Assignment 3

Nguyen Phuc Thai
a1846505

Abstract

This paper attempts to build a recurrent neural network (RNN) in the problems of forcasting future value of the Dow Jones (DJ) index. The general purpose of the report is to explore the use of RNN rather than proposing a method for optimal index/stock price forcasting. The input for the network will be available data regarding Dow Jones, with the sole focus on the index components, without any economics factor. The resulting RNN model captured the movement of the index relatively well, yet, if there the index experience periods of aggressive growth, the model failed to capture not the magnitude of the value of the index. The best model from this project scored a mean absolute percentage score of around 2%, in order words, model's prediction is only about 2% off of the actual value on average

1. Introduction

The report aims to use RNN in the problem of predicting stock price based on data on previous days, or building a model that can predict on sequential data in the financial stock market scenario. RNN is a deep learning network capable of capturing sequential data, and retaining past information for future predictoin hence is used as the model to predict future stock price based on previous trading days information. Outside of the current problem of stock price prediction, RNN has also been developed in other applications including, but not limited to, natural language processing, language translation and signal processing.

On the financial forcasting problem, other types of machine learning methods was also attempted, using support vector machine [10], "time-delay" neural network, where inputs include the "lagged return" [13]. RNN has also been aplied in this field, such as using Long-short term memory (LSTM) model , a form of RNN, [2], using LSTM to predict direction of stock price [12], and using LSTM with attention mechanism to predict stock price [11]. Aside from the nature of the model, and the trend of the stock movement learned from RNN, various economics factors can also improve the predictive power ([1], [12]). These are just few examples of machine learning, and using deep learning to make prediction of the stock market, which is an area that is still being explored extensively.

The target of this report will be forcasting the value of the Dow Jones index using Recurrent neural network, namely the Long short term memory model and the Gated Recurrent unit model (GRU) proposed in [3]. The trading history of the Dow Jones index is downloaded from yahoo finance [14], and all modeling, model processing is done using Python. Aside from Dow Jones index values history, the historical trading data of DJ current 30 component stocks will also be used as the predictors for the models These companies names are available from [15], their trading history are also downloaded via yahoo finance. Financial and economic factors are not incorporated into this report due to the project nature focus more on experimenting with RNN. The

2. Methodology

2.1. Data processing

In this project, the closing value of the Dow Jones index will be the response for the model prediction. Data on Dow Jones index and its stock components between 3 January 2000 and 2 March 2022 will be used. The ending data is not the current day because this is the last available date downloaded using "yfinance" library in Python. Available data downloaded includes "Trading date", "Opening price", "Close price", "Adjusted close price", "High price" and "Low price" on a given day and "Trading volumes" (the index's volume is unavailable). The figure below shows the movement of the response in the dataset.

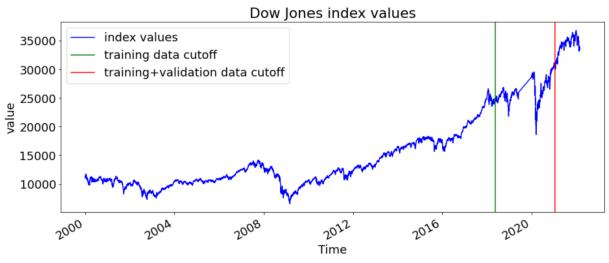


Figure 1. Dow Jones' index value in the chosen timeframe

It should be noted that the components of the index

change frequently [6], and not all current 30 companies in the index has been in the index during the chosen periods. Additionally, three companies (Visa, Saleforce and Dow) has no trading information for a significant portion of the chosen period (missing more than 1000 trading days) and is thus unsuable for modeling.

