DEEP NEURAL NETWORKS FOR BITCOIN PRICE PREDICTION

Emmanuel G. Pintelas

AM: 073650018

A thesis submitted for the degree of "*Technologies and Smart Services of Informatics and Communications Systems on Business Analytics and Data Science*".

June, 2020

Abstract

Deep Neural Networks (DNNs) are modern and powerful machine learning techniques, which achieve state-of-the-art pattern recognition performance in many research areas. They constitute artificial neural networks with multiple layers between the input and output layers, aiming on modelling complex non-linear relationships. Their popularity has grown exponentially over the past decade, due to innovations which have recently occurred in their training methods. The aim of this thesis is the implementation, the hyperparameter tuning and the optimization of deep learning models, which have recently proposed in the literature, for Bitcoin price prediction based on time series data. The prediction of Cryptocurrency price can be considered as a common type of time series problems, like the stock price prediction. Traditional time-series models, like the popular ARIMA, have been applied for cryptocurrencies price and movement prediction. Nevertheless, the growing research and rapid advances of digital technology constitute in the exponentially generation of data in size, dimension and complexity. As a result these models lack of the ability to capture the non-linearity of samples in the training set. Therefore, DNNs probably constitute the most suitable and efficient way to tackle these problems.

Keywords: Deep learning, long short-term memory network, Time series forecasting, Bitcoin forecasting

Acknowledgments

I would like to express my gratitude to my supervisor Professor V. Tampakas and co-supervisor Dr. Ioannis E. Livieris for their continuous help on my thesis. Their support helped me to understand the bitcoin market, deep neural networks and forecasting theory which was essential for my research.

Table of contents

Chapter 1
Introduction

Cryptocurrency is an electronic exchange tool which uses cryptographic functions to conduct financial transactions. Cryptocurrencies leverage blockchain technology to gain transparency, decentralization and immutability. BTC is the most famous cryptocurrency, which was created in 2009 by an anonymous group or person, reaching its peak value on December 16, 2017 by climbing to nearly $20,000. In the last ten years, 1512 alternative cryptocurrencies like Etherium and Litecoin, were created proving that the cryptocurrency market has emerged revealing its strong growth [11].

The most common form of "*investing*" in Bitcoin is buying Bitcoin currency, holding and selling in higher value in order to make profit like the stock exchange investment. Bitcoin exchanges are online marketplaces where you can trade bitcoin for traditional currencies, like BTC for USD. Bitcoin price prediction can potentially lead to high amounts of profits as the investor could buy, hold and sell the bitcoin currency the proper time based on his predictions. However, prices of Bitcoin have highly fluctuated which make them very difficult to predict. Thus, the development of an accurate prediction tool, utilizing complicated and sophisticated mathematical formulas is essential in order to assist Bitcoin investors to gain high profits.

The prediction of Bitcoin price can be considered as a common type of time series problems, like the stock price prediction [3]. Traditional time-series models, like the popular ARIMA, have been applied for cryptocurrencies price and movement prediction [4,6]. In recent years, machine learning techniques have also been applied for Bitcoin price prediction [1,2,5,7]. Nevertheless, the growing research and rapid advances of digital technology constitute in the exponentially generation of data in size, dimension and complexity. As a result, these models lack of the ability to capture the non-linearity of samples in the training set in contrast to Deep Neural Networks [12,13], [45-47] which probably constitute the most suitable and efficient way to tackle these problems.

Deep Neural Networks (DNNs) constitute modern and powerful machine learning techniques, which have a lot of commercial applications and have been applied to many fields like speech recognition, computer vision, audio recognition, natural language processing, social network filtering, machine translation, drug design, bioinformatics and medical image analysis [8-10]. More specifically, DNNs are high complex non-linear systems which constitute artificial neural networks with multiple layers between the input and output layers, aiming on modelling complex

non-linear relationships. Their popularity has grown exponentially over the past decade, due to innovations which have recently occurred in their training methods [18-19].

The aim and innovation of this thesis lies in the implementation, hyperparameter tuning, optimization and possible enhancement of deep learning models, which have recently proposed in the literature, for Bitcoin price prediction based on time series data. The data utilized for our experiments was acquired from Bitcoin Historical Data on Kaggle database and is constituted by 1-minute Bitcoin prices for the time period of January 2012 to August 2019. The implementation code of the DNN models was written in Python.

The remainder of this research is organized as follows: Chapter 2 presents an introduction to deep neural networks and basic fundamental aspects and challenges in deep learning domain such as basic optimization algorithms and bias-variance trade-off problem. Chapter 3 mainly presents how deep learning is applied on time series forecasting problems and finally Chapter 4 presents our proposed time series forecasting framework for predicting Bitcoin price and experimental results.

Chapter 2
Deep neural networks

In this chapter, we present the Deep Neural Networks (DNN) architecture, equations and optimization. A DNN model has a lot of parameters which need tuning before training in order to improve its performance requiring a lot of important decisions such as the number of layers, learning rates, number of hidden units and activation function. Bias-variance trade-off, regularization, softmax function, optimization algorithms, hyperparameter tuning and batch normalization are some significant essences we also need to explain in this section.

2.1 Introduction to deep neural networks

DNNs are of high significance in machine learning as they have a lot of commercial applications and have been successfully applied in many fields like speech recognition, computer vision, audio recognition, natural language processing, social network filtering, machine translation, drug design, bioinformatics and medical image analysis [14-16]. They constitute a part of a wider family of machine learning techniques based on artificial neural networks (ANNs).

ANNs are non-linear systems, which goal is to predict a target variable y based on an input vector X. A training set for a binary classification problem is represented as (X, y) for m training examples:

$$\{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(m)}, y^{(m)})\}$$

while $y \in \{0,1\}$. The basic architecture of a neural network is shown in Figure 1.

Figure 1 presents an ANN with 1 hidden layer which is also called shallow neural network. In general, neural networks are composed of an input layer, one or more hidden layers and an output layer. Figure 2 presents a deep neural network which has three or more hidden layers. Each layer has a number of nodes which are called neurons. Each arrow of the network which ends up on each node has a numeric value which is called weight. The neurons of the hidden and output layers have a computation function which take as input the output from the neurons of the previous layers. The neurons of the input layer don't have a computation function, they just represent the input vector X.

