Application of ARIMA, RNN and LSTM on Foreign Exchange Rate Prediction

Fang Cabrera

Abstract

Autoregressive integrated moving average (ARIMA) has been one of the widely used linear models in time series forecasting during the past decades. Recent progress in machine learning has witnessed recurrent neural network (RNN) and long short term memory (LSTM) gain popularity. This paper reviews in depth these three models and explores the efficacy of their application on foreign exchange rate prediction. Furthermore, sentiment analysis is incorporated into the LSTM model to show that there is correlation between sentiments extracted from historical news and currency exchanges.

1. Models

1.1. ARIMA

One of the most important and widely used time series models is the autoregressive integrated moving average (ARIMA) model. ARIMA used to capture autocorrelation in the series by modeling it directly. ARIMA model is nothing but a composition of three models namely Auto-regression(AR), Integration(I), and Moving Average(MA). (**?**)

The **auto-regression** part of the ARIMA model is used to do a regression on the current data point based on the previous data points mentioned. These previous data points are also called lagged points. Regression without lags would fails to account for the relationships through time and overestimates the relationship between the dependent and independent variables. p is conventionally used to denote the number of lag points needed to do the regression. $AR(p)$ is an auto-regressive model with p lags:

$$y_t = \mu + \sum_{i=1}^{p} y_i y_{t-i} + \epsilon_t \tag{1}$$

The **moving average** models account for the possibility of a relationship between a variable and the residuals from previous periods. It assumes that the series is stationary, which means there's no trend or seasonality, variance stays constant and autocorrelations remain steady throughout the entire series. $MA(q)$ is a moving average model with q lags:

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_t \epsilon_{t-i} \tag{2}$$

where θ_q is the coefficient for the lagged error term in time $t-q$.

The **ARMA** model, which include **AR** model as a special case, also assumes that the series is stationary. It captures all forms of correlation by including lags of the series and of the forecasted errors, combining both p autoregressive terms and q moving average terms. $ARMA(p, q)$:

$$y_t = \mu + \sum_{i=1}^{p} y_i y_{t-i} + \epsilon_t + \sum_{i=1}^{q} \theta_t \epsilon_{t-i} \tag{3}$$

Adding a differencing step to ARMA leads to the **ARIMA** model. Because ARIMA assumes stationarity, the differencing operator is used to remove trend and/or seasonality. d is conventionally used to represent the order of differencing. ARIMA model selection typically follows the Box-Jenkins Methodology, which breaks the process down to three steps: identification, estimation and diagnostic checking. Important during the identification step is the problem of parameter choices, where the help of autocorrelation function (ACF) and partial autocorrelation function (PACF) come in handy. ACF is the proportion of autocovariance of y_t and y_{t-k} to the variance of a dependent variable y_t:

$$ACF(k) = \rho_k = \frac{Cov(y_t, y_{t-k})}{Var(y_t)} \tag{4}$$

PACF is the simple correlation between y_t and y_{t-k} minus the part explained byt the intervening lags:

$$\rho_k^* = Corr[y_t - E^*(y_t|y_{t-1}, ..., y_{t-k+1}), y_{t-k})] \tag{5}$$

where $E^*(y_t|y_{t-1}, ..., y_{t-k+1})$ is the minimum MSE predictor of y_t.

Sample ACF and PACF are compared to those of various theoretical ARMA models, the properties of which are used as a guide to estimate plausible models and select appropriate p, d, and q.

1.2. RNN

In order for a model to deal with sequential data, the following **design criteria** need to be met with care:

- Handle variable-length sequences
- Track long-term dependencies
- Maintain information about order
- Share parameters across the sequence

From feed-forward neural networks to RNN
As seen in figure **??**, in a standard 'vanilla' neural network, information flows from input to output in one direction, which brings about a problem: it fundamentally can?t maintain information about the sequential data. Several mechanisms have been devised to aid feed-forward network, but each has its own problem. *Fixed window* has difficulty modeling long-term dependencies. *Bag of words* uses the entire sequence as set of counts stored inside a fixed length vector. But problematically, counts don't preserve order, which means in using counts any sequence information is completely abandoned. *Expansion of fixing window* to include more context seems to be feasible but because of the absence of parameter sharing, things we learn about the sequence won't transfer if they appear elsewhere in the sequence.