Two dataset will be used for the modeling purpose, the first one uses all available closing price and volumes of stocks in the index and the close price of the index itself to predict future value (55 predictors from 27 stocks and the index's close price). The second one only uses trading information of the Dow Jones index, using high price, low price and Close price to predict future value (3 predictors). The reason for this choice is because of the considerable difference in performance observed from the dataset, with the second one seem to outperform the first despite have much less information. Furthermore, open and close price of of component companies' stock were not used as these data are highly correlated to the close price data.

The data is then split into a test and training data, where the test data contains latest 5% observations because of the sequential nature of the problem. The training data is also further split, with 10% of the training data (9% for the whole dataset) of the latest observation being used for validation, and the rest are kept for training the model. Figure 1 show the index value movement during the chosen timeframe. It should be noted that the validation and test range of the dataset are relatively volatile timeframe of the index, mainly due to the effect of COVID-19.

	AAPL	AMGN	AXP	BA
count	5405.000000	5405.000000	5405.000000	5405.000000
mean	24.040308	104.895727	64.219277	113.444387
std	36.178848	61.332531	32.798929	87.002259
min	0.234286	31.070000	10.260000	25.059999
25%	1.450179	57.709999	42.500000	54.500000
50%	10.318571	69.220001	53.430000	75.519997
75%	28.552500	158.800003	81.000000	139.000000
max	182.009995	260.950012	198.380005	440.619995

Figure 2. dataset snapshot

Figure 2 shows a snapshot of 4 components of the dataset. The range of values for just four components of the dataset differs significantly, and the values of the index also have a major difference compared to its components (with mean around 15,000, and nearly 7,000 standard deviation). Hence the dataset is scaled using min max scaling so all columns ranges from -1 to 1 to help the machine learning model learns better. In particular, for a given column, the data is scaled using the following equation:

$$x_{new} = \frac{x - \frac{x_{max}+x_{min}}{2}}{x_{max} - \frac{x_{max}+x_{min}}{2}} \quad (1)$$

Where x_{new} is the scaled value, x is the original value, x_{max} and x_{min} are the maximum values and minimum value. The minimum and maximum is determined based on the the values that the model is trained on, the data that the model has no access to during training is still scaled, but the range of data could be bigger 1 or smaller than -1.

Finally, the model is then processed so that they can be used as input for the the RNN models. The deep learning model expect a 3 dimensional input, the number of observation, the sequence length or time step length (in days, for this problem) and the number of predictors. For this project, we will fixed the time frame to ten days, which means that for a particular date we want to use for prediction, the input will be the previous 10 days trading information, for instance, the 2 datasets used for training the model for validating in this project has dimension of (4613, 10, 55) and (4613, 10, 3). The response will be the close price on the very next day

2.2. Criteria and Baseline prediction

To validate the model, the Root mean squared error (RMSE) will be the main criteria, which measure how close, on average did all predictions get to the actual value in magnitude. The mean absolute percentage error will also be calculated on the final prediction to measure in percentage terms how close the model get to the actual results. Another metrics used will be the R score, which measure the variance that the predicted result captured from the actual values, and the closer value of R to 1 means, the better model's prediction capture the actual variance. These metric are calculated as follow:

$$RMSE = \sqrt{\frac{1}{n}\sum_{1}^{n}(\widehat{y_i} - y_i)^2}$$

$$MAPE = \frac{1}{n}\sum_{1}^{n}|\frac{\widehat{y_i} - y_i}{y_i}|$$

$$R = \frac{\sum_{i=1}^{n}(y_i - \overline{y})(\widehat{y_i} - \overline{\widehat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2(\widehat{y_i} - \overline{\widehat{y}})^2}}$$

Where n is the number of predictions, $\widehat{y_i}$ and y_i are the predicted value and actual value, respectively, and $\overline{y}$ and $\overline{\widehat{y}}$ are the average of the actual and predicted results.

As a baseline to compare the RNN, we will compute the close value of the index on a given day using the Dow Jones

calculation formula [6]:

$$Value = \frac{\sum Components\ close\ price}{Dow\ Divisor}$$

The latest recorded Dow Divisor is approximately 0.1517, but taking into account the unuse stock the divisor will be 0.13 for the baseline. On the whole training data (including the validation set), the RMSE is 1187.736, on the test data, it is 1684.988.