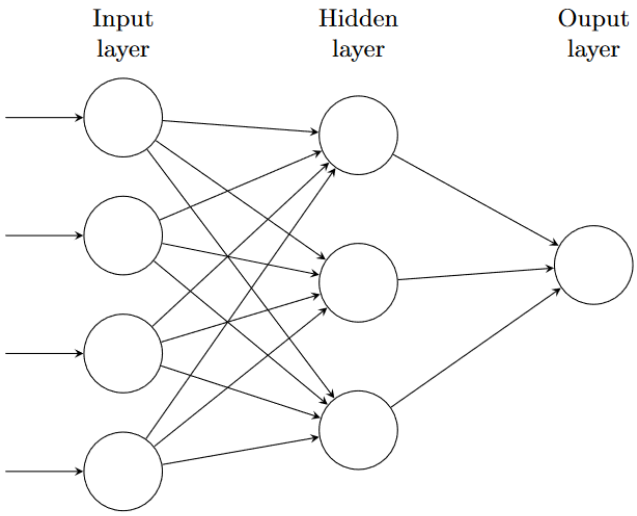


Figure 1. Shallow neural network architecture

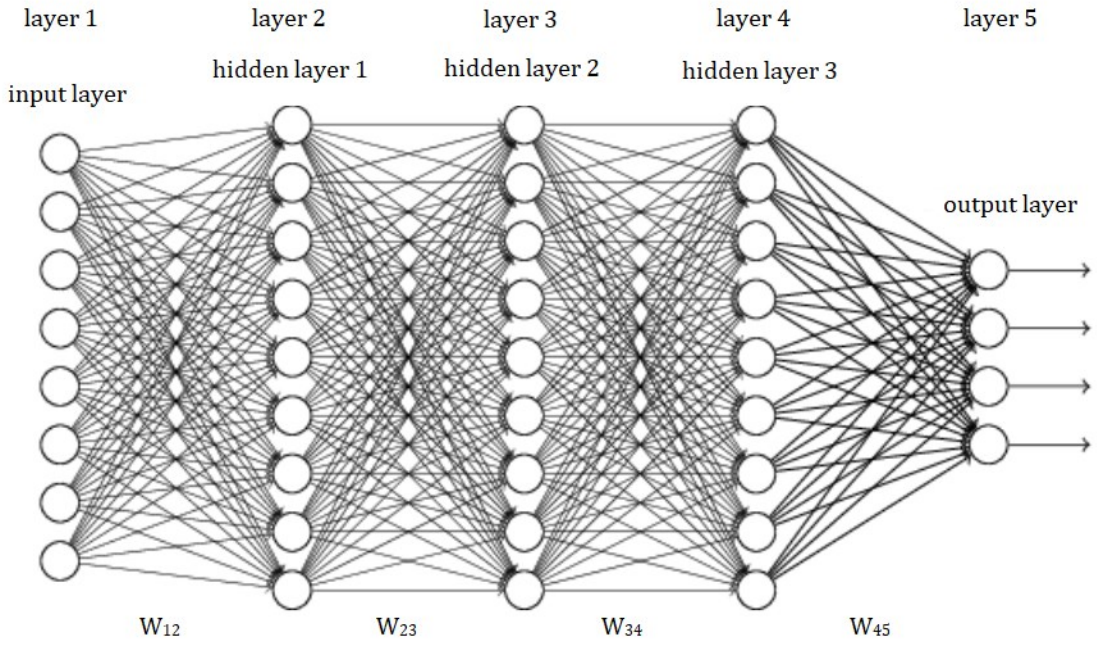


Figure 2. Deep neural network architecture

The computation function of each node nf, on each input, hidden and output layer f is given by:

$$a_f^{[nf]} = g_f^{[nf]}\left(z_f^{[nf]}\right) \tag{1.1}$$

$$z^{[nf]} = W_f^{[nf]}\, A_p^T \tag{1.2}$$

where $p = l$, $f = p + 1$, $l = 1,2,\dots,L-1$ and L is the total number of layers (input layer, hidden layers and output layer). $\boldsymbol{W_f}$ is a matrix which contains the weight vectors $W_f^{[nf]}$,

$$\boldsymbol{W_f} = \left[W_f^{[1]}, W_f^{[2]}, \dots, W_f^{[nf]}, \dots, W_f^{[Nf]}\right] \tag{1.3}$$

$$W_f^{[nf]} = \left[w_f^{[nf][1]}, w_f^{[nf][2]}, \dots, w_f^{[nf][np]}, \dots, w_f^{[Nf][Np]}\right] \tag{1.4}$$

where Nf is the total number of nodes of f layer, while N_p is the total number of nodes of p layer.

Notice that $W_f^{[nf]}$ is a vector containing the weights between the nodes np of layer p and node nf of layer f, where $w_f^{[nf][np]}$ is the weight between the node np of layer p and node nf of layer f. Moreover, A_p is a vector which contains the output values of nodes of layer p,

$$A_p = \left[a_p^{[1]}, a_p^{[2]}, \dots, a_p^{[np]}, \dots, a_p^{[Np]}\right] \tag{1.5}$$

where $a_p^{[np]}$ is the output value of node np of layer p. A_1 is the input vector and finally $g_f^{[nf]}$ is the activation function of node nf of layer f. The activation function is a function which defines the output of each node nf, given the single value of input $z_f^{[nf]}$.

$$A_1 = X \tag{1.6}$$

$$a_L^{[nL]} = \hat{y}^{[nL]} \tag{1.7}$$

$$a_p^{[np]} = 1 \tag{1.8}$$

if np is a biased mode.

2.2 Basic deep neural networks optimization problem

We recall that DNNs main object, like all prediction models, is the approximation of a target variable

$$y = f(X) + \varepsilon \tag{1.9}$$

based on an input vector X given a training set for a classification problem represe-nted as (X, y) for m training examples:

$$\left\{\left(X^{(1)}, y^{(1)}\right), \left(X^{(2)}, y^{(2)}\right), \dots, \left(X^{(m)}, y^{(m)}\right)\right\}.$$

The function $f(X)$ is the real determinist function which maps the input x to y by adding a random value ε, or else we call it noise. Instead, a prediction model builds the function $\hat{y} = f^*(X)$ which is an approximation of the real y. Therefore, we have to define a metric which will express how "*close*" the DNN's output $\hat{y}$ is to the real value y. This metric is a basic loss error function $e = E(y, \hat{y})$ where E could be for example a simple squared error: $(y - \hat{y})^2$. This basic error function will be the kernel of a cost function J, where J is the sum of all error functions for each instance X. The training problem of a DNN can be considered as the following optimization problem:

Find a hypothetic function:

$$\hat{y} = f^*(X)$$

which will minimize the function:

$$J = \frac{1}{m}\sum_{i=1}^{m} E\left(y^{(i)}, \hat{y}^{(i)}\right) \tag{1.10}$$

On a deep network topology, the output of each neuron nf on each hidden layer f is described by (1.1) and (1.2) as:

$$a_f^{[nf]} = g_f^{[nf]}\left(W_f^{[nf]}\, A_p^T\right) \tag{1.11}$$

for $f = L$, since $f = p + 1$, then $p = L - 1$ and thus:

$$a_L^{[nL]} = g_L^{[nL]}\left(W_L^{[nL]}\, A_{L-1}^T\right) \tag{1.12}$$

By (1.7) it turns out that:

$$\hat{y} = g_L^{[nL]}\left(W_L^{[nL]}\, A_{L-1}^T\right) \tag{1.13}$$

Therefore, by (1.10) and (1.13) the optimization problem for the DNN can be redefined in as follows:

Find the weight vectors $W_f^{[nf]}$ *of each matrix* $\boldsymbol{W_f}$ *for each layer* f *and node* nf*,*

which minimizes the following function:

$$J\left(\boldsymbol{W_1}, \boldsymbol{W_2}, \dots, \boldsymbol{W_f}, \dots, \boldsymbol{W_L}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left(y^{(i)}, g_L^{(i)[nL]}\left(W_L^{[nL]} A_{L-1}^T\right)\right) \tag{1.14}$$

2.3 Optimization algorithms for deep neural networks

The most popular optimization algorithm in machine learning is called gradient descent. In Figure 3, we present an example on how the gradient algorithm manage to converge on a global minimum for a very simple loss function. The training/optimization gradient descent algorithm is performed by using the following iteration formula which is defined as :

Algorithm 1.3.1. For $i = 1, \dots, m$ repeat {

$$\boldsymbol{W_f} := \boldsymbol{W_f} - \eta \frac{\partial J}{\partial \boldsymbol{W_f}}, \forall\ layer\ f$$

}

where η is the learning rate which controls the size of the step of the gradient descent.

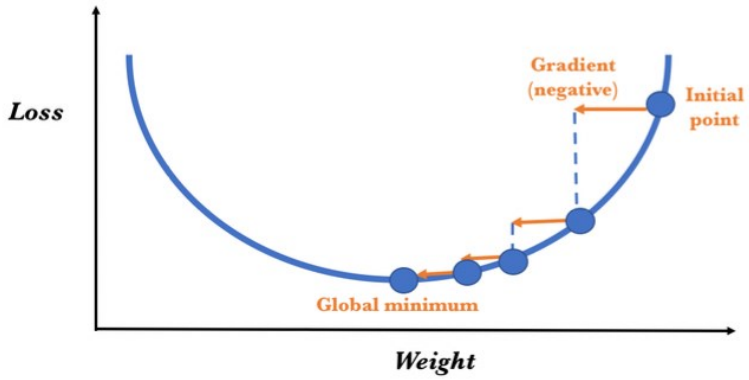


Figure 3. Visualization example of gradient descent algorithm to converge on global minimum

However, on more complicated loss functions, the converge of the optimization algorithm does not always lead to a global minimum, since for example if the parameter η is too low, the algorithm may stuck on a local minimum. Figure 4 presents a visualization example for this case. Therefore, more advanced and sophisticated optimization algorithms and methods are required in order to deal with such problems [Livieris PhD, 44].

Furthermore, for large and deep networks this approach is computationally inefficient. Thus, stochastic versions of traditional training algorithms have been proposed such as stochastic gradient descent [18] and Adam [19]. *Stochastic gradient*

descent is an iterative optimization method suitable for large scaling datasets because it converges faster since it randomly picks few samples, instead of the whole data set for each iteration and training on each single data point. *Adam* is an adaptive learning rate optimization algorithm which can be used instead of the stochastic gradient descent algorithm to update network weights. This algorithm utilizes the squared gradients to scale the learning rate while also take advantage of momentum by using moving average of the gradient instead.

