APS 1052 AI in Finance

Ruchita Rajkumar Bhadre
Student No.:1008497244

Mireille Gendy
Student No.:1000640041

UNIVERSITY OF TORONTO

Introduction

- We selected option 1 for our project: Predict a single asset (price related or volatility related)
- The purpose of this project is to develop a model that predicts the price of a stock in the future. The chosen stock was Amazon and the future window was set at 15 days in the future.
- We based our project on an already developed neural network. More specifically the code used Recurrent neural network with LSTM cells. This was developed using Keras and TensorFlow 2.
- We also incorporated financial metrics to validate the quality and accuracy of the model.
- Seed code link here.

Improvements in the seed program

The improvements made into the seed program were:

- Adding 2 more models than the seed program to test which model performed best and used the best model for further predictions.
- Incorporating financial metrics such as:
  - CAGR
  - SharpeRatio
- Using White Reality Check for evaluating the results.

Model from seed program

1) Data extraction
2) Data preprocessing
3) Model creation
4) Model training
5) Model testing
6) Target prediction

Data extraction

For this step we used the load_data function. This function utilises the yahoo finance library. Our function takes as parameter ‘ticker’ which can be a string or a df. This is the parameter which will hold the stock symbol we are interested in. For example if we want to predict price of Amazon stock, we will set ticker to AMZN. If the parameter is a string, we use the get_data function to load the data from yahoo finance. If the parameter is a df, we assign it to our df variable. Otherwise our function throws an error.

Once we extracted our data, we will make a copy of it and assign it to a result dictionary with key ‘df’.

After extracting our inputs, we will start preprocessing the data.

Data preprocessing

The preprocessing portion of the function handles tasks:

1) Data scaling
2) Identifying the inputs and target
3) Data cleaning
4) Splitting the data in testing set and training set

All these steps are done in the same function after extracting the data from yahoo finance

Data scaling

This task was accomplished by using a boolean parameter scale and feeding it to the load_data function.

If scale is set to True, the function creates a new dictionary and loops through the columns in the column_feature list. Each column is scaled using the MinMaxScaler function. The new scaled values are added to the corresponding column in the df. The scaled values are also saved in the new dictionary created.

	open	high	low	close	adjclose \
1997-05-15	0.121875	0.125000	0.096354	0.097917	0.097917
1997-05-16	0.098438	0.098958	0.085417	0.086458	0.086458
1997-05-19	0.088021	0.088542	0.081250	0.085417	0.085417
1997-05-20	0.086458	0.087500	0.081771	0.081771	0.081771
1997-05-21	0.081771	0.082292	0.068750	0.071354	0.071354
...	...	...	...	...	...
2022-07-29	134.899994	137.649994	132.410004	134.949997	134.949997
2022-08-01	134.960007	138.830002	133.509995	135.389999	135.389999
2022-08-02	134.720001	137.440002	134.089996	134.160004	134.160004
2022-08-03	136.210007	140.490005	136.050003	139.520004	139.520004
2022-08-04	140.580002	143.559998	139.550003	142.570007	142.570007

Input DF before scaling

	open	high	low	close	adjclose	volume	ticker \
1997-05-15	0.000276	0.000279	0.000166	0.000151	0.000151	0.690172	AMZN
1997-05-16	0.000150	0.000141	0.000107	0.000089	0.000089	0.136869	AMZN
1997-05-19	0.000095	0.000086	0.000085	0.000084	0.000084	0.054117	AMZN
1997-05-20	0.000086	0.000080	0.000087	0.000064	0.000064	0.047957	AMZN
1997-05-21	0.000061	0.000052	0.000017	0.000008	0.000008	0.176865	AMZN
...	...	...	...	...	...	...	...
2022-07-29	0.720515	0.729539	0.716251	0.723216	0.723216	0.066915	AMZN
2022-08-01	0.720835	0.735796	0.722204	0.725575	0.725575	0.032310	AMZN
2022-08-02	0.719553	0.728425	0.725343	0.718980	0.718980	0.025124	AMZN
2022-08-03	0.727515	0.744599	0.735950	0.747719	0.747719	0.029893	AMZN
2022-08-04	0.750868	0.760878	0.754892	0.764073	0.764073	0.029217	AMZN

Input DF after scaling

Identifying inputs and target

We include all the available features extracted from yahoo_finance as input features.

```
FEATURE_COLUMNS = ["adjclose", "volume", "open", "high", "low"]
```

To add the target column to our DF, we use the shift method and apply it only to the adjclose feature. We apply the shift in the negative direction by the number of lookup_step, in this case we want to predict the stock price in 15 days, so the lookup_step is set to 15. This in turn creates a problem as the last 15 values in the target column will be set to NAN.