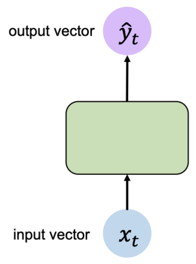

Figure 1. vanilla neural network. (**?**)

Recurrent neural networks (RNNs), on the other hand, allow information to persist through loops (see figure **??**). RNN applies a recurrence relation at every time step to process a sequence. Because the input layers activity patterns pass through the network more than once before generating a new output pattern, the network can learn extremely complex temporal patterns. RNNs make use of sequential information unlike tradition neural networks where all inputs are assumed to be independent. Recurrent neural networks do not use limited size of context. By using recurrent connections, information can cycle inside these networks for arbitrarily long time (see [5]). The network has an input layer x, hidden layer h (also called context layer or state) and output layer y. Input vector x_t is formed by concatenating vector w representing current word, and output from neurons in context layer h at time $t-1$. Hidden and output layers are then computed as follows:

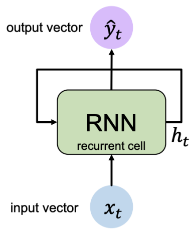

Figure 2. In RNN, the computation updates the internal state h_t, then passes the information about the state from this step of the network to the next. (**?**)

Ouput vector:

$$y_t = W_{hy}h_t \tag{6}$$

Update hidden state:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{7}$$

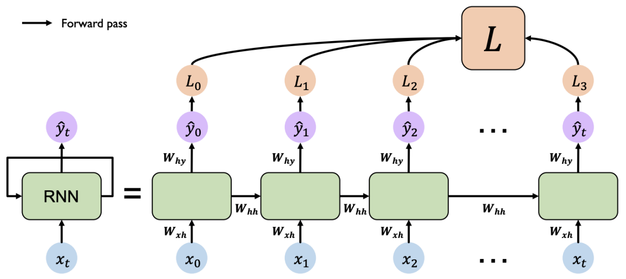

Figure 3. Recurrent neural network's computational graph across time (**?**)

1.3. LSTM

Comparison with standard RNN
One tough puzzle RNNs have to solve is the problem of long-term dependencies. Multiplying small numbers together, harder and harder to propagate errors further back into the back because the gradients are becoming smaller and smaller (the vanishing gradients problem), this in turn bias network to capture short-term dependencies. Mechanisms have been engineered to tackle with the problem. Among them, there are simple ones like using the ReLU activation function, initialize weights to identity matrix and biases to zero. A more robust solution is to use a more complex type of recurrent unit with gates that can more effectively track long-term dependencies by controlling what information is passes through and what?s used to update the cell state. Long Short Term Memory (LSTMs) networks rely on exactly such gated cells (figure **??**) to track information throughout many time steps.

In a standard RNN, repeating modules contain a simple computation node, whereas LSTM repeating models contain interacting layers that control information flow. Key ideas behind this design are:

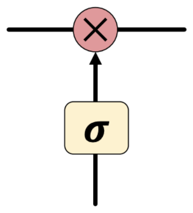

Figure 4. LSTM gate (**?**)

1. LSTMs maintain a cell state C_t in addition to standard RNN's h_t where it's easy for information to flow along relatively unchanged.
2. LSTMs use gates to add or remove cell state. Specifically, gates let information through optionally via a sigmoid neural net layer followed by point wise multiplication, where sigmoid function forces the input to the gate to between 0 and 1, which serves the purposes of flow regulation, meaning capturing how much input should pass through the gate.