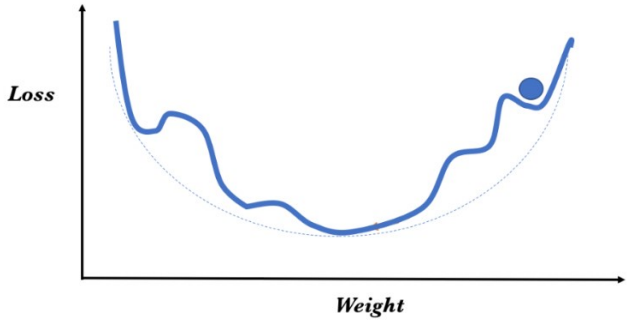


Figure 4. Visualization example of gradient descent to stuck on local minimum

2.4 Bias-variance trade-off

A fundamental problem in Machine Learning (ML) and thus Deep Learning (DL) is the bias-variance tradeoff. For the sake of a brief qualitative analysis we define three possible states that a ML model may occur after training and optimization process utilizing a training dataset. We call "*over-fitting*" (high variance) when the model managed to capture the patterns of the data but probably captured noise too and thus its prediction performance is extremely good in terms of training accuracy score but poor in terms of testing accuracy score, "*under-fitting*" (high bias) when the model failed to capture the patterns of the data and thus its prediction performance is very poor in terms of training and testing accuracy score and finally "*appropriate-fitting*" (low bias and low variance) when the model managed to capture the patterns of the data and thus its prediction performance is very good in terms of training and testing accuracy score. On Figure 5 we present a visualization example for a bias-variance trade-off problem.

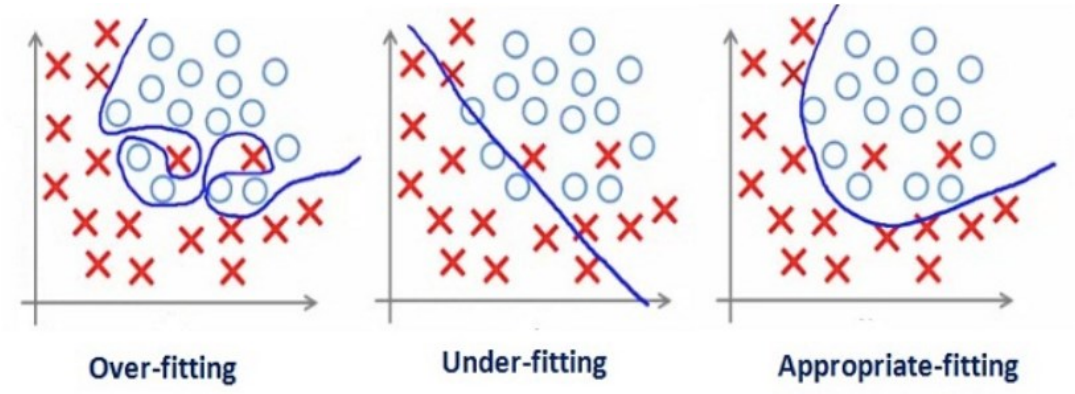


Figure 5. Visualization example for bias-variance trade-off

To illustrate this analysis strongly, we will describe it in a mathematician way too. For any random variable X, we have

$$Var[X] = E[X^2] - E[X]^2 \tag{1.15}$$

or

$$E[X^2] = Var[X] + E[X]^2 \tag{1.16}$$

where E, Var are the mean and variance functions respectively.

We recall that given an input vector X, the prediction model has to approximate the target variable $y = f(X) + \varepsilon$ by building a function $\hat{y} = f^*(X)$. Since ε is a random variable, then $E[\varepsilon] = 0$ and $var\ [\varepsilon] = \sigma^2$. Therefore,

$$E[y - f(X)] = E[\varepsilon] = 0 \tag{1.17}$$

$$Var\ [y - f(X)] = Var\ [\varepsilon] = \sigma^2 \tag{1.18}$$

Moreover, since f is a deterministic function then

$$\mathrm{E}[f(X)] = f(X) \tag{1.19}$$

Therefore,

$$E[y] = E[f(X) + \varepsilon] = E[f(X)] + E[\varepsilon]$$

which together with (1.17) and (1.19), we have

$$E[y] = f(X) \tag{1.20}$$

Moreover with (1.15) - (1.20) and since ε and $f(X)$ are independent, we have

$$\begin{aligned} Var[y] &= E[y^2] - E[y]^2 \\ &= E[f(X)^2 + 2f(X)\varepsilon + \varepsilon^2] - f(X)^2 \\ &= E[f(X)^2] + 2E[f(X)\varepsilon] + E[\varepsilon^2] - f(X)^2 \\ &= f(X)^2 + 2f(X)E[\varepsilon] + Var[\varepsilon] + E[\varepsilon]^2 - f(X)^2 \\ &= \sigma^2 \end{aligned} \tag{1.21}$$

We define as $Bias$ term for the approximated values $\hat{y}$ by a prediction model the function

$$Bias[\hat{y}] = E[\hat{y}] - E[f] = E[\hat{y}] - f(X) \tag{1.22}$$

In case, the Mean Square Error (MSE) is utilized as evaluation metric and since y, ŷ are independent we have

$$\begin{aligned} MSE &= E[(y - \hat{y})^2] \\ &= E[y^2 - 2y\hat{y} + \hat{y}^2] \\ &= E[y^2] - 2E[y\hat{y}] + E[\hat{y}^2] \\ &= E[y]^2 + \sigma^2 - 2E[y]E[\hat{y}] + Var[\hat{y}] + E[\hat{y}]^2 \\ &= \sigma^2 + Var[\hat{y}] + f(X)^2 - 2f(X)E[\hat{y}] + E[\hat{y}]^2 \\ &= \sigma^2 + Var[\hat{y}] + (f(X) - E[\hat{y}])^2 \\ &= \sigma^2 + Var[\hat{y}] + Bias[\hat{y}]^2 \end{aligned} \tag{1.23}$$

An efficient trained model should have the equation (1.23) minimum. This means that both the Variance and Bias term for the prediction $\hat{y}$ must be as low as possible.

2.5 Regularization in deep neural networks

Too complicated prediction models with a lot of weight parameters, such as DNNs are often prone to overfitting, especially when the training set is too small. One very common solution for addressing the over-fitting problem is regularization method. *Regularization* is a method which performs minor modifications to the learning algorithm in order to improve the model's performance on new data.

By this method, the term :

$$\frac{\lambda}{2\mathrm{m}}\sum_{f=2}^{L}\sum_{nf=1}^{Nf} W_f^{[nf]} W_f^{[nf]^T}$$

is added on the cost function and thus the (1.13) is rewritten to,

$$\mathrm{J}(\mathbf{W_1}, \mathbf{W_2}, \dots, \mathbf{W_f}, \dots, \mathbf{W_L}) =$$

$$\frac{1}{m}\sum_{i=1}^{m} \mathrm{E}\left(y^{(i)}, g_L^{(i)[nL]}\left(W_L^{[nL]} A_{L-1}^T\right)\right) + \frac{\lambda}{2\mathrm{m}}\sum_{f=2}^{L}\sum_{nf=1}^{Nf} W_f^{[nf]} W_f^{[nf]^T}$$

where λ is the regularization parameter. By setting to the parameter λ a value, then the deep neural network topology can approximate a simpler neural network topology or even a logistic regression function if setting λ too large. A logistic regression function can be considered a swallow neural network with just one node (neuron) with a sigmoid activation function.

In general, the regularization methods aims to simplify the too complicated initial network topology resulting in a less complicated and simpler final classification function, in order to reduce over-fitting. Another very popular regularization method is *dropout* method [41] which randomly drop units (along with their connections) from the neural network during training.

2.6 Long Short-Term Memory networks

Long Short Term Memory (LSTM) [42] are probably the most popular, successful and widely used deep learning topologies. They are able to learn long-term dependencies by utilizing feedback connections in order to "remember" past network cell states. These networks have become very popular since they have been successfully applied on a wide range of applications and have shown remarkable performance on time series forecasting [43].

In particular, LSTM constitute a special type of deep neural networks which belong to the class of Recurrent Neural Networks (RNN) domain. RNN were proposed in order to deal the problem of traditional feed forward neural networks, which is the lack of "memory" and thus the poor performance on sequences and time series problems, by utilizing cyclic connections on their hidden layer in order to acquire short term memory and be able to capture information from sequences and time series data. However, RNN suffer from vanishing gradient problem which restricts the model to learn long range dependencies. Therefore, LSTM come to solve this

problem by storing useful information on memory cells and vanishing unnecessary information, achieving in general a better performance than RNN.