Frequency of the target is ‘daily’.

```
1997-05-15    0.000070
1997-05-16    0.000078
1997-05-19    0.000050
1997-05-20    0.000039
1997-05-21    0.000056
                ...
2022-07-29         NaN
2022-08-01         NaN
2022-08-02         NaN
2022-08-03         NaN
2022-08-04         NaN
Name: future, Length: 6348, dtype: float64
```

Cleaning the data

We need to remove the NaN values from the target however we do not want to lose the associated feature values as those can be used to test the model. To solve this issue we create an np array where we store the feature column values for the last 15 rows using the .tail() method. Once the inputs are saved we drop the rows containing NaN values from the df.

Before:

	open	high	low	close	adjclose	volume
1997-05-15	0.000276	0.000279	0.000166	0.097917	0.000151	0.690172
1997-05-16	0.000150	0.000141	0.000107	0.086458	0.000089	0.136869
1997-05-19	0.000095	0.000086	0.000085	0.085417	0.000084	0.054117
1997-05-20	0.000086	0.000080	0.000087	0.081771	0.000064	0.047957
1997-05-21	0.000061	0.000052	0.000017	0.071354	0.000008	0.176865
...	...	...	...	...	...	...
2022-08-09	0.737348	0.736432	0.736816	137.830002	0.738658	0.014778
2022-08-10	0.763266	0.766393	0.762794	142.690002	0.764717	0.021682
2022-08-11	0.768396	0.765810	0.756029	140.639999	0.753725	0.016912
2022-08-12	0.758724	0.760931	0.757977	143.550003	0.769328	0.018219
2022-08-15	0.762731	0.761897	0.765392	143.179993	0.767344	0.014040

	ticker	date	future
1997-05-15	AMZN	1997-05-15	0.000070
1997-05-16	AMZN	1997-05-16	0.000078
1997-05-19	AMZN	1997-05-19	0.000050
1997-05-20	AMZN	1997-05-20	0.000039
1997-05-21	AMZN	1997-05-21	0.000056
...	...	...	...
2022-08-09	AMZN	2022-08-09	NaN
2022-08-10	AMZN	2022-08-10	NaN
2022-08-11	AMZN	2022-08-11	NaN
2022-08-12	AMZN	2022-08-12	NaN
2022-08-15	AMZN	2022-08-15	NaN

After

	open	high	low	close	adjclose	volume
1997-05-15	0.000276	0.000279	0.000166	0.097917	0.000151	0.690172
1997-05-16	0.000150	0.000141	0.000107	0.086458	0.000089	0.136869
1997-05-19	0.000095	0.000086	0.000085	0.085417	0.000084	0.054117
1997-05-20	0.000086	0.000080	0.000087	0.081771	0.000064	0.047957
1997-05-21	0.000061	0.000052	0.000017	0.071354	0.000008	0.176865
...	...	...	...	...	...	...
2022-07-19	0.617912	0.630377	0.616778	118.209999	0.633457	0.024675
2022-07-20	0.633516	0.654399	0.639995	122.769997	0.657907	0.029624
2022-07-21	0.657991	0.661664	0.655906	124.629997	0.667881	0.024314
2022-07-22	0.667664	0.665110	0.656394	122.419998	0.656031	0.020088
2022-07-25	0.655319	0.655247	0.649250	121.139999	0.649168	0.019490

	ticker	date	future
1997-05-15	AMZN	1997-05-15	0.000070
1997-05-16	AMZN	1997-05-16	0.000078
1997-05-19	AMZN	1997-05-19	0.000050
1997-05-20	AMZN	1997-05-20	0.000039
1997-05-21	AMZN	1997-05-21	0.000056
...	...	...	...
2022-07-19	AMZN	2022-07-19	0.738658
2022-07-20	AMZN	2022-07-20	0.764717
2022-07-21	AMZN	2022-07-21	0.753725
2022-07-22	AMZN	2022-07-22	0.769328
2022-07-25	AMZN	2022-07-25	0.767344

