Foreign Exchange Rates prediction with LSTM

Huy Phung, Tashi Choden, Sahil Pasricha
University of Konstanz

February 2019

Contents

1 Abstract

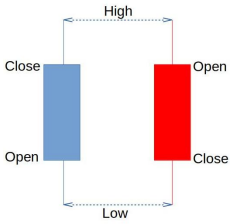
Foreign Exchange (abbreviation *Forex* or simply *FX*) Market is the decentralized market for currency investment. Forex market is the second most important market, after stock market. Supply and demand in the market determine Forex rate, in which a pair of currency can be exchanged. Forex rates has been studied in econometrics as a financial timeseries. The purpose of studying Forex rates is to explain the market behaviour or forecast future prices.
In our project, we use statistical models and deep learning model to predict the future rates of one step ahead. Our goal is to compare the effectiveness of LSTM and statistical models (ARIMA and VAR) as timeseries models, in both accuracy and performance.

2 Problem Description

2.1 Forex rates

Foreign Exchange rates (short Forex rates) are decided solely by support and demand of Forex market. Each rate represents the price to buy or sell a pair of currency (e.g. EURUSD) at the moment. The price to buy is called Bid price; the price to sell is called Ask price. The difference between Bid price and Ask price is called Spread. In this project we consider only the Bid price. However, if both Bid and Ask (and therefore Spread) were available, our analysis would be more precise.
Forex brokers update rates according to the market within milliseconds by standardized FIX (Financial Information eXchange) protocol. The time interval between FIX market update messages are not uniform; it may varies from a millisecond to few seconds. Therefore, the timeseries is of continuous time step. In order to simplify our analysis, we convert it to a data form that has discrete, uniform time step, while still keep important information. One possible way to do so is to format the rates into OHLC format. This approach is widely used in financial technical analysis. We partition the timeseries into intervals of uniform time length t. For each interval, we keep only 4 rate: the first (*open*), the last (*close*), the maximum (*high*), the minimum (*low*). Since the time intervals are uniform among dataset, we have the desired discrete, uniform time step for analysis.

(a) Original rates and OHLC form



(b) Candlestick

Figure 1: Illustration of OHLC timeseries formatting.

2.2 Prediction

In this project, we concern about the prediction of future Open, High, Low, Close prices. Other features, either originally exists (volume) or later added (mean, median, momentum), are only considered as supporting features. These features are only used for prediction of OHLC features.
The problem we are trying to solve in this project is declared as follow: given history data in OHLC form of Forex rates, namely $\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_k}$ where $\mathbf{x_i} = (x_i^O, x_i^H, x_i^L, x_i^C)$, predict future rate of *one* step ahead, $\mathbf{x_{k+1}}$.

2.3 Dataset

Acquiring real-time data is expensive, due to the fact that most FIX data providers requires subscription contract. However, Janus [2] collected an EURUSD rate dataset. The dataset consists of OHLC of BID price (no ASK price) of EURUSD rates from 2010 to 2016, thus contains 245444 values. Time interval for OHLC value is uniformly set to 15 minutes. Janus also published a smaller sample subset of the dataset, which contains only 14880 values. We would use the sample dataset later for ARIMA parameter search to reduce computational effort.
We separate 245444 records into 3 sets: train set, validation set, test set. In our default configuration, 80% of the dataset is trainset (19636 records), 10% is valid set (2454 records), 10% test set.
For better modelling and prediction, we add the following features to the original dataset:

- Median Price = (High Price + Low Price) / 2
- Mean Price = (High Price + Low Price + Open Price + Close Price) / 4

- Momentum = Volume * (Open Price - Close Price)

However, these additional features are used only *for* prediction. We do not build any model to predict these new features.

3 Model Selection and Evaluation

3.1 Akaike Information Criterion (AIC)

Akaike Information Criterion (AIC) is basically log-likelihood, but it penalizes a model by the number of parameters. AIC is widely used in statistical model selection, not only ARIMA and VAR, but also Hidden Markov Model and so on.

$$AIC = 2k - 2\ln(\hat{L})$$

in which k is the number of parameters and $\hat{L}$ is the likelihood. Since the log-likelihood is multiplied by -1, lower AIC means the model fits better to the data.

3.2 Root Mean Squared Error (RMSE)

Root Mean Squared Error is widely used to measure the difference between values predicted by a model and the actually observed values. Given y represents the actually observed values and $\hat{y}$ represents the values predicted, $RMSE$ is given by:

$$RMSE = \left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)^{\frac{1}{2}}$$

Root Mean Squared Error shows difference between y and $\hat{y}$ regardless the difference is negative or positive. However, since it does not take the range of possible values into account, it would be difficult to intepret the $RMSE$ result without knowing the possible range of predicted and actual values.

