A Comparison of Long Short-Term Memory and ARMA Model on a Bitcoin Prices Dataset

Jose R. Navarro navarroj@mit.edu
Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA. 02139

Manuel A. Mundo mmundo@mit.edu
Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA. 02139

Abstract

Our project compared traditional statistical methods to study time series such as ARMA and ARIMA to methods based on Long Short Term Memory recurrent neural nets. RNNs empirically perform better than ARMA and ARIMA.

1. Introduction

RepoURL:

$https://github.com/sizfeetunder/6867-Final\_Project$

Our project presents a contrast of two different statistical methods to the problem of studying sequential data.

In particular we contrast linear models as it is the ARMA and ARIMA model with a highly nonlinear estimation model as is a Long-Short Term Memory neural network. We also experiment with auto encoders.

2. Models

In this section, the models used to understand the data are described.

2.1. Autoregressive Moving Average

AutoRegressive (AR) Moving Average (MA) models are the most frequently used theoretical framework to model time series data. The following formal definition is taken from Brockwell and Davis, "Introduction to Time Series and Forecasting". (Peter J. Brockwell, 2016)

Definition 1. A stochastic process $\{X_t\}$ is an ARMA(p,q) process if it is stationary and, for every t,

$$X_t - \phi_1 X_{t-1} - \ldots - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + \ldots + \theta_q Z_{t-q}, \quad (2.1)$$

where Z_t is a white noise process, i.e. $\{Z_t\} \sim N(0, \sigma^2)$, and the polynomials $1 - \phi_1 y - \ldots - \phi_p y^p$ and $1 + \theta_1 y + \ldots + \theta_q y^q$ have no common factors.

These models allow researchers to understand the influence of previous elements of the process, X_{t-i} on the current random variable X_t. This is clearly a linear model with a high level of interpretability. However, we suspect a LSTM neural networks might, generally, have higher accuracy in the prediction of X_t based on previous values of the process. This claim is tried empirically in this paper as will be clear later.

2.2. Autoregressive Integrated Moving Average

Another model useful for time series analysis is ARIMA also defined in Brockwell and Davis.

Definition 2. If d is a nonnegative integer, then X_t is an ARIMA(p, d, q) process if $Y_t := (1 - B)^d X_t$ is a causal ARMA(p, q) process.

2.3. Recurrent Neural Networks

Recurrent Neural Networks are used to model sequential data such as text and speech but variants can be created to perform time series analysis.

A RNN is typically represented as an unfolded computational graph as in figure 1 below:

RNN suffer from gradient vanishing problem which has been solved by Hochreiter et al among others with the

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

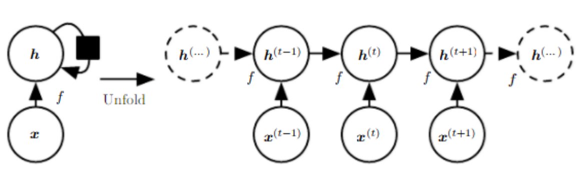


Figure 1. Unfolded Computational graph representing a RNN. From *Deep Learning* by Goodfellow et al.

introduction of LSTM recurrent networks which is the tool used in this paper.

2.3.1. Long Short-Term Memory

A Long Short-Term Memory architecture consists of a memory block as shown in figure 2 bellow. The memory block has three multiplicative units, known as input output and forget gates. (Hochreiter & Schmidhuber, 1997)

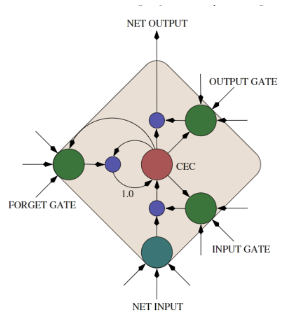


Figure 2. Memory cell for LSTM.

These gates perform the operations in the memory block are described as follow

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

where x_t is the input vector to the LSTM block; f_t, the forget gate's activation vector; i_t, the input gate's activation vector; o_t, the output gate's activation vector; h_t, the output vector of the LSTM block; c_t, the memory cell state vector. The activation functions are σ_g, a sigmoid function; σ_c and σ_h, hyperbolic tangent functions.

Additionally, $W \in R^{h \times d}$, $U \in R^{h \times h}$ and $b \in R^h$ are the weight matrices and bias vector parameters which need to be learned during training.

