Performances of the Moving Average Model and the Recurrent Neural Network on the Forecast of Stock Market Indices

Peilin Jing, Jiachen Liu, Yaoyuan Zhang, Rashi Lodhi

Abstract

This project delves into the theoretical framework of two types of models. It aims to compare and contrast the performances of the moving average model and the recurrent neural network on the prediction of stock market indices. We used the data from Yahoo Finance with daily frequency from 1 January 2000 to 31 December 2021. Using a rolling window approach, we evaluated ARIMA-SGARCH to reflect the specific time-series characteristics and have better predictive power than the simple ARIMA and Recurrent Neural Network models. To assess the precision and quality of these models in forecasting, we compared their equity, their forecasting error metrics, and their performance metrics. The main contribution of this research is to show that the hybrid ARIMA SGARCH model outperforms recurrent neural networks in predicting log-return.

1. Introduction

As we know, any kind of prediction is a challenging task, especially when the future is volatile. The stock market is unpredictable and extremely volatile by nature, with the free market theorem compared with other investments. Hence, lots of investors are willing to take risks in the hope of making profits from the volatility. Numerous factors might influence the market, such as economic factors, social events, government policies, supply, demand, etc. Yet it is still difficult to receive a good performance on stock-price predictions despite many factors being used.

In this project, we will make attempts to predict day-ahead stock prices with only historical price data using different approaches. As we learned in lectures, the ARIMA model is a popular option when it comes to time series analysis. However, due to the uniqueness of the financial time series, we will adopt an improved method to make the predictions. Another popular method for predicting stock prices is the Recurrent Neural Network. Many existing papers, research, and blogs made similarly or the same attempts to use neural networks to predict stock prices with historical data. Although we know it is not likely to produce highly accurate predictions of the stock price due to the nature of the free market, we wish to be able to compare and contrast the performances of the two different approaches.

2. Literature Review

2.1 Moving Average Model

Applying Hybrid ARIMA-SGARCH in Algorithmic Investment Strategies on S&P500 Index [6]

The article we choose to reproduce for hybrid Arima SGARCH model is wrote by Nguyen Vo and Robert Slepaczuk, which is to compare the performance of Arima and GARCH family models to forecast S&P500 log returns and find the investment strategy. The hypothesis for this article is that ARIMA(p,1,q)-SGARCH(1,1) (hybrid model) with window size equal to 1000 can generate an algorithmic trading strategy that outperforms ARIMA(p,1,q). This article has 25 pages in total length and made comparison in different models, different window sizes, and difference distribution models by using error metrics and performance statistics. After different comparison, the results shows that ARIMA(p,1,q)-SGARCH(1,1) with window size 1000 does outperform other models in both error metrics and performance statistics. Furthrmore, the hybrid methods can generate a strategy that can outperform the ARIMA model even when we change the assumption for family of GARCH , window sizes, and distributions models.

2.2 Artificial Neural Network

Stock Price Prediction Using Recurrent Neural Networks (2018) [3]

This article by Israt Jahan tries to predict volatile financial instruments using the Recurrent Neural Network algorithm. Including the appendix, the whole article is 119 pages long and is very well structured. It explained the terminologies used for every detail and provided the data and code used in the process, making it easier to reproduce. According to Jahan, the advantage of Machine Learning are its ability to handle multi-dimensional data and its continuously quality improvable structure, especially in complex environments. The disadvantages are result interpretation and error diagnosis. Jahan used a single LSTM layer on five stock price time series. She compared predicted results with actual stock prices and concluded that "we can accurately predict the stock price one day ahead using this developed RNN-LSTM model."

3.1 Data Analysis

The first step in the process was cleaning the data. Then, we transformed the adjusted price into a daily logarithmic return, which was calculated according to the following formula:

$$r_t = ln\frac{P_t}{P_{t-1}}$$

Reasons to choose log returns:

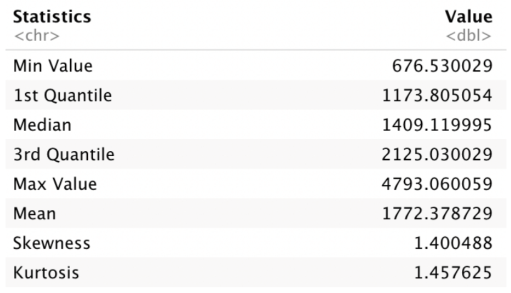
- can be added across time periods to create cumulative returns
- easy to convert between log return and simple return
- log return follows normal distribution

Advantages to log return having normal distribution: - Distribution only dependent on mean and sd of sample - forecast with higher accuracy (log return) - Stock prices cannot be normal distribution

3.1.1 Descriptive Satistics - Stock prices

Figure 1 below presents the descriptive statistics of the adjusted closing prices

```
knitr::include_graphics("images/Descriptive_Statistics.png")
```



Statistics <chr>	Value <dbl>
Min Value	676.530029
1st Quantile	1173.805054
Median	1409.119995
3rd Quantile	2125.030029
Max Value	4793.060059
Mean	1772.378729
Skewness	1.400488
Kurtosis	1.457625

Figure 1: Descriptive Statistics for Prices

As seen in Figure 2 there are a few periods, such as 2008, 2011, 2015, and 2018, that show high volatility of returns. We can expect to build more accurate forecasting models if we are able to mitigate and "smooth" such periods.

```
knitr::include_graphics("images/PricesPlot.png")
```

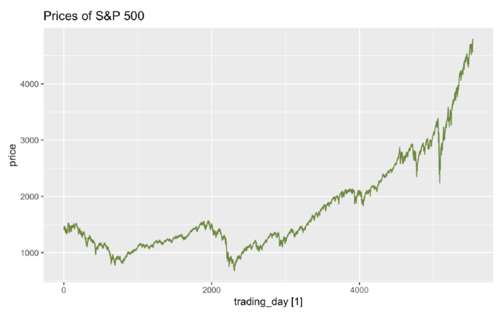


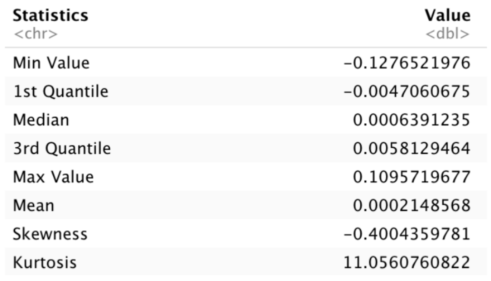
Figure 2: Prices Plot

Stock prices of SP 500 is that it is not normally distributed

3.1.2 Descriptive Satistics - Log Returns

Figure 3 presents the descriptive statistics of the adjusted closing prices

```r
knitr::include_graphics("images/LogReturnDescriptive.png")
```



