SCHOOL OF ADVANCED TECHNOLOGY

SAT301 FINAL YEAR PROJECT

Lifelong Machine Learning for Regression Problems

Final Thesis

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Engineering

Student Name	:	Yihan Zhang
Student ID	:	1823149
Supervisor	:	Steven Guan
Assessor	:	Rui Yang

Abstract

Human beings are able to learn different knowledge and skills incrementally and obtain them in the brains for a long time. Furthermore, humans can utilize the accumulated knowledge and transfer them to new tasks or domains. Lifelong machine learning is an advanced paradigm, different from traditional machine learning models, lifelong machine learning aims to make machines behave like human beings that can accumulate knowledge and deal with a variety of tasks. Regression problem is an important research field of machine learning that builds the functional relationships between variables and results. The outputs are usually the continuous prediction values. Though lifelong machine learning is not a new topic, there is a little research on the lifelong machine learning for regression problems. Stock price plays an essential role in the business. Moreover, have a precise prediction of stock price is significant for both investors and shareholders. With the help of lifelong machine learning, it will be easier to handle a large amount of data of business. Therefore, this paper will overview the development history of lifelong machine learning and propose a framework for regression problems. The experiment is to predict the stock price based on different kinds of models including long short-term memory (LSTM), gate recurrent unit (GRU) and bi-directional long short-term memory (Bi-LSTM). Structures as well as key components inside three models will be introduced in detail. The programming language of the code is Python. Training time as well as the mean square error will be compared as the criterion to measure the performances of models.

Keywords: Lifelong Machine Learning; Regression; Long Short-Term Memory; Gate Recurrent Unit; Bi-directional Long Short-Term Memory

Acknowledgements

First and foremost, I would like to thank my supervisor, Steven Guan. He gives me a lot of useful as well as practical suggestions on the topic, which helps me to finish this project, and I would like to thank Professor Rui Yang for his helpful and objective suggestions on my interim report, which helps me greatly to revise the final report.

Moreover, I would like to thank the Ph.D. student of Steven Guan, Xianbin Hong. He provided me a lot of suggestions and ideas at the beginning of the project.

Furthermore, I would like to thank all of teachers who taught me during previous four years. Without this knowledge, it is difficult to finish this project.

In addition, I would like to thank the teacher Mu Li for uploading online machine learning courses, which definitely helps me to understand the topics.

Meanwhile, I would like to thank my roommates when I lived in the dormitory. I want to thank them for providing a quiet and harmonious atmosphere in the dormitory.

Finally, I would like to thank my parents for their support and companionship. Thank them for their physically and spiritually help, especially during this Covid-19 pandemic period when I lived at home.

Contents

List of Tables

List of Figures

List of Acronyms

LML	Lifelong Machine Learning
KB	Knowledge Base
PIS	Past Information Store
KBL	Knowledge-based Learn
KM	Knowledge Miner
KR	Knowledge Reasoner
SVM	Support Vector Machine
MSE	Mean Squared Error
RNN	Recurrent Neural Network
EBNN	Explanation-Based Neural Network
ELLA	Efficient Algorithm for Lifelong Learning
LSTM	Long Short-Term Memory
GRU	Gate Recurrent Unit
Bi-LSTM	Bidirectional Long Short-Term Memory
RMSprop	Root Mean Square prop
EWC	Elastic Weight Consolidation
BERT	Bidirectional Encoder Representation Transformers
GTP-2	Generative Pre-Training 2.0
GEM	Gradient Episodic Memory
f_t	Forget Gate
I_t	Input Gate
O_t	Output Gate
C_t	Memory Cell
h_t	Hidden Layer State
w	Weight Matrices
b	Bias Vector

Chapter 1

Introduction

1.1 Motivation, Aims and Objective

Machine learning is the subfield of artificial intelligence which focus on how to teach computer to learn from data, and it can be applied in many fields except for computer science. However, traditional machine learning has some shortcomings. One defect is that machines or models are not able to learn continuously like a human, which means they are not able to memorize knowledge from the past and utilize them into the future. Therefore, the concept of lifelong machine learning (LML) is put forward. Another motivation is that, though lifelong machine learning is not a new concept, most of studies are related to classification problems and natural language processing field, there are a little research on regression field.

During these years, with the rapid development of Internet, the online business is becoming more prosperous. Stock price is an important indicator for a company, which shows the power of the company. It is significant for both investors and shareholders to have a precise prediction of the stock price. Companies with high stock prices can attract more investors to participate, which is the chain effect. Generally, the prospects of companies with high stock prices are optimistic.

In this report, firstly, the development history of lifelong machine learning as well as some related models will be introduced. Then, a simple lifelong machine learning framework for regression problems will be proposed, the steps inside the framework will be introduced separately. In addition, the three models applied in the framework including long short-term memory, gate recurrent unit and bi-directional long short-term memory will be introduced. Important features of them will be introduced in detail separately. Furthermore, experiment results about the stock prediction problem will be illustrated in the picture format, including the whole process of the experiment, from how to process the data to the prediction. The comparison results of training time and mean square error among three models will be showed. Finally, the report will conclude the whole project, discuss some limitations and correspondingly possible improvements for the future work.

1.2 Literature Review

In order to deal with the inadequate of the original representation space for the problem at hand, Michalski [1] proposed the theory of constructive inductive learning in the mid 1980s. He discussed the principle that new knowledge is easier to induce if the search is done using the right representation. To be more specific, the new knowledge can be hypothesized through two interrelated searches, one search is used for representational space for hypotheses and another search is used for the best hypothesis within the current representational space.

In 1986, Rumelhart [2] proposed the concept of recurrent neural network named by Hopfield network, which created the internal states of the network. RNNs are suitable to learn from sequential data.

In 1989, catastrophic forgetting, which is an issue in neural network was proposed by Cohen and McCloskey [3]. Through their research, the performance of neural network for previous tasks drops because it tends to forget information learned in the previous tasks while training for the new tasks. This kind of issue was also referred to as the stability and plasticity dilemma proposed by Robins and Abraham in 2005 [4], which means if a model is too stable, it could not learn the new knowledge. On the contrary, if a model is plastic, it might forget previous knowledge.

In the mid 1990s, Thrun and Mitchell proposed the explanation-based neural network (EBNN) [5]. The key of EBBN is that, when facing a new task, it utilizes the domain knowledge from previous tasks, which are the back-propagation gradients of prior learned tasks. This is the early idea of transfer learning.

In 1997, the architecture of LSTM, which is the abbreviation of long short-term memory was proposed by Hochreiter and Schmidhuber [6]. The motivation of the proposal was to deal with issues in conventional back-propagation. Meanwhile, the paper discussed some limitations of LSTM.

Lifelong machine learning was firstly named by Thrun and Mitchell in 1995 [7]. Ruvolo and Raton [8] proposed an efficient algorithm for lifelong machine learning called ELLA in 2013, which is the abbreviation of efficient lifelong learning algorithm. The accuracy of this algorithm can reach that of multi-task learning (MTL) in three orders of magnitude less time, which means it has lower computational cost and higher performance.

In 2014, Kyunghyun [9] proposed gate recurrent unit (GRU), which is a variant of LSTM that has fewer gates and parameters to train. Meanwhile, the paper proposed Encode-Decoder model for typical statistical machine translation task.

In 2016, Chen and Liu [10] introduced the architecture of lifelong machine learning and summarized the differences between traditional machine learning and lifelong machine learning. With respect to traditional machine learning models, for instance: linear regression, decision tree, AdaBoost, Support Vector Machine, they determine the target tasks firstly, then utilize the data for training, and then use the trained model to complete the tasks. These models are not intelligent because they have no memory, which is the knowledge base (KB) in LML, as the result, these models are not able to utilize what they have learned into new tasks in the future. As the result, this property leads to the situation that during the training stage, these types of models require a large amount of data. Figure 1 illustrates the brief process of traditional machine learning model.

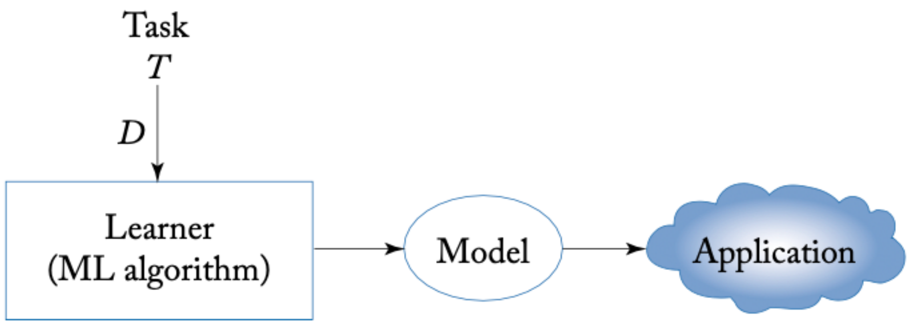


Figure 1: Traditional Machine Learning Model [10]