3.3 Mean Absolute Percentage Error (MAPE)

In order to measure the difference between predicted values and actual values with regarding to the scale, we use Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Compare to RMSE, MAPE is easier to interpret, for it is only a ratio without unit. Knowing the Max-Min range of the data beforehand is not necessary. The drawback could be the absolute value function, which is not continuous and thus make it difficult to take it as a loss function. However, we do not use it as a loss function here.

4 Statistical Models

4.1 Autocorrelation and White noises

Let $\{x_t\}$ is a timeseries with $\mu = \mu(t)$ is its mean and $\sigma = \sigma(t)$ is its variance. **Autocovariance** of lag k is defined as $C_k = E[(x_t - \mu)(x_{t+k} - \mu)]$ and **autocorrelation** is defined as $\rho_k = \frac{C_k}{\sigma^2}$.
A time series $\{e_t\}$ is a **discrete white noises** if its elements e_i are independent, identically distributed, have mean equals to zero and no autocorrelation between any of its values. Formally, $\mu_{\{e_t\}} = \mu_{\{e_t\}}(t) = 0$, $Cor(e_i, e_j) \neq 0, \forall i \neq j$.
A time series $\{x_t\}$ is a **random walk** if it satisfies that $x_t = x_{t-1} + e_t$ where $\{e_t\}$ is a discrete white noise as described above.
The following models we consider are similar to linear regression to some extents. Their rationales is that we find a linear relation between the value at time t and certain **lags** before it . Detailed explanation of the models can be found at [7] and [4].

4.2 ARIMA(p,d,q)

4.2.1 Model description

ARIMA(p,d,q) consists of three models: $AR(p)$, $MA(q)$ and integrated series of order d.
A timeseries $\{x_t\}$ is a *Auto-Regression* $AR(p)$ if it satisfies that

$$x_t = \alpha_1 x_{t-1} + \ldots + \alpha_{t-p} x_{t-p} + e_t$$

where $\{e_t\}$ is discrete white noises. So, *AR(p)* model looks back to find a linear relation with p previous values. Normally, we use Autocorrelation Plot (ACF) of log difference to find p, which is the highest lag in which we find a significant autocorrelation.

A timeseries $\{x_t\}$ is a *Moving Average* $AR(p)$ if it satisfies that

$$x_t = \alpha_1 e_{t-1} + \ldots + \alpha_{t-p} e_{t-p} + e_t$$

where $\{e_t\}$ is discrete white noises. The coefficients α_i are estimated, for example with maximum likelihood, by a sample (in our case, train set) to Instead of looks back into values, *MA(q)* looks into *differences* between timesteps, assuming that the timeseries is a random walk (thus differences between timesteps are discrete white noises). To find parameter q, normally we look at the Partial Autocorrelation Plot (PACF) to find the lag in which autocorrelation start to decay.
It is not necessarily true that we receive a discrete white noises by first order difference, namely $x_t - x_{t-1} = e_t$ and $\{e_t\}$ is discrete white noise. We possible need to take d times of differencing to get discrete white noises, namely $(x_t - x_{t-1}) - (x_{t-1} - x_{t-2}) - \ldots - (x_{t-d+1} - x_{t-d}) = e_t$. Let d be the order of difference, we have the component $I(d)$ in $ARIMA(p, d, q)$.

It is important that we select the proper parameters (p, d, q) that covers all the lags which affects the current value. In the next section, we consider another method for parameters selection that would give us the optimal parameter set (p,d,q) at once.

4.2.2 Parameters selection

Another method is that we try all possible combination of p, d and q to find the combination which gives us the lowest AIC. This method is computationally heavy, since we have to estimate the model by a hundreds of thousands of datapoints. However, it guarantees that the result parameters is optimal, namely it maximizes the likelihood to the dataset.

Algorithm 1 ARIMA(p,d,q) parameters select

procedure PARAMS SELECT($trainset, maxP, maxD, maxQ$)
  $MinAIC \leftarrow \infty$
  $OptimalModel \leftarrow None$
  for $p = 1$ to $maxP$ **do**
    for $d = 0$ to $maxD$ **do**
      for $q = 1$ to $maxQ$ **do**
        $Model \leftarrow$ `ARIMA(p,d,q)`
        $EstModel \leftarrow$ Estimate coefficients α_i of $model$ by $trainset$
        if $EstModel.AIC < MinAIC$ **then**
          $MinAIC \leftarrow EstModel.AIC$
          $OptimalModel \leftarrow EstModel$
        end if
      end for
    end for
  end for
  return $optimalModel$
end procedure

From the ACF and PCAF plot we can observe that after 20 lags, history values has no significant correlation to current value. Set $maxP$ and $maxQ$ to 25 and assume stationary on first order difference, the algorithm gives the following optimal parameters on trainset.

