Advanced Machine Learning Project - Financial Time Series Forecasting With Neural Networks

LIFFRAN Anna
anna.liffran@ensae.fr

MASSIN Keryann
keryann.massin@ensae.fr

BOURU–GAZEAU Ambroise
ambroise.bouru-gazeau@ensae.fr

Abstract

This project delves into the realm of financial time series forecasting, aiming to compare traditional methods such as ARIMA and GARCH models with more recent innovations, particularly focusing on Recurrent Neural Networks (RNNs). The objective is to explore the state-of-the-art in predicting the future evolution of financial instruments, with a specific focus on Bitcoin's price. The project is structured to introduce the data and mathematical framework related to time series, followed by an exploration of traditional methods and the implementation of Deep Learning methods such as LSTM and GRU neural networks. The ultimate goal is to gain insights into the effectiveness of neural networks in financial forecasting.

1 Introduction

One of the hot topics in finance involves accurately predicting the future evolution of a given financial instrument's price—be it in the near or long term. The better the estimates, the better the investments. In this context, various techniques are mentioned in the literature for forecasting financial time series.

Historically, ARIMA and GARCH models have been widely used and remain popular due to their robustness and strong statistical foundations. However, the emergence of Deep Learning in the past decade has introduced a range of innovative models, among which Recurrent Neural Networks (RNNs [3]) stand out. These models offer numerous advantages, including their ability to capture non-linear relationships within time series and their flexibility in handling diverse input structures.

This project aims to explore the state-of-the-art in financial time series forecasting with the ambition of comparing both traditional and more recent models. The underlying objective is to delve deeply into the application of neural networks in the realm of business.

For the aforementioned purposes, we decided to focus our work on the time series of Bitcoin's price, considering how famous of an instrument it is and for its great mathematical properties.

The current projet is divided in a few sections: we first introduce the data and the mathematical framework related to time series (stationarity and mixing assumptions); we then explore some of the traditional ways of handling financial time series (ARIMA models, GARCH models, etc.) before implementing Deep Learning methods (LTSM [2], GRU [1] neural networks).

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Data

Using the Yahoo-finance API, we were able to retrieve 30 days worth of BTC-USD (bitcoin, in $) historical prices at 1-min intervals. This time series, which we will denote (X_t), is univariate: on such a tiny interval, Yahoo-finance does not provide any information besides the average price X_t at minute t (namely, the 'Open', 'Close', 'High' and 'Low' prices for a specific t are equal).
Picture 1 illustrates the raw data we retrieved.

Figure 1: Evolution of Bitcoin price over the period 02/11/2023 - 01/12/2023

3 Affine causal models

The traditional framework often used to carry out a Time Series project consists in assuming the series $(X_t)_{t\in\mathbb{Z}}$ of interest can be seen as an affine causal process, following the dynamic:

$$X_t = M_\theta(X_{t-1}, X_{t-2}, \dots) \cdot \xi_t + F_\theta(X_{t-1}, X_{t-2}, \dots) \tag{1}$$

where $(\xi_t)_{t\in\mathbb{Z}}$ is a white noise (i.e $\mathbb{E}[\xi_t] = 0$, $\mathbb{E}[\xi_t^2] = \sigma^2 > 0$ and $\mathbb{E}[\xi_t\xi_\tau] = 0$ for $\tau \neq t$) with $\sigma^2 = 1$, and $F_\theta, M_\theta : \mathbb{R}^\infty \to \mathbb{R}$ are two functions. The whole goal of the project is then to determine the vector of parameters θ^* that allows the best forecasts for X_t, knowing its past $\mathcal{F}_{t-1}$.

Inside the class of affine causal models, Auto Regressive Integrated Moving Average (ARIMA) or Generalized Auto Regressive Conditional Heteroscedasticity (GARCH) models are well-known and considered as proven solutions in the study of Time Series. In light of this, we wanted to give them a try.

3.1 Stationarity and mixing properties

The standard hypothesis over a Time Series one can make are stationarity and mixing property.