The LSTM unit is composed by a memory cell, an input, output and forget gate. The input gate (i_t) along with a second gate (c_t^*), controls the new information which is stored into the memory cell (c_t) Eq. (4) at time t, while the forget gate (g_t) controls the past information which must be vanished or must be kept on the memory cell at time $t-1$. Finally, the output gate (ot) controls the final output information value (ht) which is given after a 1 time-step delay into the forget, input and output gate by a feedback connection loop. By this structure, the LSTM manages to create a controlled information flow by deciding which information must "forget" and which have to "remember" and thus managing to learn long term dependencies. More specifically, equations (1.24 – 1.29) describe in a mathematical way how an LSTM unit works.

$$g_t = \sigma(U_g x_t + W_g h_{t-1} + b_g) \tag{1.25}$$

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i) \tag{1.26}$$

$$c_t^* = tanh(U_c x_t + W_c h_{t-1} + b_c) \tag{1.27}$$

$$c_t = g_t \odot c_{t-1} + i_t \odot c_t^* \tag{1.28}$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o) \tag{1.29}$$

$$h_t = o_t \odot tanh(c_t) \tag{1.30}$$

where x_t denotes the input, h_t represents the hidden state, W and U are weight matrixes, b is a bias term, σ is the sigmoid function and the $\odot$ denotes component-wise multiplication.

Chapter 3
Time series forecasting

In this chapter, we present time series analysis and forecasting methods while mostly emphasize and focus on describing how machine learning and deep learning can be applied to time series forecasting. Finally, we present the Bitcoin price forecasting problem which is also our topic in this work.

3.1 Time series introduction

Time series are ordered sequences of values which are usually listed over time t. Some time series application examples are stock prices, weather forecasting, earthquake prediction and electroencephalography. Time series analysis is comprised by a variety of methods which analyze time series data aiming to extract useful statistics and various characteristics in order to make forecasts. Some time series analysis methods are exploratory analysis, autocorrelation analysis, spectral analysis, curve fitting and function approximation [23], while some traditional time series forecasting models are autoregressive integrated moving average (ARIMA) models, autoregressive fractionally integrated moving average (ARFIMA) model, nonlinear autoregressive exogenous model (NARX), autoregressive conditional heteroscedasticity (ARCH) model, Markov switching multifractal (MSMF) model, Hidden Markov model (HMM) and finally machine learning and deep learning models [23].

Time series forecasting [20] is the domain which utilizes models to predict future values based on the dynamics and correlations between observed historical data. It is constituted by two main categories which are one-step prediction [25, 26] and multi-step prediction problems [21, 22]. In both problems a time-step interval is usually utilized which defines the time delay of each future prediction. In one-step prediction problems, the model aims to predict one future value at one time-step interval, while in multi-step prediction problems the model predicts future values on consecutive time-step intervals aiming to describe the time series future evolution and trajectory [22]. In this work, we focus in one-step prediction since multi-step prediction is more difficult [27] especially on very complicated and challenging prediction problems as Bitcoin price forecasting.

3.2 Machine learning applied to time series forecasting

Machine learning can be applied to time series datasets as a supervised learning problem by preprocessing time series data in order to create a suitable data form which will be utilized to a machine learning model as a training set.

3.2.1 Brief description of supervised learning

Supervised learning utilizes algorithms to learn the mapping function $f: \mathbb{R}^N \rightarrow \mathbb{R}$ from an input to an output based on examples (instances) from datasets. A dataset **D** for a supervised learning problem is represented as $(\mathbf{X}, \mathbf{y})$ for M total instances:

$$\{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(M)}, y^{(M)})\}$$

where $X^{(i)} \in \mathbb{R}^N$ represents the instance i which contains the values of N different features, while $y \in \mathbb{R}$ represents the target variable. Therefore, we define the supervised learning problem as:

$$\boldsymbol{X} = \{X^{(1)}, X^{(2)}, \dots, X^{(i)}, \dots, X^{(M)}\} \quad (2.1)$$

$$\boldsymbol{y} = \{y^{(1)}, y^{(2)}, \dots, y^{(i)}, \dots, y^{(M)}\} \quad (2.2)$$

$$\boldsymbol{D} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(i)}, y^{(i)}), \dots, (X^{(M)}, y^{(M)})\} \quad (2.3)$$

$$\boldsymbol{Tr} \subset \boldsymbol{D} \quad (2.4)$$

$$\boldsymbol{V} \subset \boldsymbol{D} \text{ and } \boldsymbol{V} \cap \boldsymbol{Tr} = \emptyset \quad (2.5)$$

$$\boldsymbol{T} \subset \boldsymbol{D} \text{ and } \boldsymbol{T} \cap \boldsymbol{Tr} = \emptyset, \boldsymbol{T} \cap \boldsymbol{V} = \emptyset \quad (2.6)$$

where $\boldsymbol{Tr}$ is the Training set which is utilized in order to train a model, $\boldsymbol{V}$ is the Validation set which is usually utilized for validating the prediction model in order to tune its parameters and thus retrain the model till an appropriate validation score achieved, while $\boldsymbol{T}$ is Test set for evaluating the prediction model. It is worth mentioning that, the Test set represents all the new upcoming instances that a model has to predict in real world cases.

3.2.2 Time series definitions and notations

Let assume a continuous-time signal V(t). The signal V(t) could be for example the weather's temperatures or stock market prices or bitcoin prices $\forall t \in \mathbb{R}^+$. By utilizing a sampling time step interval T, a time series dataset can be created as defined below in (2.7).

$$D_T = \{V(t_{init} + T), V(t_{init} + 2T), \dots, V(t_{init} + kT), \dots, V(t_{init} + k_{Pr}T)\} \quad (2.7)$$

$$t_{Pr} = t_{init} + k_{Pr}T \quad (2.8)$$

D_T contains k_{Pr} total discrete ordered values $V(t_{init} + kT)$, the term k represents the discrete-time defined by the set $K = \{1, \dots, kPr\}$ and $k \in K$. The term t_{init} is the starting time point of sampling process and t_{Pr} represents the present time. We also define:

$$V(t_0 + kT) = V_T(k), \forall\, k \in K \quad (2.9)$$

A visualization example is also presented in Figure 6.

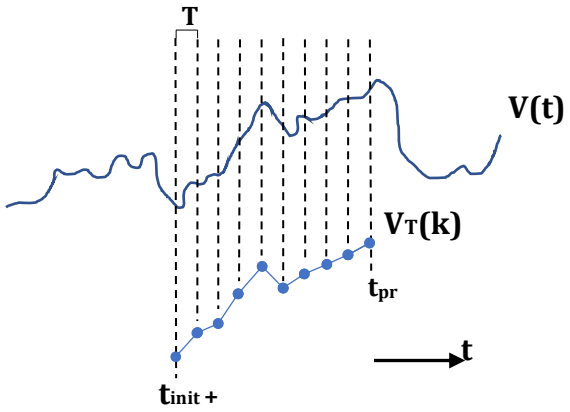


Figure 6. Extracting time series dataset by sampling a continuous-time signal V(t)

By (2.7) and (2.9) we can re-write (2.7) to

$$D_T = \{V_T(1), V_T(2), \dots, V_T(k), \dots, V_T(k_{pr})\} \quad (2.10)$$

3.2.3 Time series forecasting with static modeling

A prediction model will have to utilize the dataset D_T in a way (train model with D_T) to be able to make future forecasts. By the term static modeling, we mean that the dataset D_T will not change or be updated as time passes, while the model will have to make future predictions based on its training with the static dataset D_T.

An advantage of this approach is that it can be easily implemented in real world systems while it can be very efficient on time series problems which their patterns do not change significant over time. Instead on time series problems, such as the Bitcoin price prediction, where the patterns may change dynamically over time due to various external changes, this approach would probably fail to make future forecasts since the model will attempt to make predictions based on probably outdated patterns learned on a static dataset.

3.2.4 Time series forecasting with dynamic modeling

In general, the main object of each time series prediction model is the prediction of the future value $V_T(k_f), \forall k_{Pr}$ utilizing the dataset D_T, where

$$k_f = k_{Pr} + 1 \tag{2.11}$$

$$D_T \coloneqq D_T \cup \{V_T(k_f)\} \tag{2.12}$$

This means that the dataset D_T must be dynamic since it will have to be updated every time, as time passes, by adding the value $V_T(k_f)$ as the present value for the new updated dataset D_T. This could be of high importance in real world time series forecasting systems since future values are highly depended from the most resent instances, while patterns observed in the "very" past maybe start to lose their predictive power and lose their significance. Thus, very old instances should be removed from dataset D_T as defined below.

$$D_T \coloneqq D_T \setminus \{V_T(1), \dots, V_T(k_{old})\} \tag{2.13}$$

where k_{old} is the last discrete-time point in the set $K_{old} = \{1, \dots, k_{old}\}$ which denotes the discrete-time points of old instances.