	Open	High	Low	Close
P	1	2	1	1
D	1	1	1	1
Q	18	25	26	18
AIC	-1.43336e+05	-1.44620e+05	1.45923e+05	-1.43399e+05

Table 1: ARIMA(p,d,q) optimal parameters and AIC

4.3 VAR(p)

4.3.1 Model description

VAR is applied to multivariate data. It is similar to $AR(p)$ model; however, instead of considering $\{x_t\}$ as real values (univariate), we analyse $\{\mathbf{x_t}\}$ as a timeseries of vectors $\mathbf{x_i} = (x_i^O, x_i^H, x_i^L, x_i^C)$. Formally, a timeseries $\{\mathbf{x_t}\}$ in

which $\mathbf{x_i}$ is a row vector of n dimensions, is *VAR(p)* if

$$\mathbf{x_t} = \beta_1 \mathbf{x_{t-1}} + \dots \beta_p \mathbf{x_{t-p}} + e_t$$

where e_t is discrete white noises and β_i is a column vector of n dimensions.

4.3.2 Parameters selection

Parameters selection for VAR(p) is done in the same way as ARIMA(p,d,q): we loop through a set of possible parameters to find the one with lowest AIC.

Algorithm 2 VAR(p) parameters select

```
procedure PARAMS SELECT(trainset, maxP)
    MinAIC ← ∞
    OptimalModel ← None
    for p = 1 to maxP do
        model ← VAR(p)
        Estimate coefficients of model by trainset
        if Estimated model has lower AIC then
            MinAIC ← model.AIC
            OptimalModel ← model
        end if
    end for
    return OptimalModel
end procedure
```

Set $maxP$ to 25 with the same reason as *ARIMA(p,d,q)*. The algorithm returns optimal parameter $P = 20$ for train set with $AIC = -1.053747548995608e+07$.

5 Deep Learning Model

5.1 Recurrent Neural Network

Recurrent Neural Network (RNN) is introduced by [6] to process sequential input. In RNN, each state connects to the following state to form a directed graph. The structure of RNN makes it capable of handling sequential data with temporal dynamic behaviour, such as timeseries or natural language

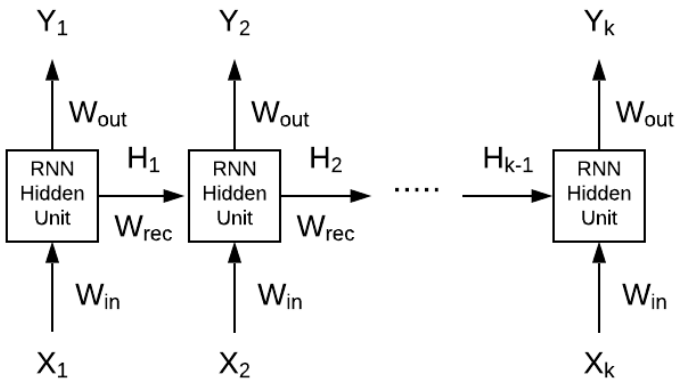


Figure 2: Recurrent Neural Network topology.

However, Pascanu [5] shows that Recurrent Neural Network has *vanishing gradient* and *exploding gradient* problems. These problems come from the topology of RNN, in which the layers were added consecutively. Let E_k be the error at k-th Recurrent unit. The gradient of E_k is calculated by chain rules over k timesteps:

$$\frac{\partial E_k}{\partial W_{rec}} = \sum_{i=0}^{k} \frac{\partial E_k}{\partial y_k} \frac{\partial y_k}{\partial h_k} \frac{\partial h_k}{\partial h_i} \frac{\partial h_i}{\partial W_{rec}}$$

Apply chain rules again on $\frac{\partial h_k}{\partial h_i}$, we have:

$$\frac{\partial h_k}{\partial h_i} = \prod_{t=i}^{k-1} \frac{\partial h_{t+1}}{\partial h_t} = \prod_{k \geq i > 1} \mathbf{W}_{rec}^T diag(\sigma'(x_{i-1}))$$

When $W_{rec} < 1$, the product decrease exponentially fast (vanishing gradient); when $W_{rec} > 1$, the product increase exponentially fast (exploding gradient).

5.2 Long-Short Term Memory

Hochreiter and Schmidhuber (1997) [1] introduced Long-Short Term Memory neural network architecture architecture to solve both vanishing gradient and exploding gradient problem from RNN. LSTM introduces in each unit

a memory cell, and a forget gate, so the past memory can be forgotten and thus does not affect the learning.