A stochastic process (X_t) is said to be (second order) stationary whenever $t \mapsto \mathbb{E}[X_t]$ and $t, h \mapsto \mathbb{C}\text{ov}(X_t, X_{t+h})$ are constant with respect to t (and $\mathbb{E}[X_t^2] < +\infty$ incidentally). This property can be checked through two widely used statistical tests: the Augmented Dickey-Fuller (ADF) test,

and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test.

On the other hand, (X_t) is mixing if its values at widely separated times are asymptotically independent, which translates to saying that:

$$\mathbb{P}(X_{t+h} \in A, X_t \in B) \underset{h \to +\infty}{\to} \mathbb{P}(X_{t+h} \in A)\mathbb{P}(X_t \in B) \quad \forall A, B \in \mathcal{A} \tag{2}$$

The autocorrelation and partial autocorrelation functions graphs are good leading indicators of mixing. Mixing implies ergodicity, which is often a valuable property.

3.1.1 Stationarity

The ADF test assumes absence of Unit Root as the null hypothesis while KPSS test assumes the opposite. After conducting both of these tests on our raw data, we obtained a set (statistic, p-value) of $(0.14, >0.96)$ for the ADF test and $(21.66, < 0.01)$ for the KPSS test. These results strongly reject the absence of Unit Root in the raw series (X_t). In view of Figure 1, this was expected.

In finance, log-returns are often studied for their interesting additive property. Here, the log-returns series (Y_t) is defined as:

$$Y_t = \log\left(\frac{X_t}{X_{t-1}}\right) \quad \forall t \in \mathbb{Z} \tag{3}$$

These log-returns are represented in Figure 2 below.

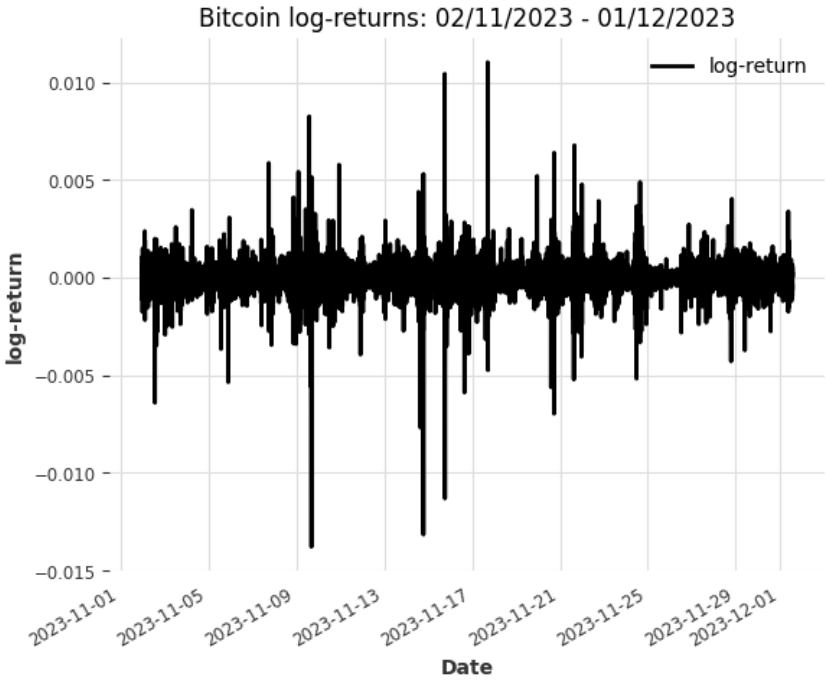


Figure 2: Evolution of Bitcoin log-return over the period 02/11/2023 - 01/12/2023

This series potential stationarity is hardly ruled out by a first glance, so we again performed ADF and KPSS tests. Indeed, these tests respectively returned the statistics $(-33.90, \simeq 0.00)$ and $(0.16, >0.10)$ which allow us to confidently consider the series (Y_t) stationary.