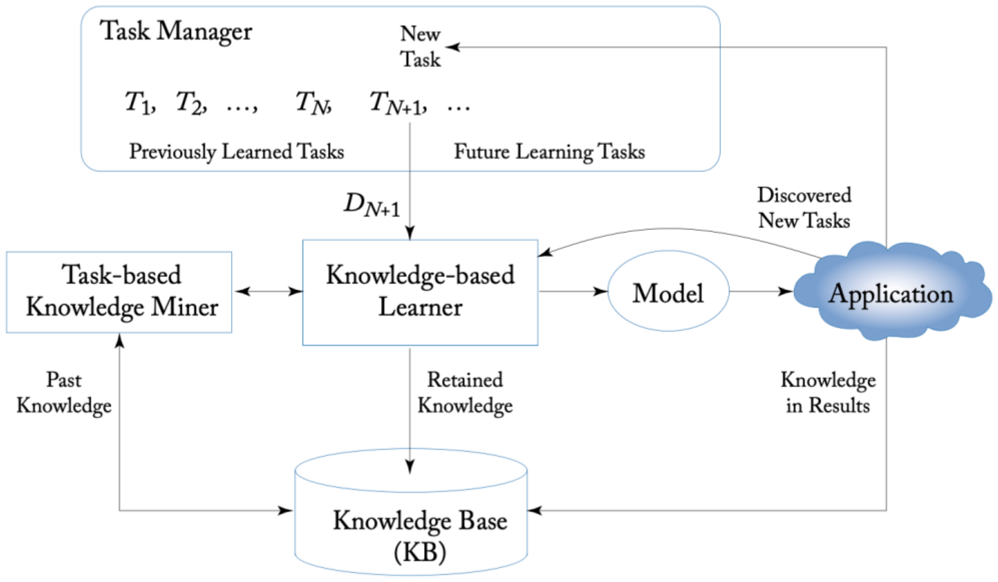


Figure 2: Lifelong Machine Learning System Architecture [10]

With the idea above, the architecture of lifelong machine learning system has been proposed. It is clear to observe that the architecture is more complicated compared with the traditional machine learning model. However, this figure is only for illustration purpose, it is not always the case that all the components will be used in the real situations. For traditional machine learning models, each model can only deal with one task, but for LML model, it has a series of tasks rather than a single task, which can be defined as: $\{ T_1, T_2, \ldots T_N, T_{N+1}\}$

Introduction of key components of LML architecture [10]:

- **Knowledge Base (KB):** It is used for storing the learned knowledge from the past
- **Knowledge-based Learner (KBL):** It is used to leverage the knowledge in the knowledge base to learn the new task.
- **Task-based Knowledge Miner:** It is used to mine knowledge from the knowledge base for the new task.
- **Task Manager:** It manages tasks in the system and presents the new learning task to the knowledge-based learner.

In 2016, Chen [10] summarizes the differences between some paradigms with lifelong machine learning: for transfer learning, it only transfers the information from the source domain to the target domain once and does not retain the knowledge for the future, which is different from the accumulative knowledge idea of lifelong machine learning. Furthermore, transfer learning is unidirectional since it is only able to transfer source domain to the target domain restricted by the little knowledge in the target domain. For multitask learning, though there are many tasks can be completely by MTL, they can be reduced to one big task because it does not accumulate any knowledge over time, which is also different from the continuous learning idea.

Li and Hoiem [11] proposed the Learning without Forgetting method (Lwf) for convolutional neural networks in 2016, which is the combination of knowledge distillation [12] and fine-tuning that can preserve performance on original tasks without access to training data for previous tasks. The theory of this technique is to optimize the shared parameters for all tasks and parameters for the new task with constraint that keeps shared parameters and parameters for old tasks do not shift much. The disadvantage [11] of this algorithm is that, as the learned tasks increase, the training time will linearly enhance. Moreover, it requires the reservoir of persistent data for each learned task.

In 2016, Kirkpatrick [13] and his team proposed the elastic weight consolidation (EWC) algorithm aiming to deal with catastrophic forgetting issues occurred in the neural network.

The idea is that: in the neural network, not all of nodes has a great impact on the results. When learning a new task, reduce the weights of those useless nodes.

The survey about lifelong learning for sentiment classification conducted by Chen in 2018 [14] have showed that the performance of LML is better than multi-task learning model as well as single task learning model including Naïve Bayes and Support Vector Machine.
In 2017, Vaswani [15] and his team proposed the self-attention mechanism and the Transformer architecture for NLP tasks. Compared with typical RNN who has limited memory length and the data at time t_i can only be calculated after calculating the data at time t_{i+1}.Transformer can be parallelized. Transformer has been applied into NLP field such as Bidirectional Encoder Representation Transformers (BERT) as well as Generative Pre-trained Transformer 2 (GPT-2). Moreover, some articles creatively applied Transformer model to computer vision fields and achieved good results. This is also considered by many scholars to create a new era in the field of computer vision and might even completely replace the traditional convolution operation.

Loper-Paz and Ranzato [16] proposed the method to evaluate the performance of continual learning models in 2017. The idea is to measure the accuracy and the ability to transfer knowledge between different tasks. To be more specific, they defined three criteria including average accuracy, backward transfer, and forward transfer. The larger these indicators represent the more perfect the model is established. If the accuracies of two models are the same, the model who has the greater backward transfer and forward is better. In addition, their proposed the gradient episodic memory (GEM) algorithm in order to solve the catastrophic forgetting issue. They applied this proposed algorithm on MNIST Permutations, MNIST Rotations and CIFAR 10. Compared with the result with EWC, iCaRL and single model, the performance of proposed algorithm is better.

Chapter 2

Methodology and Results

2.1 Methodology

In methodology section, the framework for regression problems will be introduced, including some important components and steps. In the first semester, long short-term memory (LSTM) [6] model was chosen to be applied in the framework. In the second semester, gate recurrent unit (GRU) [9] and Bi-directional long short-term memory (Bi-LSTM) were chosen to be applied in the framework for comparison experiments. The training time and mean square error will be compared as the indicators to measure the performance of models. Structures as well as components of models will be introduced separately below.

2.1.1 Environment and Packages

This project is implemented using Python 3 (3.9) since it supports many useful machine learning and data processing packages. The code was written using the software PyCharm and Jupyter Notebook (easier to draw pictures). The pictures of framework and structures of models are drawn using the open-source software draw.io.

Numpy and Pandas are packages used to do process the data. Matplotlib is the package to generate pictures. Keras, Scikit-learn and TensorFlow are packages that provide a lot of machine learning models and utility functions. Figure 3 below shows the utility packages used in the project.

