## DEEP LEARNING

## MASTER IN ARTIFICIAL INTELLIGENCE

## Universitat Politècnica de Catalunya

---

# PRACTICAL WORK 2:

# RECURRENT NEURAL NETWORKS APPLIED TO BITCOIN PRICE PREDICTION

---

Albert Espín

March 27th, 2019

## Table Of Contents

## 1. Introduction

Recurrent Neural Networks (RNNs) are a Neural Networks that can be applied as prediction models in sequential data. In particular, time series are sequences of data instances chronologically ordered. One of the fields in which time series analysis is commonly applied is the prediction of the evolution of the price of articles, e.g. in stock market analysis. Bitcoin is the first and most well-known cryptocurrency. This works aims to build a RNN-based model to predict Bitcoin price on a minute-by-minute basis, and analyze its results compared to a simpler, non-RNN model.

The developed code [1] was built taking as a base Dr. Javier Béjar's Air Quality RNN prediction model [2].

## 2. The data set

### 2.1. Data representation

The used data set [3] is a history of Bitcoin prices per minute from January 12th, 2014 (6:33h) to October 24th, 2017 (23:32). That is to say, there are 1293167 data samples. Each of them has the associated timestamp and Bitcoin price information. For this work, several fields are ignored, starting with the timestamp, since the interval is constant and it is enough knowing the order of the data, being redundant. The lowest and highest value of the minute are not considered for simplicity; there are close in value and somewhat redundant to the weighted price, which is the prediction target. The opening and closing values are also ignored for the same reason as the previous fields. That is, the only value to consider is the weighted price, which can be conceptually seen as the average price of the Bitcoin (in United States dollars, USD), for each minute.

### 2.2. Preprocessing

The data has been normalized using a minimum-maximum scaler, that is, translating the whole set of price values to the range 0-1, by assigning 0 to the lowest original value, 1 to the highest, and a linear equivalent to the rest of values, lying in between the extremes. Once the scaler has been fitted to the data, it can be used after producing the model to invert the transformation on the predicted values, to recover the original ranges of values for the predictions. The reason to normalize the data is to help the RNN and specifically back-propagation and gradient descent learn faster by reducing the magnitude of the value search space.

## 2.3. Data splits

As previously mentioned, the data is composed of 1293167 instances. The decision of how to split the data was taken trying to both have a large percentage of the data to learn (the more data, the more a model can opt to learn) and to keep a reasonably long and heterogeneous sample as the test one. Therefore, the model was trained with the first 1200000 instances (92.2% of the data), while the remaining 93167 (7.8%) were used as a test. They correspond to a period of time, during the second half of 2017, with large, fast changes in the value of the Bitcoin, which make the prediction task really challenging.

## 3. Experimental methodology

When it comes to develop a time series prediction model, a very informative first step is to develop a baseline system whose predictions are independent of any type of architecture that a more advanced model may have. The baseline system is explained in section 4.1. Along with the baseline model, a more complex, RNN-based model was built, as explained in section 4.2, where different configurations of the model parameters are tested to improve the accuracy.

The complex model was built by testing different options or values regarding the following elements or parameters:

- Number of RNN layers
- Number of recurrent neurons per layer
- Recurrent architecture (LSTM or GRU)
- Presence or absence of secondary fully-connected layers
- Dropout percentage for generalization
- Training epoch number
- Training batch size per epoch
- Optimizer method
- Learning rate

To be able to compare both models, and different configurations of parameters for the complex one, and to assess whether the complex model gives predictive advantages compared to the baseline solution, different metrics were selected, after doing research on appropriate performance metrics for time series models [5]:

- Mean Square Error (MSE) between the real and predicted values. It is calculated for the normalized values, before restoring them to the original ranges, so its value should lie between 0 and 1, where lower is better (less error).

- Coefficient of determination ($R^2$), calculated using the MSE and measuring the proportion of data variance explained by the model, with a value that should be between 0 and 1, where higher is better (greater part of the variance explained).
- Mean forecast error (Forecast bias), which is the mean of the subtracting the predicted value from the real one, for each time step, without absolute value. It is computed once the data values have been transformed to the original range, so it is expressed in USD. If it is negative, it means that the model is biased by predicting prices higher than the real ones, lower if it is positive.
- Mean absolute error (MAE), calculated as the mean of the absolute value of the differences between real and predicted values, also expressed in USD. Since it is an error, the lower it is, the better.
- Mean error percentage (ME%), calculated as the mean of the real-predicted differences in value, normalized (divided) by the real price. The lower it is, the better.

