Bitcoin Trading Strategy Based on Deep Neural Networks

December 12, 2022

1 Abstract

The object of this work is to tests different models for the prediction of the Bitcoin price and build a trading strategy that, at least, is able to overperform a buy-and-hold strategy. In the first section econometric methods such as an AR model and an ARIMA model are examined. Then Machine Learning methods are analysed, in particular various Neural Network architectures such as: LSTM, GRU and CNN. The time series examined is that of Bitcoin on January 2021, on the 1-minute time frame. In order to identify the best Bitcoin price prediction methodology that could be actually implemented on a real trading account, the strategy will be built taking into account a convenient compromise between result accuracy and an acceptable computational effort.

2 Introduction

Predicting the future price of various financial instruments has long attracted the interest of researchers, from the largest hedge funds, banks and proprietary trading companies to small investors and retail traders who aim to build an edge over other investors in order to profit by speculating in the financial markets. In this paper, several Bitcoin price prediction models are presented.

Firstly econometric models AR and ARIMA are examined. They have as s strength the computational simplicity in training the model but, at the same time, have as a weaknesses the poor predictability ability out-of-sample.

Then models from the Deep Learning field are implemented; the first is the Long Short Term Memory (LSTM). The critical point of this model is that it is based on a too complex structure with many parameters to be trained. Similar considerations apply to the Gated Recurrent Units (GRU) model. The last model taken into account is the Convolutional Neural Network (CNN). With CNN an optimal balance was found between model complexity, training time and goodness of predictions.

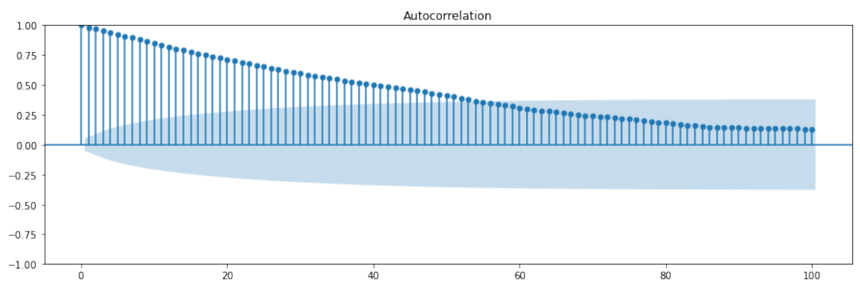
All models are trained on the same training set which is the time series of the Bitcoin closing price on the 1-minute time frame in the first seven days of observation (i.e. from 1st to 7th January 2021). Then the same models are tested on the testing set from 8th to 14th January 2021. To determine the most promising models two metrics are used: the first is the Mean Square Error (MSE) to determine the predictive capacity, and the second is the total number of parameters to be trained to evaluate the computational effort required. To conclude, the most promising model is tested with a specific strategy. If the forecasted price close in the next forecasted minute is greater than the past close, one buys 1 unit of Bitcoin, vice versa one sells.