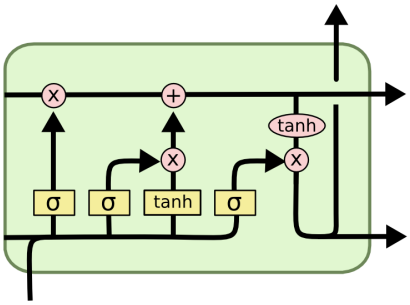


Figure 3: A LSTM hidden unit *source: colab.github.com*

5.3 Proposed Network Topology

We propose a network topology, which consists of 3 stacked LSTM layer of sequential output, with a dropout layer to tackle overfitting problem.

Sequence Input Layer
LSTM Layer of 125 hidden units
LSTM Layer of 125 hidden units
Fully Connected Layer
Dropout Layer of 0.1
LSTM Layer of 125 hidden units
Regression Layer

Table 2: Proposed LSTM Network topology

5.4 Training

Training data for LSTM network must be prepared in a different way than for ARIMA or VAR. Standardization of the training data before feeding it into LSTM network is necessary, since the scale differences among features would deteriorate the training process. Feeding the whole timeseries from the beginning to the network in order to predict the next value is ineffective, since we have observed before that autocorrelation decays and is negiligible after 25-th lag. Therefore, we partition (fold) the trainset into samples of

25 consecutives timesteps as X set, and use the observation right after that sample tobe the Y set.

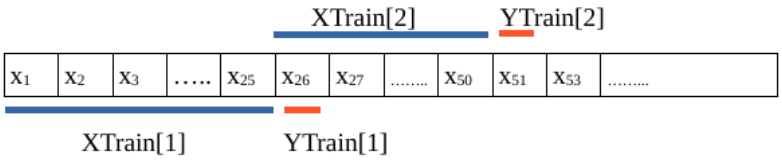


Figure 4: Train set folding.

We train the model by optimizer *adam*, with batch size 32. Other parameters can be found in the source code. As we can see from training progress, after the second epoch, the loss does not decrease any more, meanwhile the RMSE error still has the same distribution. It may imply that we could stop the training process at 3rd epoch without the loss of accuracy.

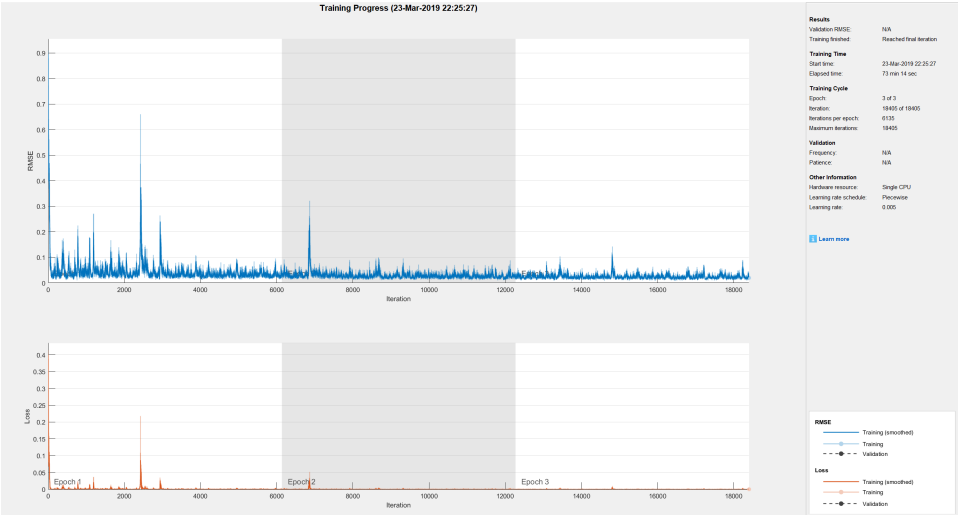


Figure 5: Training progress of LSTM on Open prices.

6 Experiments

6.1 Experiment design

We consider using LSTM network in two experiments. In the first experiments, we build and train the network to predict future values only by giving to history lags of one timeseries **(univariate)**. For example, we train the network to predict one Open price value in the future given only the Open prices in the past. This experiment is designed to compare the performance of LSTM with $ARIMA(p, d, q)$ model.
In the second experiments, we build and train the network so that it would predict one values ahead of one feature (e.g. one value of Open price ahead), given history values of *all* features in the past **(multivariate)**. This experiment is designed to compare the performance of LSTM to $VAR(p)$ model.
In both experiments, since we want to predict the future values of all Open, High, Low and Close features, we have to build and train one mode for each feature accordingly. Therefore, in each experiment we have to build and train 4 models.