## 4. Developed models: architecture and results

### 4.1. Baseline model

The baseline model is extremely simple: for each step, the value predicted for the next time step is the value of the current moment. Formally, considering t as a time step index and T the total number of steps:

$$\forall\ t \in [1, T\text{-}1]: \text{price}_{t+1} \leftarrow \text{price}_t$$

The result of this model may not be a prediction in the rigorous definition of thw word, since it just expects each immediate future step to be equal to the present one. Nevertheless, computing the performance metrics can allow to have a base point of comparison with more complex models.

The baseline results can be found in Table 1. It is noteworthy to say that, despite of the very low MSE and really high $R^2$, this does not not guarantee that the model can produce predictions that will be considered valuable, since they are just assigning the present to the immediate future. From one minute to the other, the price is changing in 11.16$, on average, which is what this model is able to predict. Ideally, one would like to have another (more complex model) that would predict the next minute's price with less difference with the real value.

Table 1: Results of the baseline model

| **MSE** | **$R^2$** | **Forecast bias** | **MAE** | **ME%** |
|---|---|---|---|---|
| 1.55633E-6 | 0.99996 | 0.10$ | 11.16$ | 0.09 |

## 4.2. Recurrent neural network model

### 4.2.1. Best model configuration

Many different combinations of parameters have been tested to obtain the best configuration. In the end, the best results have been obtaining using a single RNN layer of 64 neurons, based on the Gated Recurrent Unit (GRU) architecture, with tangent activation function for neuron activation and using the hard sigmoid for the recurrent activation. No dropout is used, and a single-unit fully-connected unit is connected to the RNN to synthesize the output. The architecture is represented in Figure 1. The learning process has been performed with a window of 6 units and predicting the 1st immediate future value, training for 25 epochs, each of them performing 500 batches with Adam optimizer and a learning rate of 0.001. Section 4.2.2 compares each of these parameter choices with alternative ones that has been tested, and formulates hypothesis to try to understand why the chosen options listed here are better than the other, as for the experiments.

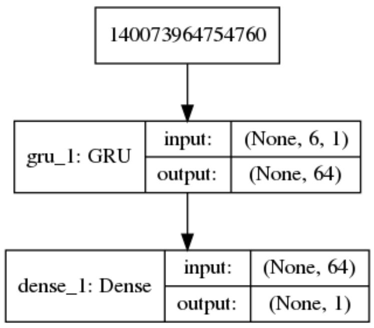

Figure 1: Model architecture.

The results of the RNN-based model are shown in Table 2. It can be observed that despite the very low MSE and high R² results obtained, that may lead to think at first that the model is very accurate, the RNN-based model is unable to outperform the baseline model in any of the computed metrics. Nevertheless, this configuration is reasonably close to the baseline results, compared with other RNN setups described in the next sections.

Table 2: Results of the best configuration for the RNN-based model.

| **MSE** | **R²** | **Forecast bias** | **MAE** | **ME%** |
|---|---|---|---|---|
| 2.60981E-6 | 0.99993 | 11.84$ | 18.45$ | 0.14 |

According to the literature [7], currency price values are time series of chaotic nature, which make them very challenging (if at all possible) to predict in the strict sense of the word, i.e. going further than naive attempts like the baseline model presented in section 4.1.

In an attempt to compare the developed model with other existing ones, research was done to find other Bitcoin price predictive models. One of them [4] uses the ARIMA (Auto-Regressive Integrated Moving Average) model, and achieves an error of 3.59%, which is clearly higher than the developed model, but it should be compared in a skeptical way since the used data is different, consisting in daily predictions instead of minute-by-minute ones. A different model for daily predictions [6] uses RNN, as this work, but do not present numeric results, only plots, that seem less accurate than the developed model.

Figure 2 presents the comparison of the real and predicted price for all the minutes of the test set. It is difficult to analyze such a long sequence (almost 100000 values) in an image, but it can be seen that, at least from the distance, it seems that the predictions are very close to the real ones (zooming is required to make a distinction). With the closer plots of Figure 3 one can see, that the distance is greater than it seemed. Nevertheless, the model succeeds to follow the approximate, rough shape with which the real series evolves in time.

