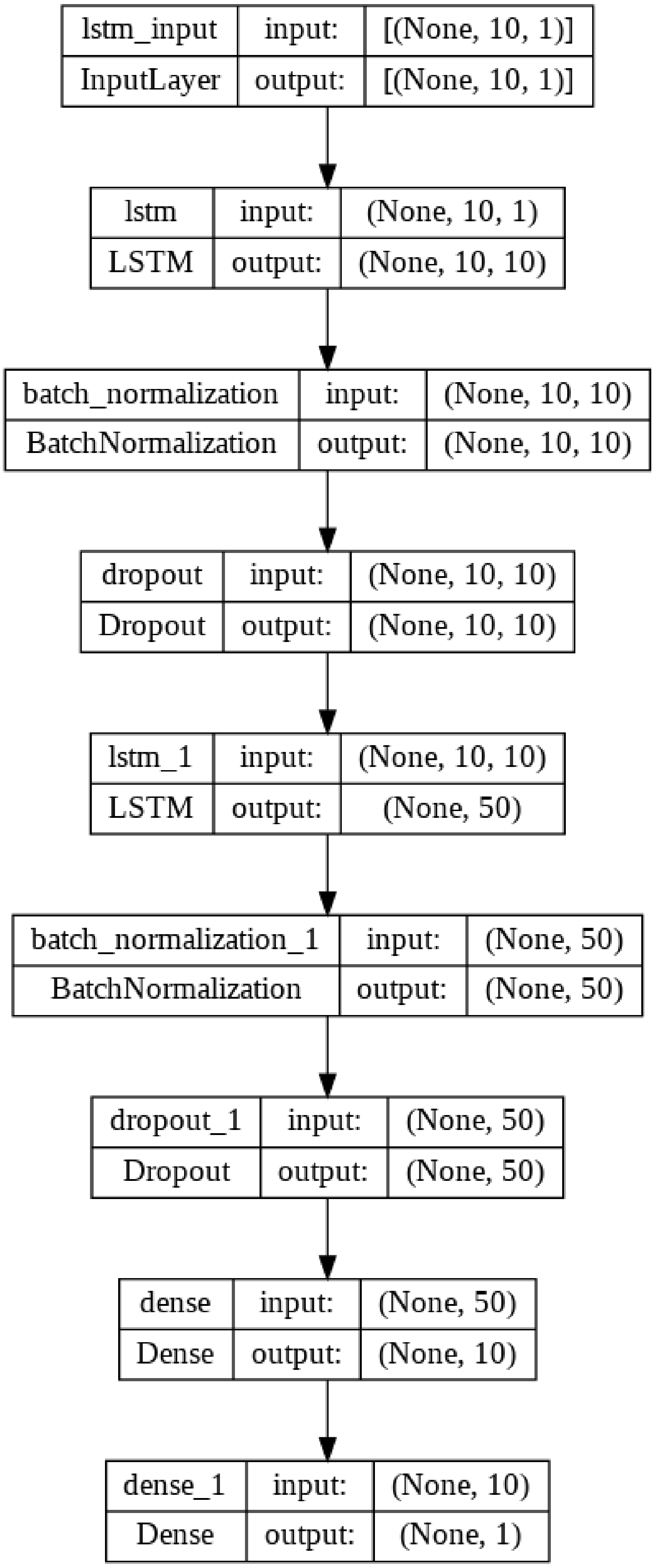
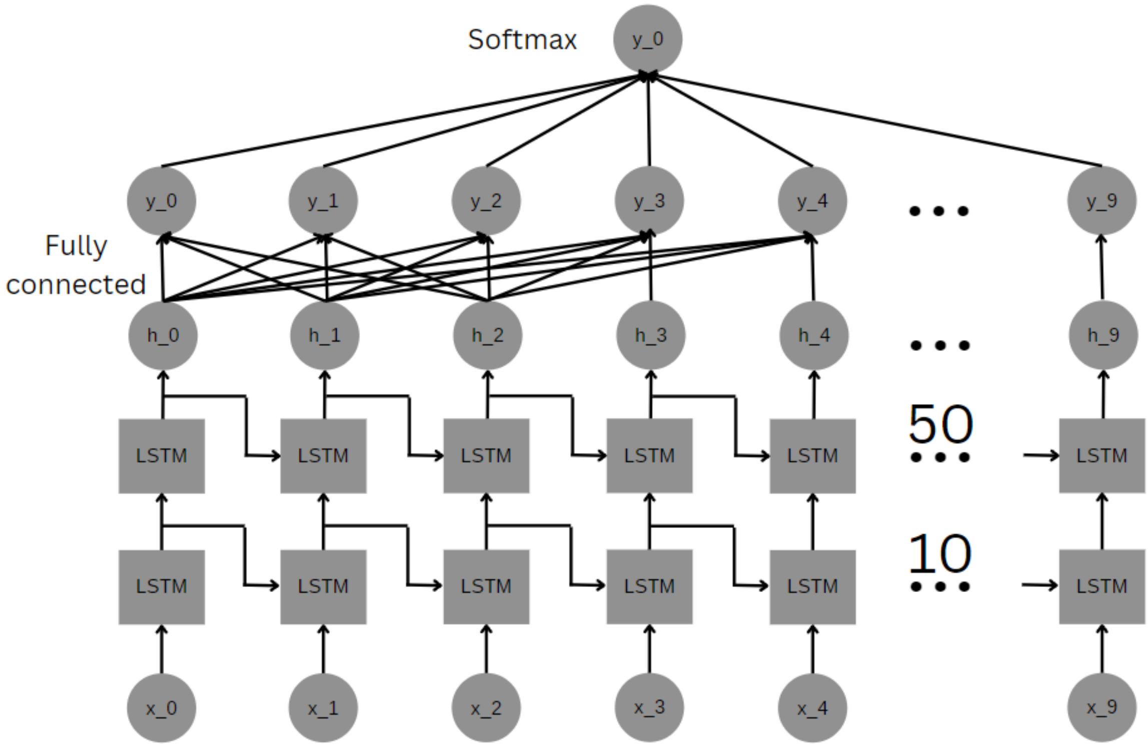
Homework 4