Another interesting approach, instead of totally removing old instances, could be to add weights on each instance depending on how old or new they are in time, in order to guide the model to give gradually more focus on learning the most recent instances, while give gradually less focus on learning instances which get older over time.

Therefore, by the term dynamic modeling, we mean that the dataset D_T will change and be updated as time passes, while the model will have to make future predictions based on its continuous training with the dynamic dataset D_T.

3.2.5 Data preprocessing

We recall that the dataset D_T represents ordered data values utilizing a sampling time step interval T. The most common method to transform the initial data D_T to another form of data $\boldsymbol{D_T}$ as described in (2.3), suitable for machine learning models

to utilize for their training process, is the sliding window method. By this, a window of a fixed horizon size $N\left(N \in \mathbb{N}\ and 1 \leq N < k_{pr}\right)$ slides all over the D_T extracting its data values by batches of size N, adding them on a new data matrix $\boldsymbol{X_T}$ which contains the new instances of N dimension, while also filling a target vector $\boldsymbol{y_T}$. The vector $\boldsymbol{y_T}$ contains the target values to the corresponding extracted batch instances of dataset D_T. M are the total number of instances of the data matrix $\boldsymbol{X_T}$, while also

$$M = k_{Pr} - N \tag{2.14}$$

Therefore, the dataset $\boldsymbol{D_T}$ is defined as:

$$\boldsymbol{D_T} = \{(X_T^{(1)}, y_T^{(1)}), (X_T^{(2)}, y_T^{(2)}), \ldots, (X_T^{(i)}, y_T^{(i)}), \ldots, (X_T^{(M)}, y_T^{(M)})\} \tag{2.15}$$

where

$$X_T^{(i)} = \{V_T(i), V_T(i+1), \ldots, V_T(i+N-1)\} \tag{2.16}$$

$$y_T^{(i)} = V_T(i+N) \tag{2.17}$$

$$\forall\, i \in \{1, \ldots, M\}$$

An example which describe in a visually way the sliding window method applied on a time series example utilizing a horizon window size of $N = 3$ is presented in Figure 7.

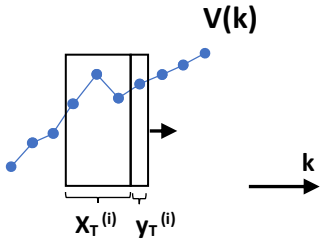


Figure 7. Sliding window method utilizing a horizon window size of N = 3

However, in order to create datasets $\boldsymbol{Tr}, \boldsymbol{V}$ and $\boldsymbol{T}$ for time series problems, as defined in (2.4) – (2.6), we have to define two splitting points k_V and k_T, where $k_V \in$ K, $k_T \in K$ and $k_V < k_T$. Therefore, the datasets $\boldsymbol{Tr}$, $\boldsymbol{V}$ and $\boldsymbol{T}$ are defined as:

$$\boldsymbol{Tr} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(i)}, y^{(i)}), \ldots, (X^{(k_V-1)}, y^{(k_V-1)})\} \tag{2.18}$$

$$\boldsymbol{V} = \{(X^{(k_V)}, y^{(k_V)}), (X^{(k_V+1)}, y^{(k_V+1)}), \ldots, (X^{(k_T+1)}, y^{(k_T+1)})\} \tag{2.19}$$

$$\boldsymbol{T} = \{(X^{(k_T)}, y^{(k_T)}), (X^{(k_T+1)}, y^{(k_T+1)}), \ldots, (X^{(M)}, y^{(M)})\} \tag{2.20}$$

3.2.6 Model's training, validation and testing

Let assume a ML model C. Since $\boldsymbol{Tr}, \boldsymbol{V}$ and $\boldsymbol{T}$ datasets are created, C can start the training process utilizing $\boldsymbol{Tr}$. After training process, it has to be validated using $\boldsymbol{V}$ in order to tune the model's parameters and thus re-train again with $\boldsymbol{Tr}$ till an appropriate validation score achieved. Finally, it will be evaluated using the $\boldsymbol{T}$. However, all of this this process can be performed utilizing the traditional static modeling approach or utilize the dynamic modeling approach as defined in eq. 2.12.

3.2.6.1 Static modeling approach

The model C is training utilizing $\boldsymbol{Tr}$ and create the function $f_C: \mathbb{R}^N \rightarrow \mathbb{R}$. Then compute

$$V_{sc} = Eval\ (\hat{\boldsymbol{y}}_V, \boldsymbol{y}_V),$$

where $Eval$ can be considered a general function, which computes prediction model's performance depending on the utilized evaluation method,

$$\boldsymbol{y}_V = [y^{(k_V)}, y^{(k_V+1)}, \ldots, y^{(k_T-1)}]$$

$$\hat{\boldsymbol{y}}_V = [f_C(X^{(k_V)}), f_C(X^{(k_V+1)}), \ldots, f_C(X^{(k_T-1)})]$$

and V_{sc} the corresponding validation score. In an iterative procedure, the model's C parameters are tuned and re-training utilizing $\boldsymbol{Tr}_T$, till $V_{sc} > V_{opt}$, where V_{opt} is a variable validation threshold score. Finally, compute

$$T_{sc} = Eval\ (\hat{\boldsymbol{y}}_T, \boldsymbol{y}_T)$$

where

$$\boldsymbol{y}_T = [y^{(k_T)}, y^{(k_T+1)}, \ldots, y^{(M)}]$$

$$\hat{\boldsymbol{y}}_T = [f_C(X^{(k_T)}), f_C(X^{(k_T+1)}), \ldots, f_C(X^{(M)})]$$

and T_{sc} is the testing score.

3.2.6.2 Dynamic modeling approach

By this approach, the ML model will be dynamically trained with the dataset $\boldsymbol{Tr}$, validated on $\boldsymbol{V}$ and tested on $\boldsymbol{T}$. This process is defined below. Initially train C on

$$\boldsymbol{Tr} = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \ldots, (X^{(k_V-1)}, y^{(k_V-1)})\}$$

creating the function $f_C^{(k_V-1)}$.

The *Validation-tuning process* is defined as:

Compute

$$V_{sc} = Eval\,(\hat{\boldsymbol{y}}_V, \boldsymbol{y}_V)$$

and tune the model's C parameters till $V_{sc} > V_{opt}$, where $\hat{\boldsymbol{y}}_v$ is computed by the process defined below.

Initially define $\hat{\boldsymbol{y}}_v = \emptyset$.

Then $\forall i \in \{k_{v-1}, \ldots, k_{T-2}\}$ compute {

$$\hat{\boldsymbol{y}}_{\mathbf{v}} \coloneqq \hat{\boldsymbol{y}}_v \cup \{f_C^{(i)}(X^{(i+1)})\}$$

$$\boldsymbol{Tr} \coloneqq \boldsymbol{Tr} \cup \{(X^{(i+1)}, y^{(i+1)})\}$$

and re-train C on $\boldsymbol{Tr}$ creating the function $f_C^{(i+1)}$.

}

The *Testing-Evaluation process* is defined as:

Let $f_{Copt}^{(k_V-1)}$ be the function that model C created by training on $\boldsymbol{Tr}_T$ choosing the best parameters for this model as resulted from Validation-tuning process. Then compute

$$T_{sc} = Eval\,(\hat{\boldsymbol{y}}_T, \boldsymbol{y}_T)$$

where $\hat{\boldsymbol{y}}_T$ is computed by the process defined below.