3.1.2 Mixing

If (Y_t) is mixing, we expect that for h sufficiently large, Y_{t+h} is approximately independent from Y_t, which translates to

$$\mathbb{E}[h(Y_{t+h})h(Y_t)] \approx \mathbb{E}[h(Y_{t+h})]\mathbb{E}[h(Y_t)] \tag{4}$$

for all $h : \mathbb{R} \to \mathbb{R}$ measurable, bounded or positive.
In particular, we can check whether $\mathbb{C}\mathrm{ov}(Y_{t+h}, Y_t) = \mathbb{E}[Y_{t+h}Y_t] - \mathbb{E}[Y_{t+h}]\mathbb{E}[Y_t] \approx 0$ and $\mathbb{C}\mathrm{ov}(|Y_{t+h}|, |Y_t|) \approx 0$ as h grows. Assuming (Y_t) mixing, the series is also ergodic, so $(|Y_t|)$ is also ergodic. Under this hypothesis, we can check autocorrelation graphs of (Y_t) and $(|Y_t|)$. On figure 3, these graphs are represented. We see that although (Y_t) presents autocorrelation with exponential decay, this is not the case of $(|Y_t|)$, hence the series (Y_t) is not mixing.

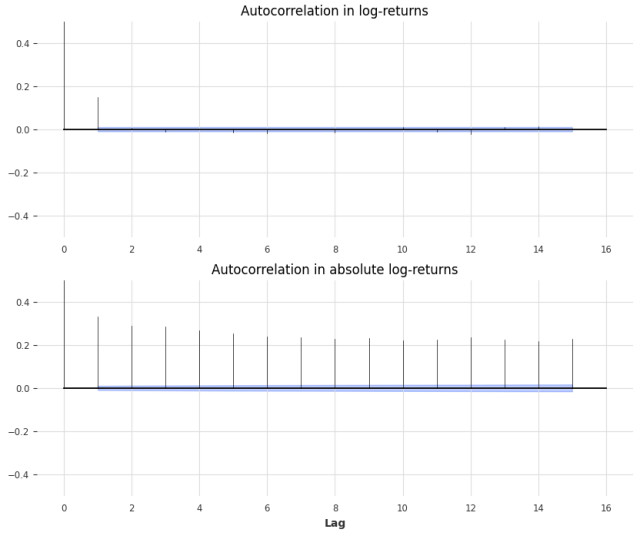


Figure 3: ACF graphs for the log-returns Y_t and its absolute value $|Y_t|$

3.2 ARIMA-GARCH models

An ARMA(p, q)-GARCH(p', q') process is, for $p, q \geq 0$ and $p', q' \geq 0$, a stationary series (X_t) such that for all $t \in \mathbb{Z}$:

$$X_t = a_0 + a_1 X_{t-1} + \cdots + a_p X_{t-p} + \xi_t + b_1 \xi_{t-1} + \cdots + b_q \xi_{t-q} \tag{5}$$

with

$$\xi_t = \varepsilon_t \sigma_t \quad \text{and} \quad \sigma_t^2 = \omega_0 + c_1 \xi_{t-1}^2 + \cdots + c_{p'} \xi_{t-p'}^2 + d_1 \sigma_{t-1}^2 + \cdots + d_{q'} \sigma_{t-q'}^2 \tag{6}$$

where

- (ε_t) is a white noise of variance 1
- $\omega_0, c_{p'}, d_{q'} > 0,\ a_p, b_q \neq 0$ and $c_1, \ldots, c_{p'-1}, d_1, \ldots, d_{q'-1} \geq 0$

3.2.1 ARIMA model on the trend

The log-returns series (Y_t) has been considered stationary, which indicates its *unconditional* variance is constant across time. In this context, fitting an ARIMA model (dynamic of X_t if ξ_t was a white noise in equation 5) is a good idea. However, the stationarity of (Y_t) does not forbid *conditionally* non-constant volatility to exist. In such a case, we talk about *Conditionnal Heteroskedasticity*. The GARCH models (dynamic of ξ_t in equation 6) are used to take into account such effects in the forecasts, resulting in a overall ARMA(p, q)-GARCH(p', q') model.

Looking at the autocorrelation graph of (Y_t) in figure 3, it makes sense to fit ARIMA models with $p, q \in \{0, 1\}$. We did so after a usual train/test split with 90% of data in the training set. Table 1 displays the results. The three models present almost the same AIC and BIC scores, so we arbitrarily decided to keep the AR(1) model. It is given by the following dynamic:

$$Y_t = 0.00091075 + 0.12946327 Y_{t-1} + \varepsilon_t \tag{7}$$

with $\mathbb{V}\mathrm{ar}(\varepsilon_t) = 0.24603986$.