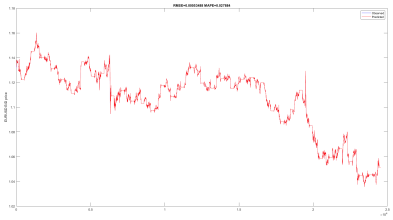
6.2 Univariate Experiment

6.2.1 Results from ARIMA(p,d,q)

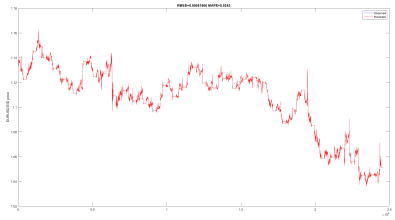
	Open	High	Low	Close
P	1	2	1	1
D	1	1	1	1
Q	18	25	26	18
RMSE on Test set	0.00053488	0.00051806	0.00052238	0.00053157
MAPE on Test set	0.027884	0.0263	0.026312	0.027741
Overall MAPE	0.0005268			
Overall RMSE	0.0270593			

Table 3: Optimal parameters and error on Test set

These following figures are the visualization of results.

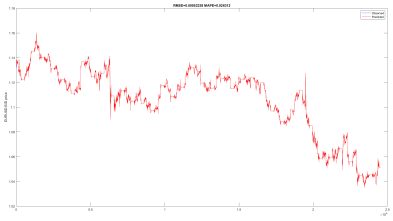


(a) ARIMA(1,1,18) Open.



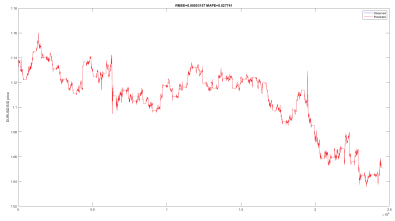
(b) High

(c) ARIMA(2,1,25) on High price.



(d) ARIMA(1,1,26) on Low price.

(e) Low



(f) Close

(g) ARIMA(1,1,18) on Close price.

Figure 6: Prediction on Test set.

Since the difference between observed and predicted data is small, we plot a small subset of values, for example Open prices from timesteps 500 to 600, in order to see the difference easier.

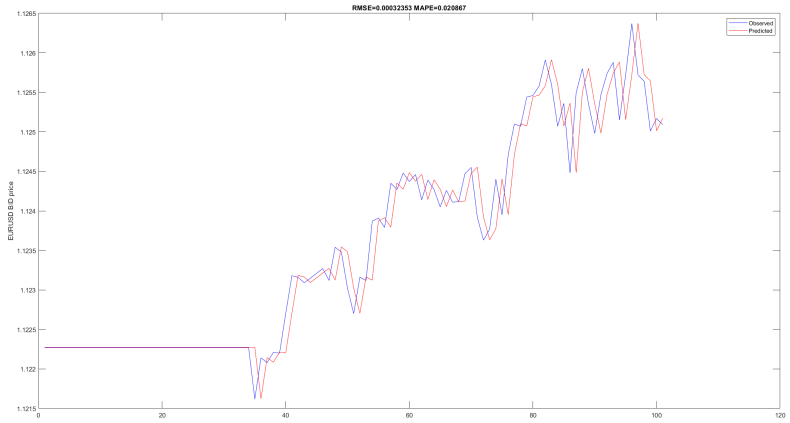


Figure 7: ARIMA(1,1,18) on Open price, sample of 100 values

6.2.2 Results from LSTM

Univariate	Open	High	Low	Close
RMSE on Test set	0.075301	0.024839	0.052389	0.048788
MAPE on Test set	5.9546	1.8086	4.2258	3.7145
Overall MAPE	0.053414			
Overall RMSE	3.925875			

Table 4: Optimal parameters and error on Test set

Visualization of prediction result. With LSTM, the difference is obviously visible.

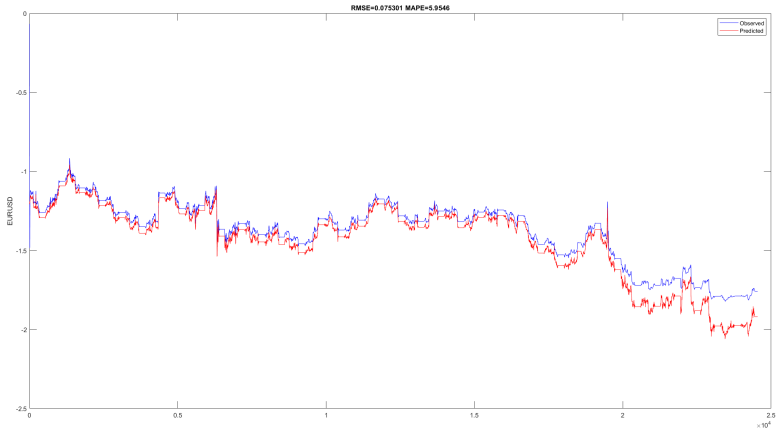


Figure 8: LSTM Univariate on Open price.

