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

Peilin Jing, Jiachen Liu, Yaoyuan Zhang, Rashi Lodhi

Abstract

1. Introduction

2. Literature Review

3. Methodology and Data

3.1 Data Analysis

3.2 Methodology

RNN-LSTM Model

Recurrent Neural Network(RNN)

A **recurrent neural network** is a class of artificial neural network that uses sequential or time series data. 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

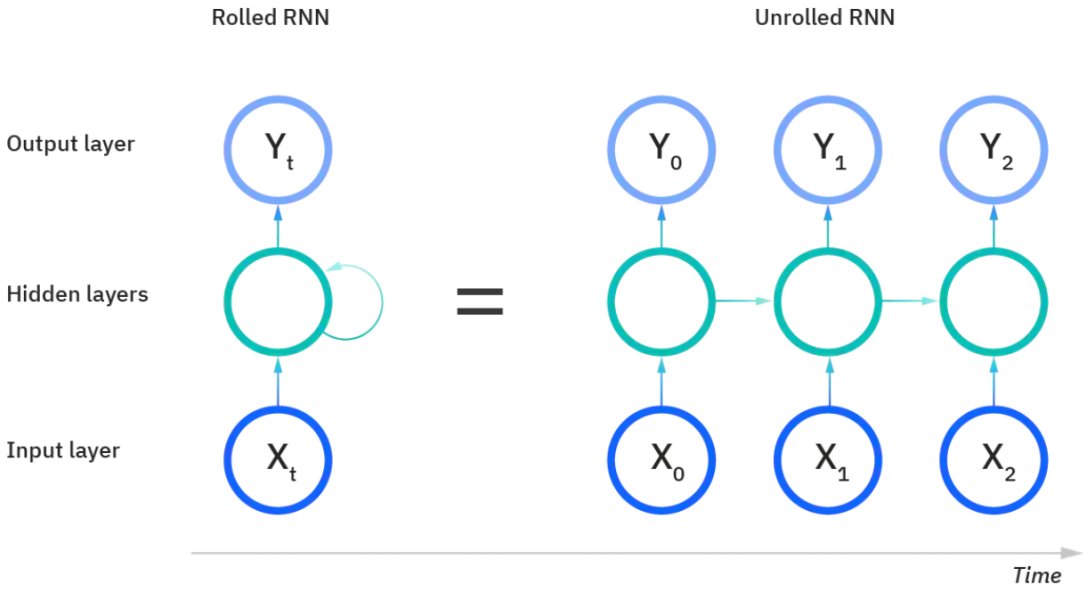

Figure 1: Figure 6. Rolled RNN and Unrolled RNN

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.

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

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

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. 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:

- **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:

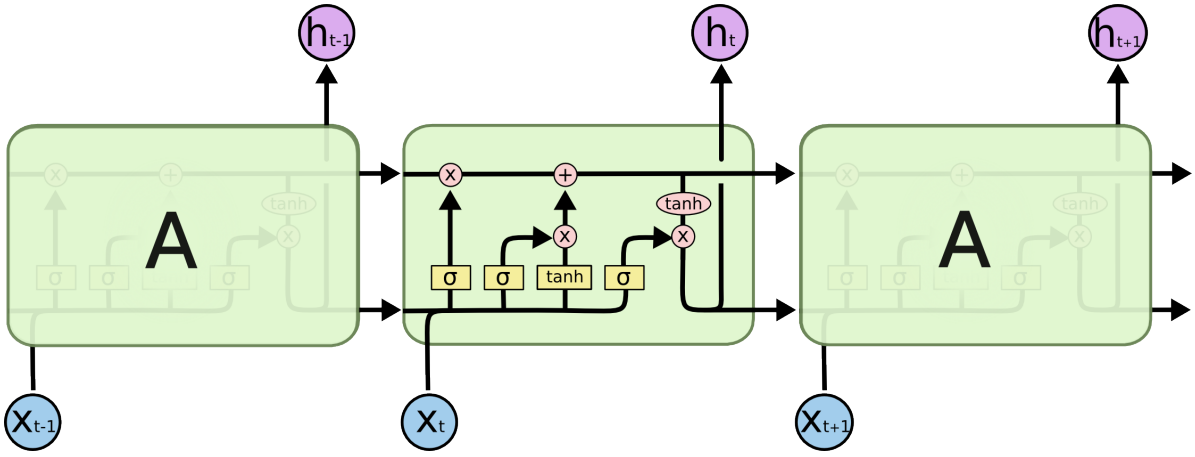

Figure 2: Figure 7. LSTM Chain

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.

'rugarch' package exploration

- **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().

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.

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

```r
# 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: 43.95554 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 50o log return volatility since it has best performance.

5. Conclusions

References