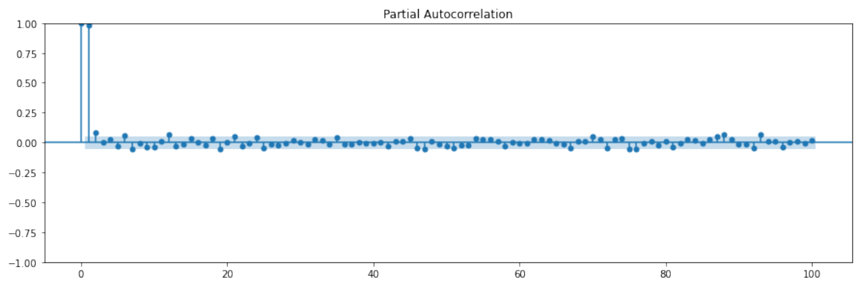
3 Econometric models: AR, ARIMA

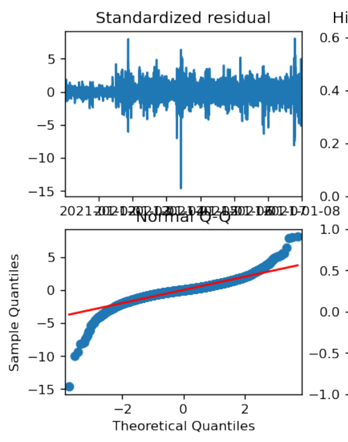
In this section econometric models such as AR model and ARIMA model are analyzed. The scheme followed consists of determining, during the training phase, the parameters of AR and ARIMA models and then, during the following testing phase, of analysing the results out-of-sample.

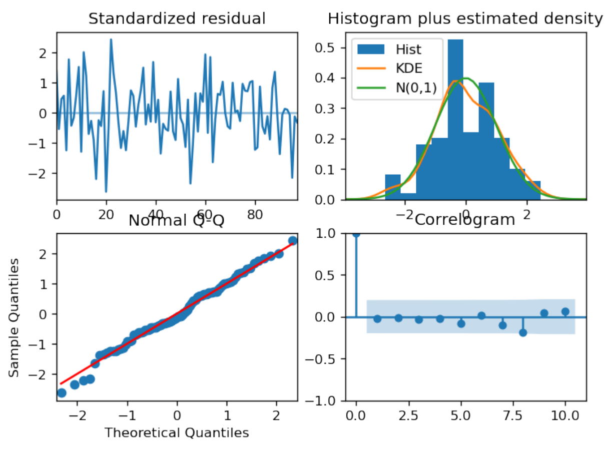
Starting with the training phase, an Augmented Dickey-Fuller (ADF) test was performed on the training set to verify the existence, or non-existence, of unit root. Recall that the Augmented Dickey-Fuller test tests the null hypothesis of the presence of unit root in a time series against the alternative hypothesis of the stationarity of the time series. The results of the analysis is that the null hypothesis was not rejected; therefore the time series has unit root. To eliminate the unit root, the time series was differentiated. Then a second ADF test was performed on the differenciated time series and, this time, the null hypothesis was rejected. After having obtained a stationary time series, the Autocorrelation Function (ACF) graph and the Partial Autocorrelation Function (PACF) graph were computed (image below). From the visual study, it can be assumed that the time series could be represented by an AR(1) model; note that the first bar of the PACF graph corresponds to lag 0 while the second bar corresponds to lag 1. Hence the parameters of the AR(1) model were computed together with the analysis of the residual (image below left). From the study of the residuals, unfortunately, it can be observed that the residuals are not distributed following a normal distribution. For this reason a second model, ARIMA, was tried. Using this model, it is assumed that the time series can be described not only by an Autoregressive (AR) component but also by a Moving Average (MA) component. The best parameters of ARIMA were then computed; these are p = 1, d = 1, q = 3. The ARIMA model was then trained with those parameters and the residuals were analysed again (image below right). This time, it can be observed that the residuals of the ARIMA model follow the normal distribution much more than those of the AR model.

Having completed the training phase now starts the testing phase. To do so the Mean Square Error (MSE) between the actual closure of the closing price and his forecast was computed. The ARIMA model obtained an MSE of 8716, which is roughly similar to the MSEs of the CNNs models that will be seen later, but unfortunately, the results are quite negative leading to a losing strategy over time.