How does LSTMs work? The operation of LSTMs boils down to three key steps:

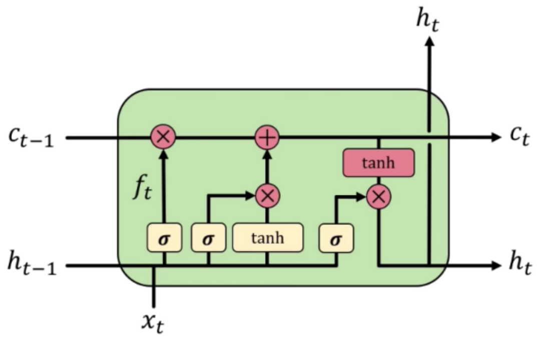

Figure 5. A LSTM cell containing the three key steps of operation: forget, update, output. (**?**)

- Forget: Decide what information is irrelevant to the previous cell state; As seen as the leftmost σ layer in figure **??**. The f_t function is parameterize by a set of weights and biases:

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

  This layer uses previous cell output h_{t-1} and input x_t, and then output a value between 0 and 1 which corresponds to somewhere between completely forget and completely keep the information.

- Update: Take prior information and current input, selectively update cell state values; This is achieved, as seen in figure **??**, by the combination of a σ layer which is used again to decide what values to update and a tanh layer to generate new vector of 'candidate values' that could be added to the state. We then apply the forget operation to previous internal cell state $f_t * C_{t-1}$ and add scaled candidate values: $i_t * \widetilde{C_t}$.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{8}$$

$$\widetilde{C_t} = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{9}$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{10}$$

- Output: Output certain parts of the cell state through an output gate. Again use a σ layer to filter through the output, and a tanh layer to squash values between -1 and 1. h_t is the filtered version of cell state.

$$0_t = \sigma(W_o[t_{t-1}, x_t] + b_o) \tag{11}$$

$$h_t = O_t * \tanh(C_t) \tag{12}$$

Now if we examine the above steps, it's clear that back-propagation from C_t to C_{t-1} requires only element-wise multiplication. Avoiding matrix multiplication means the vanishing gradient problem is solved. When LSTM units are linked up in a chain, uninterrupted gradient flow is achieved unlike in standard RNNs where repeated matrix multiplication has to be done.

2. Implementation

2.1. Data

2.1.1. Foreign exchange rate data

- Source: Open Exchange Rates API
- Range: 2016-01-01 to 2017-07-07 (552 records in total)
- Trading pairs collected: USD/EUR, USD/GBP, USD/CNY, USD/JPY
- Preprocessing:
  1. Calculate daily and weekly return based on past data;
  2. Translate the calculated rate of change into a 4-element class;
  3. Remove items where the daily return is 0;

2.1.2. News headlines

- Source: All-The-News dataset from Kaggle (**?**)
- Range: 2016-01-01 to 2017-07-07 (146,150 records in total)
- Preprocessing:
  1. Removed irrelevant fields, such as 'Publisher', 'Title', and 'URL';
  2. Clean up article content, including whitespaces, punctuation, and stopwords;
  3. Replace numbers with token 'NUM';
  4. Convert remaining words to lowercase;
  5. Perform tokenization;

2.2. ARIMA and RNN

2.2.1. Design

1. ARIMA: by plotting the data of currency exchanges rates, I observed that the variable is not stationary and this is true for all selected currencies. As a consequence, integration factor q is set to 1. Additionally I chose $p = 3$ and $r = 0$. Data was split into training set and testing set with a 7:3 ratio. MSE was calculated on feeding the test data to the model as shown in table 1.
2. RNN: Data was split in the same manner as ARIMA. The network was built with an input layer, output layer, and two hidden layers. The first hidden layer consists of 12 neurons and the second hidden layer 8.

2.2.2. Comparison

Performance of the two models is compared side to side as shown in figre **??** - figure **??**.