Statistics <chr>	Value <dbl>
Min Value	-0.1276521976
1st Quantile	-0.0047060675
Median	0.0006391235
3rd Quantile	0.0058129464
Max Value	0.1095719677
Mean	0.0002148568
Skewness	-0.4004359781
Kurtosis	11.0560760822

Figure 3: Descriptive Statistics for Log Returns

We use log returns to build models

3.2 Methodology

3.2.1 ARIMA (p,d,q)

The ARMA process is the combination of the autoregressive model and moving average designed for a stationary time series. Autoregression (AR) describes a stochastic process, and AR(p) can be denoted as shown below:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

The moving average process of order q is denoted as MA(q) and the created time series contains a mean of q lagged white noise variables shifting along the series.

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

d is the number of differencing done to the series to achieve stationarity with I (d) so the ARIMA model can be expressed as

$$\text{Expand:} \qquad y_t = c + y_{t-1} + \phi_1 y_{t-1} - \phi_1 y_{t-2} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

p is the number of autoregressive terms (AR) q is the number of moving average terms (MA)

3.2.1.1 ARCH(p), GARCH(r,s) and Hybrid ARIMA-SGARCH

The `ARCH(p)` model is given:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i u_{t-i}^2$$

- Most volatility models derive from this
- Returns have a conditional distribution (here assumed to be normal) ARCH is not a very good model and almost nobody uses it.
- The reason is that it needs to use information from many days before t to calculate volatility on day t. That is, it needs a lot of lags.
- The solution is to write it as an ARMA model.
- That is, add one component to the equation, $\beta\sigma_{t-1}$.

The `GARCH(p,q)` model is

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i u_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

Where: α is news. β is memory. The size of $(\alpha + \beta)$ determines how quickly the predictability of the process dies out.

This leads us to lastly, ARIMA-SGarch

3.2.1.2 ARIMA SGARCH - Overview

Stock prices can be tremendously volatile during economic growth as well as recessions. When homoskedasticity presumption is violated, it affects the validity or power of statistical tests when using ARIMA models. We consider the SGARCH effect. The error term of the ARIMA model in this process follows SGARCH(1,1) instead of being assumed constant like the ARIMA model.

3.2.1.3 ARIMA SCGARCH - Steps

1) We conduct a rolling forecast based on an ARIMA-SGARCH model with window size(s) equal to 1000.
2) The optimized combination of p and q which has the lowest AIC is used to predict return for the next point. At the end, the vector of forecasted values has the length of 3530 elements
3) We describe and review our implementation of dynamic ARIMA(p,1,q)-SGARCH(1,1) models with GED distribution and window size(s) equal to 1000.
4) we evaluate the results based on error metrics, performance metrics, and equity curves.

Iteration of the forecasting model ARIMA(p,1,q)-SGARCH(1,1)

```
knitr::include_graphics("images/ARIMA_iteration.png")
```

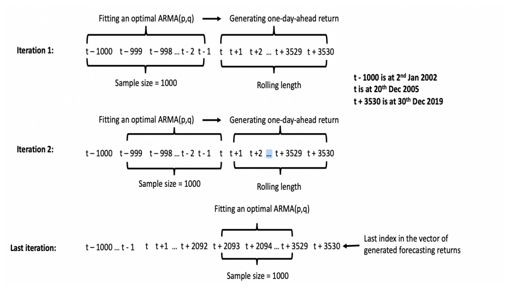

Fitting an optimal ARMA(p,q)
Generating one-day-ahead return
Iteration 1:
t − 1000 t − 999 t − 998 ... t - 2 t - 1 t t +1 t +2 ... t + 3529 t + 3530
Sample size = 1000
Rolling length
t - 1000 is at 2nd Jan 2002
t is at 20th Dec 2005
t + 3530 is at 30th Dec 2019
Fitting an optimal ARMA(p,q)
Generating one-day-ahead return
Iteration 2:
t − 1000 t − 999 t − 998 ... t - 2 t - 1 t t +1 t +2 ... t + 3529 t + 3530
Sample size = 1000
Rolling length
Fitting an optimal ARMA(p,q)
Last iteration:
t − 1000 ... t - 1 t t +1 ... t + 2092 t + 2093 t + 2094 ... t + 3529 t + 3530
Last index in the vector of generated forecasting returns
Sample size = 1000

Figure 4: ARIMA Iteration [6]

Flowchart of the forecasting model ARIMA(p,1,q)-SGARCH(1,1).

```
knitr::include_graphics("images/ARIMA_SGARCH_methodology.png")
```

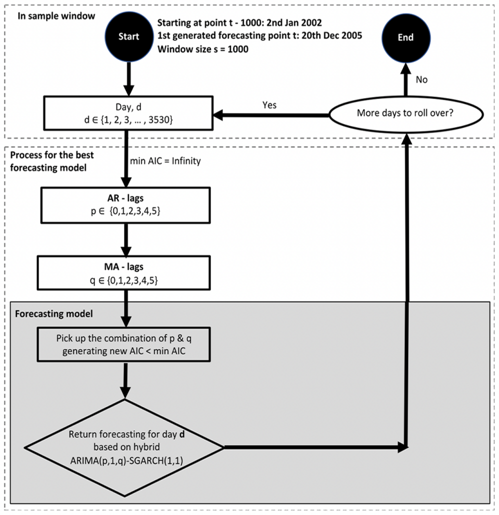

In sample window
Start
Starting at point t - 1000: 2nd Jan 2002
1st generated forecasting point t: 20th Dec 2005
Window size s = 1000
End
No
Yes
Day, d
d ∈ {1, 2, 3, ... , 3530}
More days to roll over?
Process for the best forecasting model
min AIC = Infinity
AR - lags
p ∈ {0,1,2,3,4,5}
MA - lags
q ∈ {0,1,2,3,4,5}
Forecasting model
Pick up the combination of p & q generating new AIC < min AIC
Return forecasting for day d based on hybrid ARIMA(p,1,q)-SGARCH(1,1)

Figure 5: ARIMA_SGARCH Methodology [6]

Flowchart of the forecasting model ARIMA(p,1,q)-SGARCH(1,1). This flowchart is for models with window size s = 1000.