Initially define $\hat{\boldsymbol{y}}_T = \emptyset$ and compute

$$\hat{\boldsymbol{y}}_T \coloneqq \hat{\boldsymbol{y}}_T \cup \{f_{Copt}^{(k_V-1)}(X^{(k_T)})\}$$

$$\boldsymbol{Tr} \coloneqq \boldsymbol{Tr} \cup \{(X^{(k_T)}, y^{(k_T)})\}$$

and *re-train* C on $\boldsymbol{Tr}$ creating the function $f_{Copt}^{(k_T)}$

Then $\forall i \in \{k_T, \ldots, k_{M-1}\}$ compute {

$$\hat{\boldsymbol{y}}_T \coloneqq \hat{\boldsymbol{y}}_T \cup \{f_{Copt}^{(i)}(X_T^{(i+1)})\}$$

$$\boldsymbol{Tr} \coloneqq \boldsymbol{Tr} \cup \{(X^{(i+1)}, y^{(i+1)})\}$$

and re-train C on $\boldsymbol{Tr}$ creating the function $f_{Copt}^{(i+1)}$ }

3.3 Bitcoin price forecasting problem

Bitcoin (BTC) is a new type of digital currency which utilizes blockchain technology and cryptographic functions to gain transparency, decentralization and immutability[BIBLIO]. Bitcoin is considered the first and the most popular cryptocurrency [28], which was invented by an anonymous group or person in 2009. Since then, 4000 alternative cryptocurrencies like Etherium (ETH) and Ripple (XRP) were created proving that the cryptocurrency market has emerged in financial area [29]. BTC, ETH and XRP are the three most popular cryptocurrencies, since they almost hold the 79.5% of the global cryptocurrency market capitalization.

Bitcoin price prediction can assist investors for making proper investment decisions in order to acquire higher profits while it can also help policy decision-making and financial researchers for studying Bitcoin markets behavior. Bitcoin price prediction can be characterized as a complicated and extremely noisy time series forecasting problem such as stock price prediction problem. Classic time series methods such as the well-known AutoRegressive Integrated Moving Average (ARIMA) model, have been applied for cryptocurrencies price [30] and movement prediction [31]. However, these models are not able to capture non-linear patterns of very complicated prediction problems in contrast to Deep Learning algorithms which achieve greater performance on forecasting time series problems [32].

Deep Learning algorithms (DL) are powerful machine learning algorithms which specialize in solving non – linear and very complex problems exploiting most of the times big amounts of data in order to become efficient predictor models. Cryptocurrency price prediction is by nature a very complex and non - linear problem since its values have very big fluctuations over time following an almost chaotic and unpredictable behavior which makes the accurate price prediction a very challenging task. Therefore, deep learning techniques may constitute the proper methodology to solve this problem.

Recent studies have utilized deep learning techniques for predicting Cryptocurrency price. Ji et al. [33] conducted a comparison of state-of-the-art deep neural networks such as Long Short-Term Memory (LSTM), Deep Neural Networks (DNNs), deep residual network, and their combinations for predicting Bitcoin price. Their results showed that LSTM were slightly better than other models for regression problem while DNNs outperformed all models on classification problems (ups and downs prediction). Shintate & Pichl [34] developed a trend prediction classification framework for predicting non-stationary Cryptocurrency time series utilizing deep learning. Their results revealed that their proposed model outperformed LSTM baseline model in terms of classification accuracy and F1 score, while profitability analysis showed that simple buy-and-hold strategy was superior to their model and thus it cannot yet be used for algorithmic trading. Lahmiri & Bekiros [35] utilized

LSTM and generalized regression neural networks for Cryptocurrency prediction. Their results showed that LSTM was superior to the generalized regression neural architecture concluding that deep learning is a very efficient method in predicting the inherent chaotic dynamics of cryptocurrency prices. Amjad & Shah [36] used live streaming Bitcoin data for predicting price changes (increase, decrease or no-change), building a model which utilizes the most confident predictions, in order to perform profitable trades. The classification algorithms which they used were Random Forest, Logistic Regression and Linear Discriminant Analysis. Their results seem to be very impressive since they managed to achieve a high prediction accuracy (> 60-70%) and about 5.33x average return on investments on test set.

Chapter 4
Experimental analysis

In this chapter, we present our proposed deep learning framework for Bitcoin price prediction and our experimental results of the comparison between this framework and the traditional forecasting approach. The implementation code was written in Python 3.5 while for all deep learning models Keras library [37] was utilized and Theano as back-end and for the machine learning models Scikit-learn library [38] was used.

4.1 Description of proposed method

In this work, we have used various LSTM and DNN base models for predicting the hourly price of BTC utilizing two different forecasting framework approaches. We have to mention that the basic idea of utilizing LSTM as a base learning model on BTC price prediction problem, is that it might be able to capture useful long or short sequence pattern dependencies, due to its special architecture design, assisting on prediction performance. The first prediction framework (Framework 1) constitute the traditional way for making forecasts utilizing historic time series data as described in Section 2, while in our proposed framework (Framework 2), instead of utilizing a specific time interval T, we utilize various time intervals of higher frequencies than the specific prediction frequency interval that we wish to make future forecasts, in order to utilize and exploit in a more efficient way all possible information that a historic dataset may contain. We have to mention that both framework approaches were utilized following the static modeling approach as described in Section 2, since our main objective is to identify the effectiveness or no of our proposed method.

4.1.1 Description of utilized dataset

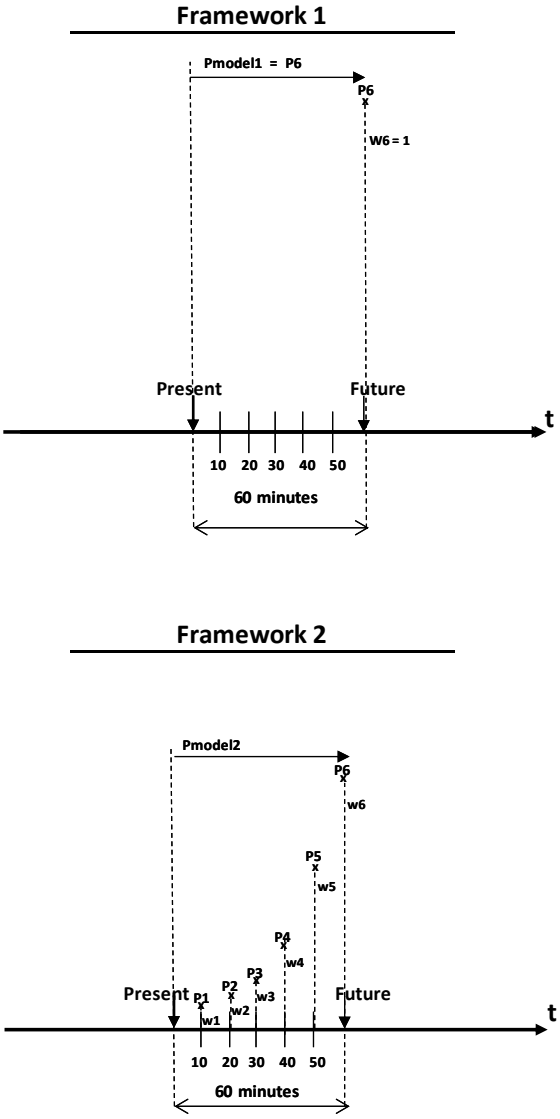
For the purpose of this research, we used data from Jan-2019 to Oct-2019, concerning the minute prices in USD and were divided into training set consisting of data from Jan-2019 to Sep-2019 (almost 340000 values) and testing set the from Sep-2019 to Oct-2019 (almost 38000 values). However, the testing points which were taken into consideration were much lower, since our models will make future forecasts per hour.

4.1.2 Description of proposed model

In this section we present in a very detailed way our proposed forecasting model for BTC price prediction.

4.1.2.1 Framework architecture

The two prediction frameworks (Framework 1-2) are presented in Figure 8.



P_1: Prediction of the 10-min model
P_2: Prediction of the 20-min model
P_3: Prediction of the 30-min model
P_4: Prediction of the 40-min model
P_5: Prediction of the 50-min model
P_6: Prediction of the 60-min model
P_{model1}: Final output prediction of Framework1
P_{model2}: Final output prediction of Framework2

A T-min model is model which is trained utilizing a T-min sampled data and can make forecasts per T minutes

$P_{model1} = P_6$
$P_{model2} = F\ (P_1, P_2, P_3, P_4, P_5, P_6)$

Let F be a customized weighted function such as:
$P_{model2} = w_1 * P_1 + w_2 * P_2 + \ldots + w_6 * P_6$,

where $w_1, w_2, \ldots, w_6$ are specified weights.