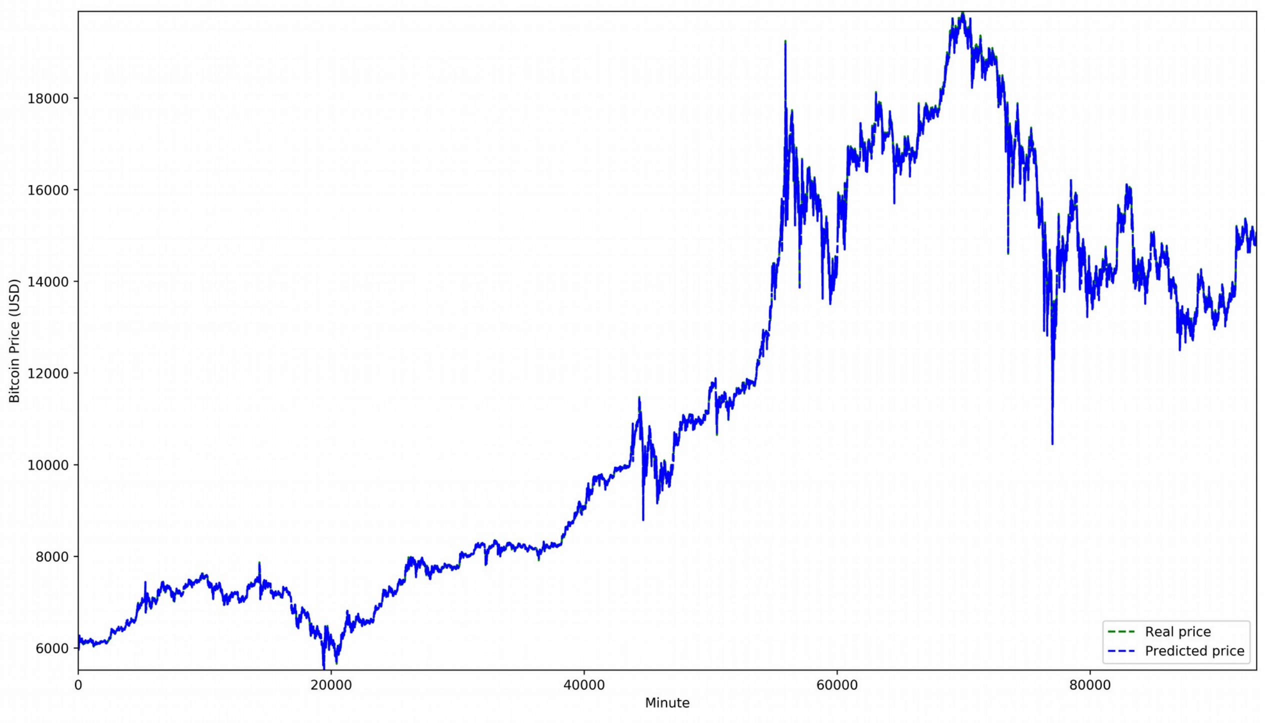

Figure 2: Comparison of the real and predicted Bitcoin USD price minute-by-minute, for the whole test set.

