Deep learning models for predicting stock price

1 Introduction to Microsoft Stock Price Dataset

This course project was based on the Microsoft Stock Price Dataset downloaded from Data Repository of Kaggle. This dataset consists of the stock price information of Microsoft from 04/01/2015 to 04/01/2021. Figure 1 shows the first 10 rows of the dataset. Each row of the dataset contains the Open Price, High Price, Low Price, Close Price and Volume associated with an individual date. The column Volume represents the number of shares traded in that particular day. The dataset consists of 1511 rows. The summary statistics of each column is shown in Figure 2.

	Date	Open	High	Low	Close	Volume
0	4/1/2015	40.60	40.76	40.31	40.72	36865322
1	4/2/2015	40.66	40.74	40.12	40.29	37487476
2	4/6/2015	40.34	41.78	40.18	41.55	39223692
3	4/7/2015	41.61	41.91	41.31	41.53	28809375
4	4/8/2015	41.48	41.69	41.04	41.42	24753438
5	4/9/2015	41.25	41.62	41.25	41.48	25723861
6	4/10/2015	41.63	41.95	41.41	41.72	28022002
7	4/13/2015	41.40	42.06	41.39	41.76	30276692
8	4/14/2015	41.80	42.03	41.39	41.65	24244382
9	4/15/2015	41.76	42.46	41.68	42.26	27343581

Figure 1 The first 10 rows of Microsoft Stock Price Dataset

	Open	High	Low	Close	Volume
count	1511.000000	1511.000000	1511.000000	1511.000000	1.511000e+03
mean	107.385976	108.437472	106.294533	107.422091	3.019863e+07
std	56.691333	57.382276	55.977155	56.702299	1.425266e+07
min	40.340000	40.740000	39.720000	40.290000	1.016120e+05
25%	57.860000	58.060000	57.420000	57.855000	2.136213e+07
50%	93.990000	95.100000	92.920000	93.860000	2.662962e+07
75%	139.440000	140.325000	137.825000	138.965000	3.431962e+07
max	245.030000	246.130000	242.920000	244.990000	1.352271e+08

Figure 2 Summary Statistics of Microsoft Stock Price Dataset

Figure 3 shows the variation of the stock price between 04/01/2015 and 04/01/2021.

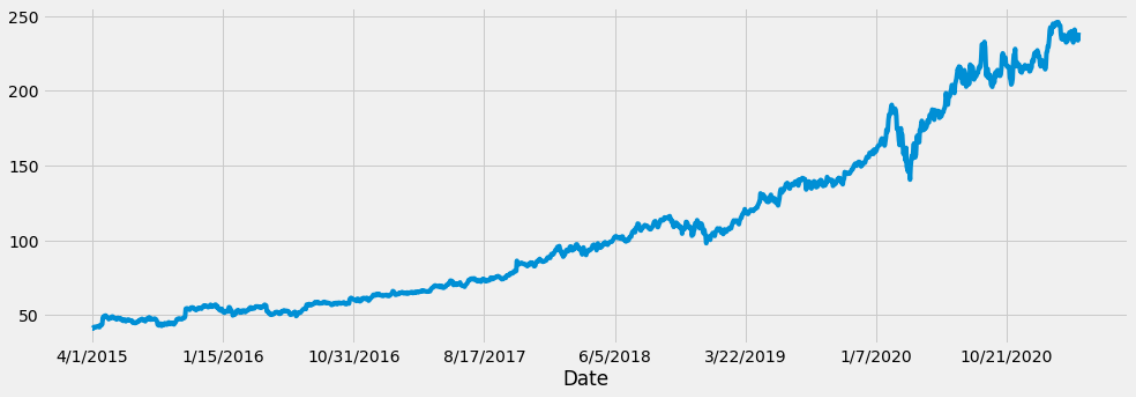

Figure 3 Stock Price Variation between 04/01/2015 and 04/01/2021

2 Organization and Main Objectives of the Project

The remainder of this project report are organized as follows: Section 3 will present the process of data loading and the development of four neural network-based models. The first two models are recurrent neural networks (RNNs) with different configuration. The third model is a Long Short-Term Memory (LSTM) network. These three models were developed from the scratch. The last model is referred to an existing LSTM architecture. Section 3 also compares the evaluation result using test dataset of the four developed models and key findings. The last section summarizes the project and proposed future improvement opportunities.