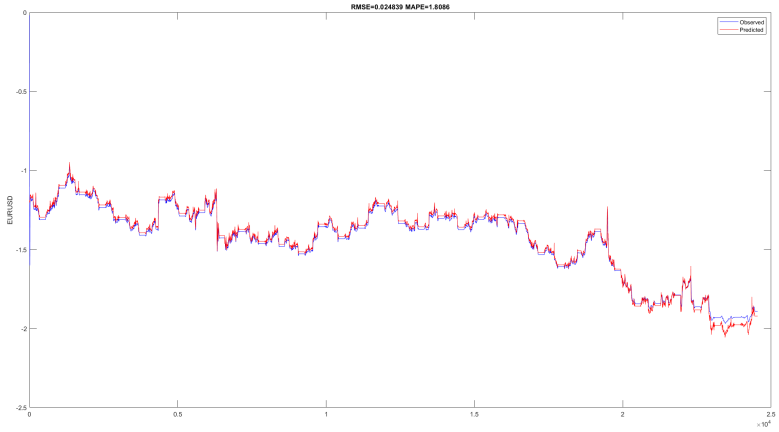


Figure 9: LSTM Univariate on High price.

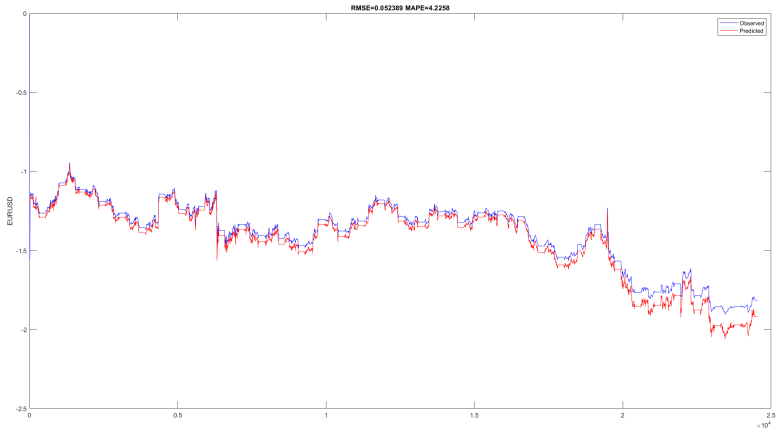


Figure 10: LSTM Univariate on Low price.

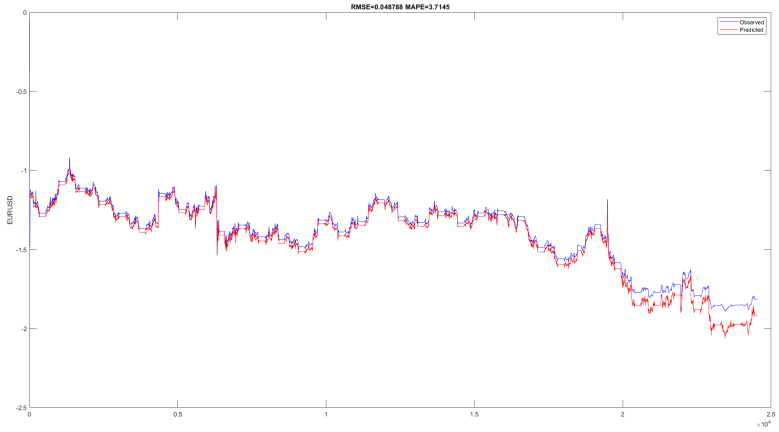


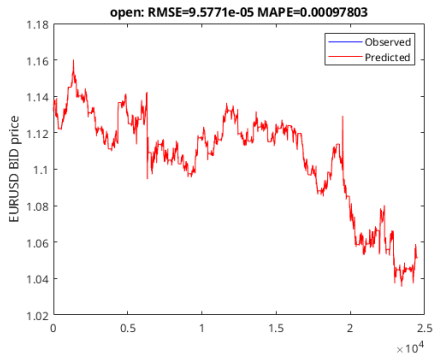
Figure 11: LSTM Univariate on Close price.

6.3 Multivariate

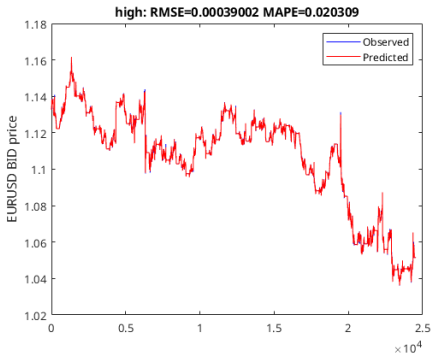
6.3.1 Results from VAR(p)

P=20	Open	High	Low	Close
RMSE on Test set	0.000095771	0.00039002	0.00037809	0.00053147
MAPE on Test set	0.00097803	0.020309	0.019355	0.027689
Overall MAPE	0.0003829817534299121			
Overall RMSE	0.0170827575			

