Yuan Cheng
EECS 605 Course Project

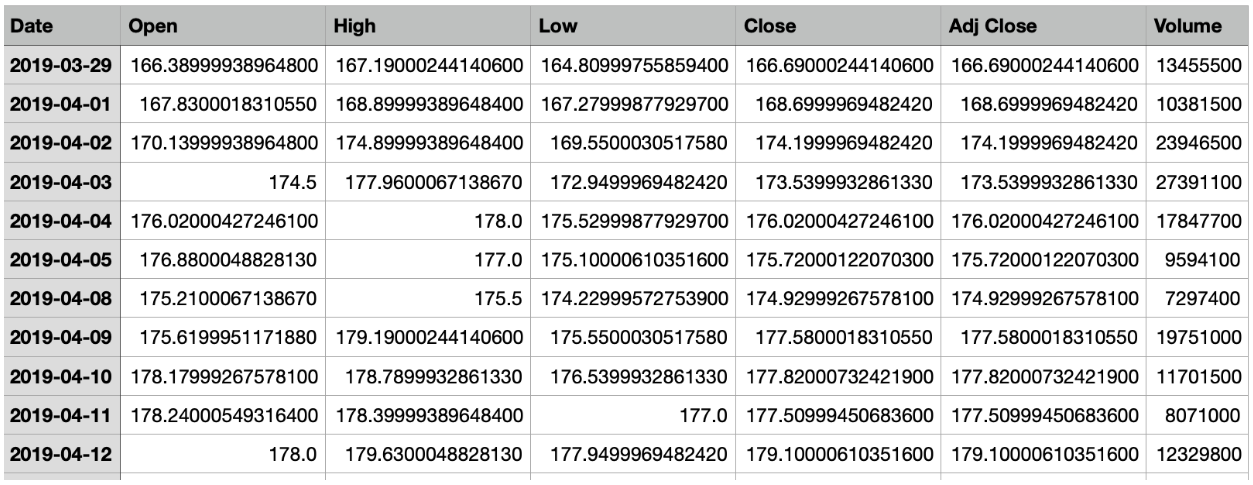
EECS-605 Project Technical Report

Background and Introduction

In my final project, I apply a machine learning algorithm to predict some data which is unknown to people. To be more specific, the task I aim to address is to use the machine learning algorithm to predict the trend of the stock price in the near future. Another important part of the project is the deployment of the algorithm on AWS and I also deploy the project through Heroku so that everyone can use the project by link created by Heroku.

Input & Output

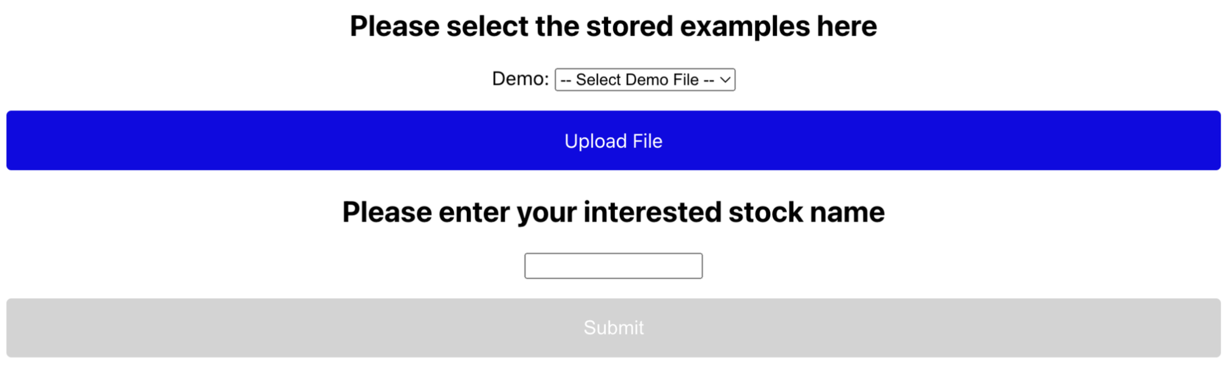
The data source I use is the public stock data provided by Yahoo Finance and luckily, Yahoo Finance provides an open Python package so that the historical stock data can be easily accessed by specifying the name of the stock, the start date, and the end date. It is noticeable that the generated data is in the form of a . CSV file which could be convenient to be applied to the preprocessing part of the project. The following is a part of the generated CSV file:



Date	Open	High	Low	Close	Adj Close	Volume
2019-03-29	166.38999938964800	167.19000244140600	164.80999755859400	166.69000244140600	166.69000244140600	13455500
2019-04-01	167.8300018310550	168.89999389648400	167.27999877929700	168.6999969482420	168.6999969482420	10381500
2019-04-02	170.13999938964800	174.89999389648400	169.5500030517580	174.1999969482420	174.1999969482420	23946500
2019-04-03	174.5	177.9600067138670	172.9499969482420	173.5399932861330	173.5399932861330	27391100
2019-04-04	176.02000427246100	178.0	175.52999877929700	176.02000427246100	176.02000427246100	17847700
2019-04-05	176.8800048828130	177.0	175.10000610351600	175.72000122070300	175.72000122070300	9594100
2019-04-08	175.2100067138670	175.5	174.22999572753900	174.92999267578100	174.92999267578100	7297400
2019-04-09	175.6199951171880	179.19000244140600	175.5500030517580	177.5800018310550	177.5800018310550	19751000
2019-04-10	178.17999267578100	178.7899932861330	176.5399932861330	177.82000732421900	177.82000732421900	11701500
2019-04-11	178.24000549316400	178.39999389648400	177.0	177.50999450683600	177.50999450683600	8071000
2019-04-12	178.0	179.6300048828130	177.9499969482420	179.10000610351600	179.10000610351600	12329800

One important design of the project is that I only require the users to type in the stock name they are interested in instead of generating the CSV data by themselves so the user interface is designed

as follow:

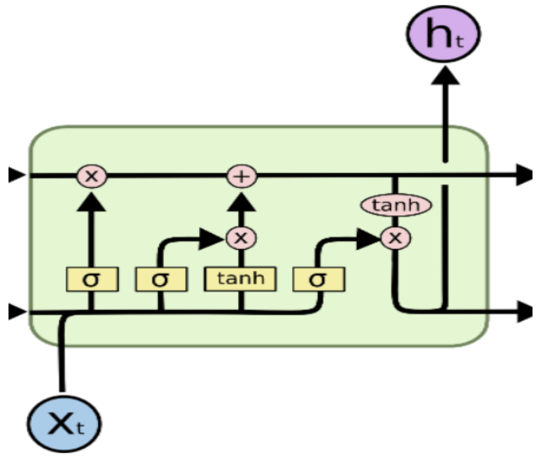

Please select the stored examples here
Demo: -- Select Demo File --
Upload File
Please enter your interested stock name
Submit

The users can either type in the stock name or they can select the pre-stored demo examples for a quick check and in the back end, the CSV file will be generated.

For output, the users can see the predicted data values compared to the real test data which would enable them to see the effectiveness of the algorithm graphically, and meanwhile, the user can really see the rmse based on the algorithm's prediction. For future modifications, I can let the user select the date when they want to see the price of the stock so the system can return the predicted value.

Training

In the back end of the project, the column named "Adj Close" is selected from the CSV file and 90 percent of data is selected as training data and the rest 10 percent as testing data. The preprocessing is important prior to the training, so I first standardize data and each data will be multiplied by the difference between the max and min values plus the minimum of the standardized data. As a result, all data will be in the range of 0 to 1. The model contains 3 layers of LSTM and 2 Dense layers and the LSTM layer has the following structure:



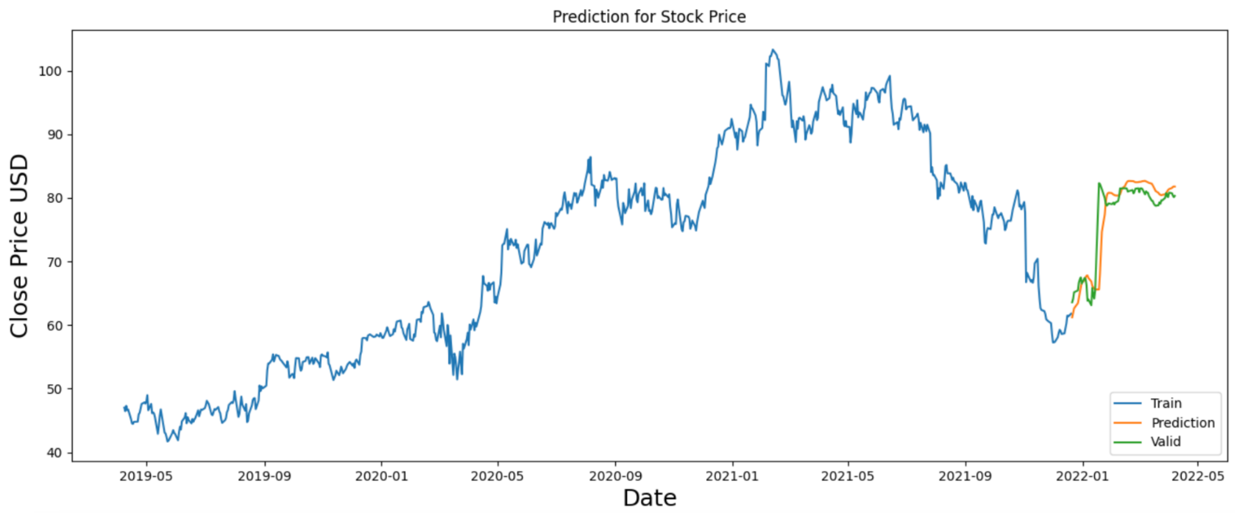
The input of the cell depends on the previous cell's output h_{t-1}, the information flow connecting all the cells, and the specific input for the current cell. Since the stock data is time-series data, the current data depends on the historical data. I design the algorithm so that the predicted data will depend on the previous 60 data. With a dense layer, the model can generate a value representing the predicted stock value.