4 Machine learning models: model selection step

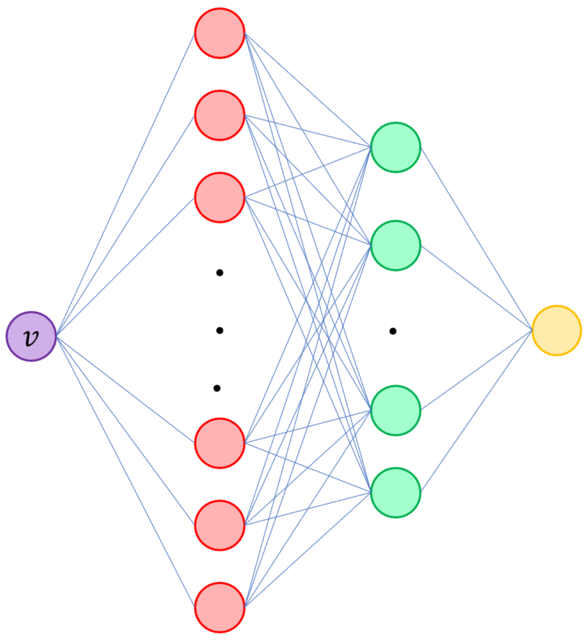
After having considered econometric models such as AR and ARIMA, in this section deep neural network models are analysed. The first step is to determine which neural architecture to implement. The architectures considered are: Long Short Term Memory (LSTM), Gated Recurrent Units (GRU) and Convolutional Neural Network (CNN). In order to make a fair comparison, the architectures considered are trained maintaining consistency, i.e. using the same number of layers and neurons per layer for all models. The tests performed on the various architectures are also trained twice; the first time using only one variable as input (past lags of the closing price), and the second time with several inputs. In each case the output will always be the prediction of the closing price in the next minute after the last minute taken as input. It should also be specified that since we are working on time series, to predict the price close in the next minute, the entire past of the time series is not taken into account, but only a subset of subsequent past lags. This window of past lags, which in this paper will be called window size, will be an additional parameter to be determined. In this first stage of model selection, a window size of 5 lags is used.

In the univariate case, the only variable taken as input is the past close of the price. So if our target is x_t (price closure at time t), as input we have a vector of size (5,1) having the 5 past closures $v = \begin{pmatrix} x_{t-1} & \dots & x_{t-5} \end{pmatrix}'$.

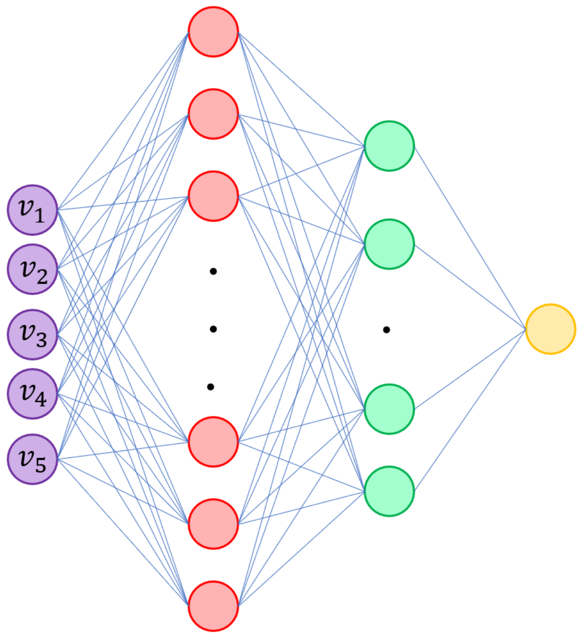
In the multivariate case, the variables taken into account in addition to the past close of the price are: trading volume of the underlying, price volatility, relative strength index (RSI) and moving average of convergence/divergence (MACD) (the latter two being two technical indicators that "should" help predict price trends). In this case, the inputs will be 5 vectors of size (5,1) of the type: $v_i = \begin{pmatrix} x_{t-1,i} & \dots & x_{t-5,i} \end{pmatrix}'$, for i = 1,...,5.

The parameters used to test the different architectures are:

- a window size of 5 lag passed;
- an input layer of size (5,1) in the univariate case or an input layer size (5,5) in the multivariate case (the purple neurons in the image below);
- a subsequent characteristic layer (LSTM, or GRU or CNN) with 1024 neurons (the red neurons in the image below);
- followed by a further dense layer with 128 neurons which uses the ReLU function as activation function (the green neurons in the image below);
- then an output layer with only one neuron from which our output will come out (the yellow neuron in the image below).
- further parameters employed for the training are a learning rate of 0.0001, the Adam optimiser and 30 training epochs.