Diagram of network and parameter used



HW04 INC491 Pattarapon Buathong 62070504012 (AE)



Source code

```
[1] import tensorflow as tf
    import numpy as np
    import pandas as pd
    pd.set_option('display.float_format', lambda x: '%.7f' % x)
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import MinMaxScaler
```

```
[2] # Import PTT CSV as DataFrame
    df = pd.read_csv('ptt.csv')
    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   object
     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: float64(4), int64(1), object(1)
    memory usage: 103.2+ KB
```

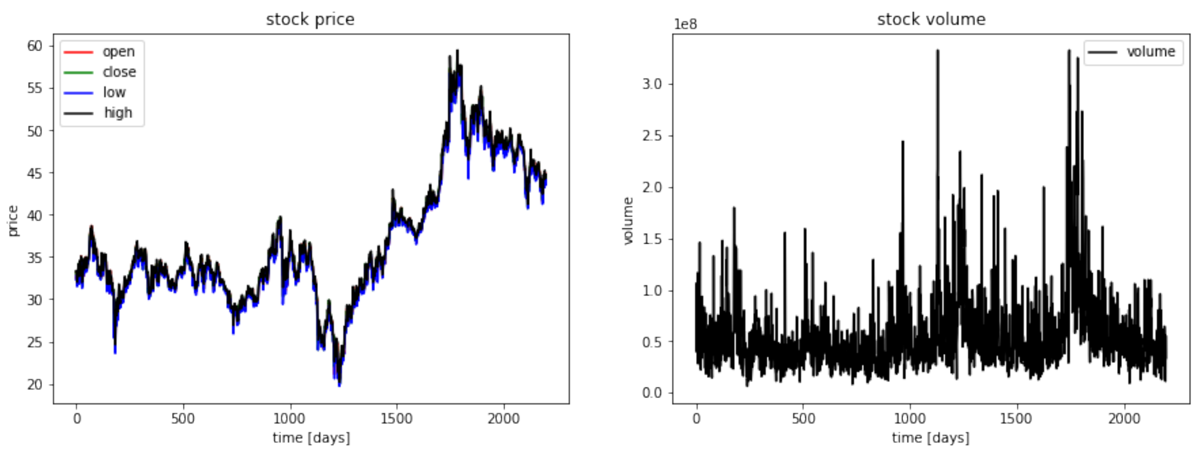
[1] Import libraries, I also set pandas float format to 7 digits.