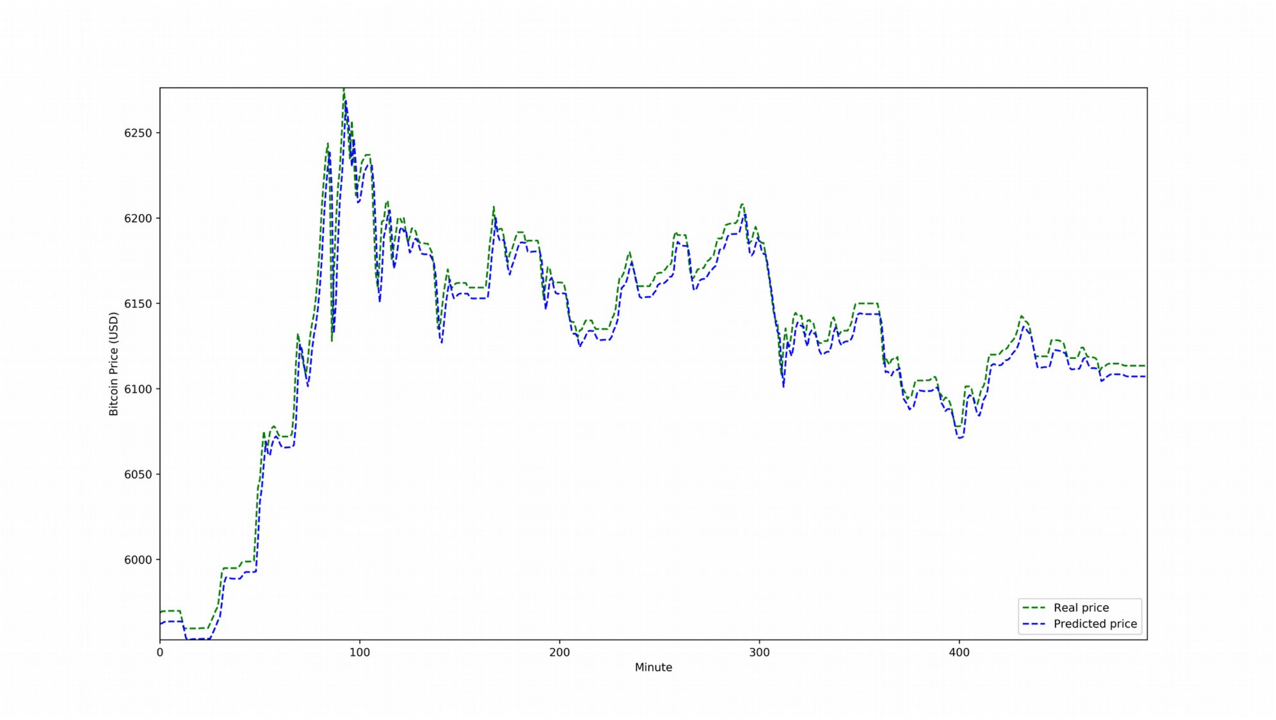

Figure 3: Comparison of the real and predicted Bitcoin USD price minute-by-minute, for a small section of the test set.

### 4.2.2. Alternative configurations compared to the chosen ones

Table 3 shows the alternative configurations that have been tested, that proved to have an overall performance considered worse than the model explained in section 4.1. Each of them only differ in one aspect from the model of 4.1, that is mentioned in the first column of each row.

It can be seen that adding a second RNN layer has made the model less accurate, learning noisy information of the data that does not generalize well and therefore has increased both the execution time (due to the larger total number of neurons) and the mean error percentage up to 0.46%, compared to the 0.14% of the model explain in section 4.1. Reducing the neurons of the single RNN layer down to 32 or even 1 has not improved the model neither, but lead to heavier underfitting that could not learn well the temporal relationships, with 0.17% and 3.59% of error, respectively. Doubling the number of neurons from 64 to 128 did not achieve any improvement, with the time series being too complex to be easier to study by just simply increasing the neuron number. Furthermore, it made the training process slower, since more units require more back-propagated gradients to be computed. Using Long Short Term Memory (LSTM) units instead of GRUs for the RNN architecture did not improve the results neither, with the error percentage reaching 0.25%. As of now it is not clear in the empirical tests of RNN literature which type of unit is better, and the justification of why GRU works better in this scenario is not clear, but, as expected, LSTM is slower due to its larger number of gates (GRUs do not have a forget gate, but integrate it in the update one).

In complex neural network models that learn too specific patterns of the training data, it is common to add dropout layers to disable some neurons in some learning steps with a certain probability. This strategy can improve the learning redundancy and lead to generalization, decreasing overfitting. The fact is that, for this time series, the model is not enough yet overfitting, i.e. it may still be able to learn relevant information from the data. For this reason, adding dropout leads to heavier underfitting and greater error: 0.30% for 0.25 dropout and 0.47% for 0.5 dropout. Therefore, it is better not to apply dropout unless the model is first modified in other ways towards overfitting.

Increasing the number of epochs allows to learn during more time, which eventually allows to achieve a forecast bias below the best model, and even below the baseline (10.72$ with 30 epochs, 8.79$ with 50 epochs), but other metrics are worse, such as the absolute mean error and the error percentage. These higher number of epochs, combined with other changes, may eventually lead to better results. On the other hand, reducing the epoch number removes part of the learning: the error percentage raises to 0.20% with 20 epochs, instead of the default 25. Modifying the batch size, however, does not improve the model: the accuracy is lower both increasing and decreasing the size, since the different arrangement of data taken in each learning step does not become determinant for this complex time series.