The main objectives of this course project are as follows:

- Practice loading time series dataset and preprocessing
- Practice setting up the configuration of RNN, compiling the model and training it
- Practice setting up the configuration of LSTM networks, compiling the model and training it
- Exploring if leveraging existing LSTM architecture can improve the predictive accuracy of the model

3 Data Loading and Model Development

3.1 Data Loading

As the dataset is contained in a csv file, the data loading process is straightforward using pandas. read_csv. After the dataset is loaded, it is confirmed there is no missing value in the dataset as follows.

```
check_nan = df_Stock.isnull().values.any()
check_nan
```

```
False
```

Figure 4 Data Completeness Check

Another important step is to set Date Column as the index of the dataframe as below. As the dataset is highly clean and consistent, we can start the model development stage.

```
df_Stock.set_index(df_Stock.Date,inplace=True)
```

3.2 Train RNN-based Model 1

For the purpose of practicing and reducing the training time, the architecture of the first RNN model is relatively simple. As the model development uses Keras, one pre-processing step is to convert the data into a format that the Keras library accepts. For Keras RNNs and LSTMs, training samples should be stored in a 3D numpy array of shape **(n_samples, time_steps, n_features)**.

Here in the first three models only the series' history will be used to predict its future, the model only takes 1 feature. However, when developing the last model, all the five columns as shown in Figure 1 will be used as the feature to predict the stock price in the future.

To this end, the utility functions that were provided in the course note are used to extract the formatted data as below.

```
def get_keras_format_series(series):
    """
    Convert a series to a numpy array of shape
    [n_samples, time_steps, features]
    """

    series = np.array(series)
    return series.reshape(series.shape[0], series.shape[1], 1)

def get_train_test_data(df, series_name, series_days, input_days,
                        test_days, sample_gap=3):
    """
    Utility processing function that splits an hourly time series into
    train and test with keras-friendly format, according to user-specified
    choice of shape.

    arguments
    ---------
    df (dataframe): dataframe with time series columns
    series_name (string): column name in df
    series_days (int): total days to extract
    input_hours (int): length of sequence input to network
    test_hours (int): length of held-out terminal sequence
    sample_gap (int): step size between start of train sequences; default 5

    returns
    ---------
    tuple: train_X, test_X_init, train_y, test_y
    """

    forecast_series = get_n_last_days(df, series_name, series_days).values # reducing our forecast series to last n days

    train = forecast_series[:-test_days] # training data is remaining days until amount of test_hours
    test = forecast_series[-test_days:] # test data is the remaining test_hours
    print(test.shape)
    print(train.shape)

    train_X, train_y = [], []

    # range 0 through # of train samples - input_hours by sample_gap.
    # This is to create many samples with corresponding
    for i in range(0, train.shape[0]-input_days, sample_gap):
        train_X.append(train[i:i+input_days]) # each training sample is of length input hours
        train_y.append(train[i+input_days]) # each y is just the next step after training sample

    train_X = get_keras_format_series(train_X) # format our new training set to keras format
    train_y = np.array(train_y) # make sure y is an array to work properly with keras

    # The set that we had held out for testing (must be same length as original train input)
    test_X_init = test[:input_days]
    test_y = test[input_days:] # test_y is remaining values from test set

    return train_X, test_X_init, train_y, test_y
```