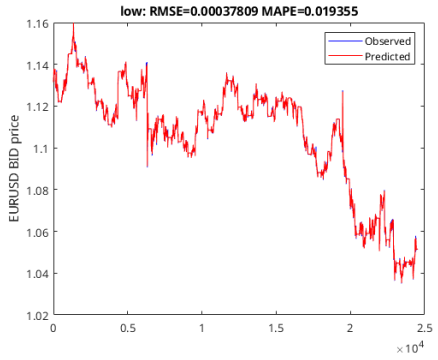
Table 5: Optimal parameters and error on Test set of VAR(20)



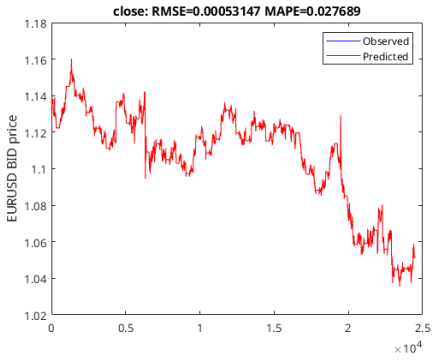
(a) Open



(b) High



(c) Low



(d) Close

Figure 12: Prediction on Test set.

Since *VAR(20)* fits the test set very well, for visualization, we draw an additional plot for less value, for example values from timesteps 500 to 600

of Open prices in order observe the minute differences between prediction and actual observation.

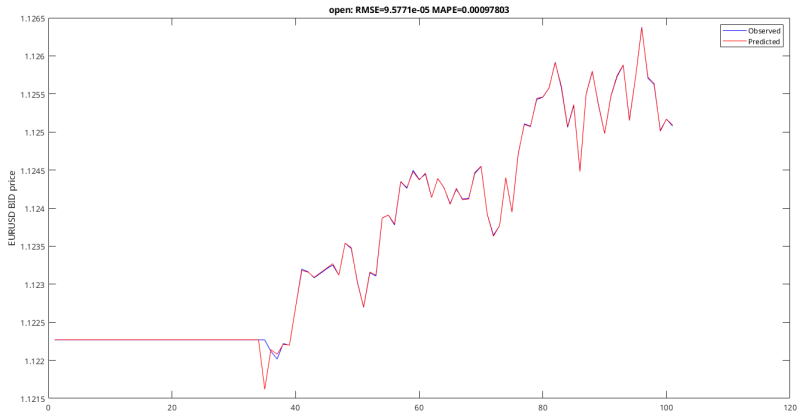


Figure 13: VAR(20) on Open price, sample of 100 values

6.3.2 Results from LSTM

	Open	High	Low	Close
RMSE on Test set	1.4085	1.376	1.3852	1.3818
MAPE on Test set	55.8928	54.7415	53.1869	54.1241
Overall MAPE	1.38792			
Overall RMSE	54.48633			

Table 6: LSTM Multivariate error on Test set.

Visualization of prediction result. With LSTM, the difference is obviously visible.

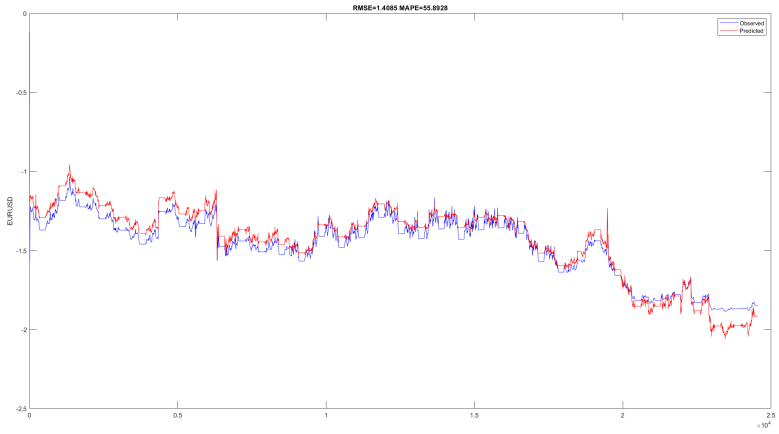


Figure 14: LSTM Multivariate on Open price.

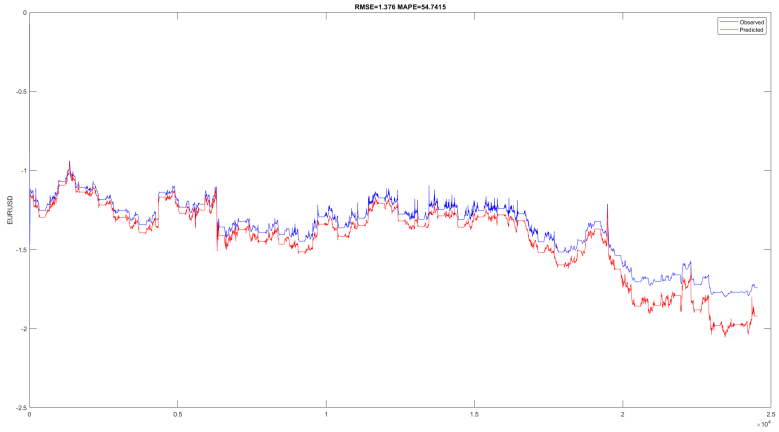


Figure 15: LSTM Multivariate on High price.