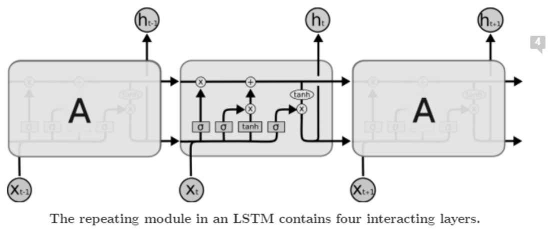

h_{t-1}
h_t
h_{t+1}
A
σ
tanh
A
X_{t-1}
X_t
X_{t+1}
The repeating module in an LSTM contains four interacting layers.

Figure 3. Memory blocks for LSTM.

The memory blocks are recurrently connected as figure 3 shows.

Taking inspiration from the success of encoder-decoder paradigm for translation (include reference for seq2seq), we consider an analogous method of capturing the distribution of a sequence of prices via an encoder and decoder. Batches of sequences of prices and volumes of length (t) where used for training. The encoder was fed each sequence and outputted a representation of the this sequence via a hidden layer which would then be sampled from via the decoder.

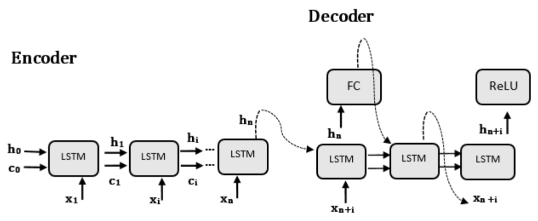


Figure 4. Architecture of our neural network. Encoder and Decoder both use LSTM layers.

GRU layers were tried in place of LSTM layers with no noticeable improvements.

3. Experimental Setup

3.1. Dataset construction

The data was obtained as a .csv file from http://api.bitcoincharts.com/v1/csv/. It was processed using pandas and split in sections according to the method used. The DataFrame and Series data structures from pandas were particularly useful.(mck)

3.2. Setup for ARMA/ARIMA

Data with different time spans was used, from year long data with resampling of one week to month long data with resampling of an hour or 30 minutes. Resampling was used to make the data less variable and to be able to read the autocorrelation and partial autocorrelation plots. Two different time series analysis models were tried with mixed results. To fit ARMA and ARIMA model data was detrended using logarithmic transformation and a differencing. Once transformations were applied, stationarity condition was also checked using the Dickey-Fuller Test.

To pick the correct lags p,q the autocorrelation and partial autocorrelation functions were plotted.

3.3. Topology of Neural Network

A LTSM single layer from tensorflow was used for naive LSTM implementation.tf1

In order to implement LSTMs auto encoder, we used pytorch (Paszke et al., 2017) python library which abstracts away the implementation of common computation layers (e.g LSTM layers). (Reference where we got the data). We note that the model was first tested on CPU (on a small sample of 10000 time stamps), and then on GPU once we confirmed that the model ran. The latter was done with the full (5 million data points)

Figure 4 shows the appropriate topology of the neural net. The net has an encoder section which is composed of one LSTM layer and the output of the n-th hidden layer to the decoder which is composed of a LSTM layer and the output of each hidden layer is passed through a fully connected layer and fed back to the LSTM layer as input.

The initial input into the decoder for the predicted sequence is the last known price ($x_{(t-1)}$) of the price sequence on which the encoder trained. Hence, once the decoder outputs its first sample (h_1) this will represent the continuation (the next predicted price): x_t

3.4. Hyperparemeters for LSTMs

Different hyperparameters were tried for the LSTM newtorks. The batch sizes of 100 and 200 datapoints were tried, learning rates of 0.5 and 0.1 were tried, size of hidden layers of 100, 200, 500 were used. We first tried low learning rates (10^{-3},10^{-2},10^{-1}) which caused very poor performance, hence we considered picked learning rates (0.1, 0.5). Note that the hidden layer size controls the dimensionality of the summary of the length (t) sequence fed into the encoder.

3.5. Training

Loss is RMSE and training is done through *adam* optimizer with a learning rate of 0.5 and 0.1.

4. Results

This section presents the project results.

4.1. ARMA/ARIMA

Detrending was also performed as can be seen in figure 6 below. Logarithmic transformation of the data was necessary and also differencing helped make the data stationary as ARIMA and ARMA models assume data is stationary.