Model	AIC	BIC
ARMA(1,1)	55052	55086
MA(1)	55053	55079
AR(1)	55051	55077

Table 1: Akaike and Bayesian Information Criterion of different models

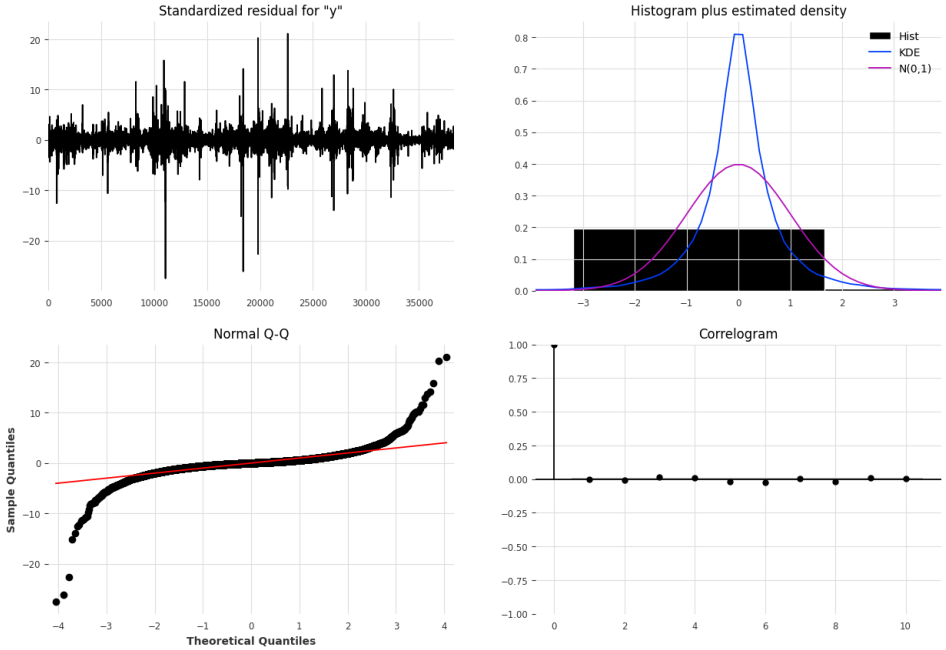


Figure 4: Residual diagnosis for the model AR(1)

3.2.2 GARCH model for the residuals

To decide whether or not it would be interesting to fit a GARCH model on the residuals of the AR(1) model, the procedure consists in

- first checking whether the residuals are a white noise (in case there is no need for a GARCH model). This can be done by conducting a Ljung-Box (LB) Test, which assumes the null hypothesis of white noise property.
- second, if the residuals are indeed not a white noise, checking for the presence of ARCH effects. This can be done with the help of the Lagrange-Multiplier (LM) Test, which assumes the null hypothesis of absence of such effects.

On figure 4, residuals of the chosen model are analysed. We do not see apparent trend (top left-hand corner) but the histograms and Q-Q plots (top right-hand and bottom left-hand corners) hint non-normality of residuals, which indicates that the model's residuals still contains information. As explained before, we conducted LJ and LM tests (see table 2) and the results encourage us to fit a GARCH model. Looking at the ACF and PACF plots of the squared residuals (ε_t^2) (figure 5), it is

Test	test statistics	p-value
Ljung-Box Test	72.98187	1.17381e-11
Lagrange-Multiplier Test	2041.7939033610214	0.0

Table 2: LB and LM Tests results on the series (ε_t)

interesting to test all combinations of $p \leq 9$ (from PACF graph) and $q \leq 10$ (from ACF graph). In