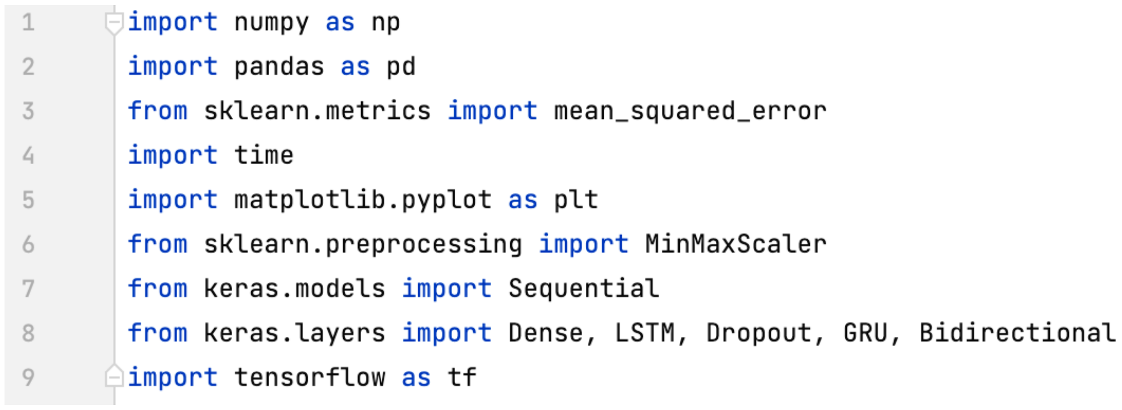

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
import time
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
import tensorflow as tf
```

Figure 3: Utility Packages

2.1.2 Proposed Framework for Regression Problems

Figure 3 shows the proposed framework based on long short-term memory.

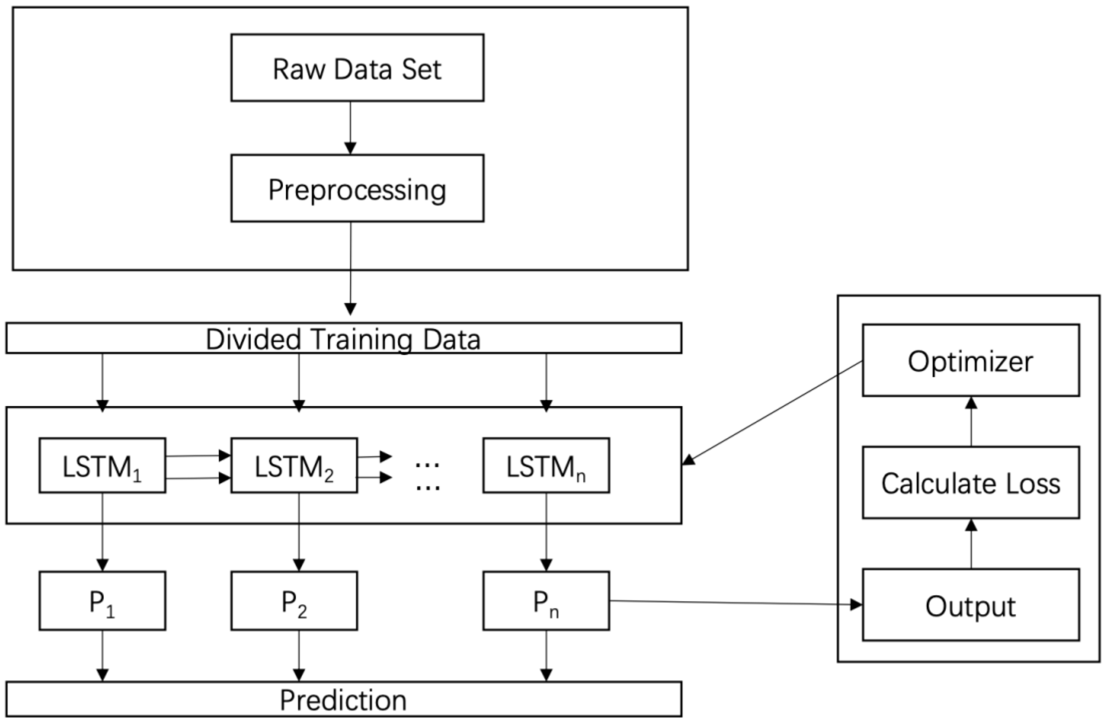
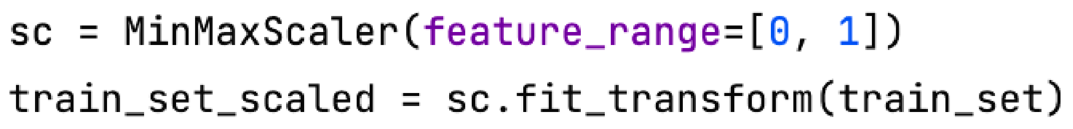


Figure 4: Proposed LSTM Based Framework for Regression Problems

To be more specific, it consists of six main steps. First of all, raw data need to be processed after observation, including changing dimensions and normalization. Normalization is a technique that scale the values into a range, which can appropriately reduce the training time and enhance the accuracy to some extent [17]. In this project, Min-max normalization was applied, the corresponding formular and the implementation code are as follows:

$$value' = \frac{value - min(value)}{max(value) - min(value)}$$

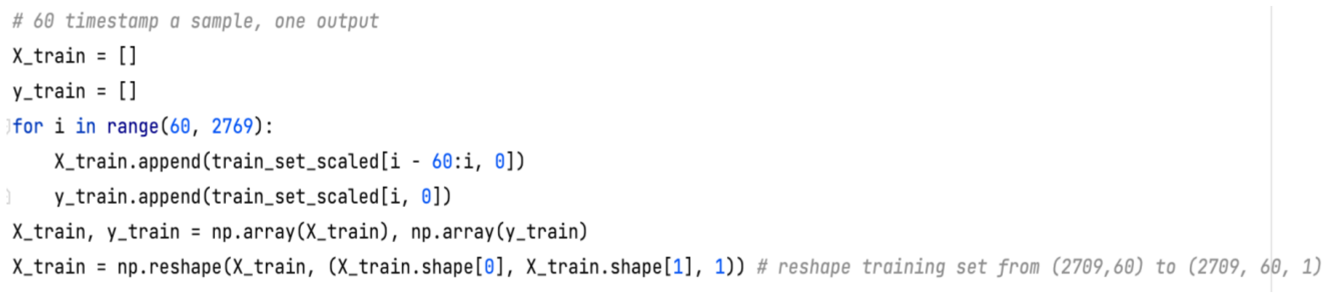


```
sc = MinMaxScaler(feature_range=[0, 1])
train_set_scaled = sc.fit_transform(train_set)
```

Figure 5: MinMaxScaler Function

It is worth noting that, both training set and testing set need to do be normalized.

Then the preprocessed data will be divided into two parts, one part of data is used for training and another part of data is used as testing data. Normally, the ratio of training set and testing set is 70% and 30%. Furthermore, since the project aims to predict stock price, the time-series data should be constructed using rolling method. In this project, stock price of one day is set aside every 60 days as a result. The implementation code of division step is as follows:



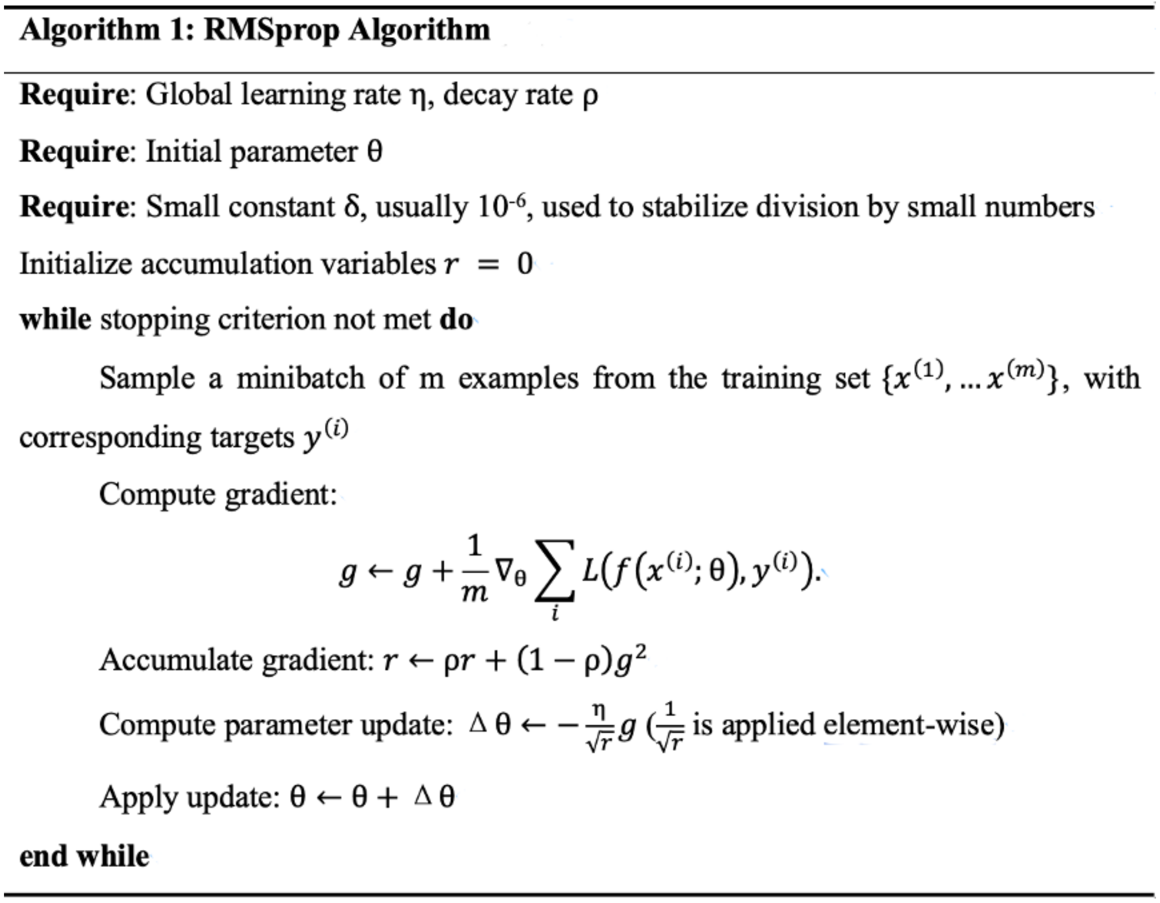
```
# 60 timestamp a sample, one output
X_train = []
y_train = []
for i in range(60, 2769):
    X_train.append(train_set_scaled[i - 60:i, 0])
    y_train.append(train_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1)) # reshape training set from (2709,60) to (2709, 60, 1)
```

Figure 6: Construct Training Set Code

Next step is to define layers and construct different models. During the training step, MSE , the abbreviation of mean squared error is selected as the loss function in this prediction task, the formular is as follow:

$$\mathrm{loss} = \frac{1}{\mathrm{i}} . \sum \left(\mathrm{y}_\mathrm{i}^2 - \widehat{\mathrm{y}_\mathrm{i}^2} \right)$$

Where i represents the number of samples, y_i is the ground truth, $\widehat{y_i}$ is the prediction value. Root Mean Square Prop (RMSprop) is selected as the optimizer, the algorithm of RMSprop is as follows [18]:



```
Algorithm 1: RMSprop Algorithm
Require: Global learning rate $\eta$, decay rate $\rho$
Require: Initial parameter $\theta$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers
Initialize accumulation variables $r = 0$
while stopping criterion not met do
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \dots x^{(m)}\}$, with corresponding targets $y^{(i)}$
    Compute gradient:
        $g \leftarrow g + \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
    Accumulate gradient: $r \leftarrow \rho r + (1 - \rho) g^2$
    Compute parameter update: $\Delta \theta \leftarrow -\frac{\eta}{\sqrt{r}} g$ ($\frac{1}{\sqrt{r}}$ is applied element-wise)
    Apply update: $\theta \leftarrow \theta + \Delta \theta$
end while
```

Table 1: RMSprop Algorithm [18]

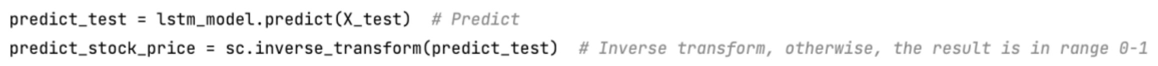
Adam algorithm is a combination of momentum as well RMSprop, the offset is corrected to make the parameters more stable. Adam has high computational efficiency and low memory requirements. Pseudocode of Adam algorithm is as follows [19]:

```
**Algorithm 2: Adam Algorithm**
Require: $m_t = 0, n_t = 0$ (Initialize moment vector to zero)
Require: α (Learning rate)
Require: $f(\theta)$ (Stochastic objective function with parameter θ)
Require: $\theta_0$ (Initial parameter vector)
while $\theta_t$ not converged do
        $t \leftarrow t + 1$
        $g_t \leftarrow \nabla\theta f_t(\theta_{t-1})$
        $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
        $n_t = \beta_2 n_{t-1} + (1 - \beta_2) g_t^2$
        $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$
        $\widehat{n_t} = \frac{n_t}{1-\beta_2^t}$
        $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\widehat{m_t}}{\sqrt{\widehat{n_t}}+\epsilon}$
end while
```

Table 2: Adam Algorithm [19]

m_t is the first order momentum term and n_t is the second momentum term. $\widehat{m_t}$ and $\widehat{n_t}$ are the revised value. α is learning rate. β_1 and β_2 are nonnegative weighted parameters and they are usually set to 0.9 and 0.999.

In order to enhance the performance of models, hyper-parameters such as the learning rate should be tuned correspondingly. In addition, after the prediction, the values are still in the range of zero to one, which is not the real stock price. Therefore, the data need to inverse the normalization. The implementation code is as follows:



```
predict_test = lstm_model.predict(X_test)  # Predict
predict_stock_price = sc.inverse_transform(predict_test)  # Inverse transform, otherwise, the result is in range 0-1
```

Figure 7: Inverse Normalization Implementation Code

Finally, it is better to illustrate the data using pictures instead of numbers, therefore, the results will be generated using pictures.

2.1.3 LSTM Introduction

The picture below illustrates the overall structure as well as components of long short-term memory:

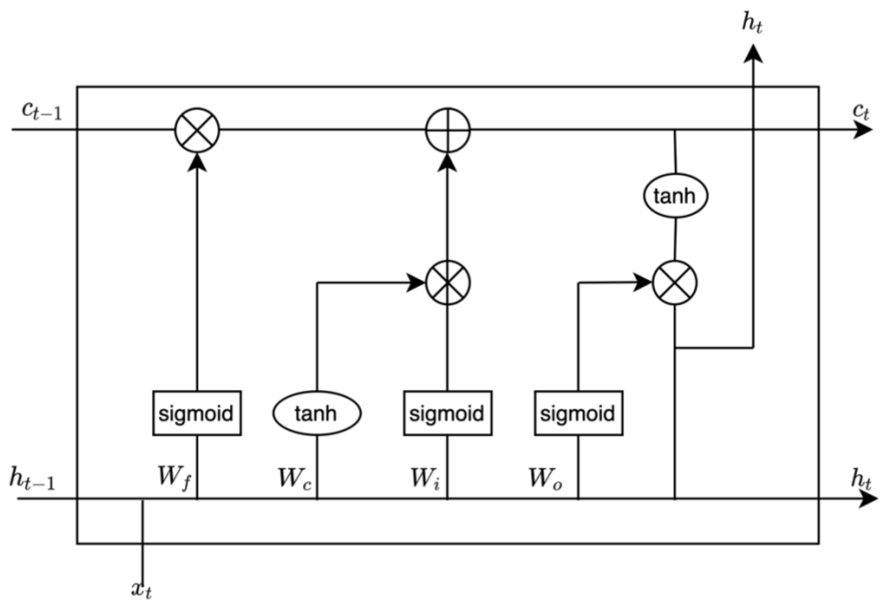


Figure 8: General Structure of Long Short-Term Memory

The appearance of LSTM solves the gradient vanish and explosion problems [6] of typical RNN, and LSTM has the ability to memorize the dependency between input sequence and time. It achieves excellent effects in handwriting recognition and speech recognition.

Input of LSTM: c_{t-1}, h_{t-1} and x_t.

Output of LSTM: c_t and h_t.

Basically, LSTM has three types of gates: forget gate, input gate and output gate [6], which will be introduced below in detail.

Forget gate decides how much information will be retained or throw through the sigmoid layer, it accepts h_{t-1} and x_t as the input. Output 0 to 1 values for c_{t-1} and label it as f_t. 0 means the gate will completely drop the value and 1 means the gate will completely retain the value.

The formular of forget gate is:

$$f_t = \text{sigmoid}\left(W_f.[h_{t-1}, x_t] + b_f\right)$$

$$\text{, where } sigmoid(x) = \frac{1}{1+\mathrm{e}^{-\mathrm{x}}} = \frac{e^x}{e^x+1} = 1 - sigmoid(-x)$$

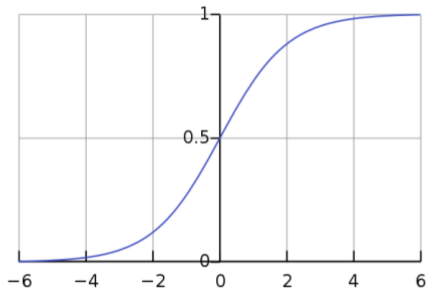


Figure 9: Sigmoid Function

Input gate has two parts: one is sigmoid layer, deciding what value should be updated, the probability is i_t and another one is a tanh layer, which will create a candidate vector $\widetilde{\mathrm{C}}_\mathrm{t}$, which will be added into the state. The formular of input gate it is:

$$\mathrm{i}_t = \text{sigmoid}(\mathrm{W_i}.[\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_i})$$

$$\widetilde{C_t} = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

$$\text{,where } tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$sinh(x) = \frac{e^x - e^{-x}}{2}$$

$$cosh(x) = \frac{e^x + e^x}{2}$$

Tanh function will constrain values into -1 to 1.

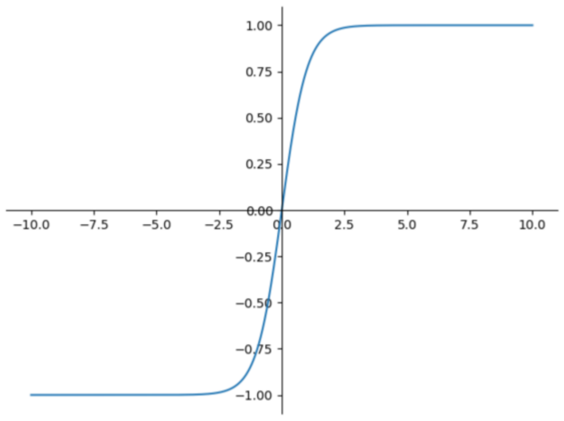


Figure 10: Tanh Function

The formular for updating the state, which means the previous state C_{t-1} multiply the value of forget f_t plus the temporal state multiply the memory value i_t

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

The output gate determines which part of state C_t should be output through sigmoid layer. Then use tanh layer to let the value in the range between -1 and 1, then multiply with output of sigmoid layer, the final result labels with h_t. The formular of output gate is as follows:

$$o_t = sigmoid(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

2.1.4 GRU Introduction