3.2.2 RNN-LSTM Model

3.2.2.1 Recurrent Neural Network(RNN)

A **recurrent neural network** is a class of artificial neural network that uses sequential or time series data [1]. Unlike Feedforward Neural Network, RNN allows the output from some nodes to affect subsequent input to the same nodes by using connections between nodes to create cycles. As a result, the hidden layers produce the outputs with the input information and prior “memory” received from previous learning.

Unroll RNN

Recurrent Neural Network(RNN)

Another distinguish characteristic of RNN is that they share parameters across each layer of the network. Unlike feedforward neural networks having individual weight and bias for each node in one layer, recurrent neural networks share the same weight parameter within each layer. However, these weights are still adjusted during the processes of backpropagation and gradient descent to facilitate reinforcement learning.

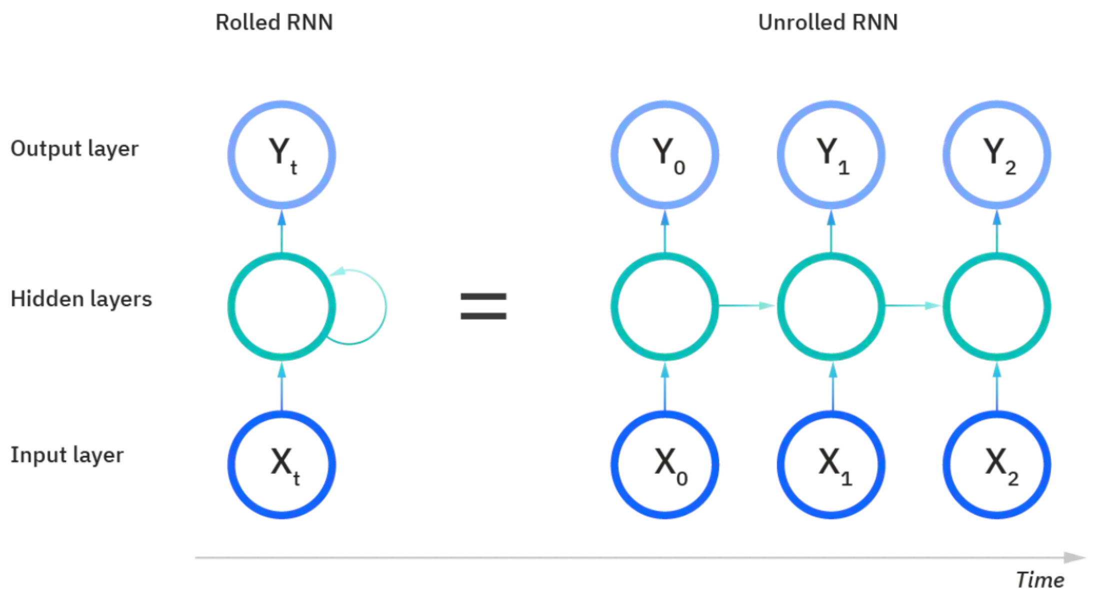


Figure 6: Rolled RNN and Unrolled RNN [1]

In feedforward neural network, backpropagation algorithm was used to calculate the gradient with respect to the weights. Recurrent neural network, on the other side, leverage backpropagation through time (BPTT) algorithm to determine the gradient as BPTT is specific to sequential data.

Activation Functions

In neural networks, an activation function determines whether a neuron should be activated and typically maps the input to $[0, 1]$ or $[-1, 1]$. The followings are two of the most commonly used activation functions and will be adopted later:

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Tanh (Hyperbolic tangent)

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU (Rectified Linear Unit) Activation Function

$$R(x) = max(0, x)$$

3.2.2.2 Long Short-term Memory (LSTM)

Long short-term memory network, usually known as LSTM, is a specific RNN architecture first introduced by Sepp Hochreiter and Juergen Schmidhuber as a solution to vanishing gradient problem [2]. Recall with an RNN, similar with human reading a book and remembering what happened in the earlier chapter, it remembers the previous information and use it for processing the current input. The shortcoming of the NN is that it is not able to remember long term dependencies due to the vanishing gradient. The LSTM is designed to alleviate and avoid such issues.

The LSTM consists of three parts [5]:

- **Forget Gate**: Choose whether the information coming from the previous time stamp should be remembered or can be forgotten
- **Input Gate**: Learn new information from the input to this cell
- **Output Gate**: Passes the updated information tot the next time stamp

Forget Gate

In an LSTM cell, the cell first need to decide if the information from previous time stamp should be kept or forgotten. The equation of the forget gate is:

$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f)$$

Where

- x_t = input to the current time stamp
- h_{t-1} = hidden state of the previous time stamp
- W_f = weight matrix associated with hidden state
- b_f = constant

After that, a `sigmoid` function is applied over f_t and make it a number between 0 and 1. Then f_t is multiplied with the previous cell state. If $f_t = 0$, the network will forget everything from the previous time stamp while $f_t = 1$ represents that the network will remember everything.

Input Gate and new information

Next we decide what new information we will store in the cell state. First, the input gate decides which values we'll update with `sigmoid` activation function:

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i)$$

Where

- W_t = weight matrix of the input associated with hidden state

Next, the new information is sent through a `tanh` layer to create the new candidate values:

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

New Cell State C_t

With previous work, the LSTM cell now updates the new cell state for the current time stamp as:

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

The current cell state C_t combines how much we decide to remember from the previous cell state C_{t-1} scaled by the forget gate and how much we wish to take in from the new current input $\tilde{C_t}$ scaled by the input gate.