Figure 5 Utility Functions for Extracting Formatted Dataset

In this model development, I used the last 1500 days of the stock price to train and test the model. The RNN model will take in 30 days High stock price in order to predict High stock price of the next time step. Here I used the last 50 days of data for visually testing the model. The train-test split and the obtained dataset summary are as below. Model summary is shown in Figure 6.

```
series_days = 1500
input_days = 30
test_days = 50

train_X, test_X_init, train_y, test_y = \
    (get_train_test_data(df_Stock, 'High', series_days,
                         input_days, test_days))
```

```
print('Training input shape: {}'.format(train_X.shape))
print('Training output shape: {}'.format(train_y.shape))
print('Test input shape: {}'.format(test_X_init.shape))
print('Test output shape: {}'.format(test_y.shape))
```

```
Training input shape: (474, 30, 1)
Training output shape: (474,)
Test input shape: (30,)
Test output shape: (20,)
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_3 (SimpleRNN)    (None, 50)                2600

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 2,651
Trainable params: 2,651
Non-trainable params: 0
_________________________________________________________________
```

Figure 6 Model Summary of RNN 1

The simple baseline RNN contains 50 cells and trained in 50 epochs. The training process of the model is shown as below. The training time of this model is negligible. For the purpose of testing, the trained model is applied to the last 50 days data. As shown in Figure 6, the prediction is a recursive process. The first 30 days of the testing dataset are used as the initial predictors, and the stock price of the 31st day is predicted. Then, I shifted forward one day to use 2nd to 31st day's stock price as the predictors to predict the 32nd day's stock price, and so on so forth.

As shown in this graph, the prediction of the model is significantly different from the true values. The model is badly underfit and essentially just making constant predictions. This is not unexpected as this is a very simple model.

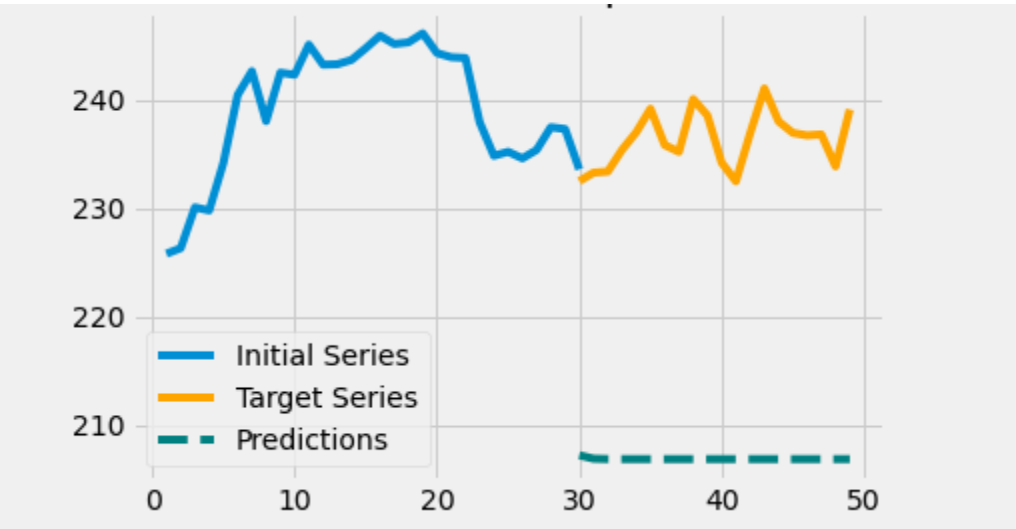

Figure 7 Visual Test of RNN Model 1

3.2 Train RNN-based Model 2

The second CNN model is more sophisticated than the first model. In comparison with the first CNN model, this model contains 100 cell units and was trained in 1000 epochs. The model summary is shown in Figure 8. The model summary shows that the model contains over 10, 000 parameters. Even if this is not a very complicated model, the training process has already taken several minutes.

```
model = fit_SimpleRNN(train_X, train_y, cell_units=100, epochs=1000)
predict_and_plot(test_X_init, test_y, model,
                 'PM Series: Test Data and Simple RNN Predictions')
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_2 (SimpleRNN)    (None, 100)               10200

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 10,301
Trainable params: 10,301
Non-trainable params: 0
_________________________________________________________________
```