GRU is the abbreviation of gate recurrent unit is variant of LSTM, according to its name, it also uses the gate mechanism to control how much information should be memorized and dropped. Compared with LSTM, the structure of GRU is simpler and hence has fewer parameters. Figure 11 illustrates the general structure as well as components of GRU.

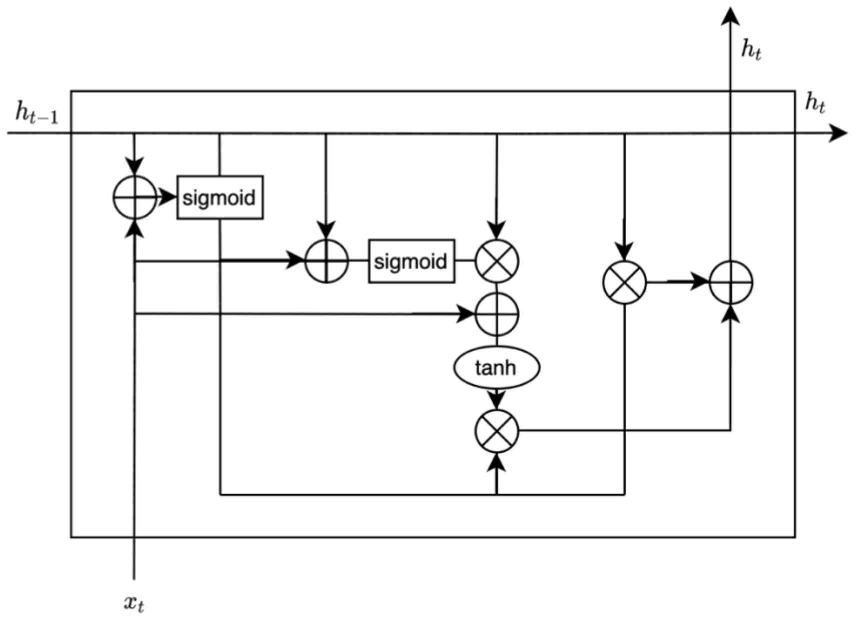


Figure 11: General Structure of Gate Recurrent Unit

Update gate as well as reset gate are two key components of GRU. In addition, reset is the combination of forget and input gate of LSTM, which makes the structure simple.

For reset gate that is used to decide how much information will be forgotten, it receives the previous cell's information and current input. Then use the sigmoid function to transform the values of the sum of weighted input and current state in the range of zero to one. The formular of reset gate is as follows:

$$r_t = sigmoid\left(W^{(r)}x_t + \mathrm{U}^{(\mathrm{r})}\mathrm{h}_{\mathrm{t-1}}\right)$$

The current memory content $\widetilde{h_t}$ is calculated by the sum of multiplication of input x_t with its weight W and multiplication of h_{t-1} with its weight U. Then the result will be transformed using tanh function:

$$\widetilde{h_t} = tanh(Wx_t + r_t \odot Uh_{t-1})$$

For update gate, similarly, it uses the previous state h_{t-1} and current x_t, dot product with their weights, then use a sigmoid function to transform the value in the range of zero to one.

$$z_t = sigmoid\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

Finally, the output h_t is calculated as follows and will be transferred to next cell:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \widetilde{h_t}$$

2.1.5 Bi-LSTM Introduction

One disadvantage of LSTM and GRU is that they can only learn from the previous data. Bi-LSTM, the abbreviation of Bi-directional long short-term memory, is also an improved variant of LSTM, but it is the combination of forward LSTM and backward LSTM, which enables to encode the information from back to forward. Figure 12 illustrates the general structure as well components of Bi-directional LSTM:

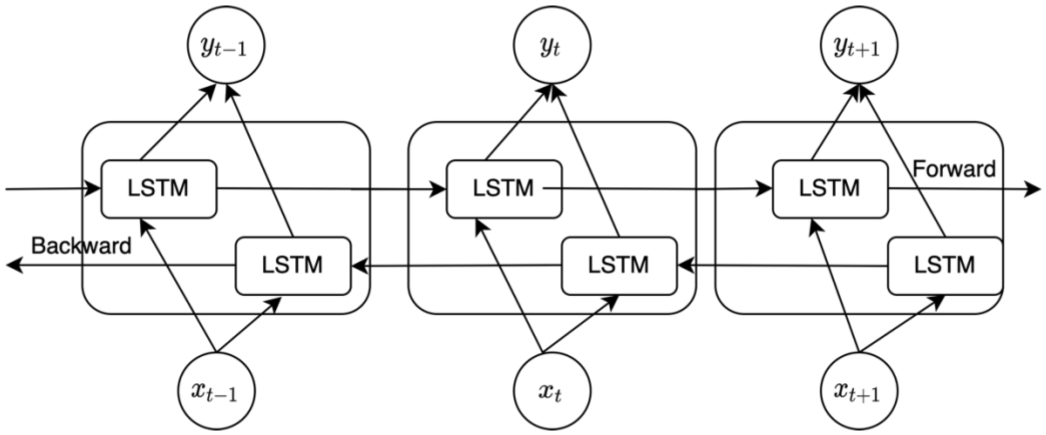


Figure 12: General Structure of Bi-directional LSTM

2.2 Results

2.2.1 Data Set Introduction

The data collected the stock price of IBM from 2006 to 2017. The reason why the project chose the stock price of IBM is that: IBM is a successful company, which has a long period history. Therefore, it can provide sufficient and stable data for training. Furthermore, the dataset is open source so that there is no need to spend time crawling the data.

Through the observation of dataset, there are 3020 rows. Since 70% of data is used for training and remaining 30% data is used for testing, there are 2769 rows of training data and 251 rows of testing data.

According to the picture below, the line in blue color represents the training data and the line in orange color represents the testing data. The x-axis represents the date and the y-axis represents the stock price of that date. The testing results of different models will be showed after the comparison among three models.

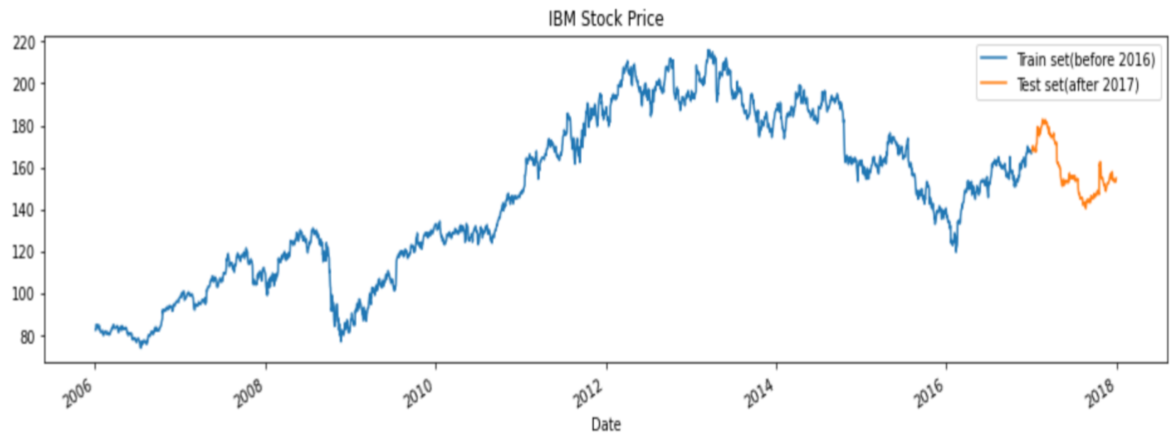


Figure 13: Overview of Data Set

2.2.2 Structure and Hyper-Parameters Details

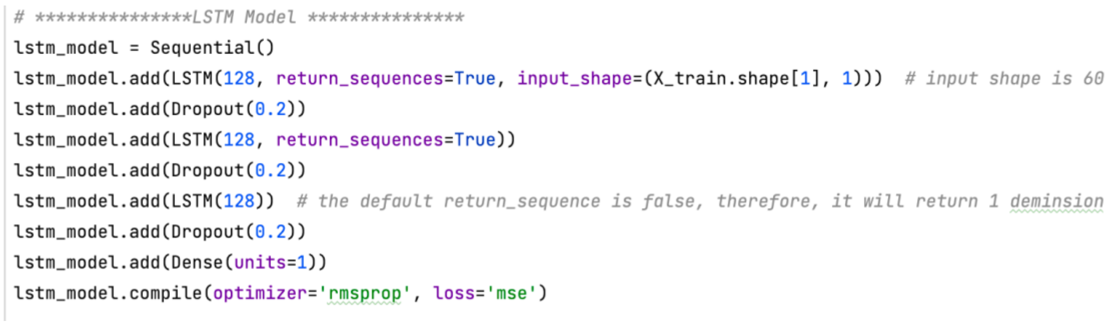
With respect to the designed structure of the LSTM based model, we use three LSTM layers as hidden layers, after each layer, we use a Dropout layer to drop twenty percent of data in order to prevent overfitting [20], finally, dense layer is used to output. The epoch is set to 20, the selected batch size is 32, the optimizer is RMSprop, which is an improved optimize algorithm of AdaGrad, and the loss function is the MSE. By the default settings of LSTM model of Keras, the value of learning rate is $\frac{1}{1000}$. The value of decay is zero, which means the learning rate remains unchanged.

Input Shape	60 dimensions
1st Layer	LSTM Layer

2nd Layer	Dropout Layer
3rd Layer	LSTM Layer
4th Layer	Dropout Layer
5th Layer	LSTM Layer
6th Layer	Dropout Layer
7th Layer	Dense Layer
Epoch	20
Batch Size	32
Optimizer	RMSprop
Learning Rate	0.001 (default)
Decay	0
Loss	MSE

Table 3: Structure and Hyper-parameters of Model

Figure 14 shows the implementation code for structure of LSTM based model.