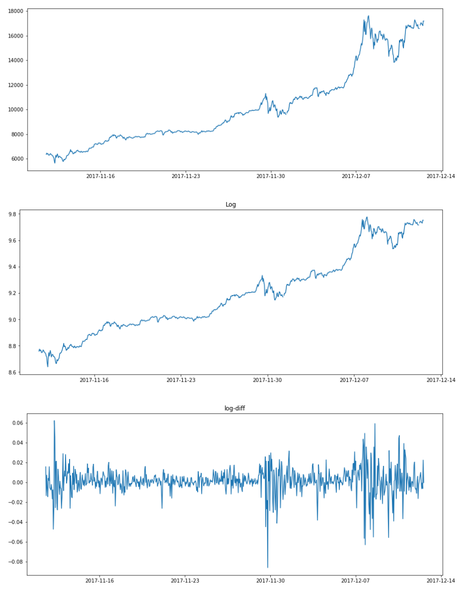


Figure 5. Detrending. Above plot is initial dataset, second plot is a log transformation, in this case trend is almost linear, third plot shows logarithmic and differencing transformations being applied.

StatsModels, a python library (Seabold & Perktold, 2010), was used in order to implement the ARMA model. Since the ARMA model is denoted as Eq. (2.1) above, hyper parameters are p and q which denote the number of previous prices to look at and the number of previous errors the model should consider, respectively. Preliminary results are considered which where obtained by taking p parameter to be in [1,5,10]. ARMA models were fitted for these values on previous 100 time-stamps and then were tasked with predicting the next immediate 4 time-stamps . Actual prices in the next 4 time-stamps with the predicted ones were compared and MSE (mean squared error) was calculated. We sampled the 100 time-stamps uniquely 60 times and found MSE for each sample. MSE for each p considered:

Note that, we expected that larger p values would lead to less error since the model has more data to work

lag, (p)	MSE
1	9.7514974036202844
5	9.4308973740076762
10	9.3925080749340033

Table 1. Change in mean squared error, MSE, due to change in Lag parameter.

with (pprevious values) in order to make a prediction.

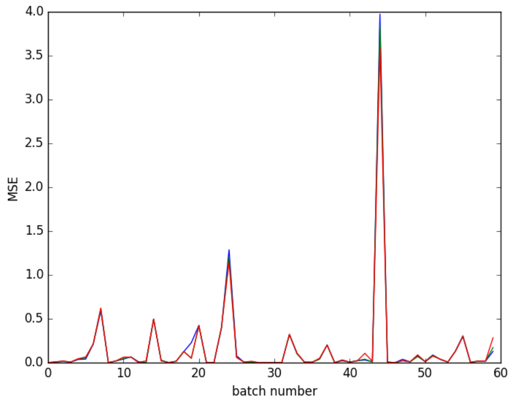


Figure 6. Change in MSE due to change in batch number.

The slight distinction of color in the top most peak (blue, green, red, in that order) represent the MSE for lags p=1, p=5, p=10.

After these preliminary results the autocorrelation and partial autocorrelation functions where calculated and plotted. An example of one such plot is presented in Figure 5 below.

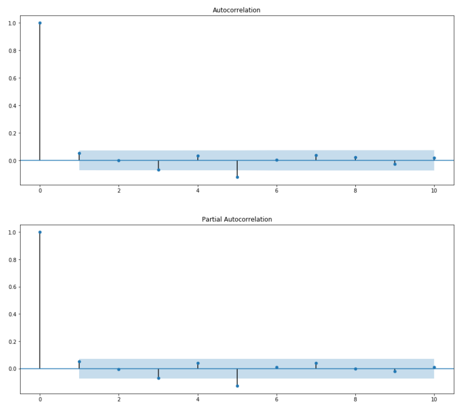


Figure 7. Autocorrelation Function with lags=10. Partial Autocorrelation Function with lags=10

The graph indicates that p=1,2 and q=1,2 should be the parameter values tried for ARMA and ARIMA modeling of the data.

Table 2 contains the residual sum of squares RSS after fitting ARMA models with the specified orders. It also cointains calculations for ARIMA root mean squared error of predictions to actual prices.

Table 2. Change in mean squared error, MSE, due to change in Lag parameter.

Order (p,d,q)	ARMA RSS	ARIMA RMSE
(2,1,1)	0.1272	5190.1172
(1,1,1)	0.1284	5206.0992
(1,1,2)	0.1284	5250.9906
(2,1,2)	0.1272	conv err

4.2. LSTM

We present some additional results. We start by trying to use dropout and given that we are using only one hidden layer with four inputs the accuracy gets reduced by a lot. Figure 8 shows the results in training and verification set. The predicted results on the verification set is presented in green and the training set predictions is shown in orange. The original data is shown in shown always in blue.
Additionally, dropout was added, however since only one hidden layer is used dropout reduces the overall accuracy of the predictions both in the training and testing set, as expected.