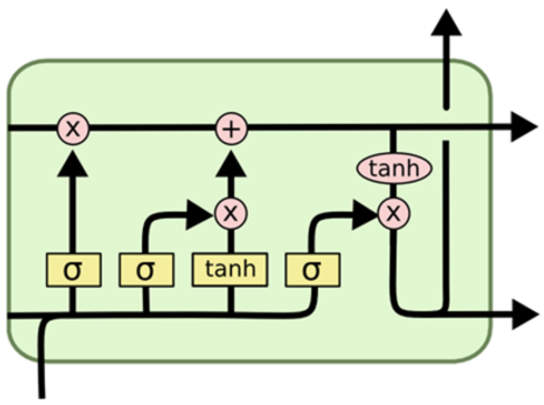


Univariate model

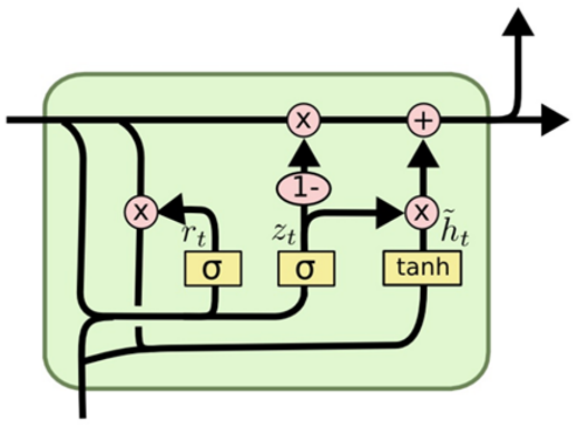


Multivariate model

The architectures which use as characteristic layer the LSTM and GRU are two subcategories of recurrent neural networks (RNNs). The RNN architecture is a class of artificial neural networks comprising neurons connected to each other in a loop. The interconnection between the layers allows one of them to be used as a state memory; furthermore by providing a temporal sequence of values as input, it is possible to model a temporal dynamic behaviour dependent on information received at previous time instants. Unfortunately, one of the problems of RNNs is the explosion and vanishing of gradients during back-propagation of time series data. For this reason, not a 'simple' recurrent neural network was used, but two more complex architectures that help to deal with these problems: LSTM and GRU. At this point, before showing the results, it is good to do a little background on how these architectures work, which will also be useful later when we will talk about computational complexity. The process of LSTM is slightly complex, as can be seen from the image below, because each time it takes input from three different states (1.the current input state, 2.the short-term memory of the previous cell, 3.the long-term memory). The LSTM uses gates to adjust the information to be kept or discarded during the loop operation before passing the long-term and short-term information to the next cell. We can imagine these gates as filters that remove unwanted information and remember relevant ones. The gates used by LSTM are three in total: Input Gate, Forget Gate and Output Gate. The input gate decides which information will be stored in long-term memory. The forget gate decides which information in the long-term memory will be retained or discarded. The output gate takes the current input, the previous short-term memory and the newly calculated long-term memory to produce a new short-term memory that will be transmitted to the cell in the next time step. The GRU process is less complex than that of LSTM. GRU incorporates two gate operation mechanisms called Update gate and Reset gate. The update gate is responsible for determining how much previous information is to be passed to the next state. The reset gate is used by the model to decide how much past information is to be dropped.