```
# ***************LSTM Model ***************
lstm_model = Sequential()
lstm_model.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], 1)))  # input shape is 60
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(128, return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(128))  # the default return_sequence is false, therefore, it will return 1 deminsion
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(units=1))
lstm_model.compile(optimizer='rmsprop', loss='mse')
```

Figure 14: Structure of LSTM Based Model Code

2.2.3 Relationship between Epoch and Loss

The picture below illustrates the relationship between loss value and epoch. There are 20 epochs in total. It is lucid that in the beginning, the loss drops quickly from 0.025 to 0.01. After five epochs while the loss becomes 0.005, the loss drops slowly though still decreases.

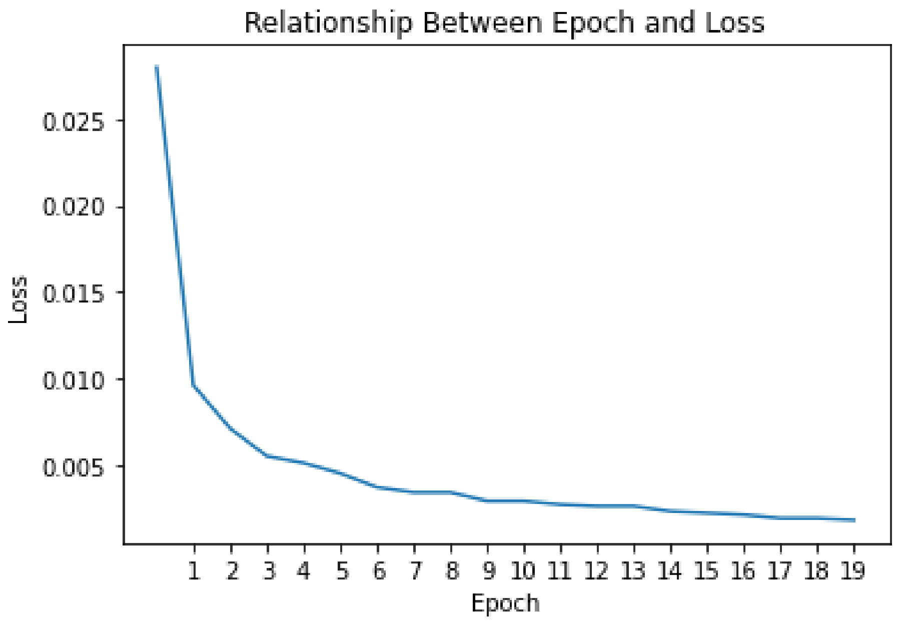


Figure 15: Relationship between Epoch and Loss

2.2.4 Training Time Comparison

The comparison experiment is also conducted in this project. The models are as follows:

LSTM based model

GRU based model

Bi-LSTM based model

The table and picture below are the comparison of training time among different models including LSTM, GRU and Bi-LSTM.

It can be depicted from the figure that GRU based model has the shortest training time.

	LSTM	*GRU*	*Bi-LSTM*
Round 1	44.4729	24.5032	38.4417
Round 2	23.7673	21.1196	39.3136
Round 3	24.3078	23.7143	39.4897
Average	30.8493	23.1124	39.0817

Table 4: Training Time Comparison of Different Models (Unit: s)

With respect to the figure below, the x-axis represents the round, and there are three rounds and one average round. The y-axis represents the training time, and the unit is second.

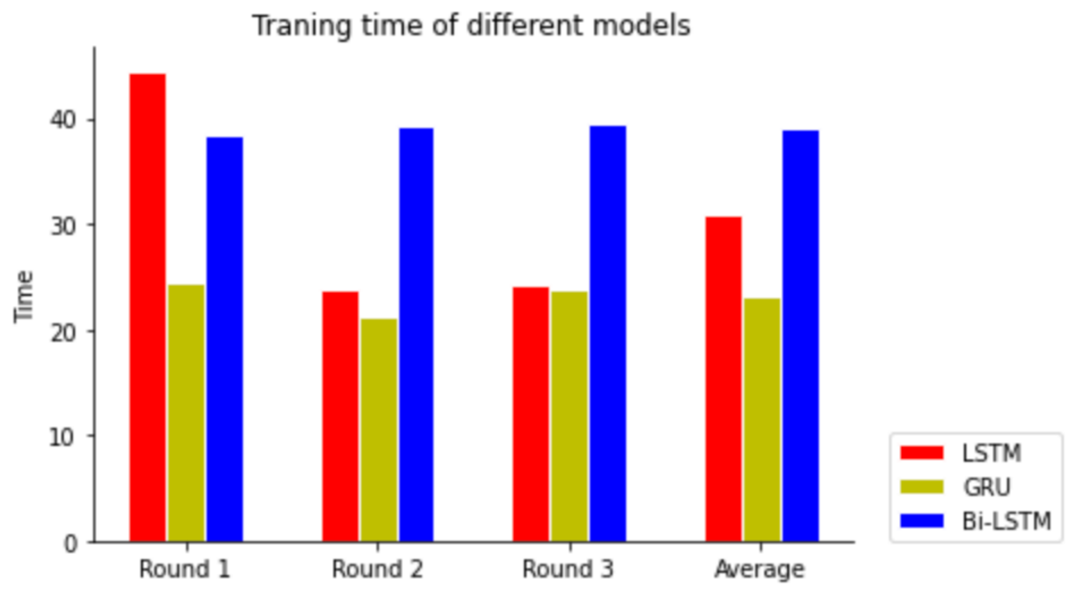


Figure 16: Training Time Comparison Bar Chart

2.2.5 Mean Square Error Comparison

The table and picture below are the comparison of mean square error of different models including LSTM, GRU and Bi-LSTM. There are three rounds involved in the experiment, take their average value:

	LSTM	*GRU*	*Bi-LSTM*
Round 1	4.3340	4.6110	4.6208
Round 2	7.6449	5.9706	5.2370
Round 3	4.4109	3.6978	5.7812
Average	5.4633	4.7598	5.2130

Table 5: Mean Square Error Comparison of Different Models (20 epochs)

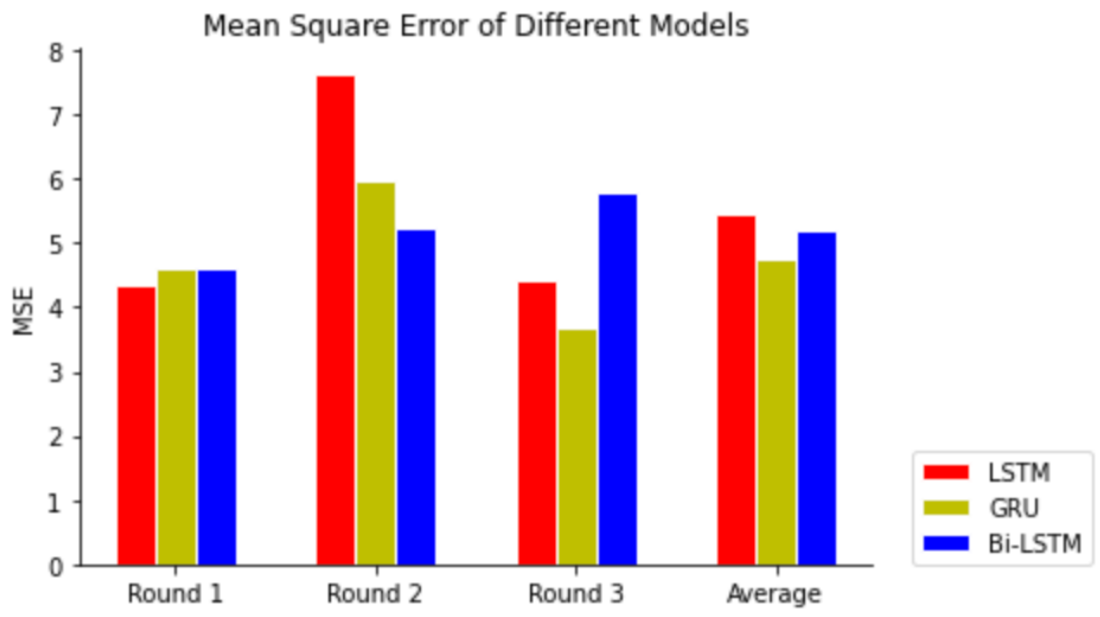


Figure 17: Mean Square Error Comparison Bar Chart

GRU has the lowest mean square error, which mean it has the highest accuracy in the prediction task. It is surprised that the result of Bi-LSTM is not good as that of GRU and even worse than LSTM in round one and two. The reason might be the parameters are not tuned well or the number of rounds is not large enough to draw the conclusion.

2.2.6 Prediction Results

The pictures of prediction results are as follows: The line in the red color represents ground truth values of stock price. The line in the blue color represents prediction value. It is clear that all of them perform well in the prediction test though the accuracies are not exactly the same.

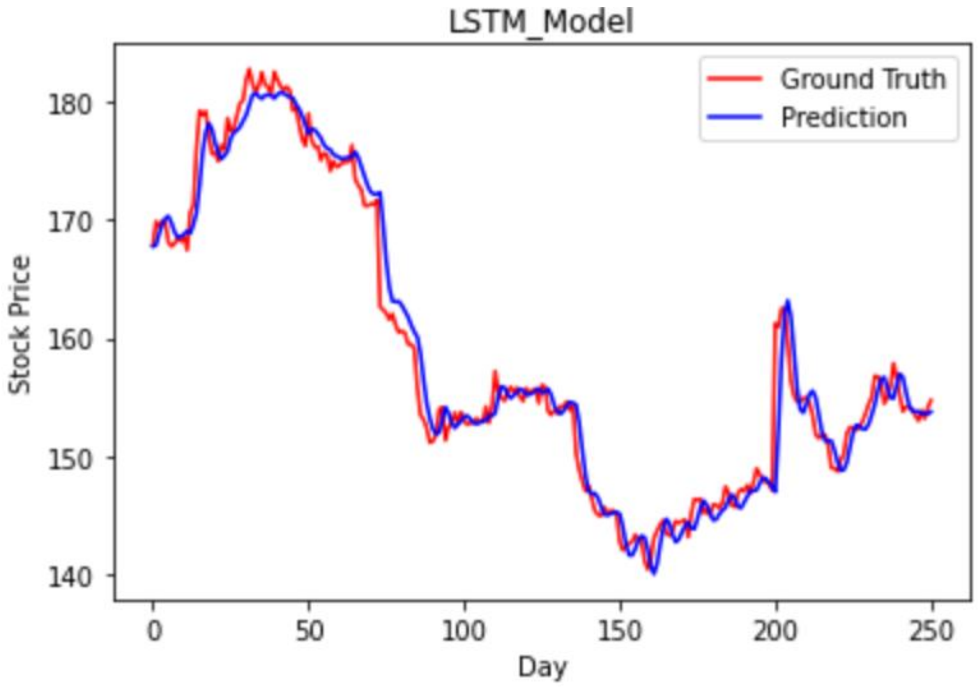


Figure 18: Ground Truth and Prediction of LSTM Based Model

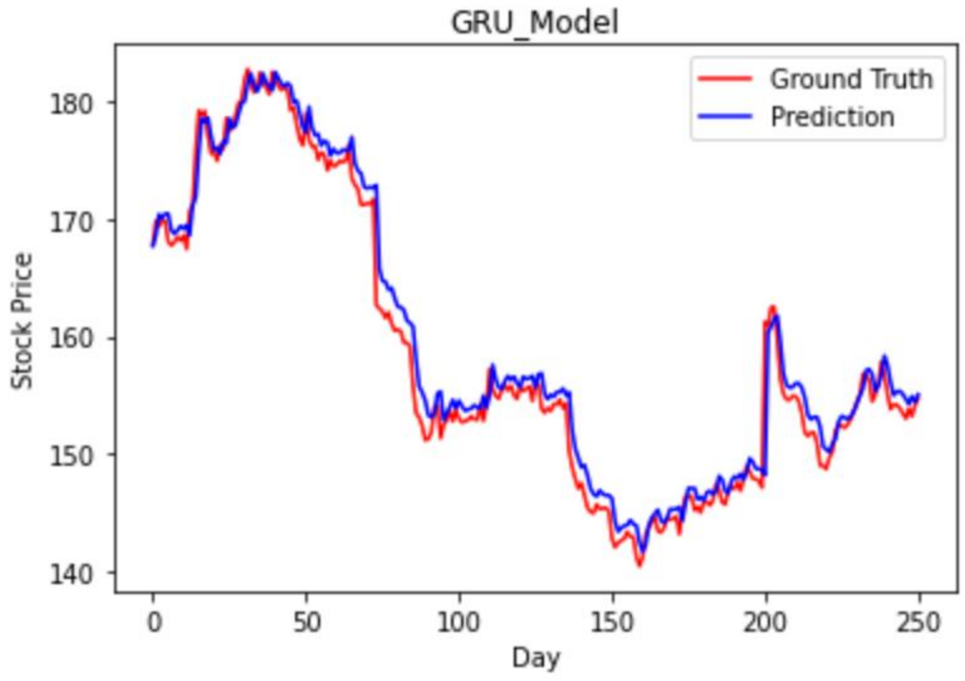


Figure 19: Ground Truth and Prediction of GRU Based Model

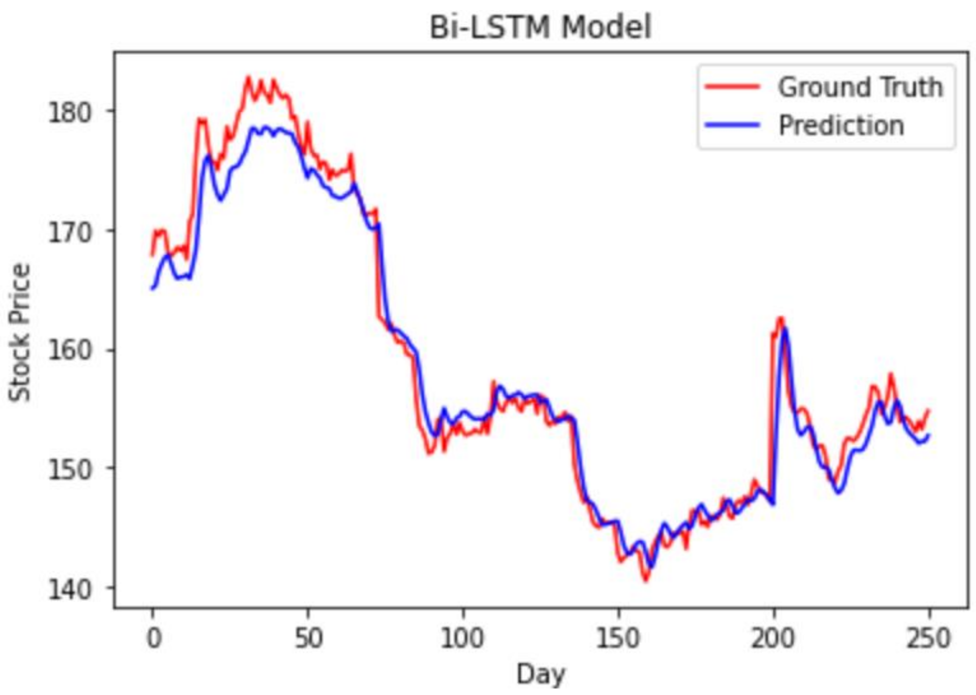
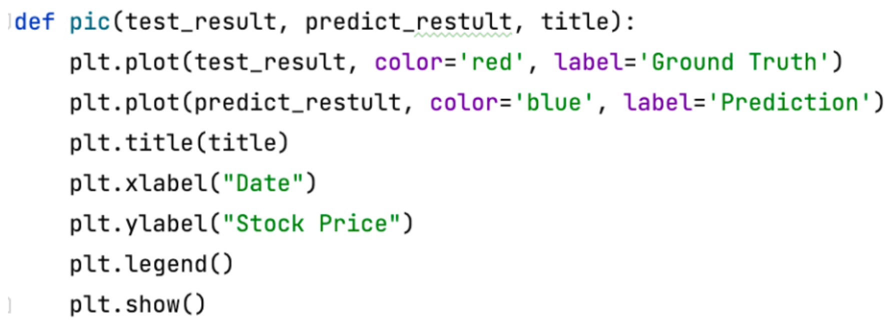


Figure 20: Ground Truth and Prediction of Bi-LSTM Based Model

Figure 21 shows the utility function used to illustrate the predictions of three models.