[2] Import ptt.csv into DataFrame and check the information of it.

```
plt.figure(figsize=(15, 5));
plt.subplot(1,2,1);
plt.plot(df.Open.values, color='red', label='open')
plt.plot(df.Close.values, color='green', label='close')
plt.plot(df.Low.values, color='blue', label='low')
plt.plot(df.High.values, color='black', label='high')
plt.title('stock price')
plt.xlabel('time [days]')
plt.ylabel('price')
plt.legend(loc='best')
#plt.show()

plt.subplot(1,2,2);
plt.plot(df.Volume.values, color='black', label='volume')
plt.title('stock volume')
plt.xlabel('time [days]')
plt.ylabel('volume')
plt.legend(loc='best');
```

Next, I visualized the stock price as a graph, I also visualized stock volume.



Here is the result of the aforementioned Pyplot code.

```
# Some Date are in General form, need to convert to datetime
df['Date'] = pd.to_datetime(df['Date'])
```

```
[5] train = df[(df['Date'] >= '1/4/2011') & (df['Date'] <= '12/31/2016')]
    print(train)
```

```
           Date       Open       High        Low      Close     Volume
0    2011-01-04 32.5000000 33.3000000 32.3000000 33.0000000  105964752
1    2011-01-05 33.0000000 33.2000000 32.8000000 33.2000000   60864168
2    2011-01-06 33.3000000 33.4000000 32.9000000 33.2000000   39651640
3    2011-01-07 33.1000000 33.1000000 32.2000000 32.2000000   55886640
4    2011-01-10 32.1000000 32.2000000 31.6000000 31.8000000   95325912
...         ...        ...        ...        ...        ...        ...
1461 2016-12-26 36.6000000 36.8000000 36.3000000 36.5000000   15572000
1462 2016-12-27 36.5000000 36.6000000 36.4000000 36.5000000   15425000
1463 2016-12-28 36.5000000 36.9000000 36.4000000 36.9000000   33264000
1464 2016-12-29 36.7000000 37.7000000 36.7000000 37.5000000   57663000
1465 2016-12-30 37.4000000 37.7000000 37.2000000 37.2000000   44540000

[1466 rows x 6 columns]
```

```
[6] test = df[df['Date'] >= '1/1/2017']
    print(test)
```

```
           Date       Open       High        Low      Close    Volume
1466 2017-01-04 37.4000000 38.1000000 37.3000000 38.0000000  75879000
1467 2017-01-05 38.2000000 38.8000000 38.2000000 38.7000000  75282000
1468 2017-01-06 38.7000000 38.9000000 38.5000000 38.9000000  45129000
1469 2017-01-09 38.9000000 39.0000000 38.2000000 38.3000000  40455000
1470 2017-01-10 38.3000000 38.9000000 38.2000000 38.8000000  43224000
...         ...        ...        ...        ...        ...       ...
2194 2019-12-24 45.0000000 45.0000000 44.0000000 44.2500000  32913200
2195 2019-12-25 44.2500000 44.2500000 43.7500000 44.2500000  11687500
2196 2019-12-26 44.2500000 44.5000000 44.2500000 44.5000000  11117700
2197 2019-12-27 44.5000000 44.7500000 43.5000000 44.2500000  55385800
2198 2019-12-30 44.2500000 44.7500000 44.0000000 44.0000000  33688500

[733 rows x 6 columns]
```

[4] Some data in the Date column are in general form (in Excel), I converted the whole column into datetime datatype.