Figure 8. The utilized prediction frameworks

The basic idea of our Framework 2 is the exploitation of all possible information which lies in a continuous time signal $v(t)$, starting to observe it at a starting time t_0, in order to make forecasts on a specific future time step interval (1 hour step in our case). More specifically, let assume a continuous signal $v(t)$ which expresses the BTC price $\forall t \in \mathbb{R}^+$ starting to observe it at $t = t_0$. In our case the signal $v(t)$ is

approximated by the signal $\hat{v}(t)\ \forall\, t \in \mathbb{N}$, which expresses the BTC prices per 1-minute. Therefore, by sampling the signal $\hat{v}(t)$ utilizing various time step intervals $\mathrm{T_x}$ we can acquire various time series datasets D_x such as:

$$D_6 = \{\hat{v}(\mathrm{t_{init6}}), \hat{v}(\mathrm{t_{init6}} + \mathrm{T_6}), \dots, \hat{v}(\mathrm{t_{init6}} + \mathrm{k_{Pr6}T_6})\} \quad (2.21)$$

$$D_5 = \{\hat{v}(\mathrm{t_{init5}}), \hat{v}(\mathrm{t_{init5}} + \mathrm{T_5}), \dots, \hat{v}(\mathrm{t_{init5}} + \mathrm{k_{Pr5}T_5})\} \quad (2.22)$$

$$D_4 = \{\hat{v}(\mathrm{t_{init4}}), \hat{v}(\mathrm{t_{init4}} + \mathrm{T_4}), \dots, \hat{v}(\mathrm{t_{init4}} + \mathrm{k_{Pr4}T_4})\} \quad (2.23)$$

$$D_3 = \{\hat{v}(\mathrm{t_{init3}}), \hat{v}(\mathrm{t_{init3}} + \mathrm{T_3}), \dots, \hat{v}(\mathrm{t_{init3}} + \mathrm{k_{Pr3}T_3})\} \quad (2.24)$$

$$D_2 = \{\hat{v}(\mathrm{t_{init2}}), \hat{v}(\mathrm{t_{init2}} + \mathrm{T_2}), \dots, \hat{v}(\mathrm{t_{init2}} + \mathrm{k_{Pr2}T_2})\} \quad (2.25)$$

$$D_1 = \{\hat{v}(\mathrm{t_{init1}}), \hat{v}(\mathrm{t_{init1}} + \mathrm{T_1}), \dots, \hat{v}(\mathrm{t_{init1}} + \mathrm{k_{Pr1}T_1})\} \quad (2.26)$$

$$\hat{v}_x(k) = \hat{v}(\mathrm{t_{init}}x + k\mathrm{T}_x), k \in \mathbb{N} \quad (2.27)$$

and

$$\mathrm{k_{Pr}}x = \frac{\mathrm{t_{Prx}} - \mathrm{t_{init}}x}{\mathrm{T}_x} \quad (2.28)$$

where $x \in \{1,2,\dots,6\}$, D_x denotes the dataset with values $\hat{v}_x$ sampled by time step interval

$$\mathrm{T}_x = \mathrm{T} \cdot x(minutes)$$

the $\mathrm{t_{Prx}}$ denotes the present time $\forall\, x$, T is the minimum time step interval from the time series datasets (in our case this is 10 min) and $\mathrm{k_{Pr}}x$ are the total discrete points that a time step T_x defines in a time distance $\mathrm{t_{Prx}} - \mathrm{t_{init}}x$.

We have to mention that since our object is to build a prediction model which will make forecasts per 1-hour, then the traditional approach (Framework 1) will just utilize the dataset D_6 to train a forecasting model since this dataset is created by a sampling time step interval of $60\ min$. However, the Framework 2 will utilize all six datasets in order to make forecasts per 1-hour. In particular, six base models will be utilized and each one will be trained on each dataset D_x while every model will be optimized in making forecasts on its own specified time step interval and its own defined horizon size hor_x. For example, the model 1 will make forecasts per 10-min, the model 2 will make forecasts per 20-min and so on. Then the final prediction will arise by the synchronized combination of all six prediction models.

It is obvious that, since we want to make forecasts per 1-hour, the dominant and most important prediction is of model's 6 (60-min model) since it was optimized to make forecasts per 1-hour. However, if we were able to know for example a

prediction for the price before 10-min (a 50-min model would do that), it could provide us an additional information for the final prediction decision which is after 10-min. Therefore, instead of utilizing the P_6 (prediction of model 6) as the final prediction, by knowing the prediction of a 50-min model we can take into consideration the P_5 for the final prediction. One simple way to do that, is to put weights w_x on each prediction which will denote importance factors regarding to their contribution to the final prediction, based on a function F. Since the P_6 is more important prediction, this function could be $P_{Final} = 0.8\ P_6 + 0.2\ P_5$. In a similar approach, if we were able to know the prediction for the price before 20-min (a 40-min model would do that) the final prediction could be

$$P_{Final} = 0.7\ P_6 + 0.2\ P_5 + 0.1\ P_4$$

Thus, in Framework 2 the final prediction would be

$$P_{final} = w_6 P_6 + w_5 P_5 + w_4 P_4 + w_3 P_3 + w_2 P_2 + w_1 P_1 \qquad (2.29)$$

4.1.2.2 Models synchronization

Nevertheless, a main problem occurs in our Framework 2, which is the proper definition of t_{initx} of each model x. This means that the t_{initx} must be defined properly in order to synchronize the six prediction models aiming to combine their predictions in the most efficient and optimum way. In Figure 9 we illustrate how the models are synchronized in order to make forecasts on the same present time. By the term synchronization, mathematically we mean that

$$t_{Pr1} = t_{Pr2} = t_{Pr3} = t_{Pr4} = t_{Pr5} = t_{Pr6} = t_{Pr} \qquad (2.30)$$

where t_{Pr} is the actual present time that we want to make forecasts. Let define t_0 the initial starting point of observing the signal $\hat{v}(t)$ in which we extract the datasets $D_x\ \forall\ x \in 1,2,\dots,6$ utilizing a time step interval $T_x = T \cdot x(min)$. Then, the total discrete time points that a time step T defines on the time distance $t_{Pr} - t_0$ is

$$k_{Pr} = \frac{t_{Pr} - t_0}{T} \qquad (2.31)$$

We also define the starting time point of each dataset D_x as

$$t_{initx} = t_0 + k_{initx} \cdot T \qquad (2.32)$$

where $k_{initx} \in \mathbb{N}$.

By (2.28), (2.30), (2.31) and (2.32) we have

$$k_{Prx} T_x = t_{Pr} - t_{initx} \Leftrightarrow k_{Prx} T_x = t_{Pr} - t_0 - k_{initx} \cdot T \Leftrightarrow$$

$$k_{Prx}T \cdot x = k_{Pr}T - k_{initx} \cdot T \Leftrightarrow$$

$$k_{Pr} = x \cdot k_{Prx} + k_{initx} \Leftrightarrow$$

(since $k_{Prx} \in \mathbb{N}$ we have)

$$k_{initx} = k_{Pr} \bmod x \tag{2.33}$$

Finally, by (2.32) and (2.33) we have

$$t_{initx} = t_0 + (k_{Pr} \bmod x) \cdot T \quad \forall x \tag{2.34}$$

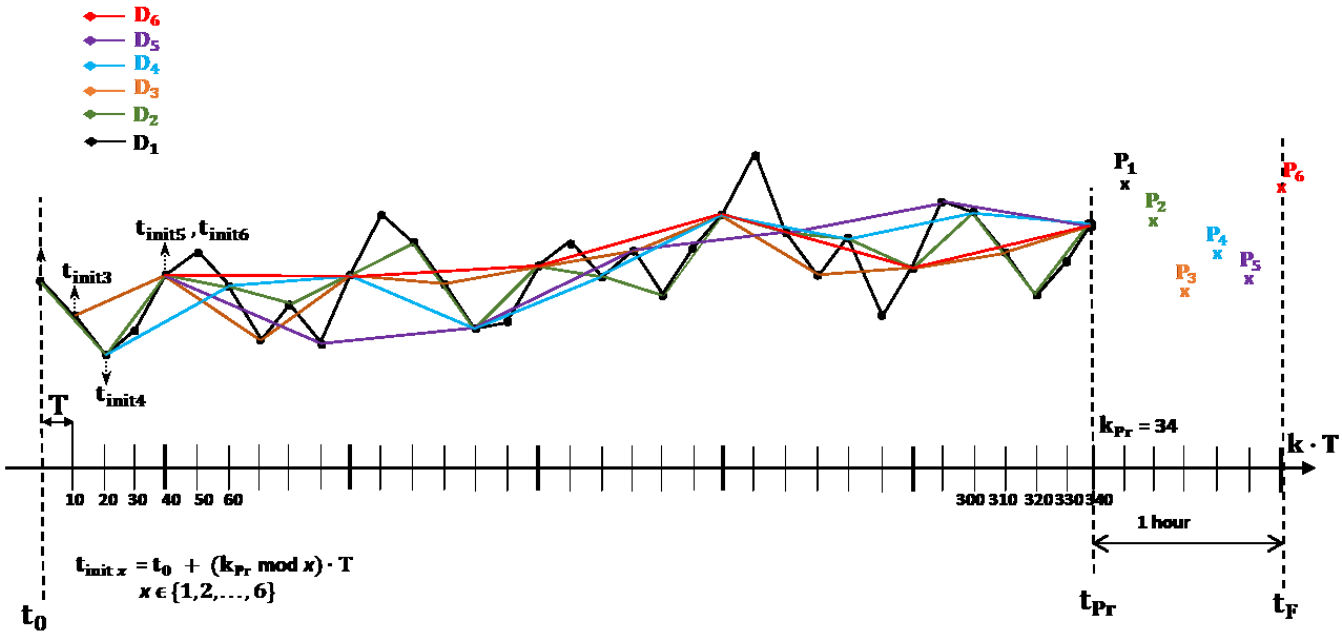


Figure 10. An example presenting how the time series models are synchronized to make forecasts in present time