```
def pic(test_result, predict_restult, title):
    plt.plot(test_result, color='red', label='Ground Truth')
    plt.plot(predict_restult, color='blue', label='Prediction')
    plt.title(title)
    plt.xlabel("Date")
    plt.ylabel("Stock Price")
    plt.legend()
    plt.show()
```

Figure 21: Function for Generating Pictures Code

2.2.7 Discussion of Results

Through the experiment, all of three models can predict well in the stock price prediction task, the mean square errors are in the range of 3 to 7, which are acceptable. It is surprised that Bi-LSTM performs not good as GRU and sometimes even worse than LSTM. Maybe it is due to the performance of my personal computer, the overall structures such as layers and batch size of models are not complicated and large. Therefore, the performance of Bi-LSTM could be better if we add more layers into it. Besides, the parameters of it might not be the optimal values.

In addition, the experiment only uses the stock price data from IBM company, and the testing data is only in 2017. Therefore, it might not perform well on other datasets.

Meanwhile, during the training of neural network, the results are not always the same because of the random initial weights and dropout layer, which means the accuracies are not exactly the same. Only a few rounds of testing are likely to make coincidence.

Chapter 3
Conclusion and Future Work

3.1 Conclusion

In conclusion, this project proposed a simple lifelong machine learning framework for regression problems and applied the proposed framework to predict the stock price using different models including long short-term memory, gate recurrent unit and bidirectional long short-term memory. Through the experiment, all of three models can complete the prediction tasks well. According to the comparative experiment result of mean square error and training time, all of models perform well in the prediction tasks, GRU can reach a better accuracy with less training time compared with LSTM and Bi-LSTM.

However, the stock price is not only related to time, but also can be influenced by various factors. For instance, Covid-19 has brought great profit for some medical and mask companies because the demand of these product has increased sharply, which is an event that is not predictable.

Though the concept of Lifelong Machine Learning is not a new topic and there are many scholars have made efforts and sweat for it, there is no algorithm can realize the real lifelong machine learning that can cope with all kinds of tasks in all fields based on current knowledge and technology. However, without trying and testing, people will never achieve this goal. There is still a long way to reach the goal of lifelong machine learning, which is also the final goal of machine learning.

3.2 Future Work

There are some limitations on the conclusions. Firstly, the number of tests is not enough to give a fair conclusion of the performance since in each round, the hyper-parameters could be random especially in the dropout layer, the dropped data is not fixed. Therefore, in the future, more testing rounds can be involved in order to prevent the coincidence. Secondly, hyper-parameters of models might not be the optimal since parameters are tuned manually. In the future, some advanced techniques which are able to tune the parameters automatically could be considered. Moreover, the experiment only used the stock price data of IBM, in the future, more datasets could be added to reinforce and verify the model. Moreover, advanced structure such as Transformer could be applied in the framework. Back to one original motivation of this project: it is better to embed the prediction models into the websites in the future, which is a feasible way to gain more profit for business.

Chapter 4
Knowledge Learned from Project

Through this project, I have learned a lot. First, I have learned how to find academic papers and how to read academic papers effectively. Through reading papers, I have basically understood the developments of lifelong machine learning and the techniques related to this topic. This discovery progress is interesting and meaningful. However, complicated algorithms and complex formulae let me realize my weakness in mathematics.

Through the experiment, I have utilized the Python as well as machine learning knowledge learned in other modules, which enhanced my programming skills and the understanding of machine learning especially the deep learning part. I have learned how to use code to implement long short-term memory, gate recurrent unit and bidirectional long short-term memory. In addition, I have studied the implementation code of packages and the theory behind utility functions. Meanwhile, I have learned how to draw pictures using software.

Though the project is difficult and the progress is tortuous, after seeing the outcome, I feel it is worthy to make effort to do it. Anyway, lifelong machine learning is an interesting and emerging topic that I would like to pay attention to this topic and study on it continuously.

References

[1] R. S. Michalski, “Learning = inferencing + memorizing,” Foundation of Knowledge Acquisition, pp. 1-41, Springer, 1993.

[2] D. E. Rumelhart, G.E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533-536, Oct. 1986, doi: 10.1038/323533a0.

[3] M. McCloskey, and N. J. Cohen, “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem,” Psychology of Learning and Motivation, vol. 24, pp. 109-165, 1989, doi: 10.1016/s0079-7421(08)60536-8.

[4] W. C. Abraham, and A. Robins, “Memory retention – the synaptic stability versus plasticity dilemma,” Trends in Neurosciences, vol. 28, no. 2, pp. 73-78, Feb. 2005, doi: 10.1016/j.tins.2004.12.003.

[5] S. Thrun, “Explannation-based neural network learning,” Explannation-Based Neural Network Learning, pp. 19-48: Springer, 1996.

[6] S. Hochreiter, and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[7] S. Thrun, and T. M. Mitchell, “Lifelong robot learning,” *Robotics Autonomous Systems*, vol. 15, no. 1, pp. 25-46, 1995.

[8] P. Ruvolo, and E. Eaton, “ELLA: An efficient lifelong learning algorithm,” *In Proceedings of the International Conference on Machine Learning*, Atlanta, GE, USA, 16–21 June 2013, pp. 507–515.

[9] Cho. Kyunghyun, et al. “Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation,” *arXiv preprint arXiv: 1406.1078*, 2014.

[10] Z. Chen, and B. Liu, "Lifelong machine learning," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 10, no. 3, pp. 1-145, 2016.

[11] Z. Li, and D. Hoiem, “Learning without forgetting,” in *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935-2947, 2016, doi:10.1109/TPAMI.2017.2773081.

[12] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” 2015, doi: 10.48550/ARXIV.1053.02531.

[13] J. Kirkpatrick et al, “Overcoming catastrophic forgetting in neural networks,” in *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp.3521-3526, 2016, doi:10.1073/pnas.1611835114.

[14] Z. Chen, N. Ma, and B. Liu, “Lifelong learning for sentiment classification,” *arXiv preprint arXiv*:1801.02808, 2018.

[15] A. Vaswani et al, “Attention is all you need,” in *Proceddings of the 31st International Conference on Neural Information Processing Systems*, pp. 6000-6010, Dec, 2017.

[16] D. Loperz-Paz, and M. A. Ranzato, “Gradient Episodic Memory for Continual Learning,” *Advances in neural information processing systems*, vol. 30, pp. 6467-6476, 2017

[17] J. Sola, and J. Sevilla, “Importance of input data normalization for the application of neural networks to complex industrial problems,” in *IEEE Transacitons on Nuclear Science*, vol. 44, no. 3, pp. 1464-1468, June 1997, doi: 10.1109/23.589532.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts: The Mit Press, 2016.

[19] D. P. Kingma, and J. L. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv*:1412.6980, 2014.

[20] N. Srivastava, G. Hinton, et al. “Dropout: a simple way to prevent neural networks from overfitting,” vol. 15, no. 1, pp. 1929-1958, Jan, 2014.

Appendix A. Implementation Code

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
import time
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU,
Bidirectional
import tensorflow as tf

print("Process Start!")
dataset = pd.read_csv("stockprice.csv", index_col='Date',
parse_dates=['Date'])
train_set = dataset[:'2016'].iloc[:, 1:2].values  # High value
 from 2006 to 2016
test_set = dataset['2017':].iloc[:, 1:2].values  # High value
in 2017


def pic(test_result, predict_restult, title):
    plt.plot(test_result, color='red', label='Ground Truth')
    plt.plot(predict_restult, color='blue', label='Prediction'
)
    plt.title(title)
    plt.xlabel("Date")
    plt.ylabel("Stock Price")
    plt.legend()
    plt.show()


sc = MinMaxScaler(feature_range=[0, 1])
train_set_scaled = sc.fit_transform(train_set)

# 60 timestamp a sample, one output
X_train = []
y_train = []
for i in range(60, 2769):
    X_train.append(train_set_scaled[i - 60:i, 0])
    y_train.append(train_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
# Input of LSTM: (samples, sequence_length, features)
# reshape: training set (2709,60)  ---> (2709, 60, 1)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape
[1], 1))
# ***************LSTM Model ***************
lstm_model = Sequential()
lstm_model.add(LSTM(128, return_sequences=True, input_shape=(
X_train.shape[1], 1)))  # input shape is 60
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(128, return_sequences=True))
lstm_model.add(Dropout(0.2))
lstm_model.add(LSTM(128))  # the default return_sequence is
false, therefore, it will return 1 deminsion
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(units=1))
lstm_model.compile(optimizer='rmsprop', loss='mse')

time_start = time.time()  # Record start time
lstm_model.fit(X_train, y_train, epochs=20, batch_size=32)
time_end = time.time()  # Record finish time
time_sum = time_end - time_start  # Unit is second
print("Time for training LSTM model is: ", time_sum, " second
!")

dataset_total = pd.concat((dataset['High'][:"2016"], dataset['
High']["2017":]), axis=0)
inputs = dataset_total[len(train_set):].values
inputs = inputs.reshape(-1, 1)
inputs_scaled = sc.fit_transform(inputs)
dataset_total = pd.concat((dataset['High'][:"2016"], dataset['
High']["2017":]), axis=0)

inputs = dataset_total[len(dataset_total) - len(test_set) - 60
:].values
inputs = inputs.reshape(-1, 1)
inputs = sc.transform(inputs)

X_test = []
for i in range(60, 311):
    X_test.append(inputs[i - 60:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1
], 1))
predict_test = lstm_model.predict(X_test)  # Predict
predict_stock_price = sc.inverse_transform(predict_test)  #
Inverse transform, otherwise, the result is in range 0-1
```

```
pic(test_set, predict_stock_price, "LSTM_Model")
# Calculate MSE of LSTM model's result
LSTM_MSE = mean_squared_error(test_set, predict_stock_price)
print("MSE of LSTM is: ", LSTM_MSE)
# ***************GRU Model ***************
print("Start GRU Training")
gru_model = Sequential()
gru_model.add(GRU(128, return_sequences=True, input_shape=(
X_train.shape[1], 1), activation='tanh'))
gru_model.add(Dropout(0.2))
gru_model.add(GRU(128, return_sequences=True))
gru_model.add(Dropout(0.2))
gru_model.add(GRU(128, activation='tanh'))
gru_model.add(Dropout(0.2))
gru_model.add(Dense(1))
print("Construction Finished!")
# Compile
gru_model.compile(optimizer='rmsprop', loss='mse')
print("Compile Finished")
# Train
time_start = time.time()  # Record start time
gru_model.fit(X_train, y_train, epochs=20, batch_size=32)
time_end = time.time()  # Record finish time
time_sum = time_end - time_start  # Unit is s
print("Time for training GRU model is:", time_sum, " second!")
# Prepare testing set
X_test = []
for i in range(60, 311):
    X_test.append(inputs[i - 60:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1
], 1))
GRU_predicted = gru_model.predict(X_test)
GRU_predicted_stock_price = sc.inverse_transform(GRU_predicted
)
pic(test_set, GRU_predicted_stock_price, "GRU_Model")
# Calculate MSE of GRU model's result
GRU_MSE = mean_squared_error(test_set,
GRU_predicted_stock_price)
print("MSE of GRU is: ", GRU_MSE)
# ***************Bi-LSTM Model ***************
bilstm_model = Sequential()
```

```
bilstm_model.add(Bidirectional(LSTM(128, return_sequences=True
), input_shape=(X_train.shape[1], 1)))
bilstm_model.add(Dropout(0.2))
bilstm_model.add(Bidirectional(LSTM(64, return_sequences=True
)))
bilstm_model.add(Dropout(0.2))
bilstm_model.add(Bidirectional(LSTM(64)))
bilstm_model.add(Dropout(0.2))
bilstm_model.add(Dense(1))
# Compile
bilstm_model.compile(optimizer='rmsprop', loss='mse')
# Train
time_start = time.time()  # Record start time
bilstm_model.fit(X_train, y_train, epochs=20, batch_size=32)
time_end = time.time()  # Record finish time
time_sum = time_end - time_start
print(time_sum)
# Prepare x_test
X_test = []
for i in range(60, 311):
    X_test.append(inputs[i - 60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1
], 1))
bilstm_predicted = bilstm_model.predict(X_test)
print(bilstm_predicted)
print(bilstm_predicted.shape)
bilstm_predicted_stock_price = sc.inverse_transform(
bilstm_predicted)
# Visualization
pic(test_set, bilstm_predicted_stock_price, "Bi-LSTM Model")
bilstm_MSE = mean_squared_error(test_set,
bilstm_predicted_stock_price)
print("MSE of Bi-LSTM is: ", bilstm_MSE)
```