Output Gate

Finally, the cell needs to decide what it is going to output and by how much. The filter of the output is the output gate, with the following equation:

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

The equation of the output gate is very similar with the forget gate and the input gate. Then, we push the cell state C_t through the `tanh` activation function to maintain the value staying in between -1 and 1, and multiply it by the output gate:

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

Overall Module

The previous steps conclude the architecture of the LSTM. The whole process can be summarized and displayed as the following:

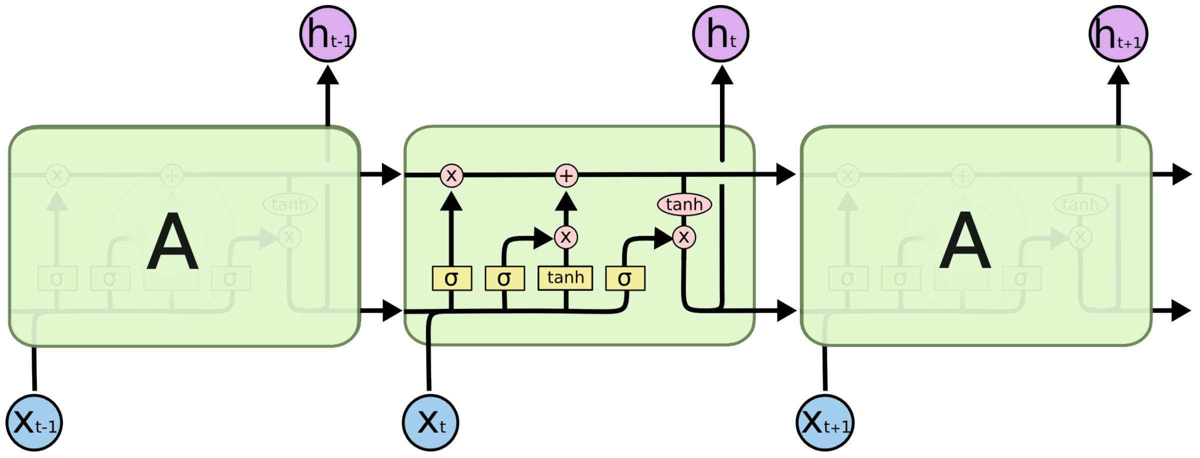


Figure 7: LSTM Chain [4]

4. Results

4.1 Model: Arima-SGARCH

In the paper, it had several different models, including SGARCH, EGARCH and Arima, etc. In the best model Arima-SGARCH, it tested for different window size and ,distribution model, and made the conclusion that hybrid Arima(p,1,q)- SGARCH(1,1) with w=1000 is the best model with best window size and distribution model according to the error metrics. In this section, we used 'rugarch' package to fit the hybrid Arima-GARCH model, compare the hybrid model, and do rolling forecast. so first, we will introduce a little bit about this package.

4.1.1 'rugarch' package exploration [7]

- **ugarchspec()**: Method for creating a univariate GARCH specification object prior to fitting.
- **ugarchfit()**: Method for fitting a variety of univariate GARCH models.
- **ugarchroll()**: Method for creating rolling density forecast from ARMA-GARCH models with option for refitting every n periods with parallel functionality.
- **ugarchboot()**: Method for forecasting the GARCH density based on a bootstrap procedures (see details and references).
- **ugarchforecast()**:Method for forecasting from a variety of univariate GARCH models.

- **ugarchfilter()**: Method for filtering a variety of univariate GARCH models.
- **ugarchpath()**: Method for simulating the path of a GARCH model from a variety of univariate GARCH models.

We used the ugarchspec() to define our model, and ugarchfit() to fit our GARCH model. Then, we used forecasting functions in our project to do volatility forecast, which are ugarchroll() and ugarchforecast().

4.1.2 Specify model

Specify sGarch model

```
# Specify sGARCH model
spec <- ugarchspec(
    variance.model =
      list(model = "sGARCH",
           garchOrder = c(1,1)),
      mean.model =
      list(armaOrder = c(0,0),
      include.mean = TRUE),
    distribution.model = "ged"
)
```

We choose the best model from the paper and reproduce it first. The best model is hybrid model ARIMA(p,1,q)-SGARCH(1,1) with GED distribution (SGARCH.GED 1000), so we define the model = "sGARCH" and define the distribution model as ged.

Information Criteria for sGARCH

```
Akaike       -6.518885
Bayes        -6.512893
Shibata      -6.518886
Hannan-Quinn -6.516795
```

Specify eGarch model

```
# Specify eGARCH model
spec <- ugarchspec(
    variance.model =
      list(model = "eGARCH",
           garchOrder = c(1,1)),
      mean.model =
      list(armaOrder = c(0,0),
      include.mean = TRUE),
    distribution.model = "ged"
)
```

Information Criteria for eGARCH

```
Akaike       -6.553511
Bayes        -6.546321
Shibata      -6.553514
Hannan-Quinn -6.551004
```

SGARCH and EGARCH comparison

These two models have similar information criteria, so we can say that it's somehow close to the comparison result in the paper since it has similar error metrics and performance metrics.

4.1.2 Forecast for fitted model

Forecast using ugarchforecast()

```
*------------------------------------*
*       GARCH Model Forecast         *
*------------------------------------*
Model: sGARCH
Horizon: 5
Roll Steps: 0
Out of Sample: 0

0-roll forecast [T0=2021-12-10]:
       Series   Sigma
T+1 0.0007399 0.01151
T+2 0.0007399 0.01153
T+3 0.0007399 0.01155
T+4 0.0007399 0.01158
T+5 0.0007399 0.01160
```

It’s convenient to use ugarchforecast() for forecast future returns, but it will have look-ahead bias, which it use the information that is not yet available or known. So we use…

Rolling forecast using ugarchroll()

We defined our window size in the rolling forecast, and according to the paper, there are three different window size, which are 1000, 500, 1500. We changed the window size in ugarchroll() function and hold all other things the same to do comparison.

Rolling Forecast for window size 1000

```
# Example code with window size 1000
roll <- ugarchroll(spec = spec,
                   data = data,
                   n.ahead = 1,
                   n.start = 3000,
                   refit.every = 50,
                   refit.window = "moving",
                   solver = "hybrid",
                   window.size = 1000,
                   keep.coef = TRUE)

show(roll)
```

```
*-------------------------------------*
*              GARCH Roll             *
*-------------------------------------*
No.Refits       : 51
Refit Horizon   : 50
No.Forecasts    : 2521
GARCH Model     : eGARCH(1,1)
Distribution    : ged

Forecast Density:
              Mu  Sigma Skew Shape Shape(GIG) Realized
2011-12-06 6e-04 0.0147    0 1.372          0   0.0011
2011-12-07 6e-04 0.0139    0 1.372          0   0.0020
2011-12-08 6e-04 0.0132    0 1.372          0  -0.0211
2011-12-09 6e-04 0.0159    0 1.372          0   0.0169
2011-12-12 6e-04 0.0148    0 1.372          0  -0.0149
```

```
2011-12-13 6e-04 0.0163    0 1.372          0  -0.0087

..........................
             Mu  Sigma Skew  Shape Shape(GIG) Realized
2021-12-03 9e-04 0.0162    0 1.3263          0  -0.0084
2021-12-06 9e-04 0.0159    0 1.3263          0   0.0117
2021-12-07 9e-04 0.0142    0 1.3263          0   0.0207
2021-12-08 9e-04 0.0137    0 1.3263          0   0.0031
2021-12-09 9e-04 0.0118    0 1.3263          0  -0.0072
2021-12-10 9e-04 0.0121    0 1.3263          0   0.0095

Elapsed: 36.76427 secs
```