2.3. RNN

Structurally, RNN is similar to a normal feed forward neural network [7], but a cell in RNN allows connection to the past state. For a past state at time step t, it is caculated as:

$$h_t = f_W(h_{t-1}, x_t)$$

Where t is the time step, h denotes hidden state, x_t is the input and W denotes the weight in the model, or the model's parameter. But because of the nature of the cell, and the need for propagating through time using the same set of cells, it is prone to the vanishing gradient problems if used on a long sequence [8].

2.3.1 LSTM

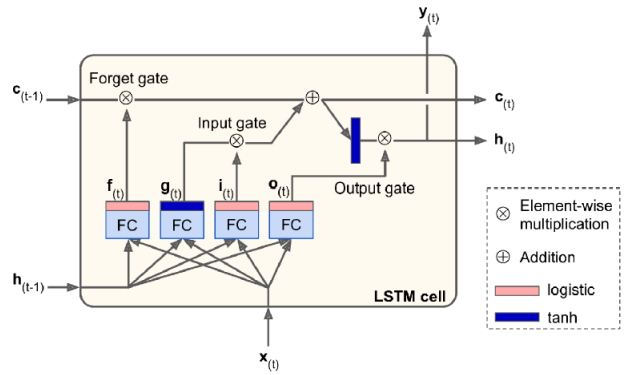
Therefore, a variation of the RNN the LSTM introduced by [9] tackles this issue by using long term cell states alongside the hidden states.

In the above functions, $\otimes$ means element wise product, $\sigma()$ and tanh() represent the sigmoid and tanh function, W_x..'s represent the weight matrix connecting to the input, W_h..'s represent the weight matrix connecting to the previous hidden state, in the function, b's represent the bias, x's represent the input vector, h's represent the hidden state. f represent the forget gates that will, i represent the input gate, o represent the output gate and g uses information from the input previous hidden state, and $c_{(t)}$ and $y_{(t)}$ are the resulting cell state and output, respectively.

Aside from having the previous hidden state h and the input sequence x, a cell state is introduced in LSTM, and this feature helps most with preserving long term memories is the cell state, and its only goes through linear computation as shown in figure 3 [7]. It does not only bring on information, as the forget gate helps the cell discard unnecessary information. LSTM is implemented in this project using the Pytorch library.

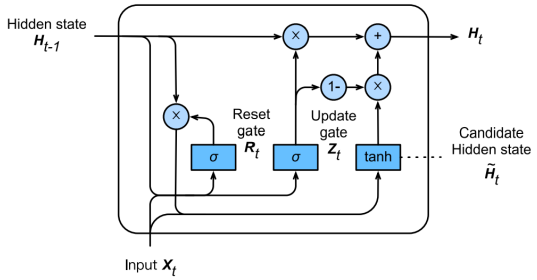
2.3.2 GRU

More recently introduced by [3], GRU also have feature design to allow it to remove redundant information and update its cells, but without an extra hidden state.



$$\mathbf{i}_{(t)} = \sigma\left(\mathbf{W}_{xi}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right)$$
$$\mathbf{f}_{(t)} = \sigma\left(\mathbf{W}_{xf}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right)$$
$$\mathbf{o}_{(t)} = \sigma\left(\mathbf{W}_{xo}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right)$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)$$

Figure 3. Inside a LSTM cell, adapted from [7]



$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$
$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

Figure 4. Inside a GRU cell, adapted from [16]

Where R_t denotes the reset gate, allowing the cell to forget and Z_t denote the update gate, $\tilde{h_t}$ represent a candidate outcome state. GRU also help with the vanishing gradient problem as the next hidden state is computed via an interpolation between the past state and the next state, which is ad-

ditive in nature. This addition in the computation allow the gradient to not vanish during back propagation [4]. GRU is implemented in this project using the Pytorch library.