[5] I separated the data into train (2011-2016) dataset.

[6] I separated the data into test (2017-2019) dataset.

```
[7] train.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1466 entries, 0 to 1465
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1466 non-null   datetime64[ns]
 1   Open    1466 non-null   float64
 2   High    1466 non-null   float64
 3   Low     1466 non-null   float64
 4   Close   1466 non-null   float64
 5   Volume  1466 non-null   int64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 80.2 KB
```

```
[8] # Normalize stock
sc = MinMaxScaler(feature_range = (0, 1))
sc_vol = MinMaxScaler(feature_range = (0, 1))

train_norm = train.copy()
train_norm['Open'] = sc.fit_transform(train_norm.Open.values.reshape(-1,1))
train_norm['High'] = sc.fit_transform(train_norm.High.values.reshape(-1,1))
train_norm['Low'] = sc.fit_transform(train_norm.Low.values.reshape(-1,1))
train_norm['Close'] = sc.fit_transform(train_norm['Close'].values.reshape(-1,1))
train_norm['Volume'] = sc_vol.fit_transform(train_norm.Volume.values.reshape(-1,1))

test_norm = test.copy()
test_norm['Open'] = sc.fit_transform(test_norm.Open.values.reshape(-1,1))
test_norm['High'] = sc.fit_transform(test_norm.High.values.reshape(-1,1))
test_norm['Low'] = sc.fit_transform(test_norm.Low.values.reshape(-1,1))
test_norm['Close'] = sc.fit_transform(test_norm['Close'].values.reshape(-1,1))
test_norm['Volume'] = sc_vol.fit_transform(test_norm.Volume.values.reshape(-1,1))
```

[7] Check if new Date column is in datetime format already or not, it is.

[8] Data normalization using Scikit-learn's MinMaxScaler, I separated the scaler into sc and sc_vol. Because the volume data is huge number compared to the prices. So, if we performed inverse transform later, the inversed outcome will be wrong.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Here is the formula of MinMaxScaler.

```
train_norm
```

	Date	Open	High	Low	Close	Volume
0	2011-01-04	0.6428571	0.6683673	0.6528497	0.6633166	0.3052875
1	2011-01-05	0.6683673	0.6632653	0.6787565	0.6733668	0.1671251
2	2011-01-06	0.6836735	0.6734694	0.6839378	0.6733668	0.1021420
3	2011-01-07	0.6734694	0.6581633	0.6476684	0.6231156	0.1518768
4	2011-01-10	0.6224490	0.6122449	0.6165803	0.6030151	0.2726962
...	...	...	...	...	...	...
1461	2016-12-26	0.8520408	0.8469388	0.8601036	0.8391960	0.0283758
1462	2016-12-27	0.8469388	0.8367347	0.8652850	0.8391960	0.0279255
1463	2016-12-28	0.8469388	0.8520408	0.8652850	0.8592965	0.0825740
1464	2016-12-29	0.8571429	0.8928571	0.8808290	0.8894472	0.1573186
1465	2016-12-30	0.8928571	0.8928571	0.9067358	0.8743719	0.1171172

1466 rows × 6 columns

```
[10] test_norm
```

	Date	Open	High	Low	Close	Volume
1466	2017-01-04	0.0319635	0.0403587	0.0379147	0.0502283	0.2061038
1467	2017-01-05	0.0684932	0.0717489	0.0805687	0.0821918	0.2042581
1468	2017-01-06	0.0913242	0.0762332	0.0947867	0.0913242	0.1110373
1469	2017-01-09	0.1004566	0.0807175	0.0805687	0.0639269	0.0965872
1470	2017-01-10	0.0730594	0.0762332	0.0805687	0.0867580	0.1051478
...	...	...	...	...	...	...
2194	2019-12-24	0.3789954	0.3497758	0.3554502	0.3356164	0.0732710
2195	2019-12-25	0.3447489	0.3161435	0.3436019	0.3356164	0.0076498
2196	2019-12-26	0.3447489	0.3273543	0.3672986	0.3470320	0.0058882
2197	2019-12-27	0.3561644	0.3385650	0.3317536	0.3356164	0.1427472
2198	2019-12-30	0.3447489	0.3385650	0.3554502	0.3242009	0.0756680