Rolling forecast error metrics

Error Metircs for 1000 window size

```
GARCH Roll Mean Forecast Performance Measures
---------------------------------------------
Model       : eGARCH
No.Refits   : 51
No.Forecasts: 2521

        Stats
MSE 0.0001064
MAE 0.0065300
DAC 0.5518000
```

Error Metrics for 500 window size

```
GARCH Roll Mean Forecast Performance Measures
---------------------------------------------
Model       : eGARCH
No.Refits   : 51
No.Forecasts: 2521

        Stats
MSE 0.0001065
```

```
MAE 0.0065300
DAC 0.5411000
```

Error Metrics for window size 1500

```
GARCH Roll Mean Forecast Performance Measures
---------------------------------------------
Model       : eGARCH
No.Refits   : 51
No.Forecasts: 2521

        Stats
MSE 0.0001064
MAE 0.0065340
DAC 0.5518000
```

Compared these the error metrics for these three models with different window size, w = 1000 and w = 1500 have the similar MSE, but w = 1000 has lower MAE, and w = 1000 and w = 500 has similar MAE but w = 1000 has lower MSE. It's obviously that hybrid Arima-SGARCH model with w = 1000 is the best model among these three models. Also, hybrid Arima(p,1,q)-SGARCH(1,1) with w=1000 will be the best model for predicting S&P 500 log return volatility since it has best performance.

4.2 Model: RNN-LSTM

4.2.1 Workflow

As the target is to reproduce the result of the article "Stock Price Prediction Using Recurrent Neural Networks" by Israt Jahan [3], understanding the article itself is essential. The whole workflow of the article is in the following order: data collection, train-test split, data scale and preprocess, RNN setup and define LSTM cell, define loss and optimization function, train the RNN and calculate the loss, and then go back to LSTM cell step and tune the model parameters, and recalculate loss. The model itself will repeat this process until loss does not decrease, which means the model converges and is ready to be used.

4.2.2 Data Gather, Split, and Scale

The data article used is the closing price of Netflix stock from "2013-01-01" to "2017-12-31". A total of 1259 days of data were gathered, and 1008 of them were used as training data. Basically, the data of first four years. A Min-Max Scale technique was then applied to the training dataset, as shown below, and a reverse scale was needed after predicting.

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

In order to reproduce, we downloaded same data from Yahoo finance.

```r
start_date = "2013-01-01"; end_date = "2017-12-31"

nflx_prices <- tq_get("NFLX", get = "stock.prices",
                  from = start_date, to = end_date)

stock <- nflx_prices |> arrange(date) |>
  select(date, close) |> column_to_rownames('date')

head(stock, 3)
```

```
              close
2013-01-02 13.14429
2013-01-03 13.79857
2013-01-04 13.71143
```

Then, we do a data split and Max-Min scale. To exactly mimic the scale process in the article and to make sure reverse scale process is accurate, we wrote our own function. The first 1008 data point is assigned to `train_set`.

```r
# Max Min Scale
max_min_scale <- function(x, name = 'value') {
  df <- data.frame((x- min(x)) /(max(x)-min(x)))
  colnames(df) <- name; df}
max_min_scale_reverse <- function(y, x) {
  min(x) + y * (max(x)-min(x))}

train_set <- stock[1:1008,]
test_set <- stock[1009:nrow(stock),]
train_scaled <- train_set %>% max_min_scale
head(train_scaled, 3)
```

```
        value
1 0.000000000
2 0.005554876
3 0.004815041
```

LSTM model requires its input data to be in three dimension shape (x, y, z). The dimension x represents the number of batches, y represents the time series length, and z represents batch size. In the article, their input shape is different because they are using TensorFlow version 1, which is different from what we are using, TensorFlow version 2. Since 22 is approximately the number of trading dates in a month, we believe it is a reasonable choice as a time series window

```r
data_prep <- function(scaled_data, prediction = 1, lag = 22){
  x_data <- t(sapply(1:(dim(scaled_data)[1] - lag - prediction + 1),
                  function(x) scaled_data[x: (x + lag - 1), 1]))
  x_arr <- array(data = as.numeric(unlist(x_data)),
                 dim = c(nrow(x_data), lag, 1))
  y_data <- t(sapply((1 + lag):(dim(scaled_data)[1] - prediction + 1),
                     function(x) scaled_data[x: (x + prediction - 1), 1]))
  y_arr <- array(data = as.numeric(unlist(y_data)),
                 dim = c(length(y_data), prediction, 1))
  return(list(x = x_arr, y = y_arr))}

x_train = data_prep(train_scaled)$x
y_train = data_prep(train_scaled)$y
cat('x_train_shape: (', dim(x_train), ') \ny_train_shape: (', dim(y_train), ')')
```

```
x_train_shape: ( 986 22 1 )
y_train_shape: ( 986 1 1 )
```

4.2.3 RNN Setup

There are several important parameters to set up the RNN and LSTM cell. The `num_neurons` is the number of hidden layers in the LSTM cell. The more hidden layers we set, the more complicated the model is, which does not mean the model is better. Another parameter is `learning_rate`, which indicates how much change we want to make on hidden layer parameters each iteration. The larger the learning rate, the faster the model outcome changes, but this does not mean a quicker convergence. A common problem with large learning rate models is that they change parameters too much and end up missing the best parameter set in the middle. The article set the `num_neurons` to be 500, which is too many for TensorFlow version

2 models. Hence we reduced it to 50. The `learning_rate` was 0.002, and we adopted it in our model.