2.4. Modeling process

To choose the best predictor, LSTM and GRU will be used on the training data of the two datasets (one containing trading data on 27 components of the Dow Jones, and one only on Dow Jones), and calculating the resulting validation root mean squared error. The model on the dataset with the lowest validation error will be chosen, and it will be refitted on both the training and validation data before being tested on the test data.

As the data is scaled, during the training process, after the model is trained, the output of the model will be inverse scaled according to the formula:

$$\widehat{y_{out}} = \widehat{y} * (y_{max} + \frac{y_{max} + y_{min}}{2}) + \frac{y_{max} + y_{min}}{2}$$

Where $\widehat{y_{out}}$ is the index value predicted by the model after being inverse scaled, $\widehat{y}$ is the model's output, y_{min} and y_{max} are the minimum and maximum response within the data the model is trained on

The model will be trained using the Adaptive moment (Adam) optimizer for its nature of taking into account both the first and second moment of the gradients during training, and trained for 100 epochs throughout the project. For each type of RNN, the parameter tuned in this project will be the learning rate from the optimizer, the number of LSTM or GRU layer within a model, the number of cells within each layer of the LSTM or GRU layer, and dropout implementation, which is randomly dropping cells with in the layer. The overall structure of the models experimented in this project will be Input → LSTM/GRU layers (number of layer can change) → Fully connected layer → output.

3. Experimentation and result

The aim of the experiment is to explore aspect of the RNN models to improve their predictive power in predicting index value.

3.1. LSTM

Tables 1 and 2 show the performance of the LSTM model on the training and validation data based on different learning rate and model architecture. Both tables show similar trends, where learning rate 0.01 is optimal compared to 0.1 as and 0.001 as either increasing or decreasing the learning rate from 0.01 makes the model perform worse, as shown in models with 2 layers and 32 cells in each layer and models with 64 LSTM cells in its single LSTM layer.

Given the same number of cell in each layer, models with fewer layers tend to perform better as observed in LSTMs with 32 cells increased from from 1 layer to 3, LSTMs with 16 and 64 cells increased from 1 layer to 2, on both datasets. Finally, the number of cells in a LSTM layers is the parameters has varying effect depending on the number of layers and dataset. For 1 layer model, increasing the number of cells improve its validation error, but not after 64, for 27 stocks dataset, the number goes from 2994.782 for 16 cells, 1343.716 for 32 cells and 1216.627 for 64 cells, but increase to 1959.393 at 128 cells. For the other dataset, it goes from 1314.677 for 16 cells, 898.984 for 32 cells and 637.023 for 64 cells, but increase to 862.514 for 128 cells. But in two-layer models, the trend is less clear, for the larger data RMSE goes from 2230.971 for 16 cells, decrease to 1736.317.716 for 32 cells and increase to 2278.873 for 64

Learning rate	Number of layers (neuron in each layer)	training error	validation error
0.01	1 (32)	322.261	1343.716
0.01	2 (32)	331.694	1736.317
0.01	3 (32)	320.334	1744.332
0.1	2 (32)	3494.240	11929.900
0.001	2 (32)	413.772	3271.497
0.01	1 (64)	364.042	**1216.627**
0.01	2 (64)	358.183	2278.873
0.1	1 (64)	1421.806	7468.748
0.001	1 (64)	491.409	2087.093
0.01	1 (16)	321.567	2994.782
0.01	2 (16)	342.996	2230.971
0.01	1 (128)	344.656	1959.393