733 rows × 6 columns

Next, I check the new train and test data that are already normalized.

```
[11] close_norm = train_norm['Close'].values.tolist()
```

```
[12] timesteps = 10

     x_train = []
     y_train = []
     for i in range(len(close_norm)-timesteps):
         x_train.append(close_norm[i:i+timesteps])
         y_train.append(close_norm[i+timesteps])

     x_train = np.array(x_train)
     x_train = x_train.reshape((-1,timesteps,1))
     y_train = np.array(y_train)
```

Next, use the dividing windows timesteps as shown in the class. The time steps is 10, and separated into x_train (features) and y_train (label).

```
[13] tb = df[df['Date'] >= '1/1/2017']
     close_test = tb['Close'].values.tolist()

     features_test = []
     label_test = []
     for i in range(len(close_test)-timesteps):
         features_test.append(close_test[i:i+timesteps])
         label_test.append(close_test[i+timesteps])

     features_test = np.array(features_test)
     features_test = features_test.reshape((-1,timesteps,1))
     label_test = np.array(label_test)
```

```
[14] tt = test_norm['Close'].values.tolist()

     x_test = []
     y_test = []
     for i in range(len(tt)-timesteps):
         x_test.append(tt[i:i+timesteps])
         y_test.append(tt[i+timesteps])

     x_test = np.array(x_test)
     x_test = x_test.reshape((-1,timesteps,1))
     y_test = np.array(y_test)
```

Do the same for testing set. But I did it two times, first is for the raw test data, it is for calculations later. Second is for the normalized test data, it is for the validation in LSTM training iterations.

```
[15] 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, 1)
     y_train.shape =  (1456,)
     x_test.shape =  (723, 10, 1)
     y_test.shape =  (723,)
```

```
[16] model = tf.keras.models.Sequential([
         tf.keras.layers.LSTM(10, return_sequences=True),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.Dropout(0.2),
         tf.keras.layers.LSTM(50, return_sequences=False),
         tf.keras.layers.BatchNormalization(),
         tf.keras.layers.Dropout(0.2),
         tf.keras.layers.Dense(10, activation='relu'),
         tf.keras.layers.Dense(1, activation=None)
     ])

     loss = tf.keras.losses.MeanSquaredError()

     model.compile(loss=loss, optimizer='adam', metrics=['mse'])

     tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='log')
```

[15] Check the shape of x_train, y_train, x_test, y_test

[16] Create the model structures, I use the pattern of LSTM → BatchNormalization → Dropout. The reason I use BatchNormalization is because the batch parameter is set at 32 in model fitting. Adam optimizer is used, and mean squared error is for loss measurement.

```
[18] model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 10)            480

 batch_normalization (BatchN  (None, 10, 10)           40
 ormalization)

 dropout (Dropout)           (None, 10, 10)            0

 lstm_1 (LSTM)               (None, 50)                12200

 batch_normalization_1 (Batc  (None, 50)               200
 hNormalization)

 dropout_1 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 10)                510

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 13,441
Trainable params: 13,321
Non-trainable params: 120
_________________________________________________________________
```

Here is the model summary result.

```
[17] model.fit(x_train, y_train, epochs=200, validation_data=(x_test, y_test), batch_size = 32, callbacks=[tensorboard_callback])
46/46 [==============================] - 0s 11ms/step - loss: 0.0021 - mse: 0.0021 - val_loss: 0.0040 - val_mse: 0.0040
Epoch 173/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0022 - mse: 0.0022 - val_loss: 0.0040 - val_mse: 0.0040
Epoch 174/200
46/46 [==============================] - 0s 10ms/step - loss: 0.0023 - mse: 0.0023 - val_loss: 0.0046 - val_mse: 0.0046
Epoch 175/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0040 - val_mse: 0.0040
Epoch 176/200
46/46 [==============================] - 0s 10ms/step - loss: 0.0021 - mse: 0.0021 - val_loss: 0.0046 - val_mse: 0.0046
Epoch 177/200
46/46 [==============================] - 0s 10ms/step - loss: 0.0018 - mse: 0.0018 - val_loss: 0.0037 - val_mse: 0.0037
Epoch 178/200
46/46 [==============================] - 0s 9ms/step - loss: 0.0023 - mse: 0.0023 - val_loss: 0.0043 - val_mse: 0.0043
Epoch 179/200
46/46 [==============================] - 0s 9ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0047 - val_mse: 0.0047
Epoch 180/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0039 - val_mse: 0.0039
Epoch 181/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0040 - val_mse: 0.0040
Epoch 182/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0037 - val_mse: 0.0037
Epoch 183/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0022 - mse: 0.0022 - val_loss: 0.0041 - val_mse: 0.0041
Epoch 184/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0023 - mse: 0.0023 - val_loss: 0.0037 - val_mse: 0.0037
Epoch 185/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0020 - mse: 0.0020 - val_loss: 0.0043 - val_mse: 0.0043
Epoch 186/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0023 - mse: 0.0023 - val_loss: 0.0043 - val_mse: 0.0043
Epoch 187/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0023 - mse: 0.0023 - val_loss: 0.0039 - val_mse: 0.0039
Epoch 188/200
46/46 [==============================] - 0s 10ms/step - loss: 0.0026 - mse: 0.0026 - val_loss: 0.0036 - val_mse: 0.0036
Epoch 189/200
46/46 [==============================] - 1s 11ms/step - loss: 0.0024 - mse: 0.0024 - val_loss: 0.0035 - val_mse: 0.0035
Epoch 190/200
46/46 [==============================] - 1s 12ms/step - loss: 0.0020 - mse: 0.0020 - val_loss: 0.0035 - val_mse: 0.0035
Epoch 191/200
```

Next, fit the model. Use x_train as X, y_train as y, 200 epochs of training, validation data is x_test and y_test that we created earlier, batch size is 32, and callback to Tensorboard to observe the loss and accuracy of each epoch trained.

```
[19] model.evaluate(x_train, y_train)

46/46 [==============================] - 0s 4ms/step - loss: 0.0010 - mse: 0.0010
[0.0010267210891470313, 0.0010267210891470313]

[20] model.evaluate(x_test, y_test)

23/23 [==============================] - 0s 3ms/step - loss: 0.0033 - mse: 0.0033
[0.0033358423970639706, 0.0033358423970639706]

[21] predict = model.predict(x_test)
     predict = sc.inverse_transform(predict)

23/23 [==============================] - 1s 3ms/step
```

Now I evaluate the model on train and test set.

Then I predict using x_test from the test set. Since x_test is normalized, we have to use the same scaler to convert it back to normal form (inverse transformation).

```
[32] last = np.array(close_test[timesteps-1:-1])  # arr 9 to before last
     last = last.reshape((-1,1))
     signal = ((predict - last)/last * 100) >= 1 # 1 percent
     # down = ((predict - last)/last * 100) <= -1
     # neutral = ((predict - last)/last * 100) < 1 and ((predict - last)/last * 100) > -1

     count = 0
     profit = 0
     for i in range(len(signal)-1):
       if i>8: # start at arr 9, day 10
         if (signal[i] == True): # Up signal
             profit = profit + (label_test[i+1] - label_test[i])
             print('Day', i)
             print('Buy: ', label_test[i])
             print('Sell: ', label_test[i+1])
             print('Profit: ',profit, 'Baht')

     print('Profit for buying/selling 1 share = ', profit)
     print('Buy and hold will give a profit =', 44-37.4)
```

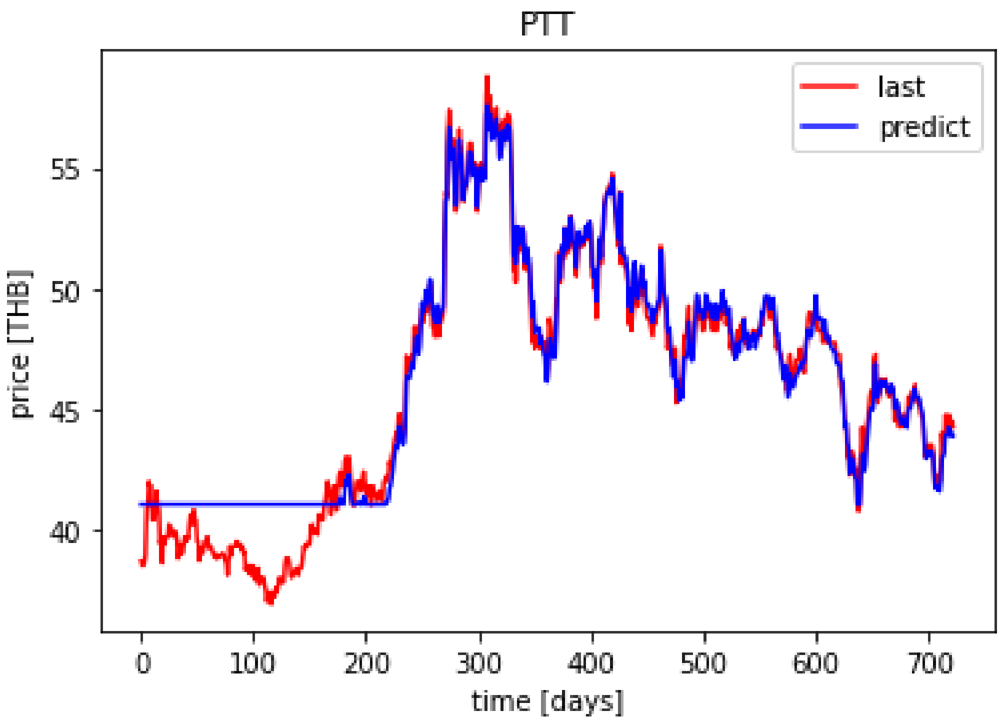
For the buy signal, see if the close price of the 11th day is +1% more than the close price of the 10th day. If so, we buy, if not, we do nothing. Then compared with the buy and hold method.

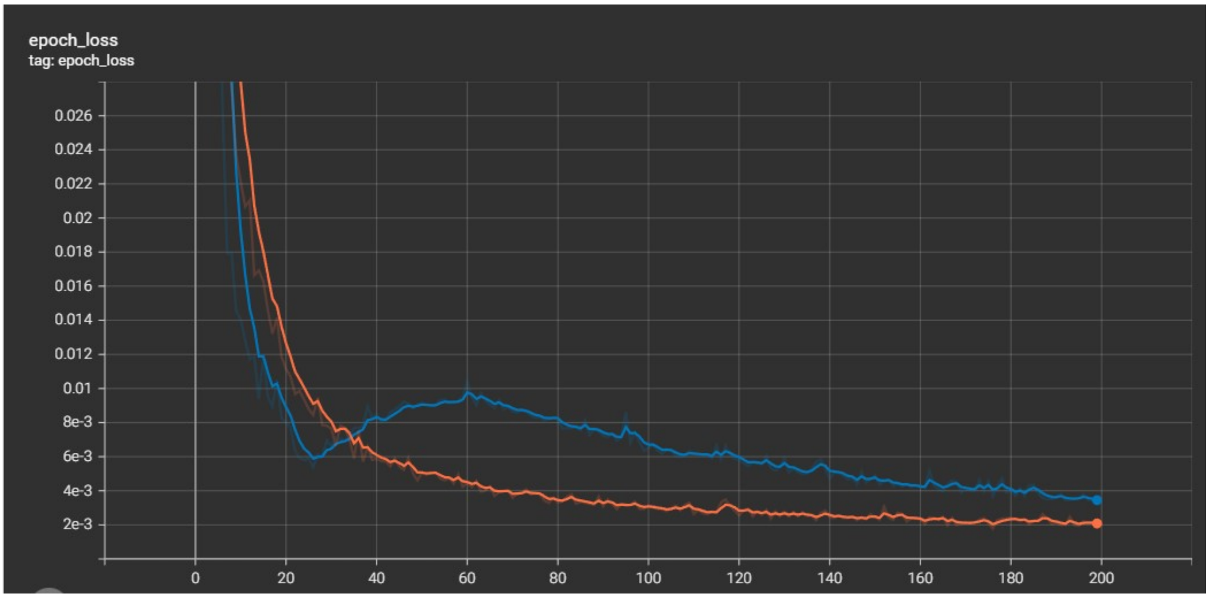
```
Day 623
Buy:  45.5
Sell:  45.0
Profit:  7.549999999999997 Baht
Day 624
Buy:  45.0
Sell:  44.25
Profit:  6.799999999999997 Baht
Day 628
Buy:  43.25
Sell:  42.25
Profit:  5.799999999999997 Baht
Day 670
Buy:  45.25
Sell:  45.0
Profit:  5.549999999999997 Baht
Profit for buying/selling 1 share =  5.549999999999997
Buy and hold will give a profit = 6.600000000000001
```

I got 5.55 Baht for this method and 6.6 for buy and hold, so buy and hold is better in this case.

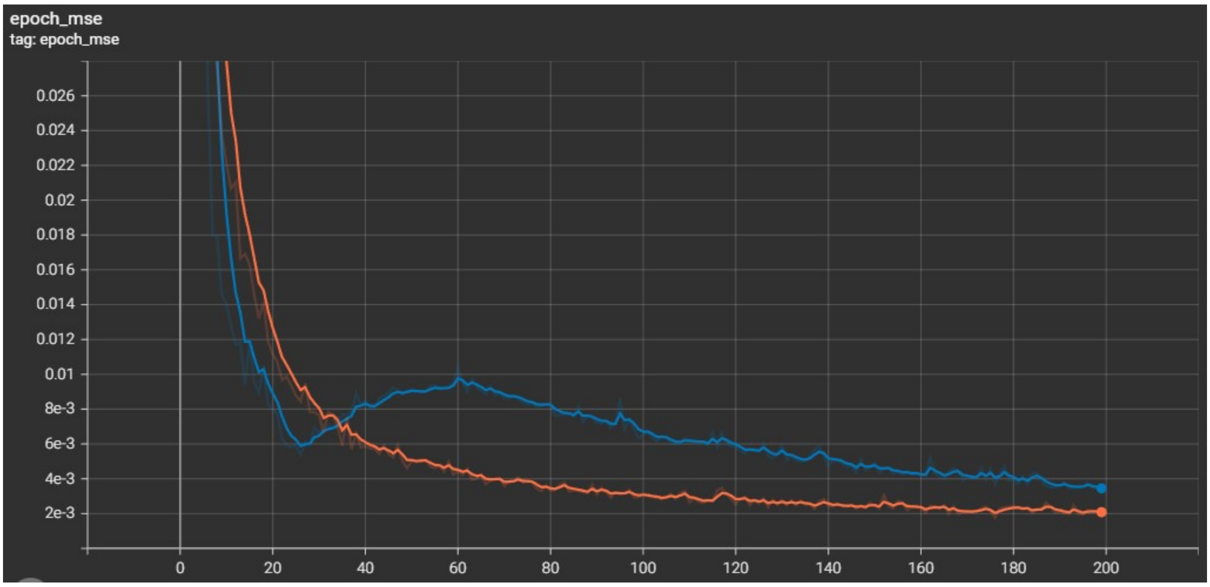
```
plt.plot(last, color='red', label='last')
plt.plot(predict, color='blue', label='predict')
plt.title('PTT')
plt.xlabel('time [days]')
plt.ylabel('price [THB]')
plt.legend(loc='best')
```

Plot the prediction compared to price





This is graph of loss per epoch, orange is train, blue is validation (test).



This is graph of accuracy (MSE) per epoch, orange is train, blue is validation (test).