File - /Users/yihanzhang/Desktop/picture.py

```
import numpy as np
import matplotlib.pyplot as plt
# Training Time Comparison
fig, ax = plt.subplots()
ax.set_ylabel('Time')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.rcParams['axes.unicode_minus'] = False
barWidth = 0.2
# values of y-axis
bars1 = [44.4729,23.7673,24.3078,30.8493]
bars2 = [24.5032,21.1196,23.7143,23.1124]
bars3 = [38.4417,39.3136,39.4897,39.0817]
r1 = np.arange(len(bars1))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
plt.bar(r1, bars1, color='r', width=barWidth, edgecolor='white
', label='LSTM')
plt.bar(r2, bars2, color='y', width=barWidth, edgecolor='white
', label='GRU')
plt.bar(r3, bars3, color='b', width=barWidth, edgecolor='white
', label='Bi-LSTM')
# label of x-axis
plt.xticks([r + barWidth for r in range(len(bars1))], ['Round
1','Round 2','Round 3','Average'])
plt.title("Traning time of different models")
plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
plt.show()
```

```
# Mean Square Error Comparison
fig, ax = plt.subplots()
ax.set_ylabel('MSE')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.rcParams['axes.unicode_minus'] = False
barWidth = 0.2
# values of y-axis
bars1 = [4.3340,7.6449,4.4109,5.4633]
bars2 = [4.6110,5.9706,3.6978,4.7598]
bars3 = [4.6208,5.2370,5.7812,5.2130]
r1 = np.arange(len(bars1))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
plt.bar(r1, bars1, color='r', width=barWidth, edgecolor='white
', label='LSTM')
plt.bar(r2, bars2, color='y', width=barWidth, edgecolor='white
', label='GRU')
plt.bar(r3, bars3, color='b', width=barWidth, edgecolor='white
', label='Bi-LSTM')
# label of x-axis
plt.xticks([r + barWidth for r in range(len(bars1))], ['Round
1','Round 2','Round 3','Average'])
plt.title("Mean Square Error of Different Models")
plt.legend(bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)
plt.show()
```