Table 1. LSTM performance on the 27 stocks dataset

Learning rate	Number of layers (neuron in each layer)	training error	validation error
0.01	1 (32)	186.563	898.984
0.01	2 (32)	200.718	1033.771
0.01	3 (32)	219.284	1392.589
0.1	2 (32)	519.722	4302.402
0.001	2 (32)	482.008	3324.858
0.01	1 (64)	156.480	**637.023**
0.01	2 (64)	200.024	877.382
0.1	1 (64)	210.817	6925.016
0.001	1 (64)	246.033	1010.540
0.01	1 (16)	191.549	1314.677
0.01	2 (16)	207.718	1672.931
0.01	1 (128)	211.152	862.514

Table 2. LSTM performance on the Dow Jones only dataset

cells. As for the Dow Jones only, its started at 1672.931 for 16 cells, decrease to 1033.771 and decrease to 877.382 for 64 cells

As shown on both tables and on both datasets, the best LSTM model so far is the single layer, 64-cell LSTM. Furthermore, comparing tables 1 and 2 line-by line, the model built on the Dow Jones only dataset is outperforming the model built on dataset with more predictors. One possible explanation is that the larger dataset contains significantly more noises, and is not helping the model predict the future index value well

Dropout was implemented on the LSTM. In pytorch, dropout in LSTM is only implemented in the last layer of the LSTM, so the testing of dropout on both dataset is only tested on two-layer LSTMs on both model using learning 0.01, and dropout rate 0.2, as summed up in the table 3

Dataset	neuron in each layer	training error	validation error
Including 27 companies	32	456.124	2048.929
Including 27 companies	64	353.480	1593.791
Dow Jones only	32	335.723	1577.969
Dow Jones only	64	282.863	1021.584
Including 27 companies	128	395.536	2181.973
Including 27 companies	64 (3 layers)	437.985	1910.666

Table 3. Dropout effect on LSTM

Comparing the first 4 models in table 3 to the four similar models in tables 1 and 2, all models included dropout perform worse than without dropout, except for the model including all stocks with 64 cells in each layer. Because of that, further tuning were done with the sixty-four-cell model, but neither increasing number of layers nor increasing the number of cell improves its validation results.

Hence the best LSTM model is the single LSTM layer, 64 cells and learning rate 0.01.

3.2. GRU

Tables 4 and 5 show the performance of the GRU model on the training and validation data based on different learning rate and model architecture. There are considerable similarities in tuning parameters for GRU compared to tuning parameters for LSTM.

Compared to LSTM, there are similarities in the performance trends between the two datasets, but there are some noticeable difference too. Concerning similarities, the learning rate, 0.01 is again the optimal learning rate with the Adam optimizer on both datasets, as either increasing or decreasing the learning rate tend to lead to deteriorating validation results, given the same model structure. The only exception is observed for model built on Dow Jones only dataset, with two layers and 32 cells, where decreasing learning rate to 0.001 from 0.01 improved the model (deceasing RMSE from 1292.468 to 913.585), but it is not

Learning rate	Number of layers (neuron in each layer)	training error	validation error
0.01	1 (32)	430.095	1110.901
0.01	2 (32)	317.327	1611.727
0.01	3 (32)	339.294	1715.945
0.1	2 (32)	1922.375	7859.063
0.001	2 (32)	486.909	2837.252
0.01	1 (64)	394.974	1514.552
0.01	2 (64)	325.051	1266.461
0.01	3 (64)	313.269	1797.987
0.1	1 (64)	1023.921	5408.268
0.001	1 (64)	434.464	2698.369
0.01	1 (16)	445.825	**750.176**
0.01	2 (16)	405.078	945.039
0.01	1 (8)	484.379	1235.145
0.01	2 (8)	398.585	1981.086

Table 4. GRU performance on the 27 stocks dataset

Learning rate	Number of layers (neuron in each layer)	training error	validation error
0.01	1 (32)	161.898	741.989
0.01	2 (32)	221.161	1292.468
0.01	3 (32)	255.304	1446.995
0.1	2 (32)	206.260	3138.546
0.001	2 (32)	252.278	913.585
0.01	1 (64)	153.867	550.168
0.01	2 (64)	165.297	779.868
0.01	3 (64)	183.726	1045.858
0.1	1 (64)	215.546	8204.734
0.001	1 (64)	189.276	627.593
0.01	1 (16)	209.946	964.445
0.01	2 (16)	279.536	1533.852
0.01	1 (128)	154.460	**505.991**
0.01	2 (128)	181.741	714.085
0.01	1 (256)	165.704	1393.556

Table 5. GRU performance on the Dow Jones only dataset

enough to justify decreasing learning rate for improvement.