Figure 8 Model Summary of RNN 2

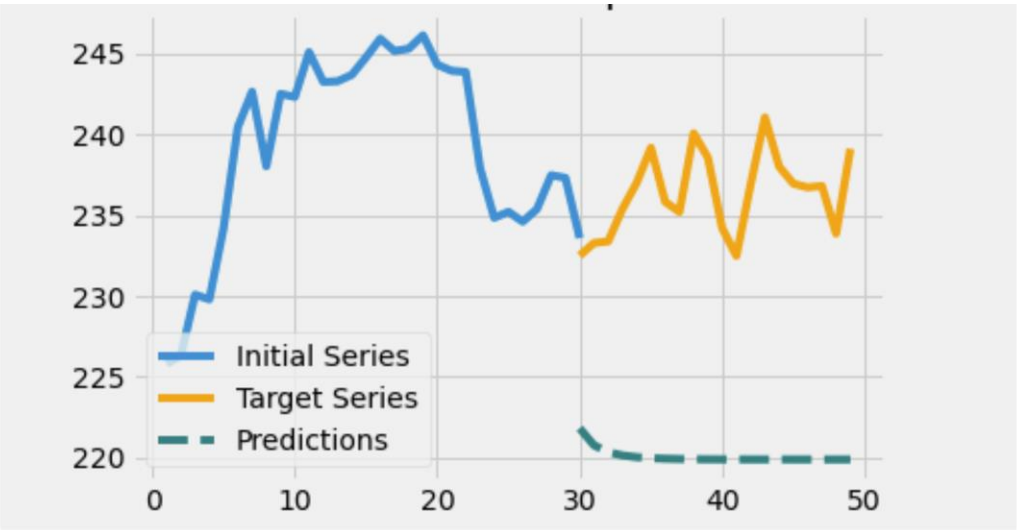

Figure 9 Visual Test of RNN Model 2

Figure 9 shows the testing result of RNN Model 2. Unfortunately, the model prediction is still poor. The prediction around 31st day stock price is slightly better than RNN model 1.

3.3 Train LSTM Model 1

In this section, a LSTM model is trained as an enhancement to the RNNs. In comparison with RNN, the advantage of LSTM is that LSTM performs better in capturing long-term auto correlation by using additional special units. LSTM includes a 'memory cell' that can maintain information in memory for long periods of time. The function of fitting the LSTM is as below. As shown in Figure 10, the predictive capability of the LSTM does not have substantial improvement. The main reason is that the architecture of this LSTM model is relatively simple. We will continue to develop a more complicated LSTM in next section.

```
def fit_LSTM(train_X, train_y, cell_units, epochs):
    """
    Fit LSTM to data train_X, train_y

    arguments
    ---------
    train_X (array): input sequence samples for training
    train_y (list): next step in sequence targets
    cell_units (int): number of hidden units for LSTM cells
    epochs (int): number of training epochs
    """

    # initialize model
    model = Sequential()

    # construct a LSTM layer with specified number of hidden units
    # per cell and desired sequence input format
    model.add(LSTM(cell_units, input_shape=(train_X.shape[1],1))) #,return_sequences= True))
    #model.add(LSTM(cell_units_l2, input_shape=(train_X.shape[1],1)))

    # add an output layer to make final predictions
    model.add(Dense(1))

    # define the loss function / optimization strategy, and fit
    # the model with the desired number of passes over the data (epochs)
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(train_X, train_y, epochs=epochs, batch_size=64, verbose=0)

    return model
```

The LSTM contains 70 cells and trained on 1000 epochs as below. Still as before, the last 1500 days' data are used to train the model and last 50 days data are used to test the model.

```
series_days = 1500
input_days = 30
test_days = 50

train_X, test_X_init, train_y, test_y = \
    (get_train_test_data(df_Stock, 'High', series_days,
                         input_days, test_days))

model = fit_LSTM(train_X, train_y, cell_units=70, epochs=1000)

predict_and_plot(test_X_init, test_y, model,
                 'PM_Nongzhanguan Series: Test Data and LSTM Predictions')
```