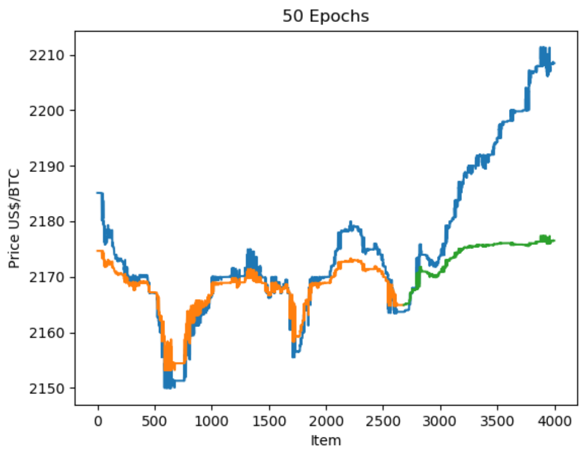


Figure 8. 50 epochs of learning.Loss is reduced from 0.3 to 2.4e-03

In figure **??** we present results for an LSTM layer with 4 units. The neural net was trained for 10 epochs and the results improved significantly to those of figure 1.

Table 3. Coefficients of ARMA and ARIMA processes estimated for our data.

Order (p,d,q)	ar.L1	ar.L2	ma.L1
(2,1)	-0.8832	0.0820	0.9420
(1,1)	0.0292	0.1272	
(1,1,1)	0.0523	0	-1
(2,1,1)	0.0730	0.0177	-0.9974
(1,1,2)	-0.9236	-0.1	-0.8979

When increasing the number of epochs to 20 the training errors decrease to Train Score: 0.95 RMSE and Test Score: 3.01 RMSE. Figure 9

When increasing the number of epochs to 20 the training errors decrease to Train Score: 0.94 RMSE Test Score: 2.55 RMSE. Figure 10 includes recurrent dropout, however the improvements thanks to recurrent dropout are not very noticeable, as evidenced below. When additional epochs are added as shown in Figure 9 the error in the validation set is reduced.

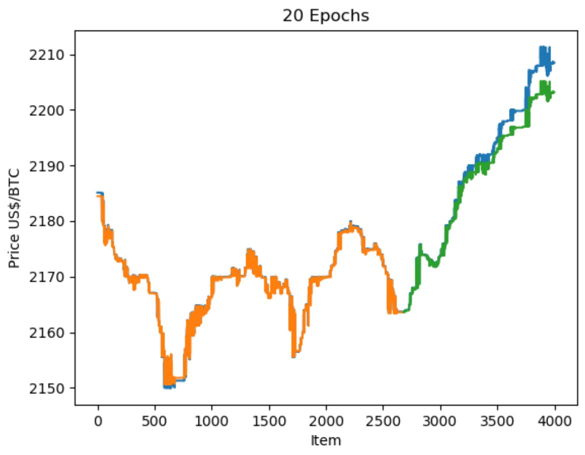
The last result has training error of 1.14 and validation error of 1.30. The prediction in the validation set is presented in figure 12. This net has a LSTM layer with 10 units and, at least graphically, high improvement in the validation set can be noticed.

4.3. AutoEncoder

The following are results from training the autoencoder architecture.Figure 10 shows how the losses decrease rapidly as epochs advance. The different parameters used in the network are shown in the legends.

We had a total of 5 million data points from 2017 year. The data was composed of time entries with appropriate timestamps, BTC price, and volume traded around this timestamp. We reserved 10,000 points for evaluation. Data points were composed of 200 timestamp entries (along with price and volume) and then the next 10 timestamps as target price values. For reference, the encoder received a sequence length of 200 timestamps and attempted to predict the next 10. Once the model was trained, it was evaluated on the evaluation data set by feeding the input sequence and comparing its output the to target.

The different hyper-parameters were compared on 10 epochs (this was chosen since we noticed the the error didnt fluctuate much afterwards). We note that even though the larger the batchsize led to faster results, we had lesser performance on the training loss (figure 10). We found that the RMSE of the encoder-decoder plateaued at around 40 for all settings tested. The best hyper-parameters with the greatest performance was hidden dim=100 and batch size=100. We found that having larger sized hidden layers leads to no noticeable improvement in error. This confirms our previous intuition that the main utility of the encoder is to do high dimensionality reduction. If the hidden size was either the same size of larger of what is being encoded (in this case the 200 timestamps of price and volumes), nothing that wasnt already in the original sequence can be extrapolated.