Looking at the first differences, on the Dow Jones only dataset, given the same number of hidden cells in a GRU layer, increasing the number of layers worsen the validation results consistently accross the three tested number of hidden cells. The same is also observed in the twenty-seven-stock dataset, except for models with 64 hidden GRU cells, where increasing layer from one to two improve performance (1514.552 to 1266.461 in RMSE), but increasing the layer again makes it perform worse.

Another difference is observed in the number of hidden cell. Looking at table 4, for single GRU layer a models, 16 cells is the optimal number of cell (with RMSE 750.176), and either increasing or decreasing the number of cell make it perform worse. The same trend is observed for two-layer GRU, as 16 seems to be the optimal number of cells, but models with single layer tend to have better performance With three-layer, increasing the number of cell from 32 to 64 worsen performance in validation error (1715.945 to 1797.987).

As for the Dow Jones only dataset in table 5 for single layer, increasing the number of cells between 16 to 128 improves the model, but increasing it again to 256 worsen the result. The same trend is observe two-layer and three-layer GRUs, where increasing cells improve model performance, but it was not tested as extensively as the single layer, as it was identified that increasing layer worsen performance of GRU built on the Dow Jones only dataset.

Overall, as with the LSTM, GRU also performs better when built on the Dow Jones only dataset, which also support the speculation that larger dataset contains significantly more noises.

Dropout was also implemented on the GRU, and also on model with more than 1 layer summed up in the table 6, models without the number of layer mentioned are two-layer model

At first only two-layer models with 32 and 64 cells were tested for each dataset. For the twenty-seven-stock data, the 32-cell model performed worse with dropout, but the one with 64 cells improved compared to its version without dropout. Hence two more models with dropout were tested via increasing cell number or increasing layer number, but both did worse.

The same were observed in the Dow Jones only data, but only increasing layer number were tested as comparing the model with 32 cells and 64 cells, the smaller performed better so increasing cell number were not tested. However, increasing the number of layer for the model with dropout on the dataset with only Dow Jones trading data did not improve performance

Therefore the best model is the GRU model built on the Dow Jones only dataset, with 1 layer, 128 hidden cell, as it achieve validation RMSE of 505.991

Dataset	neuron in each layer	training error	validation error
Including 27 companies	32	405.121	1872.451
Including 27 companies	64	402.976	1373.947
Including 27 companies	64 (3 layers)	487.576	1912.692
Including 27 companies	128 (2 layers)	391.792	1785.997
Dow Jones only	32	466.262	1009.100
Dow Jones only	64	300.729	968.289
Dow Jones only	32 (3 layers)	376.949	1416.183

Table 6. Dropout effect on LSTM

3.3. Performance on test data

As the best performing model has been identified, this model is retrained on the whole training data (training data + validation data). On the test set, this model gets an RMSE of 853.148, relatively higher than the validation error, but much better than the 1684.988 RMSE that the baseline prediction performed. Looking at the mean absolute percentage error, the model got 2.282%, or its prediction is only about 2% off the actual value, which is not far away. Finally, the R score from this model is 0.6702, signalling that the predicted result and actual result are adequately similar.