Different optimizers were tested, but none of them learned as fast an well as the base Adam. Despite the fact that RMSprop is usually recommended for RNNs, it greatly increased the error, up to 1.59%, due to the complexity of the data it was unable to learn so well. The Nesterov momentum of Nadam did not improve the results neither, that reached 0.39% of error: the descent may be too complex to accommodate this momentum. Different learning rates (0.01 and 0.0001 instead of 0.001) did not produce very different results

(0.14% and 0.16%, respectively), meaning that the key problem for this data is not how fast the learning takes place, but if it actually takes place in an effective way.

Table 3: Results of alternative configurations of the RNN-based model

| **Configuration change** | **MSE** | **R²** | **Forecast bias** | **MAE** | **ME%** |
|---|---|---|---|---|---|
| 2 RNN layers | 3.00632E-5 | 0.99930 | 60.07$ | 69.10$ | 0.46 |
| 1 RNN neuron | 0.00180 | 0.95799 | 549.53$ | 549.73$ | 3.59 |
| 32 RNN neurons | 3.755927E-6 | 0.99991 | 17.42$ | 23.22$ | 0.17 |
| 128 RNN neurons | 2.70684E-6 | 0.99994 | 16.9$ | 21.39$ | 0.17 |
| LSTM architecture | 6.73593E-6 | 0.99984 | 18.5$ | 32.83$ | 0.25 |
| 500-neuron dense layer | 1.66677E-5 | 0.99961 | 55.89$ | 57.66$ | 0.42 |
| Dropout 0.25 | 5.76688E-6 | 0.99986 | -35.52$ | 37.16$ | 0.30 |
| Dropout 0.5 | 2.58831E-5 | 0.99940 | -62.97$ | 68.50$ | 0.47 |
| 20 epochs | 4.27518E-6 | 0.99990 | 22.01$ | 26.45$ | 0.20 |
| 30 epochs | 2.68944E-6 | 0.99994 | 10.72$ | 18.91$ | 0.14 |
| 50 epochs | 2.68130E-6 | 0.99994 | 8.79$ | 19.68$ | 0.15 |
| Batch size 250 | 3.82469E-6 | 0.99991 | 20.41$ | 24.59$ | 0.18 |
| Batch size 1000 | 4.35678E-6 | 0.99988 | 25.35$ | 29.71$ | 0.22 |
| Nadam optimizer | 2.11560E-5 | 0.99950 | 54.85$ | 58.52$ | 0.39 |
| RMSprop optimizer | 0.00037 | 0.99146 | 229.04$ | 243.55$ | 1.59 |
| 0.0001 learning rate | 2.64888E-6 | 0.99994 | 13.12$ | 19.08$ | 0.14 |
| 0.01 learning rate | 2.88746E-6 | 0.99993 | 16.22$ | 21.21$ | 0.16 |

## 5. Conclusions and future work

Due to the chaotic and apparently unpredictable nature of the problem, the developed RNN-based model to predict minute-by-minute Bitcoin prices has not been able to yield results that can be considered as a predictive advantage with respect to the baseline model of considering that the next value will be equal to the current price. Further experiments with even more RNN configurations and other types of architectures, such as RNN combined with CNN, may be tested to attempt to obtain better results. Completely different models, such as neuro-fuzzy ANFIS predictive system have been also used in the time series field in the past, so they might be tested as well. It would be also interesting to use more variables to estimate the price, such as the volume of trading operations done in a minute and some statistics about them, such as the price balance (positive or negative), the total quantity of bitcoins exchanged and the number of trading events.

## 6. References

Developed code:
[1] Github repository

Code used as a base:
[2] Dr. Javier Béjar's Air Quality prediction with RNN

Data set:
[3] Minihat's Bitcoin price history data set

Other sources of information:

[4] Asthana, Ayushi (Towards Data Science), 2018. Bitcoin Price Prediction Using Time Series Forecasting

[5] Brownlee, Jason (Machine Learning Mastery), 2017. Time Series Forecasting Performance Measures With Python

[6] Gazi Yalçın, Orhan (Towards Data Science), 2018. Predict Tomorrow's Bitcoin (BTC) Price with Recurrent Neural Networks

[7] Jiang, C., & Song, F. (2010, March). Forecasting chaotic time series of exchange rate based on nonlinear autoregressive model. In *2010 2nd International Conference on Advanced Computer Control* (Vol. 5, pp. 238-241). IEEE.