The activation function article used was ReLU, the loss is MSE, and the optimizer is Adam. To reproduce their result, we used the same setting.

```
num_neurons = 50
learning_rate = 0.002
# Define LSTM
get_model <- function(){
  model <- keras_model_sequential() |>
    layer_lstm(units = num_neurons,
               batch_input_shape = c(1, 22, 1),
               activation = 'relu', # Activation function is ReLU
               stateful = TRUE) |>
    layer_dense(units = 1)

  model %>% compile(loss = 'mse', metrics = 'mae', # Loss is MSE
    optimizer=optimizer_adam(learning_rate=learning_rate)) # Optimizer Adam
}
```

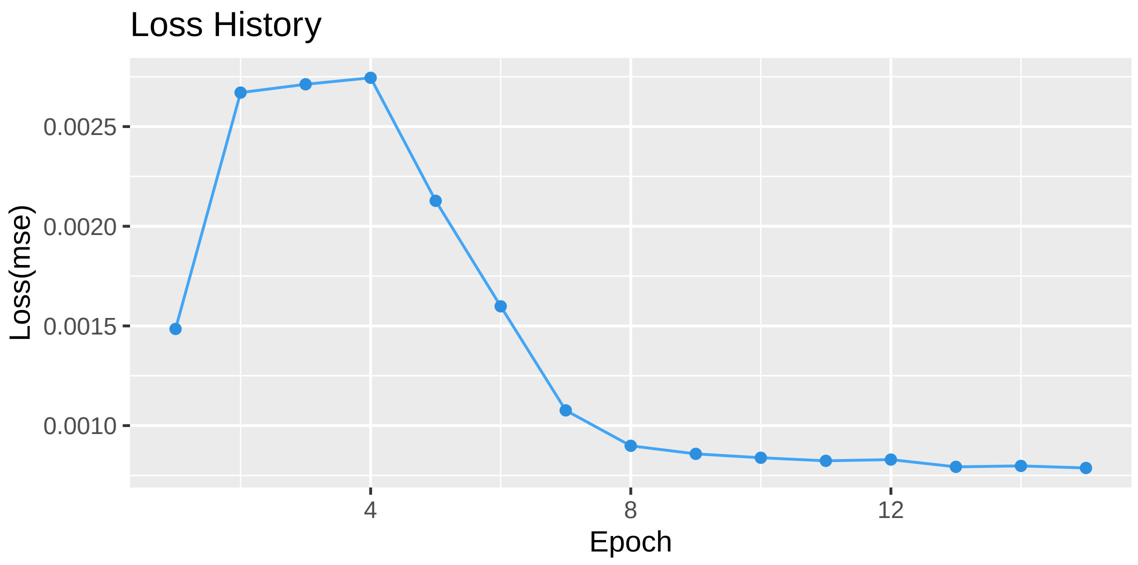
```
Loaded Tensorflow version 2.9.2

Model: "sequential"
________________________________________________________________________________
 Layer (type)                       Output Shape                    Param #
================================================================================
 lstm (LSTM)                        (1, 50)                         10400
 dense (Dense)                      (1, 1)                          51
================================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
________________________________________________________________________________
```

The model structure is as shown above. A dense layer is added to flatten the shape of the LSTM output to match our desired output shape.

4.2.4 Model Fit and Make Prediction

After set up data and RNN model, we start fitting. In the code below, the `epochs` means the number of iterations we want the model to do. For each iteration, the model back feed the loss result from last iteration and update the parameters. We set `epochs = 15` which means we want the model to update the parameters 15 times.

```
set_random_seed(1209)
lstm_model <- get_model()
lstm_model %>% fit(x = x_train, y = y_train, batch_size = 1,
    epochs = 15, verbose = 0, shuffle = FALSE) -> history
```

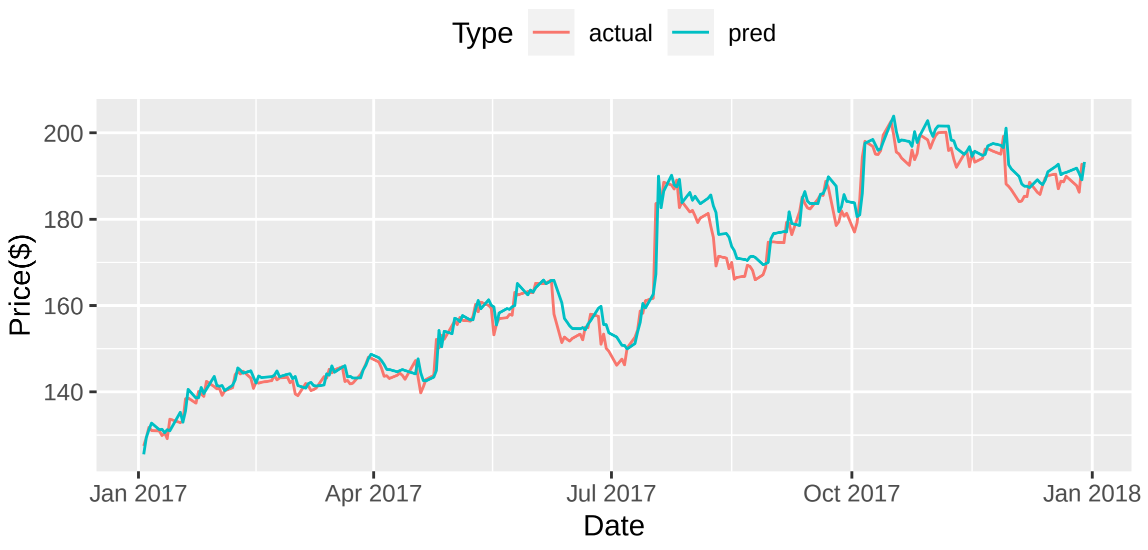


In the first several epochs, the loss increased, which means the model adjusted the parameters in the wrong direction, but later it found the way to reduce loss, and slowly at around epoch 10, converged.

Now that our model is well trained, we can make prediction use it. A for loop is used for the prediction as we need to scale the test data, make prediction, and scale it back.

```
model_prediction_lstm <- function(test_len, whole_data, model){
  t <- test_len # test size
  Tt <- nrow(whole_data) # whole data size
  t_start <- Tt - t + 1 # test start date
  predictions <- vector(length = t)
  for (i in 1:t){
    n <- t_start - 1 + i
    test_set = pull(whole_data)[(n-22):n]
    test_scaled <- test_set %>% max_min_scale
    x_test = data_prep(test_scaled)$x
    y_pred_scaled <- stats::predict(model, x_test, verbose = 0)
    y_pred <- max_min_scale_reverse(y_pred_scaled, test_set)
    predictions[i] <- y_pred
  }
  pred_df <- data.frame('date' = rownames(whole_data)[t_start:Tt] %>% as_date(),
    'pred' = predictions, 'actual' = pull(whole_data)[t_start:Tt])
  pred_df
}
```

```
        date     pred actual
1 2017-01-03 125.5396 127.49
2 2017-01-04 129.4479 129.41
3 2017-01-05 131.2491 131.81
```



The prediction results look pretty nice, the green line is close to the red line as we wanted.