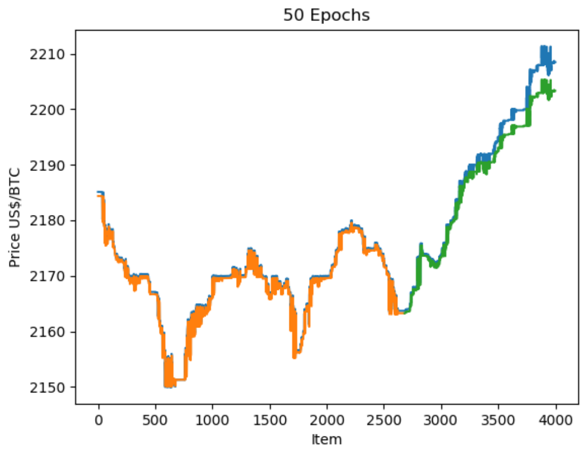


Figure 9.

The next graph shows a prediction vs actual prices during 20 different timestamps. We can see that the price difference is very little.

5. Conclusion

We have compared performance of traditional time series analysis models and more modern LSTMs and auto encoders neural networks. It has been evident that even changing experimental conditions Neural

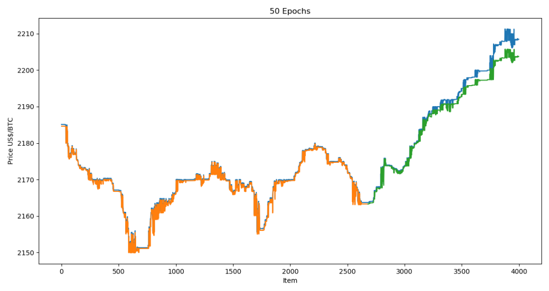


Figure 10. This frog was uploaded via the project menu.

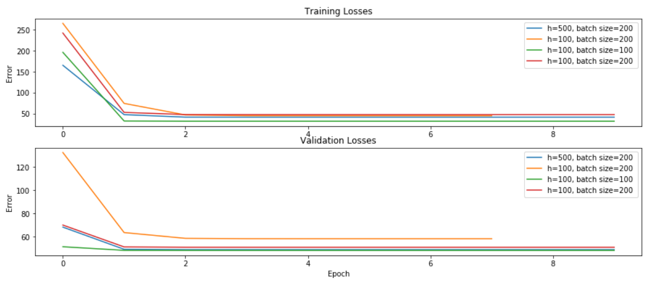


Figure 11. Training and Validation losses as functions of Epoch.

Networks perform better at prediction than traditional methods. ARMA and ARIMA are easier to interpret from the model's estimated coefficients. Additionally, LSTMs and Auto Encoders proved to require much more computational power.

6. Future Work

Here we note further modifications that were not yet tested, but we hope can lead to promising results. The encoder/decoder was trained in series (the output of the encoder fed directly to the decoder), but we would like to consider the impact of: Training them separately (In parallel). We would like to test the performance of the encoding by first choosing a training length to encode/small *hidden size* and then on the output of the encoder we apply a preliminary decoder that samples the same length of trained data points and compares it to what was supposed to be encoded. In other words, we are testing the encoders ability to successfully capture the sequence. At the same time we have the hidden output of the encoder trying to predict the next sequence of prices.By both at the same time, were able to choose an optimal encoding and size of hidden dim that might lead to higher quality predictions

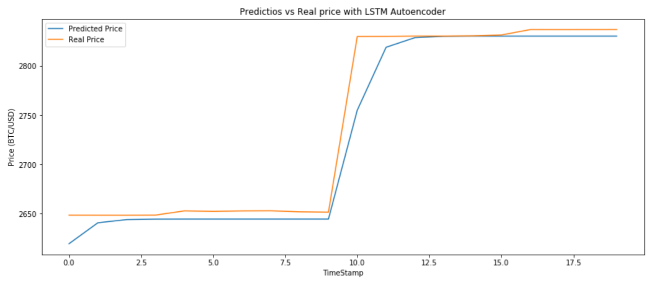


Figure 12. Real vs Predicted Bitcoin prices in US Dollars for 20 timestamps.

Acknowledgments

We would like to thank ardavan for reviewing our milestones and proving feedback and professor sontag for lectures on lstms and autoencoders and reference to goodfellow's book. We would also like to thank devs who have made statsmodels, pytorch, tensorflow, matplotlib, pandas easily available.

References

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. Automatic differentiation in pytorch. 2017.

Peter J. Brockwell, Richard A. Davis. Introduction to time series and forecasting. 2016.

Seabold, Skipper and Perktold, Josef. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.