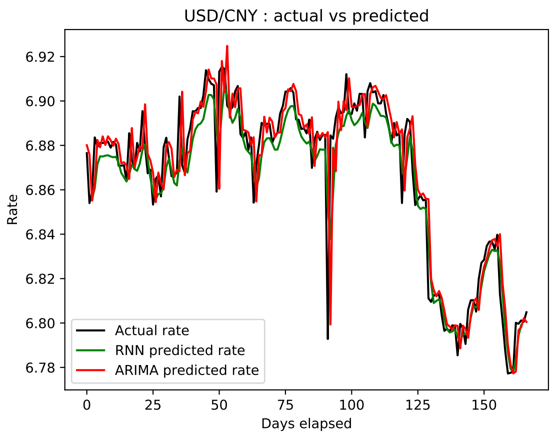

Figure 6. USD/CNY

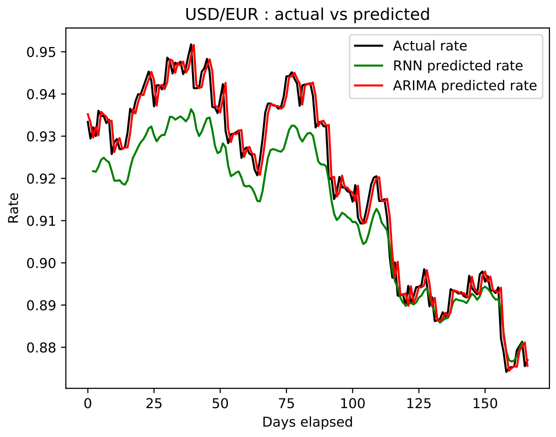

Figure 7. USD/EUR

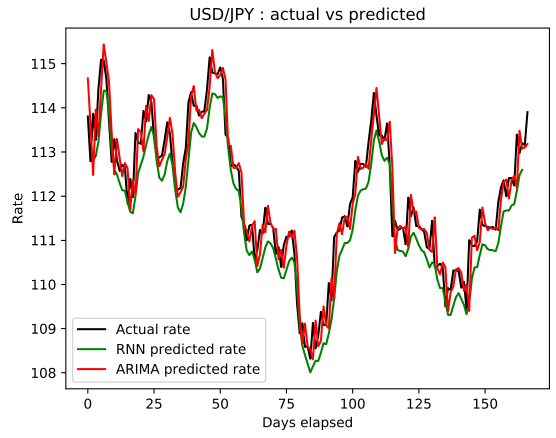

Figure 8. USD/JPY

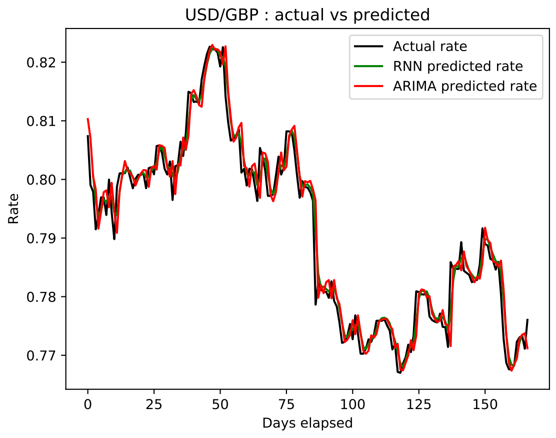

Figure 9. USD/GBP

Past studies report that the neural network model performed much better than the ARIMA model (**?**). But after analyzing these two models on currency exchange rate predictions, it is clear that ARIMA model is performing better than the neural network model. It's clear from table 1 that ARIMA performs better persistently.

CURRENCY	ARIMA	RNN	ACTUAL
USD/CNY	6.80048727 $\surd$	6.798996	6.80535
USD/EUR	0.87560187 $\surd$	0.8809963	0.87727
USD/JPY	113.17996635 $\surd$	112.593834	113.912
USD/GBP	0.771196 $\surd$	0.7734577	0.776225

Table 1. Prediction accuracies for ARIMA and RNN on date 2017-07-08 with $\surd$ indicating the better performing model

CURRENCY	ARIMA	RNN
USD/CNY	0.00024910	0.00024999
USD/EUR	1.26045833E-05	8.01440898E-05
USD/JPY	0.29101745	0.64706694
USD/GBP	1.26978747E-05	1.70068335E-05

Table 2. Mean square error obtained for predictions based on the given four currency exchange rates versus ARIMA model and neural network model

2.3. LSTM

2.3.1. DESIGN

1. SentiWordNet: The news data I obtained from Kaggle reach into other aspects of life beyond the financial world. Due to the high level of noise, I used SentiWordNet (**?**) to obtain the absolute scores of news titles for each day by parsing them against a predefined knowledge base per the country. Ten top scored news titles are then fed into the LSTM model. The news selection process is shown as algorithm **??**.