4.2.5 Result Comparison with Article

The article showed its results both in graphs and in percentage error. To better compare, we reproduced the graph in exact same format and color.

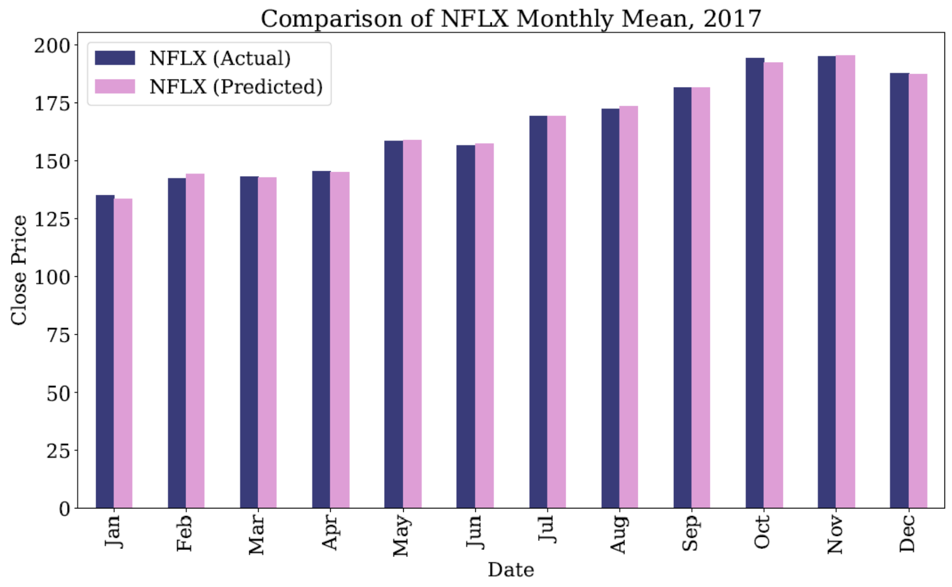
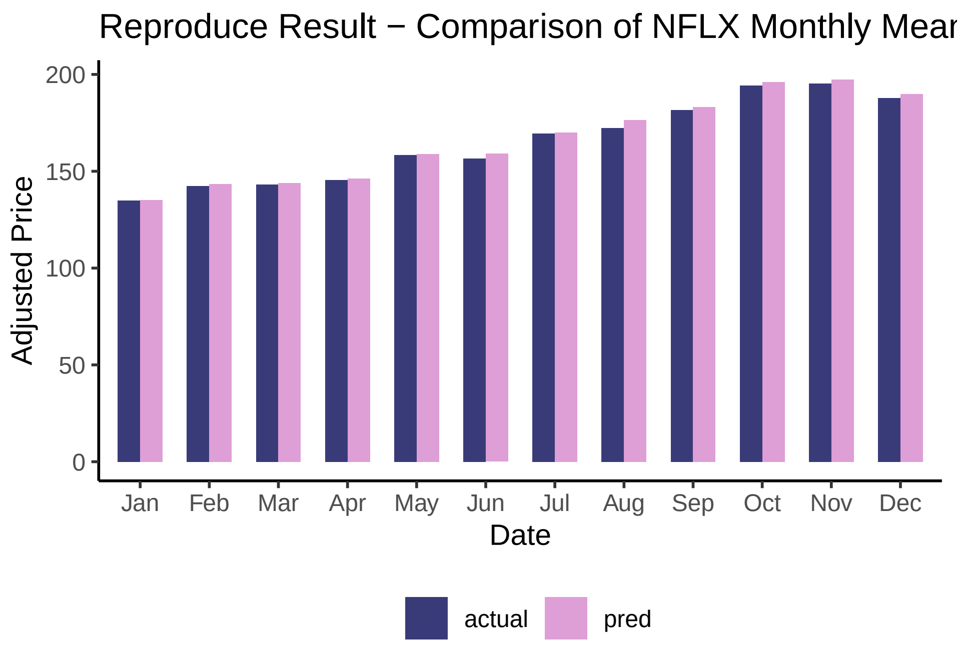


Figure 8: Article Result - Comparison of NFLX Monthly Mean [3]



As we can see, the graphs are quite similar. And if we look at the statistics below, our percentage error stats results are actually lower than the the article's, and the coefficient of determination (R^2) is very close. We consider this as a successful reproduce.

```
    |% of Error| Article  Our Results
1           mean  2.0000 1.4553167271
2            std  1.5400 1.3480666801
3       variabce  2.3800 1.8172837741
4            min  0.0000 0.0002329987
5 X25_percentile  0.8000 0.5238703257
6 X50_percentile  1.6500 1.1002221473
7 X75_percentile  2.9600 1.9550019530
8            max  9.2500 8.8855777088
9      R_squared  0.9589 1.0556739845
```

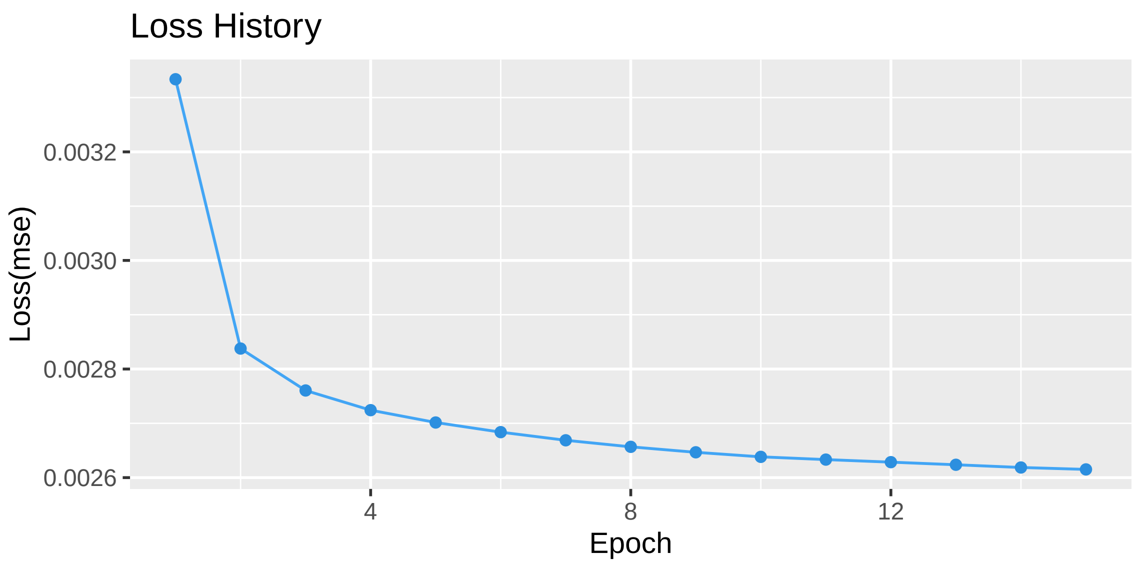
4.2.5 Result Comparison with ARIMA SGARCH

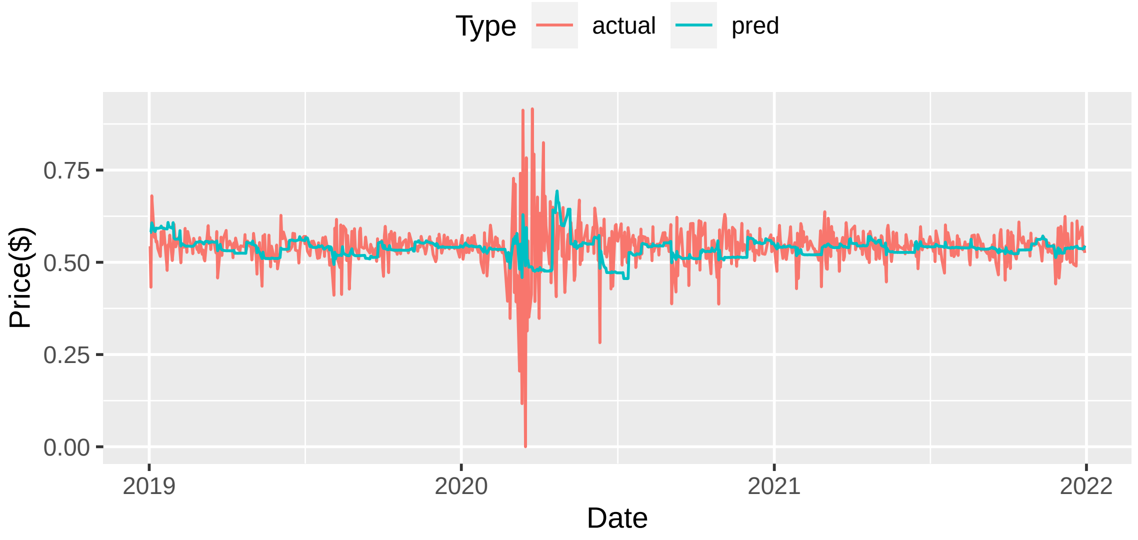
Our final goal is to compare the ARIMA SGARCH model and RNN model. To do that, we re-fit the same model to log-return of the S&P500 in the same period.

```r
# Data Preparation
sp_prices <- tq_get("^GSPC", get = "stock.prices",
                 from = "2000-01-01", to = "2021-12-31")
return <- sp_prices |> arrange(date) |>
  mutate(return = log(adjusted) - log(lag(adjusted))) |>
  drop_na(return) %>% select(date, return)
# scale
return$return <- max_min_scale(return$return) %>% pull(value)
# train test split
test_date <- "2019-01-01"
train_data_n <- return |> filter(date <= test_date) %>% column_to_rownames('date')
test_data_n <- return |> filter(date >= test_date) %>% column_to_rownames('date')
# data reshape
x_train_new = data_prep(train_data_n)$x; y_train_new = data_prep(train_data_n)$y
cat('x_train_shape: (', dim(x_train_new), ') \ny_train_shape: (', dim(y_train_new), ')')
```

```
x_train_shape: ( 4756 22 1 )
y_train_shape: ( 4756 1 1 )
```

As shown above, we have much more batches this time.





```
          MSE         MAE         RMSE        Rsq
1 0.003758965 0.03874852 0.004088116 0.2087921
```

The graph above shows that the results are not as good as before. The LSTM model is afraid to go too high or too low and just wonders in the middle.

5. Conclusions

```
  Error Metrics    sGARCH    RNN_LSTM
1           MSE 0.0001065 0.003758965
2           MAE 0.0065340 0.038748520
```

As the results shown above, the ARIMA-SGARCH model outperformed RNN-LSTM model, and by a lot. This is not what we initially predicted, which makes it very interesting. Neural networks usually are seen as powerful and complicated. It is said to mimic the human brain, which does many things other models cannot do, like image recognition and the natural language process. For example, the

Two reasons might explain the finding here. Firstly, Neural Network is inspired by the human brain, but the stock market is so volatile that even a real human brain cannot make an accurate prediction. Secondly, we only used one layer or LSTM, which is one of the most simple NN structures. In real life, a useful Neural Network can have hundreds of layers and be trained with much more data points.

All in all, simple layer LSTM is not as good as the ARIMA-SGARCH model when predicting stock market log-return. But we see some promising results from LSTM predicting stock price. The ARIMA-SGARCH model is very good for log-return prediction, and the RNN-LSTM model has potential for some financial instruments and is worth future exploring.

References

[1] IBM Cloud Education, "Recurrent Neural Network." IBM. September 14, 2020. https://www.ibm.com/cloud/learn/recurrent-neural-networks#toc-types-of-r-q1VkG6gmhttps://www.ibm.com/cloud/learn/recurrent-neural-networks#toc-types-of-r-q1VkG6gm

[2] Hochreiter, Sepp and Schmidhuber, Jürgen, "Long Short-Term Memory." Neural Computation. 1997. http://dx.doi.org/10.1162/neco.1997.9.8.1735

[3] Jahan, Israt, "Stock Price Prediction Using Recurrent Neural Networks." North Dakota State University. June, 2018. https://hdl.handle.net/10365/28797

[4] Saxena, Shipra, "Introduction to Long Short Term Memory(LSTM)." Analytics Vidhya. March 16, 2022. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

[5] Olah, Christopher, "Understanding LSTM Networks." colah's blog. August 27, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Vo, Nguyen and Slepaczuk, Robert. "Applying Hybrid ARIMA-SGARCH in Algorithmic Investment Strategies on S&P500 Index." Entropy. January 20, 2022. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8870867/pdf/entropy-24-00158.pdf.

[7] "A Short Introduction to the RUGARCH Package." Unstarched. January 11, 2014. http://www.unstarched.net/r-examples/rugarch/a-short-introduction-to-the-rugarch-package/.