Train/ test split

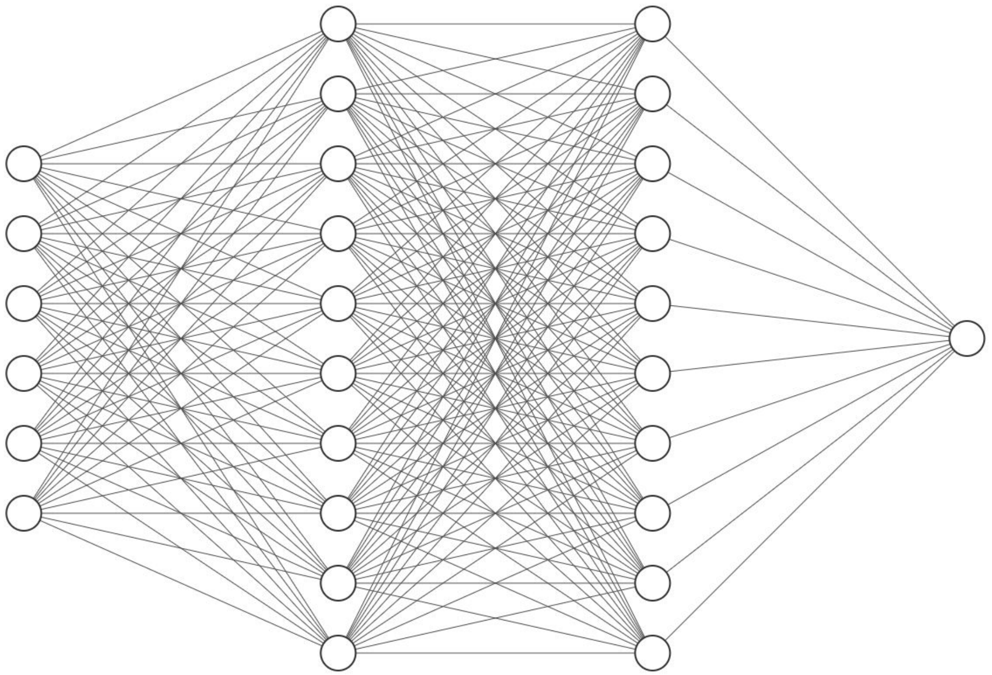
Now that our data is clean and our inputs and targets are ready, the last step is to split the training and testing sets. We do this by using the train_test_split() function and set the test size to 20% of the data.

20% was found to be the best test size as 80% of the data is still remaining to have sufficient examples for the model to pick up on the patterns in the data so that it is complex enough. Thus, preventing underfitting.

It was also important to not have the model too fitted to the data and ensure that it is still flexible enough to predict examples that were never seen before. Thus preventing overfitting.

Model Creation

After extracting and prepping the data, the next step is to create the model. The model is created using Keras. We utilise the sequential() method to initialise the model. The architecture of the model is dynamic. The layers are set through a parameter, n_layers. We then start a loop that counts through the number of layers. In the first layer we set the parameter batch_input_shape to set the number of neurons and the batch size we also set the return_sequences parameter to True such that the input to the following layer is the output of the previous layer. The following hidden layers have an output of the units parameter. Finally at the last layer, the return_sequences parameter is set to false.

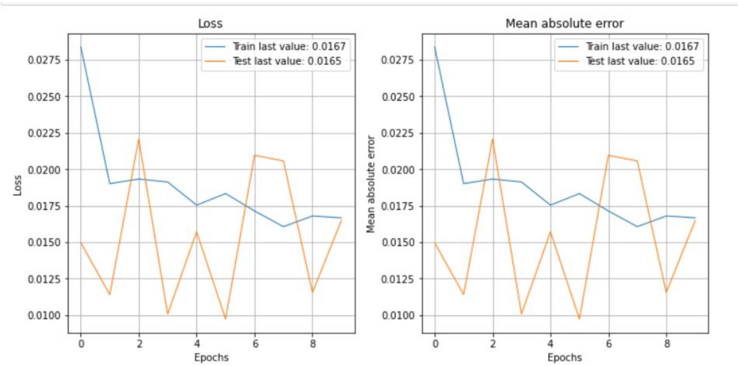


A simplified network is displayed in the figure on the left. Due to complexity of the network, the exact architecture cannot be captured in the figure.

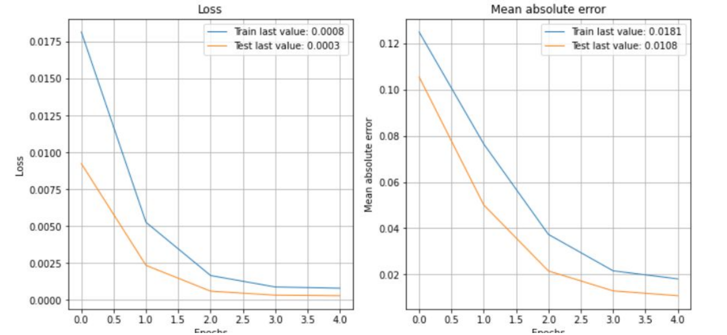
Model training

We trained 3 models by modifying certain parameters.

