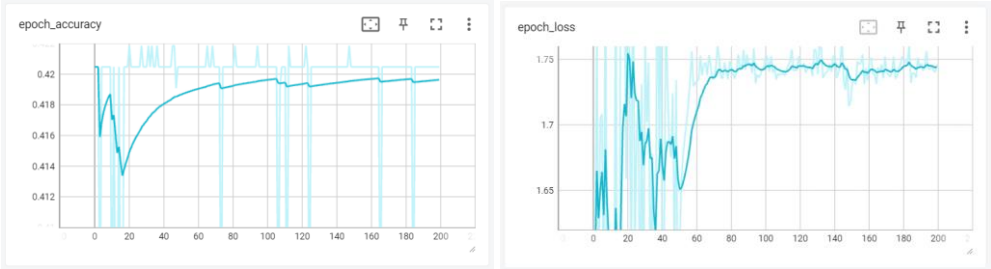
```
Accuracy (train set): 0.537
Accuracy (test set): 0.420
Loss (train set): 0.700
Loss (test set): 1.736
```



เปรียบเทียบ Accuracy และ Loss ของ Training Set vs Test set



Accuracy และ Loss ของ Training Set



Accuracy และ Loss ของ Test Set

write a code to apply a strategy “hold-one-day when the trend is up” to the test set (2017-2019).

- Calculate the total gain using this strategy and compare it with the buy-and-hold strategy.

```
# วนลูปผ่านข้อมูลทดสอบ
profits_hold_one_day = 0
for i in range(len(X_test)):
    # ทำนายทิศทาง
    prediction = model.predict(X_test[i:i+1, :, :])
    predicted_class = np.argmax(prediction)

    # ตรวจสอบทิศทางการทำนายและดำเนินการตามกลยุทธ์
    close_t_plus_1 = test[i + timesteps, -2]  # ราคาปิดของวันที่ 11
    close_t = test[i + timesteps - 1, -2]  # ราคาปิดของวันที่ 10

    if predicted_class == 2:  # Up trend
        profit = close_t_plus_1 - close_t
        profits_hold_one_day += profit

# คำนวณกำไรของกลยุทธ์ "hold-one-day when the trend is up"
print(f'Total profit with "hold-one-day when the trend is up" strategy: {profits_hold_one_day:.3f} Baht')

# กำไรจากกลยุทธ์ buy-and-hold
buy_and_hold_buy_price = 37.4
buy_and_hold_sell_price = 44.0
buy_and_hold_quantity = 1  # ซื้อหุ้น 1 หน่วย
buy_and_hold_profit = (buy_and_hold_sell_price - buy_and_hold_buy_price) * buy_and_hold_quantity

# แสดงผลลัพธ์กำไรของกลยุทธ์ buy-and-hold
print(f'Profit with buy-and-hold strategy: {buy_and_hold_profit:.3f} Baht')
```

Code ข้างต้นเขียนไว้เพื่อทดสอบกำไรจากกลยุทธ์ "hold-one-day when the trend is up" และเปรียบเทียบกับกลยุทธ์ Buy-and-Hold ตามที่ได้ระบุไว้ในโจทย์

โปรแกรมทำงานตามขั้นตอนต่อไปนี้:

1. การทำนายและคำนวณกำไรจากกลยุทธ์ "Hold-One-Day When the Trend is Up":
    - ในลูป for ที่วนลูปผ่านข้อมูลทดสอบ (X_test) โมเดลถูกใช้ในการทำนายทิศทางของราคาหุ้นในแต่ละวัน ทิศทางการทำนายถูกดึงออกมาจาก output layer ของโมเดล (ผลลัพธ์ที่ได้เป็นคลาสที่มีความน่าจะเป็นสูงที่สุด)
    - หากโมเดลทำนายว่าเป็น Uptrend (predicted_class == 2) ก็จะคำนวณกำไรที่มีจากการซื้อหุ้นในวันที่ 10 และขายในวันที่ 11
    - กำไรทั้งหมดจะถูกสะสมและแสดงผลที่สิ้นสุดของการทำนาย
2. การคำนวณกำไรจากกลยุทธ์ Buy-and-Hold:
    - กำไรจากกลยุทธ์ Buy-and-Hold ถูกคำนวณโดยหาต้นทุนในการซื้อหุ้นในวันที่ 1/4/2017 และกำไรจากการขายหุ้นในวันที่ 12/30/2019
    - แสดงผลลัพธ์ของกำไรจาก Buy-and-Hold.

Result:

```
Total profit with "hold-one-day when the trend is up" strategy: 0.693 Baht
Profit with buy-and-hold strategy: 6.600 Baht
```

สรุปผลจากโมเดล (แต่โมเดลยังไม่ได้ดีพอที่จะเชื่อถือ):

- กลยุทธ์ "Hold-One-Day When the Trend is Up" ทำการซื้อ-ขายหุ้นบ่อยกว่าเนื่องจากจะดำเนินการตามทิศทางของโมเดลทุกวันที่โมเดลทำนายว่าราคาจะเพิ่มขึ้น (Uptrend) ไม่ว่าจะมีกำไรน้อย
- กลยุทธ์ Buy-and-Hold ซื้อและถือหุ้นไว้จนถึงวันที่ 12/30/2019 โดยไม่มีการขายระหว่างทาง จึงสามารถรับกำไรมากกว่าจากการทำกำไรทุกวันเล็กน้อย
- การเปรียบเทียบระหว่างกลยุทธ์สองแบบนี้บ่งบอกว่า กลยุทธ์ "Hold-One-Day When the Trend is Up" ไม่ได้ให้กำไรมากนักเมื่อเทียบกับ Buy-and-Hold ที่สามารถกำไรมากกว่าได้ แต่นี่หมายถึงผลลัพธ์จากโมเดลนี้ ความจริงควรต้องปรับปรุงโมเดลและกลยุทธ์เพื่อให้ทำกำไรได้มากกว่านี้