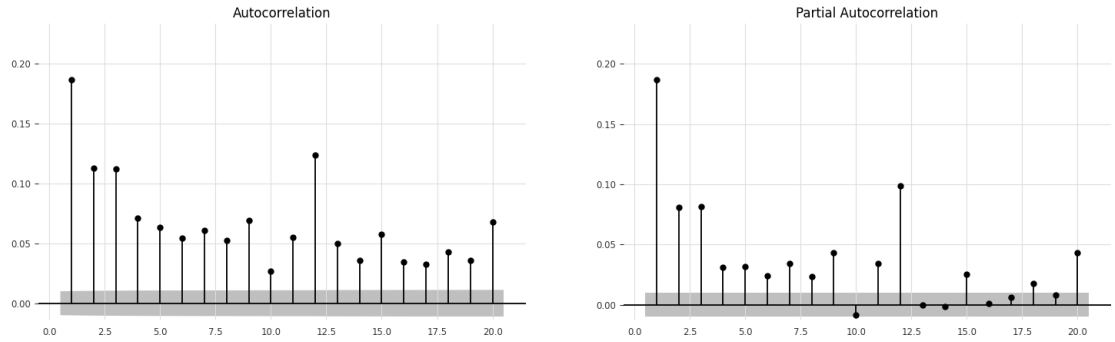


Figure 5: ACF and PACF for squared residuals (ε_t^2)

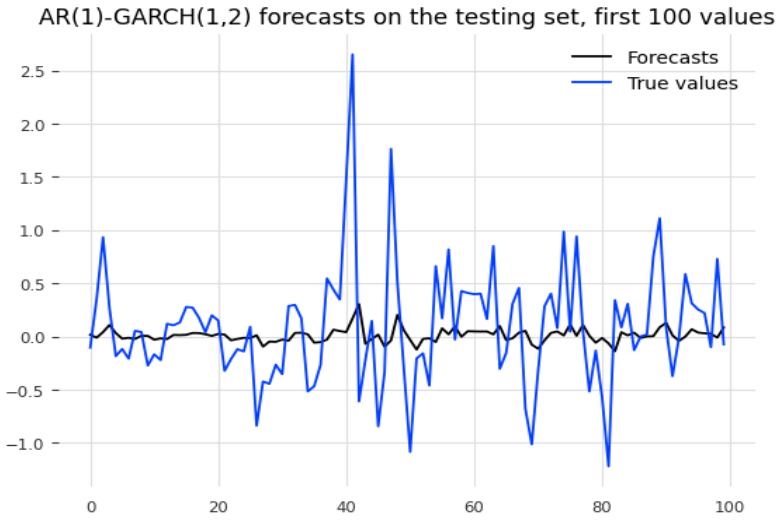


Figure 6: Forecasts of our AR(1)-GARCH(1,2) model over the first 100 time steps next to the training set

practice, the only GARCH(p, q) models that did not present ill-specified coefficients were those with $p \leq 1$ and $q \leq 2$. Looking at their AIC and BIC scores, we selected the GARCH$(1, 2)$ model.

3.2.3 Performances

In the previous discussions, we were careful in ensuring that the assumptions under our modelization choices were backed by statistical tests. We can finally apply our model to a set of unseen data (the testing set). Figure 6 demonstrates that the model actually performs poorly. On the whole testing set (4,261 samples), the root mean squared error is 0.4104 and the observed correlation between the forecasts and the true values is 0.1727. Indeed, this was expected as the Bitcoin is one of the most volatile asset on the market and the use of affine models could not capture its variability.

4 Neural networks based techniques

In light of the disappointing results the traditional ARIMA-GARCH analysis may provide for forecasting the Bitcoin prices, we will now explore how the Deep Learning techniques can be useful.

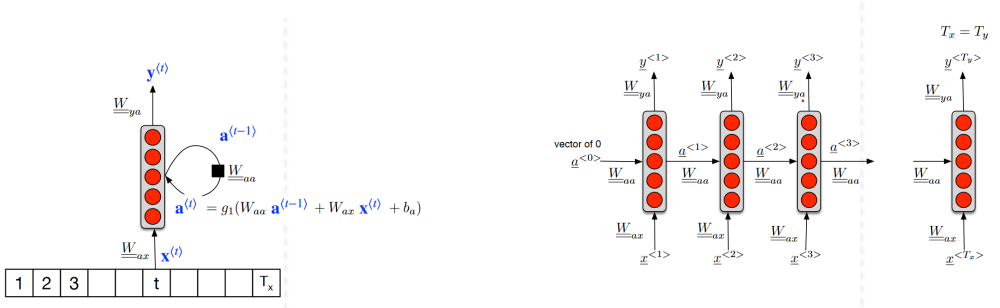
Recurrent Neural Network (RNN) is a class of neural network architecture where the output of a layer can be used as input of the same layer which differ from fully connected feedforward neural network. These kind of architecture are useful for sequential datas such as financial instrument's price. Beside basic RNNs, LSTM and GRU neural network will also be considered.