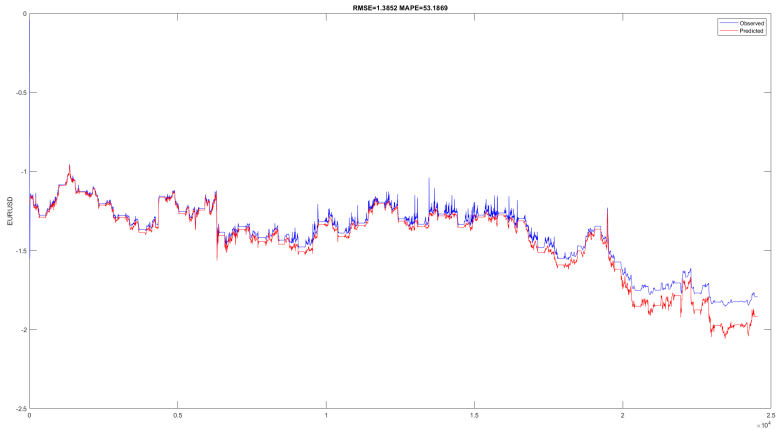


Figure 16: LSTM Multivariate on Low price.

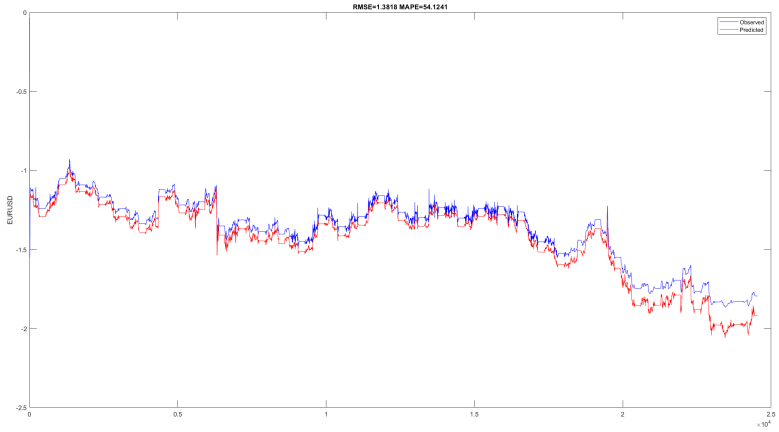


Figure 17: LSTM Multivariate on Close price.

7 Conclusion

According to the results, ARIMA and VAR deliver much better performance than LSTM neural network. More importantly, ARIMA and VAR is mathematically transparent and explanable.
However, the statistical models also have drawbacks. First, due to the fact

that all of them uses likelihood estimator to estimate the coefficients by the trainset, these models are prone to overfit. The more parameters a model has, the more likely to be overfit. That is the reason why we used AIC, for it penalizes the number of parameters. Second, ARIMA and VAR depends on the assumption that the relation between the value at current timesteps and its lags is linear. However, it can be assumed that more complicated functions are needed to reflect that relation. Third, the process of finding optimal parameters for statistical models could take even longer time than training a LSTM model.

In our project, the accuracy from LSTM models does not match that of statistical models. However, LSTM is still promising due to the fact that it can simulate any complicated functions, in compare with the statistical models which relies on linear functions. Furthermore, Dave Y. Kim and Mahmoud Elsaftawy [3] shows that for the same problem, use a different LSTM topology and training strategy, LSTM based model could achive RMSE 0.00029617 and MAPE of 0.021310060616307612, which is similar to our results achieved by statistical models ARIMA and VAR.

References

[1] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. 1999.

[2] Michal Januszewski. *EURUSD-15m-2010-2016*. `https://www.kaggle.com/meehau/EURUSD/data`.

[3] Dave Y. Kim and Mahmoud Elsaftawy. *EURUSD 15 minute interval price prediction*. `https://www.kaggle.com/kimy07/eurusd-15-minute-interval-price-prediction`.

[4] Quark Gluon Ltd. *Quantitative Trading*. `https://www.quantstart.com/articles/`.

[5] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[6] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[7] Ruey S. Tsay. *Analysis of financial time series*. Wiley series in probability and statistics. Wiley-Interscience, Hoboken, NJ, 2. ed. edition, 2005.