Long Short Term Memory Architecture



Gated Recurrent Unit Architecture

The results obtained by these two architectures are rather poor both from the point of view of predictive capacity (analysed using the mean square error on the test set) and from the point of view of the computational effort required to achieve the result (analysed taking into account the number of parameters to be trained, and thus more parameters implies more training time required). The results obtained are reported in the following table:

Model	MSE	Parameters
LSTM Univariate	19359	4.3 Million
GRU Univariate	15744	3.3 Million
LSTM Multivariate	25318	4.3 Million
GRU Multivariate	21004	3.3 Million

Recurrent neural networks are generally considered the best neural network architectures for time series prediction (and for modelling sequences in general), but recent studies show that convolutional networks may have comparable or even better performances. The convolutions can be implemented more efficiently than RNN-based solutions and do not suffer from vanishing or exploding gradients. The power of the Convolutional Neural Networks comes from the operators it uses. The first is the convolution operator, which is a way of extracting features from a signal (in our case a time series). Convolution consists of taking a kernel and making a sliding dot product of the signal. The second operation that characterises CNNs is the pooling operator, a kind of non-linear activation that smartly reduces the size of the signal. Therefore, because of all these reasons, the third model analysed is a Convolutional Neural Network (CNN). This time, the results were much more promising as shown in the following table:

Model	MSE	Parameters
CNN Univariate	9387	520000
CNN Multivariate	12300	530000

The conclusion of the model selection step is that an univariate model always overperforms its multivariate model and the CNN model is the preferable one by having a lower MSE and simultaneously having fewer parameters to be trained. Now the goal will be to find the optimal parameters for a CNN univariate model.

5 Tuning parameters for the CNN

Once identified that CNN is the most promising architecture and having defined that the best model is the univariate one, the goal is now to identify which parameters are optimal to define the final model. In particular, the parameters to look for are:

- the window size;
- the kernel size for the CNN neurons;
- the number of CNN neurons;
- the number of dense neurons.

To optimise these parameters, several models has been trained. This was the most computationally intensive step, as the models had to be retrained every time a parameter changed. The first parameter tested was the window size. The values taken into account were: window size = 2,3,5,10 and 15. The results obtained are as shown in the table below:

Window Size	**MSE**	**Economical Return**
2	8992	-27234
3	9654	378
5	9387	9860
10	9699	4880
15	9971	6630

The Mean Squared Error of the various model performed remains roughly the same on each trial. As a consequence of that, the choice of the optimal window size is made by taking into account the economical return. To do so, the trading strategy was run on each predicted time series (our output) for each window size, to determine which model returned the highest economical return; then that window size is selected. Using a window size = 2, a large loss was obtained while using all other window sizes the economical return resulted always positive. The model with the highest profitability was the one using a window size = 5. It is important to underline that if, for example, there was a model with higher economical returns than that obtained with the window size = 5 but with a bigger window size (for example 30) it has to be considered the additional computational effort requested by the bigger window size. Remember that a larger windows size corresponds to more operations to be performed and therefore more computational effort.

Once the optimal window size has been identified, the optimal kernel for the CNN neurons must now be found. Recall that the kernel in 1-dimensional convolutional neural networks is nothing more than a filter used to extract features from the time series. The values considered are: kernel size = 2,3,4 and 5. The results obtained are shown in the table below:

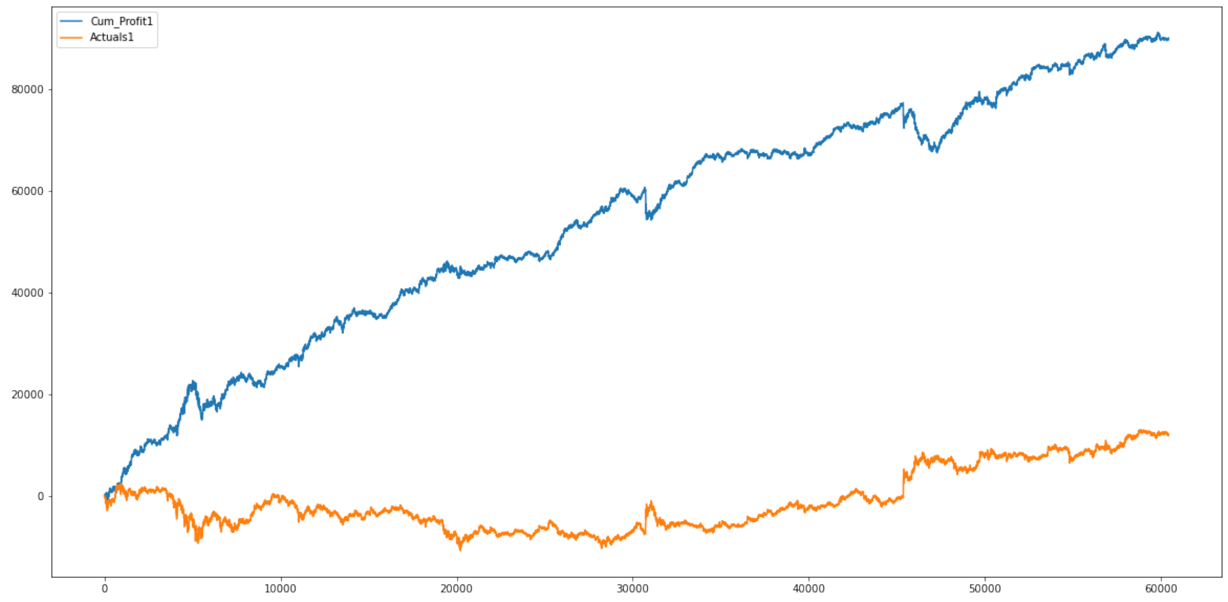
Kernel Size	**MSE**	**Economical Return**
2	9025	22371
3	9202	6680
4	9900	2438
5	10629	-18822

As before, both MSE and economic returns were compared. The results obtained are clear and sharp. As the kernel increases, both the MSE and economical returns get worse. This is why the kernel size, which will be used from now on, is the value 2.

Finally is time to find the optimal number of parameters for CNN and dense neurons. To this purpose, the function 'Bayesian Optimisation Tuner' of the keras library (a python library) was launched; this allows to find the best number of neurons by defining a range in which we want to look for the optimum. In the case analysed, a range of [0,2048] neurons was used for CNN, and a range of [0,1024] neurons was used for dense. The results obtained were that the best number of CNN neurons is 1024, and 384 of dense.

6 Backtesting of the trading strategy

Now that also all the optimal parameters for the architecture have been identified, it is time to test the trading strategy to see whether or not it is profitable in the long run. So it is not enough to test the strategy over a single week but it is necessary to iterate it over several weeks. To reach this goal, the architecture with the optimal parameters previously identified was trained and tested six times over six consecutive weeks. To clarify well what is done in this backtesting phase, six pairs of weeks(H_t) are taken into account $H_t = \left(\ week_t \quad , \quad week_{t+1} \ \right)$, for t = 1,...,6. In each H_t, the optimal architecture with the parameters previously found is trained on $week_t$ and the trading strategy is tested on $week_{t+1}$. The results obtained were very convincing, as the objective of this work was to try to find a trading strategy that would, at least, outperform a buy-and-hold strategy. In particular, with a buy-and-hold strategy over the period considered, a gain of about 12000 dollars would have been made, while buying and selling with the described strategy would have made a gain of about 90000 dollars. It is important to remind that these are only paper results, no transaction costs were taken into account. Over the entire six-week period, the graph below compares on one hand the cumulative profits earned by the trading strategy (blue line) and on the other hand the profits of a buy-and-hold strategy (orange line).



7 Conclusion

In this work was constructed a trading strategy that was able to overperform a buy-and-hold strategy. Several methodologies were tested; at first two econometric models (an AR model and an ARIMA model) both resulting with negative results. Then, different Neural Network were built to try to identify a better model to predict the Bitcoin price. The architecture considered were LSTM, GRU, and CNN. The first two turned out to be less appropriate than CNN by having higher MSE and more computational effort needed to train the models. The best parameters for CNN were then searched. Finally the strategy has been backtested over a period of six weeks. The final result is that the strategy, from a theoretical point of view, outperforms the buy-and-hold strategy by almost 8 times.

8 References

- https://keras.io/api
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Book: Deep Learning - Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron