Recurrent Neural Networks for Regression

Creating the data

To predict the next day’s stock price using the previous three days' "Volume and Open, High, and Low" prices. First of all, load the original provided dataset (q2_dataset), which contains date, close/last, Volume, Open, High, and Low.

And store the values of "Volume, Open, High, and Low" values for the number of days (here is 3), where each sample has 12 features and next days (4th day here) "Open" value in 13th column of list as the target and append them all together as the new dataset.

Then, randomize the created dataset and split it into 70% training and 30% testing (as specified in the problem statement), and save them as train_data_RNN.csv and test_data_RNN.csv files in the data directory, respectively.

Pre-processing

In the preprocessing step, First load the train dataset (train_data_RNN.csv file) and then separate the features (as x_train) and target (y_train).

Now Standard scaling normalization is performed to the features in training set (x_train). The data is normalized to have a mean of 0 and a standard deviation of 1. The input data was then reshaped into the appropriate shape for the LSTM model.

Design steps

For network design, I initially experimented with SimpleRNN layers, but encountered challenges with high loss and convergence issues. To address this, I introduced LSTM layers, known for their effectiveness in handling time-series data with both long-term and short-term dependencies.

To optimize the model's performance, I iteratively adjusted the unit numbers of each layer, starting from small values and gradually increasing them. The goal was to achieve improved results on the testing set. While exploring different approaches, I attempted to add dropout layers, but this resulted in increasing training loss and extended training epochs, maybe due to the losing useful information.

Additionally, I tried incorporating more LSTM layers to increase the number of parameters, but this adjustment also increased runtime without significant improvement in model performance.

Network Architecture

The final network has two LSTM layers with 48 and 82 units, respectively, followed by a fully connected (Dense with 1 unit) output layer with linear activation, the model has total parameters of 52,651.

The model is trained for 155 epochs with a batch size of 8 to reach the optimum solution and minimize the loss. For assessing the model's performance, I selected the mean square error (MSE) as the loss function, and the mean absolute error (MAE) as the evaluation metric. The "Adam" optimizer is used, which is a popular version of gradient descent because it automatically tunes itself and gives good results.

```python
model = Sequential()

# Input layer
model.add(LSTM(units = 48, return_sequences = True, input_shape = (x_train.shape[1],1)))

# 1st Hidden layer
model.add(LSTM(units = 82, return_sequences = False))

#Output layer
model.add(Dense(units = 1, activation='linear'))

model.summary()

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

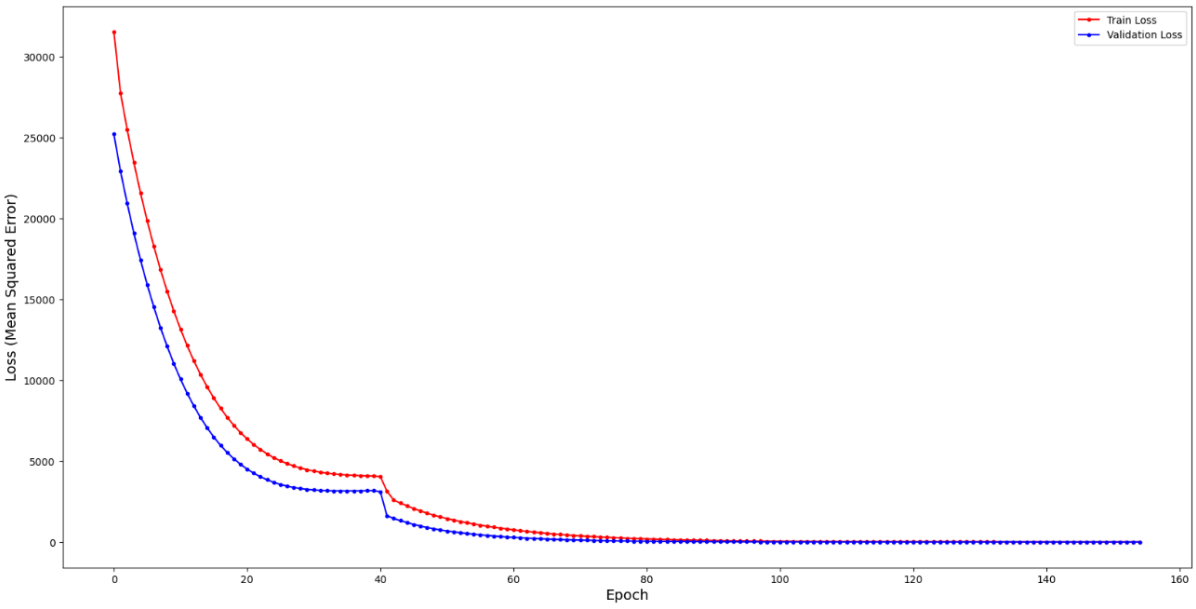
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 12, 48)            9600

 lstm_1 (LSTM)               (None, 82)                42968

 dense (Dense)               (None, 1)                 83

=================================================================
Total params: 52,651
Trainable params: 52,651
Non-trainable params: 0
_________________________________________________________________
```

Output of training loop

From the below training-validation loss figure, it can be seen the loss of training and validation decreases with the increase of training epochs. In the beginning, it decreases very fast, however, after a few epochs, it converges to about 4500, and then begins to decrease after that, finally it converged to training loss about 10, validation loss about 9. Training MAE is about 2.0 and validation MAE is about 2.06.

```
Epoch 155/155
88/88 [==============================] - 1s 16ms/step - loss: 10.6921 - mae: 2.0061 - val_loss: 9.1693 - val_mae: 2.0624
```

This is possibly caused by the effect of Adam optimizer. Adam updates the learning rate, which is different to classical stochastic gradient descent that maintains a single learning rate and the learning rate does not change during training.

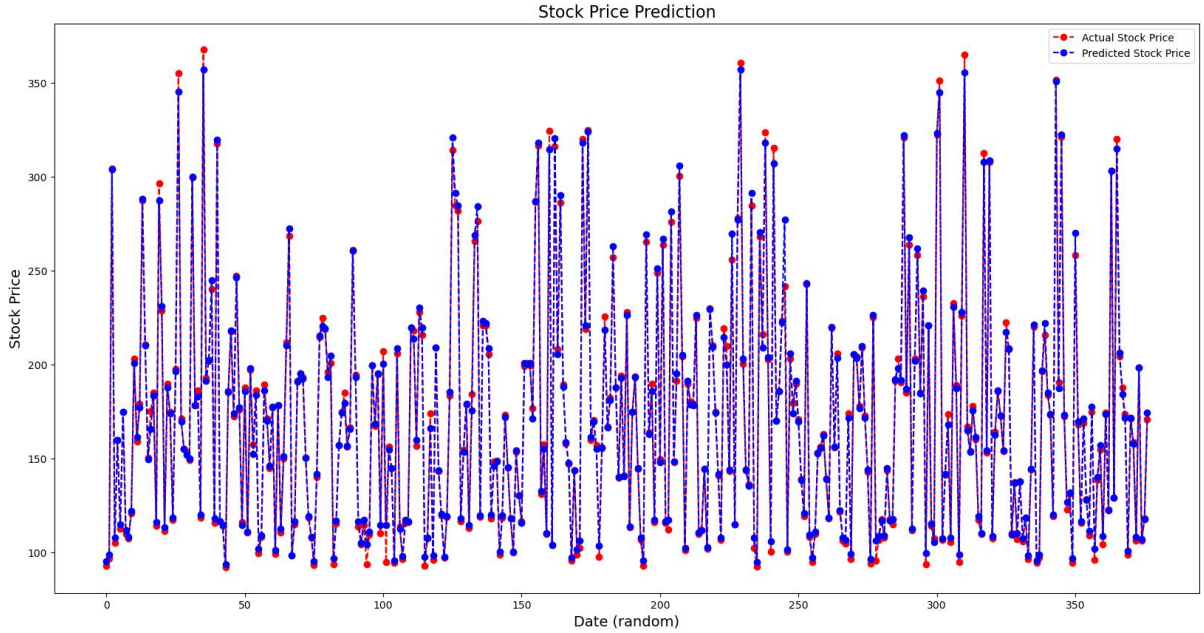


Output of testing

After loading the test data (test_data_RNN.csv) from data directory and applying the data pre-processing as done with train data, and loading the trained model (21046099_RNN_model.model) from models directory, to predict and evaluate the test set on model.

The MSE loss on testing dataset is about 13.5, and the MAE is 2.16. From the below figure, we can see most predictions are similar to the true value, but there are also a few predictions that are not close to the testing targets.

```
12/12 [==============================] - 1s 9ms/step
12/12 [==============================] - 1s 10ms/step - loss: 13.3553 - mae: 2.1640
```



Use more days for features

To use more days as features to train the network, I change the variable "days". After increasing the number of days to 4, the training MAE improved to 1.85, and with days set to 5, the training MAE reached 1.97. The model's performance showed some enhancement when including more days as features, as this provided the network with an increased amount of historical context to make predictions.

However, it is crucial to note that increasing the number of features does not always guarantee better predicted results. The inclusion of more days may introduce higher complexity and noise, leading to a more challenging training process and potentially poorer performance on the test data.