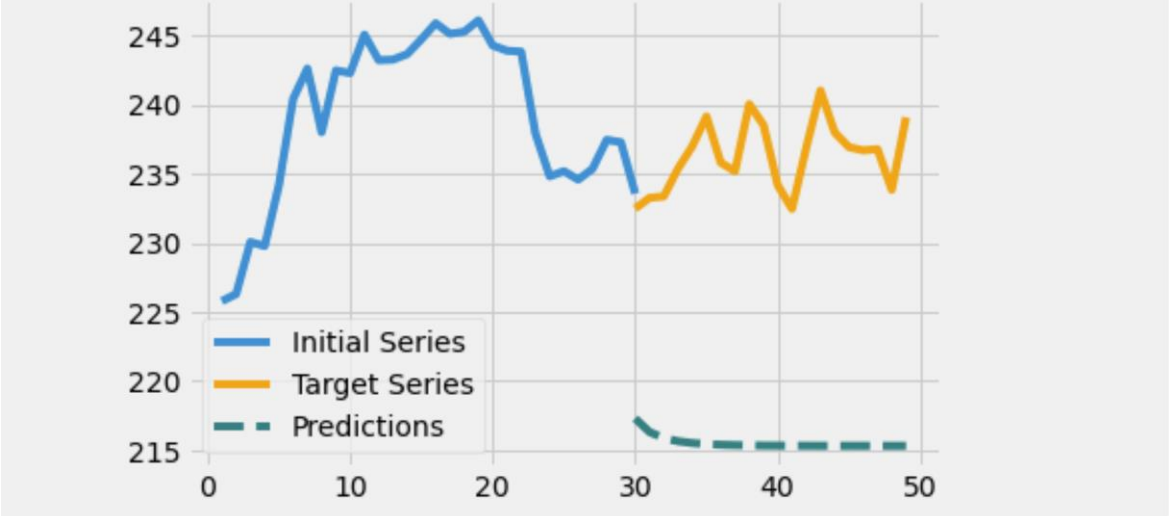

Figure 10 Visual Test of LSTM Model 1

3.4 Train LSTM Model 2

The last LSTM made several key improvements. Firstly, the previous three models only used the time series of High stock price to predict future High stock price. However, the LSTM Model 2 takes all the five columns (Open, High, Low, Close and Volume) as predictors to predict the future High stock price. Secondly, the structure of the LSTM Model 2 is much more complicated than LSTM Model 1 as shown in Figure 11 which contains over 240, 000 parameters. Third, the Min-Max scaler used to standardize the stock price before training the model.

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 30, 30)            4320

 dropout (Dropout)           (None, 30, 30)            0

 lstm_2 (LSTM)               (None, 30, 60)            21840

 dropout_1 (Dropout)         (None, 30, 60)            0

 lstm_3 (LSTM)               (None, 30, 80)            45120

 dropout_2 (Dropout)         (None, 30, 80)            0

 lstm_4 (LSTM)               (None, 30, 100)           72400

 dropout_3 (Dropout)         (None, 30, 100)           0

 lstm_5 (LSTM)               (None, 120)               106080

 dropout_4 (Dropout)         (None, 120)               0

 dense_5 (Dense)             (None, 1)                 121

=================================================================
Total params: 249,881
Trainable params: 249,881
Non-trainable params: 0
_________________________________________________________________
```

Figure 11 LSTM Model 2 Structure

The model is trained in 20 epochs and the batch size is 48. The training process is as below. The training process is very efficient, which takes less than 1 minute. To better test the performance of the model in predicting long term trend, Figure 12 test the predictive capability of the model on the last 300 days of stock price. The figure shows that the model can in general make a very good prediction of the stock price.

```
Epoch 1/20
25/25 [==============================] - 5s 70ms/step - loss: 0.0088
Epoch 2/20
25/25 [==============================] - 2s 72ms/step - loss: 0.0031
Epoch 3/20
25/25 [==============================] - 2s 74ms/step - loss: 0.0026
Epoch 4/20
25/25 [==============================] - 2s 73ms/step - loss: 0.0025
Epoch 5/20
25/25 [==============================] - 2s 72ms/step - loss: 0.0027
Epoch 6/20
25/25 [==============================] - 2s 72ms/step - loss: 0.0027
Epoch 7/20
25/25 [==============================] - 2s 70ms/step - loss: 0.0027
Epoch 8/20
25/25 [==============================] - 2s 71ms/step - loss: 0.0027
Epoch 9/20
25/25 [==============================] - 2s 73ms/step - loss: 0.0026
Epoch 10/20
25/25 [==============================] - 2s 73ms/step - loss: 0.0023
Epoch 11/20
25/25 [==============================] - 2s 73ms/step - loss: 0.0024
Epoch 12/20
25/25 [==============================] - 2s 74ms/step - loss: 0.0024
Epoch 13/20
25/25 [==============================] - 2s 73ms/step - loss: 0.0022
Epoch 14/20
25/25 [==============================] - 2s 76ms/step - loss: 0.0023
Epoch 15/20
25/25 [==============================] - 2s 74ms/step - loss: 0.0019
Epoch 16/20
25/25 [==============================] - 2s 75ms/step - loss: 0.0020
Epoch 17/20
25/25 [==============================] - 2s 75ms/step - loss: 0.0023
Epoch 18/20
25/25 [==============================] - 2s 78ms/step - loss: 0.0021
Epoch 19/20
25/25 [==============================] - 2s 75ms/step - loss: 0.0018
Epoch 20/20
25/25 [==============================] - 2s 75ms/step - loss: 0.0019
```

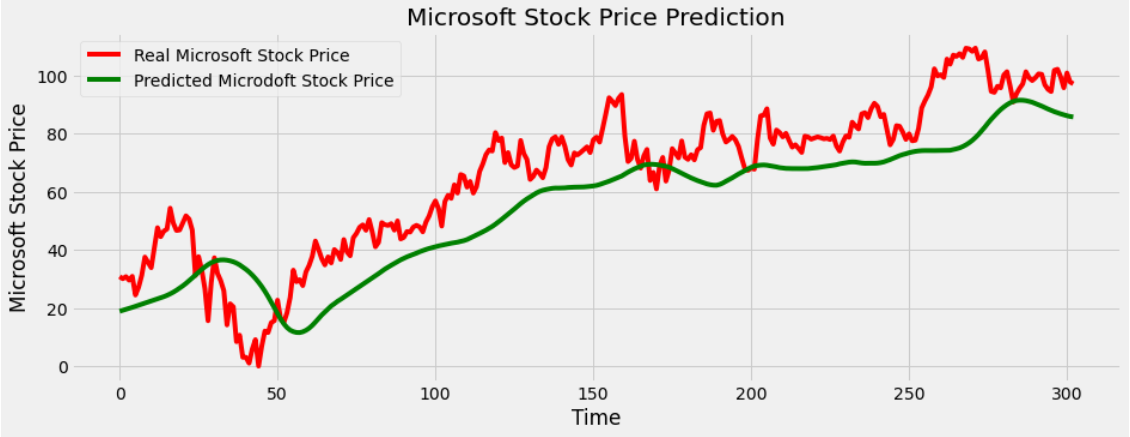

Figure 12 Test of LSTM Model 2

4 Summary of Key Findings and Future Work

This course project developed two RNN-based models and two LSTM-based models to predict stock price using the Microsoft Stock Price downloaded from Data Repository of Kaggle. RNN Model 1 has a very simple structure to be used as baseline model. As a result, the model can barely predict the future stock price. The structure of RNN Model 2 is much more complicated. However, the predictive performance is only slightly better than the first model. LSTM Model 1 is developed from the scratch which contains a relatively simple structure. The performance of LSTM Model 1 is comparable to RNN Model 2. Lastly, a highly complicated LSTM Model 2 was trained, which can make highly accurate prediction of stock price of up to 300 days.

There are three main reasons for the high performance of the LSTM Model 2. First, the architecture is relatively complex. Second, all the five columns are used as predictors. Third, the features are standardized before model training.

To sum up, the course project found that, with a proper setup, the LSTM has the potential to make relatively accurate prediction of stock price.

Due to the time limit, this course project did not set up a standard process of parameter fine-tune. In the future, the hyper-parameter of the four models can be tuned in a more appropriate and automatic approach, which may lead to even better performance of the LSTM-based model.