The optimizer I select is adam because of its good efficiency, adaptive learning rate, and robustness to the outliers; and the loss function I select is mean square error which could be a natural selection for stock data.

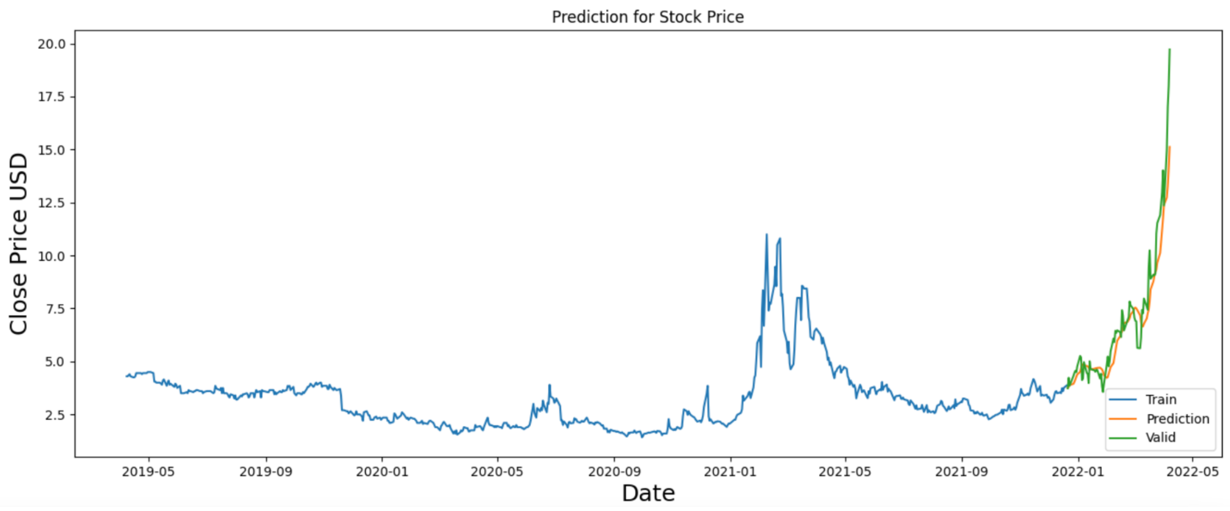
With a trained model, I save the model as an onnx file which is lite and portable, and with such a file, I can easily apply the model to the different input data.

Good & Bad Examples

During the experiment period, in the most cases, the prediction of the stock price would be good and reasonable, for example:



This represents the stock price of Blizzard and we can clearly see that the shape of prediction and the shape of validation are quite similar. However, for some stocks with extreme fluctuations, the model could not predict very well:



The model could not predict such high fluctuations in SGLY, so the model's ability is limited while working on such stocks.

Cloud Architecture

Besides the model training, the design of cloud architecture could be difficult especially when the types of input and output are quite different from the previous assignments. Initially, several packages are necessary including Pandas, Matplotlib, and Numpy, but due to the limited size of the Lambda function in AWS, I cannot directly compress the necessary packages and code files together and upload them to the Lambda function. So I tried to use 2 Lambda functions with separate APIs to run the different functionalities for the project and concatenated the API calls so I only needed to upload some of the files for each Lambda function. However, such implementation could be costly and complicated which makes the developers hard to maintain and the latency of calling 2 Lambda APIs could make the project slow.

I implement the second architecture using docker image which enables me to avoid the limitation of the size. So I create a docker file and a requirements.txt file to install the required packages in the docker and push the docker image to ECR in AWS and I only need 1 Lambda to import the pushed docker image. With only 1 Lambda and 1 API to call, the entire architecture could be easy and clear. While working on the drop-down menu functionality, I use the second Lambda and the second API to download the stored examples in the S3 bucket.