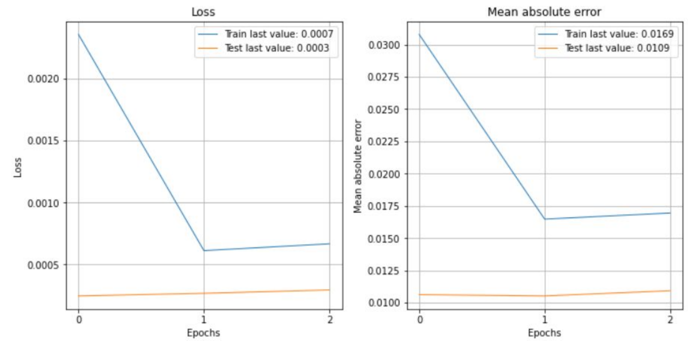
Model 1: MAE loss, Adam optimizer, 300 units and 3 layers



Model 2: Huber loss and SGD optimizer, 300 units and 3 layers

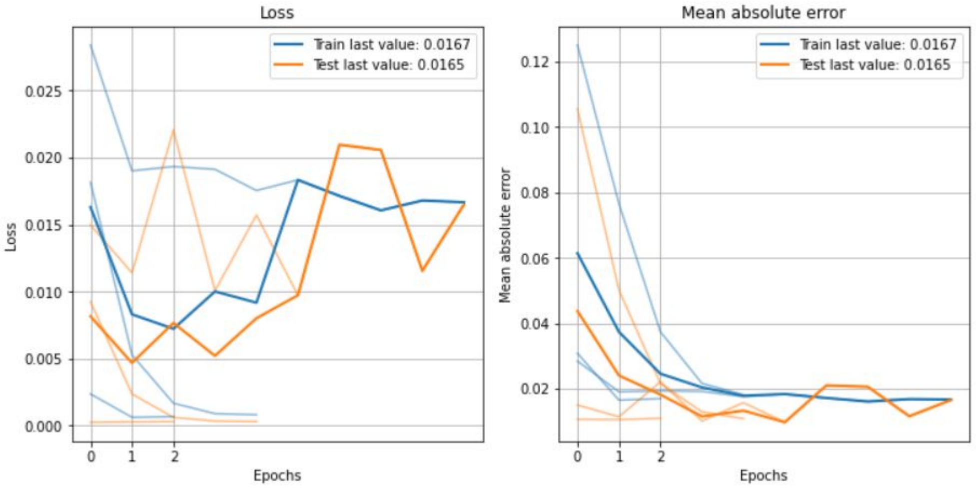


Model 3: Huber loss, Adam optimizer, 256 units and 2 layers



Model testing

```
In [26]: histories=[history1,history2,history3]
         plot_history(
             histories,
             show_standard_deviation=False,
             show_average=True
         )
         plt.show()
         plt.close()
```



Model 3 performed the best with loss=0.000582 and MAE=0.016. There model 3 is the best model that we will use for predicting stock prices on our data.

Target prediction

- Target prediction is done by writing a predict function which takes model and data as its inputs that were returned by create_model() and load_data().
- First the last sequence from data is retrieved and its dimensions are expanded. Then predictions are made using model.predict method whose output is scaled from 0 to 1.
- After this step we get the price by inverting the scaling and finally the function returns the predicted price.
- Then the model is evaluated by model.evaluate method and mean absolute error is calculated by inverse scaling.
- Accuracy is calculated by counting the number of positive profits. Total profit is calculated by adding sell and buy profits together and finally profits per trade is calculated by dividing total profit by number of trades.

Explaining the metrics

- Mean Absolute Error:We receive an error of around 20, which means that the model forecasts are, on average, more than $20 off from the genuine values. This will vary from ticker to ticker, and when prices rise, the error will rise as well. As a result, you should only use this metric to compare your models when the ticker is steady (e.g., AMZN).
- Accuracy Score:This represents the level of accuracy of our predictions. This computation is based on all trades from the testing samples that resulted in positive profits.
- Profit per trade:It is calculated as the total profit divided by the total samplings used for testing.
- Buy/Sell Profit: Using the get final df() function, we computed the profit that would result from opening trades on each testing sample.