Standard RNN layer work as described in Figure 7. The datas are coming sequentially, updating at each step the 'activation' vector, or hidden state, $\mathbf{a}^{<t>}$ with the inputs features $x^{<t>}$ in relation with the weights W_{ax} and with the previous version of the activation vector $\mathbf{a}^{<t-1>}$ modified by the weights W_{aa}. To compute the output of the layer at a certain step t, another set of weights, W_{ya},is used on the activation vector.

$$y^{<t>} = g_z(W_{ya}a^{<t>+b_y})$$

With g_z an activation function.

It is important to notice that the weights matrices W are the same for each data of the input sequence independently of it position in the sequence. These weights are the parameters of the model modified during the training.



(a) 'Rolled' representation of RNN

(b) 'Unfolded' representation of RNN

Figure 7

The Figure 7 depicts the case where each data of the input sequence is related to a data of the output sequence, it is called a Many-to-Many configuration, but it is possible to get a single value in output, a Many-to-One configuration. In which case the activation vector used at the end has been updated by the whole sequence beforehand.

The main drawback of standard RNN comes when updating the coefficient of the parameters. Using gradient descent, the recursivity of a RNN layer create a vanishing gradient problem, caused by applying the derivative of the same activation function several times and thus leads to dead neurons learning.

LSTM and GRU are RNN architectures widely used in sequence prediction due to their ability in avoiding vanishing gradient problem and thus getting information from many time steps before the prediction. Each layer of a LSTM neural network now contains a memory cell used to compute the hidden state. In the LSTM unit, gates are made to update, forget or output the memory cell. These gates are activated according to the new data of the sequence and the hidden state of the previous step. Memory cell allows to create shortcuts in the learning process, reducing vanishing gradient issues. GRU units are simplified version of LSTM with only an update and reset gate and whose performance are close even though the gap in the number of parameters of the networks. Both these solutions are efficient but at the cost of a increased number of parameters compared to basic RNN.

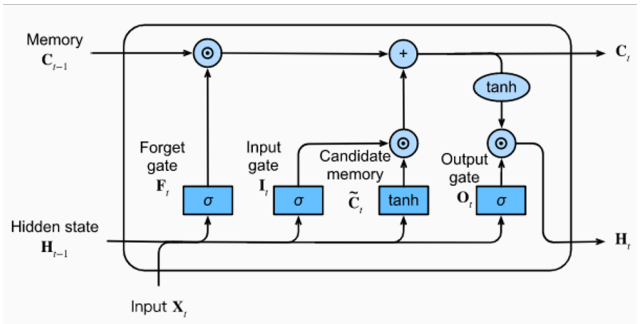


Figure 8: Diagram of a LSTM unit layer

Here we use a neural network to perform a regression task thus the loss function minimized during the training is the mean squared error.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Activation functions in the units are set as hyperbolic tangent.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The train dataset is built by the window sliding method. It means that the price time series is cut in sequences of equal length, here set to 5, but shifted by only on time step. The goal is to predict the next price, our target, with the 5 previous ones in input of the model. We obtain a dataset that we split as follow : 80% of the datas are used as train dataset, 20% as test dataset. The validation dataset is a fraction of the train dataset set to 25% (so 20% of the whole dataset). Train datas are taken at the beginning of the time series, test at the end of the time series and validation datas at the end of the train dataset as describe in Figure 9. Datas are transformed using MinMaxScaler fitted on the train dataset. The formula of the scaler is given by :

$$\text{MinMaxScaler}(X) = \frac{X - X_{\text{train}}^{(\min)}}{X_{\text{train}}^{(\max)} - X_{\text{train}}^{(\min)}}$$