Figure 5 shows a closer comparison between the prediction and actual results:

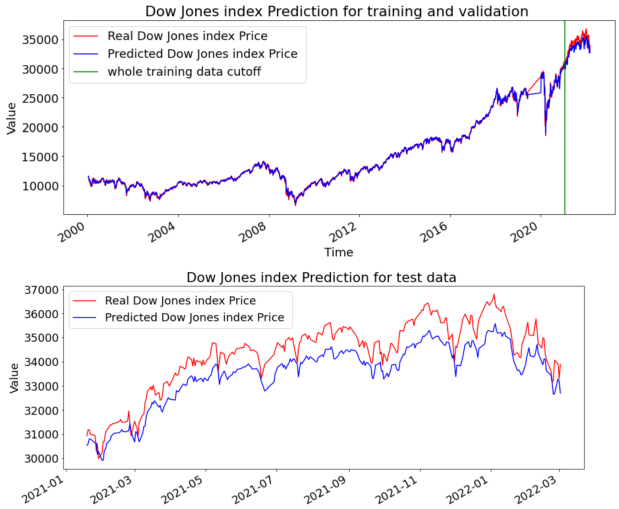


Figure 5. final model prediction on the whole data (top) and on the test data (bottom)

As shown in figures 5 and 1, the test data is cut off in the middle of the growth period of the index value. And figure 5 shows that the model underestimated the magnitude of the growth. However, the bottom of figure 5 shows that despite the underestimation, the model still captures the trend of the index relatively well.

4. Implementation

The coding implementation for this project can be found at `https://github.com/phucthai67/COMP-SCI-7318-assignment-3.git`

5. Conclusion

This report outline the process of exploring the Recurrent Neural Network, in particular, the Long-short term memory and the Gated recurrent unit, in predicting the stock market, specifically the Dow Jones index. The models was built using the trading history of either only the Dow Jones index, or a combination of the index and its stock components.

Regarding the stock forcasting problem, this is notoriously difficult due to the amount of noise and market only exhibit semi-strong form of efficiency [5], which means excess return can be gained in the market and studying all publicly available information and past information is not enough to earn excess return. Hence, it is also an area of high interest for many, not just in machine learning.

The majority of experimentation of the RNN revolves around changing the architecture of the network (the number of cells and the number of LSTM/GRU cells), changing the learning rate of the Adam optimizer to experiment with the model and implementing dropout on the model.

Overall, it was found that the model built on the dataset containing only the trading history of the Dow Jones index perform better than the model containing its companies components. This could be the result of its stock component containing a considerable amount of noises, and therefore did not help the model captured information very well. Between the two types of RNN experimented in this project, only a small difference in performance is observed, and the GRU perform marginally better than the LSTM. On a closer look at the implementation, it was found that, for this particular project, having fewer number of layers and having adequate amount of hidden cells generally helps the model performed better (which for this project was found to be 128 in the optimal model, increasing or decreasing this value will deteriorate performance).

The best model built in this report is a GRU model, using Adam optimizer with learning rate 0.01, have one GRU layer followed by a fully connected layer and 128 hidden cells. This model has 853.148 RMSE, 2.282% MAPE and 0.6702 R score. This result was calculated on the test data which was recorded late 2021, and early 2022, or post COVID 19, and in the growth period of the market as observed in 1. Despite getting relatively close to the actual value as shown by the MAPE of just more than 2%, the model under-estimated the growth, yet capture the value movement relatively well as shown by the R score. Yet the R score remains undesirable for practical use.

A major limitation of this report is the amount of data available to be used. For instance, the stock components used in this project does not reflect the stock components of the Dow Jones index within the chosen timeframe, as the companies used as the components are only the most recent companies. Additionally, prediction of future value for stock or index may not only depend only on its past movement or trading history, but there are other major economic factors that could be taken into account. For example, Bhandari et al. [1] considers macro economic factor such as interest rate, unemployment rate and consumer index, or technical indicator such as "Moving average convergence divergence" as part of the features that can be used to make future prediction. Also using a wide variety of data, including more technical measures such as price-to-book ration, price-to-earning ratio turnover ration or information about shareholder, Shen and Shafiq [12] employes feature engineering techniques, namely principal component analysis to filter and compress the inputs for the deep learning model.