In [49]:
```python
# printing metrics
print(f"Future price after {LOOKUP_STEP} days is {future_price:.2f}$")
print(f"{LOSS} loss:", loss)
print("Mean Absolute Error:", mean_absolute_error)
print("Accuracy score:", accuracy_score)
print("Total buy profit:", total_buy_profit)
print("Total sell profit:", total_sell_profit)
print("Total profit:", total_profit)
print("Profit per trade:", profit_per_trade)
```

```
Future price after 15 days is 137.71$
huber_loss loss: 0.00024629771360196173
Mean Absolute Error: 2.049548998676125
Accuracy score: 0.5771065182829889
Total buy profit: 761.969560354948
Total sell profit: 174.89358113706112
Total profit: 936.8631414920092
Profit per trade: 0.7447242778155876
```

Plot of actual price Vs predicted price

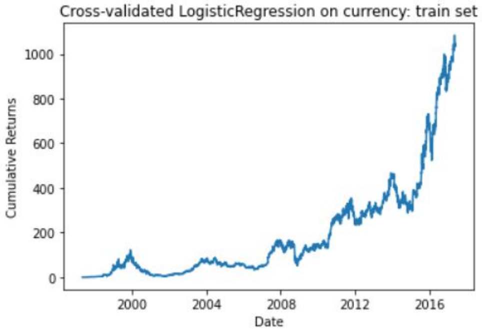
The red curve represents the expected prices, and the blue curve is the actual test set. As we predicted, the stock price rose and has been falling.

This plot displays the prices of the testing set scattered across our whole dataset together with associated forecasted values because we set SPLIT_BY_DATE to False.

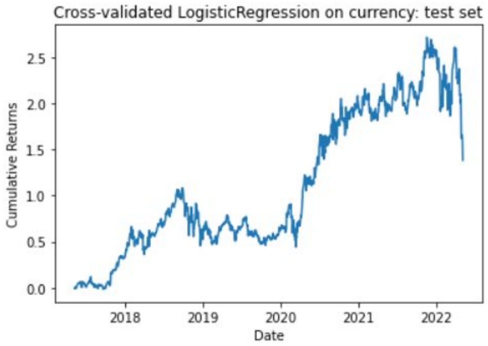
The testing set will comprise the final TEST SIZE proportion of the entire dataset if SPLIT_BY_DATE is set to True.

Financial Metrics

- The Compound Annual Growth rate(CAGR) is a statistic that represents the rate of growth that an investment would have experienced had it grown at the same pace each year and reinvested its earnings at the conclusion of each period.
- When a new asset or asset class is introduced to a portfolio, the Sharpe ratio is used to analyze how the portfolio's overall risk-return characteristics have changed.



In-sample: CAGR=0.416898 Sparpe ratio=0.858962



Out-of-sample: CAGR=0.18994 Sparpe ratio=0.683054

CAGR over training data is 41.7% ie we can get around 42% of return on our investment.

We also get a 85.9% risk-adjusted performance on training data.

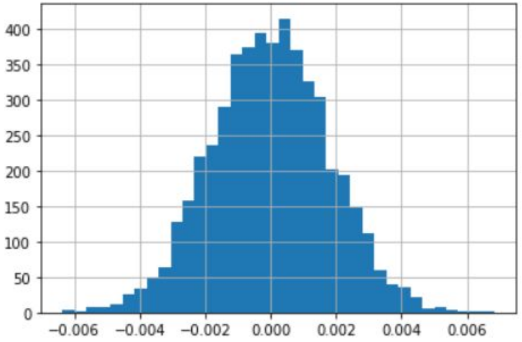
CAGR over test data is 18.9% ie we can get around 19% of return on our investment.

We get 68.3% risk-adjusted performance over test data.

White Reality Check

The p-value that we get after using the white reality check is equal to 0.2992. The p-value should be as less as possible. Therefore, the null hypothesis cannot be rejected.

```
average return 0.000934
[-0.00349948  0.00349491]
Do not reject Ho = The population distribution of rule returns has an expected value of zero or less (because p_value is not sm
all enough)
p_value:
0.2992
```



Guide to the project folder

1.Data folder contains the input data.

2.Results csv folder contains the csv file of the results generated.

3.There is an HTML file included named, AI_Finance_Project.html which contains the notebook displayed along with the outputs of the code executed by us.

4.An ipynb file is also included for the Professors to execute on their machine. The ipynb file contains all the necessary imports and installation code required to run the notebook.

References

http://alexlenail.me/NN-SVG/index.html

ridge_selprec_WRC.py

https://www.investopedia.com/