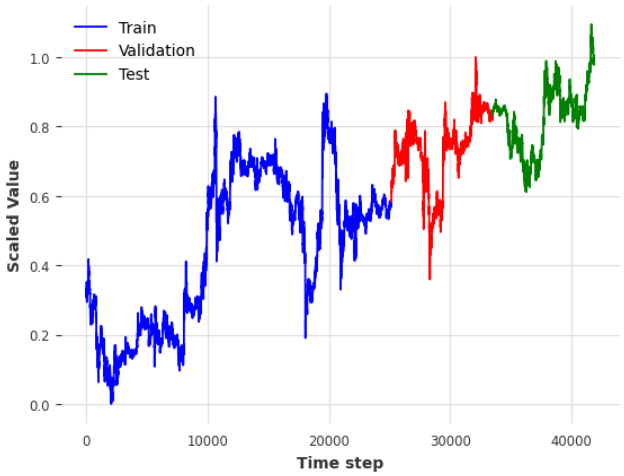


Figure 9

To compare the architectures, we define a grid of hyperparameters used to compute a selection based on the loss obtained on the validation dataset :

- Dropout $\in \{0, 0.3\}$
- Depth $\in \{2, 3, 4, 5\}$
- Units $\in \{32, 64\}$

The Dropout variable is a regularization parameter implying that some random neurons will not be used in the training process. Depth is the number of layers, this pack of layers is followed by a fully connected feedforward layer in order to get a single value output. Units is finally the number of neurons in each layer. It leads to 16 model evaluations per architecture. The grid search took between 1 hour for the fastest, the RNN, up to 3 hours and a half for the GRU.
The selected parameters for the final models are in the Table 3.

We can then compare the RMSE (Root of the Mean Squared Error) between the selected models in the Table 4. The RMSE of the non-scaled datas can be computed by multiplying each RMSE by $X_{\text{train}}^{(\max)} - X_{\text{train}}^{(\min)}$.

Table 3: Parameters retained after grid search selection

Parameters	**RNN**	**LSTM**	**GRU**
Number of neurons	32	32	64
Number of layers	5	3	2
Activation function	tanh	tanh	tanh
Dropout	0	0	0
Optimization Algorithm	Adam	Adam	Adam
Learning rate	0.001	0.001	0.001

Table 4: RMSE of the models on the different datasets

Dataset	**RNN**	**LSTM**	**GRU**
Train	0.00465	0.005	0.00442
Validation	0.00523	0.00529	0.00453
Test	0.00715	0.00411	0.00336

The performances of the standard RNN on the training and validation datasets are slightly better than the LSTM, but LSTM are known for working better on long-term dependency task. The most notable fact is that the RMSE over the test dataset is lower than the RMSE over the validation or even the the training dataset for LSTM and GRU. Looking back at 9, it might be because the test dataset contain smaller variation than in the train dataset. The models might struggle with predicting correctly these sudden variations but overall LSTM and GRU models handle better new datas with no prior information than standard RNN.

Note that we only focus here on one step ahead prediction and that none of the models have been trained to predict sequences of prices. Furthermore, the prediction where univariate and adding linked features to the dataset might increase the performance.

5 Conclusion

In summary, our project compared traditional methods such as ARIMA-GARCH models with advanced deep learning models, including LSTM and GRU neural networks, for forecasting Bitcoin prices.
Interestingly, traditional methods like ARIMA and GARCH, though historically popular, displayed limitations in adapting to the dynamic nature of cryptocurrency prices. The results clearly favored Neural Networks models, among which GRU models stood for their performance in capturing complex patterns in the Bitcoin prices Series.
However, the performance of the GRU models, while still outperforming random guessing, demonstrated a need for further refinement.
This study underscores the growing importance of embracing deep learning techniques, particularly GRU neural networks, in financial forecasting. The moderate success of GRU models, while emphasizing their practical importance, serves as a reminder of the challenges inherent in predicting cryptocurrency prices. As technology evolves, integrating innovative approaches becomes crucial for making informed investment decisions in the ever-changing landscape of financial markets.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[3] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.