In [1], LSTM models are used to predict the next day closing price of the S&P 500 index, the best model obtained in that report also employs single layer model, similar to the work done in this report. However, with a better-procured inputs, the optimal model built in [1] achieved MAPE score of 0.7989%, and R score of 0.9976, significantly better performance compared to the best model in this project.

For future experimentation, the work could incorporate a wider array of data, such as economic factors (including interest rate, exchange rate of the relevant market and consumer indexes), technical indicator as those used in [1] to make future forcast of index/stock price. Given the large array of data potentially relevant for this problem, feature engineering techniques such as principal component analysis, or feature learning tools such as autoencoder to extract most relevant feature for the problem, or seeking financial domain knowledge to choose better features as a starting point prior to using machine learning tools. More testing of the Recurrent network could also be implemented to further improve the model such as testing the model with different optimizer, varying the number of hidden cells in each RNN layer within one model.

References

[1] Bhandari, HN, Rimal, B, Pokhrel, NR, Rimal, R, Dahal, KR & Khatri, RKC 2022, 'Predicting stock market index using LSTM', *Machine Learning with Applications*, vol. 9.

[2] Chen, K, Zhou, Y, & Dai, F, 'A LSTM-based method for stock returns prediction: A case study of China stock market' *2015 IEEE international conference on big data (Big Data) (2015)*, pp. 2823-2824

[3] Cho, K, van Merrienboer, B, Bahdanau, D & Bengio, Y 2014, 'On the Properties of Neural Machine Translation: Encoder-Decoder Approaches'.

[4] Chung, J, Gulcehre, C, Cho, K & Bengio, Y 2014, 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling', *Cornell University*, DOI:10.48550/ARXIV.1412.3555

[5] Fama, E,F 1970 'Efficient Capital Markets: A Review of Theory and Empirical Work', *The Journal of Finance*, vol. 25, no. 2, pp. 383–417.

[6] Ganti, A 2022 'Dow Jones Industrial Average (DJIA): What It Is, Its Companies', *Investopedia*, viewed 16 November 2014, ¡https://www.investopedia.com/terms/d/djia.asp¿.

[7] Geron, A 2019,'Processing sequence using RNNs and CNNs' in Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems, 2nd edn, O'Reilly Media, Incorporated, Sevastopo

[8] Hochreiter, S 1998 'The vanishing gradient problem during learning recurrent neural nets and problem solutions', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107-116.

[9] Hochreiter, S and Schmidhuber, J, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780.

[10] Ince, H & Trafalis, TB 2008, 'Short term forecasting with support vector machines and application to stock price prediction', *International Journal of General Systems*, vol. 37, no. 6, pp. 677–687.

[11] Qiu, J, Wang, B & Zhou, C 2020, 'Forecasting stock prices with long-short term memory neural network based on attention mechanism', *PloS One*, vol. 15, no. 1.

[12] Shen, J & Shafiq, M,O 2020, 'Short-term stock market price trend prediction using a comprehensive deep learning system', *Journal of Big Data*, vol. 7, no. 1, pp. 66–66.

[13] Xiaohua, W, Phua, P,K & Weidong, L 2003, 'Stock market prediction using neural networks: Does trading volume help in short-term prediction?', in Proceedings of the International Joint Conference on Neural Networks, 2003, vol. 4, IEEE, pp. 2438–2442 vol.4.

[14] Yahoo Finance, n.d *Yahoo Finance*, viewed 12 November 2022, `https://au.finance.yahoo.com/`

[15] DOW 30, n.d *CNBC*, viewed 12 November 2022, `https://www.cnbc.com/dow-30/`

[16] Zhen, Z 2022, 'Recurrent Neural Network' lecture notes distributed in COMP SCI 7318 University of Adelaide, on 10 October 2022.