Algorithm 1 Greedy News Selection

```
**Input:** data $X$, size $m$, KBase $K$, priority queue $Q$
**for** $i = 1$ **to** $m - 1$ **do**
  Initialize $X_i.score = 0$. wordNum $n$.
  **for** $j = 1$ **to** $n - 1$ **do**
    **if** $X_{ij}$ is in K **then**
      increment $X_i.score$
    **end if**
  **end for**
  push $X_i$ onto Q, where priority is $X_i.score$.
**end for**
return top 10 in $Q$
```

2. *word2vec* embedding is then used to convert headlines into vectors (**?**). LSTM network then takes as input date and its news headlines selected by algorithm **??**.

3. output: *variation class*, as shown in table **??**.

Change	Tag
$<= -2.5\%$	NL
(-2.5%, 0%)	NS
(0%, 2.5%)	PS
$>= 2.5\%$	PL

Table 3. output of LSTM layer

4. *k-fold*: random sampling as typically used in k-fold algorithm does not preserve sequence information of time series. Thus I used a sliding window technique and split the dataset into 2k windows, we use the first k windows as train dataset and use k, k + 1 as test dataset. Figure **??** shows an example window split for k = 5.

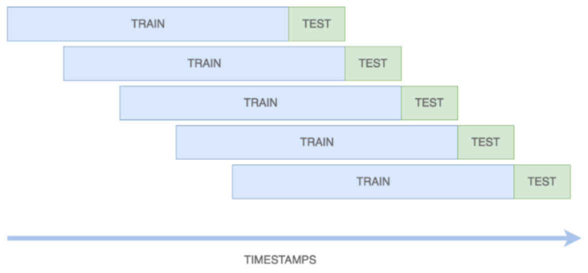

Figure 10. implementation of *k*-fold split, *k*=5

2.3.2. TRAINING AND EVALUTAION

All the LSTM networks are trained with the same hyper-parameters. Under each *k-fold*, the network went through 100 epochs of training with a learning rate of 0.0025. MSE is used as loss function, and the class-accuracy is used for accuracy calculation. The trained model is then evaluated using the testing set. I have chosen k = 7 in all training sessions.

2.3.3. RESULT

The resulting 7-fold mean-accuracy measures on each model are shown below in table 4. Baseline measures are generated by randomly assigning classes to inputs.

Currency	Acc	Baseline Acc
EUR	0.5466	0.2363
CNY	0.5604	0.2967
JPY	0.4871	0.2673
GBP	0.5057	0.1776

Table 4. Results table using mean accuracy of 7-fold validation, with *word2vec* embedding and LSTM model

3. Conclusion

From the implementation of the three models above, we can clearly see that ARIMA performed the best on this small scale experiment. Using sentiment gathered from top news to predict forex movement was an interesting attempt, but the LSTM model I built did not achieve as high accuracy as ARIMA/RNN. I'll further explore into this topic, hopefully come up with a hybrid approach that incorporates sentiments into a more sophisticated multilayer network rather than relying on sentiments as the only dimension.

References

G. Peter Zhang. Time series forecasting using a hybrid arima and neural network model. 2003.

MIT 6.S191. Introduction to deep learning. URL `introtodeeplearning.com`.

Andrew Thompson. All the news: 143,000 articles from 15 american publications. URL `https://www.kaggle.com/snapcrack/all-the-news.`

Babu AS and Reddy SK. Exchange rate forecasting using arima, neural network and fuzzy neuron. *J Stock Forex Trad 4:155. doi:10.4172/2168-9458.1000155*, 2015.

Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC 10)*, 2010.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *In Advances in neural information processing systems*, pages 3111 – 3119, 2015.

Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. 2011.

Arman Khadjeh Nassirtoussi, Saeed Aghabozorgi, Teh Ying Wah, and David Chek Ling Ngo. Text mining of news-headlines for forex market prediction: A multi-layer dimension reduction algorithm with semantics and sentiment. *Expert Systems with Applications 42*, pages 306 – 324, 2015.

Jason Brownlee. How to create an arima model for time series forecasting with python. URL `http://machinelearningmastery.com/arima-for-time-series-forecasting\-with-python/`.

Denny Britz. Recurrent neural networks tutorial, part 1 introduction to rnns. 2015. URL `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial\-part-1-introduction-to-rnns/`.