In the example of Figure 9, $k_{Pr} = 34$, therefore by (2.34) we have

$$t_{init1} = t_{init2} = t_0, \quad t_{init3} = t_0 + T, \quad t_{init4} = t_0 + 2T, \quad t_{init5} = t_{init6} = t_0 + 4T$$

These starting points defined by the formula described in 2.34 will guarantee the synchronization of all time series models in order to make forecasts in present time.

However, in real world applications in order to achieve a maximum synchronization for every new prediction (in our case forecasting per 1-hour), one has to utilize the dynamic modeling approach as described in Section 2, since on every

new prediction per 1-hour, the models will not be totally synchronized on the new time point and thus the t_{initx} parameters will have to be dynamically computed all the time by the 2.34 formula for every new prediction attempt. Since our main object was to compare this proposed framework with the traditional one, we utilized the static modeling approach for both frameworks. Therefore, during the testing process we did not manage to acquire a maximum synchronization effect for every test prediction since the t_{initx} parameters were computed just once.

4.1.2.3 Final modelization of framework 2

Therefore, the final modelization of Framework 2, (based on Section 2 methodology) is defined as:

$$\mathbf{D} = \{(\mathbf{X_1}, \mathbf{Y_1}), \dots \ (\mathbf{X_x}, \mathbf{Y_x}), \ \dots, (\mathbf{X_6}, \mathbf{Y_6})\}$$

$$\mathbf{X_x} = X_x^{(t_{initx})}, \dots, X_x^{(t_{initx} + \ i \cdot T_x)}, \dots, X_x^{(t_{initx} + \ (k_{Prx} - hor_x)\, T_x)}$$

$$\mathbf{Y_x} = Y_x^{(t_{initx})}, \dots, Y_x^{(t_{initx} + \ i \cdot T_x)}, \dots, Y_x^{(t_{initx} + \ (k_{Prx} - hor_x)\, T_x)}$$

$$X_x^{(t_{initx} + \ i \cdot T_x)} = \{\, \hat{v}(t_{initx} + i \cdot T_x), \ \hat{v}(t_{initx} + (i+1) \cdot T_x), \ \dots, \ \hat{v}(t_{initx} + (i + hor_x - 1) \cdot T_x)\}$$

$$Y_x^{(t_{initx} + \ i \cdot T_x)} = \hat{v}(t_{initx} + (i + hor_x) \cdot T_x) \qquad (2.36)$$

4.1.3 Validation metrics

The most common validation metrics for measuring the performance of regression problems are Mean Absolute Error (MAE) and Root mean squared error (RMSE) [39]. Since the cryptocurrency price prediction problem can be considered a regression problem, in our experiments we utilized these two evaluation metrics. However, we have included only the RMSE score to present in this study, since the MAE score had almost the same behavior.

Nevertheless, the MAE and RMSE may constitute an incomplete way for validating cryptocurrency price prediction problems since a prediction model may have excellent RMSE and MAE performance but cannot properly predict the cryptocurrency price direction move (classification problem). A BTC trader or investor may be more interested in knowing the future price direction movement rather than knowing the exact future cryptocurrency price. Profitability analysis for algorithmic trading strategies reveal that classification models were more effective than regression models [40]. Therefore, on future works we should utilize utilized an

additional performance metric: Accuracy (Acc). More specifically, Acc measures the classification accuracy of our models for predicting the price movement direction (if the price will increase or decrease) based on the predicted prices that our regression model made.

4.2 Experimental results

In Table 2-3 we present the experimental results of our proposed DL time series forecasting framework (Model 2) against the traditional DL time series forecasting approach (Model 1) for the best identified topologies of various DL base models on each framework. We have to mention that exhausted and thoroughly experiments were performed in order to identify the DL topologies which bring the best performance results. Table 1 presents these final DL base models.

LSTM base model	LSTM Topology (# neurons per layer)	DNN base model	DNN Topology (# neurons per layer)
LSTM 1	20	DNN 1	20
LSTM 2	30	DNN 2	30
LSTM 3	40	DNN 3	40
LSTM 4	50	DNN 4	50
LSTM 5	30 - 10	DNN 5	50 - 10
LSTM 6	30 - 20	DNN 6	50 - 20
LSTM 7	30 - 30	DNN 7	50 - 30
LSTM 8	30 - 40	DNN 8	50 - 40
LSTM 9	30 - 50	DNN 9	50 - 50

Table 1. Best identified topologies for our DL base models

DL base model	Horizon 2		Horizon 3		Horizon 7	
	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
LSTM 1	0,0155	**0,0091**	0.0105	**0.0084**	**0.0118**	0.0166
LSTM 2	0,0090	**0,0084**	0.0195	**0.0095**	**0.0111**	0.0129
LSTM 3	0,0110	**0,0080**	0.0301	**0.0105**	**0.0102**	0.0121
LSTM 4	0,0143	**0,0084**	0.0095	**0.0091**	0.0113	**0.011**
LSTM 5	0,0083	**0,0081**	0.0094	0.0094	0.0115	0.0115
LSTM 6	**0.0094**	0.0103	0.0108	**0.0085**	**0.0126**	0.0139
LSTM 7	0.0105	**0.0081**	0.0092	**0.0087**	0.0131	**0.0126**
LSTM 8	0.0087	**0.0079**	0.0102	**0.0094**	0.0124	**0.012**
LSTM 9	**0.0101**	0.0103	**0.0101**	0.0106	0.0146	**0.0119**

Table 2. RMSE performance of LSTM base models for each forecasting framework (Models 1-2)

DL base Model	Horizon 2		Horizon 3		Horizon 7	
	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
DNN 1	0.0106	**0.0077**	0.0123	**0.0081**	0.0134	**0.0113**
DNN 2	0.0128	**0.0072**	0.0135	**0.0088**	0.0184	**0.0098**
DNN 3	0.0125	**0.0072**	0.0096	0.0096	0.0185	**0.0126**
DNN 4	0.0153	**0.0075**	0.009	**0.0086**	0.0126	**0.0103**
DNN 5	0.01	**0.0078**	0.0085	0.0085	**0.0084**	0.0145
DNN 6	0.0072	0.0072	0.0083	**0.008**	0.0111	**0.0091**
DNN 7	0.0076	**0.0074**	0.0097	**0.0077**	0.0146	**0.0135**
DNN 8	0.0084	**0.0082**	0.0101	**0.009**	**0.01**	0.0128
DNN 9	**0.0076**	0.0083	**0.009**	0.0096	**0.0122**	0.0125

Table 3. RMSE performance of DNN base models for each forecasting framework (Models 1-2)

Framework 2 (Model 2) clearly outperforms framework 1 (Model 1) especially on small horizon size. This is probably due to the fact that the contribution of other models to the final combined prediction starts to become noisy for big horizon sizes, cause of the extremely chaotic and unpredictable nature of BTC dataset especially for very small time step intervals (lower frequency models such as the 10-min model). We also observe that DNN base models bring the best results on average comparing to the LSTM base model.

In conclusion, the main disadvantage of framework 2, is that taking into consideration the predictions of other models (10-min – 50 min) in order to give extra information to the 1-hour model, they may instead add noise on the final prediction,

especially for cases when the 1-hour model is very accurate. However, these cases seems to be much infrequent since the average prediction score of the combined model is much better and thus in general it will lead to more accurate and robust predictions.

References

[1] McNally, S., Roche J., & Caton, S. "Predicting the Price of Bitcoin Using Machine Learning," 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Cambridge, 2018, pp. 339-343.

[2] Chen, M., Narwal, N., & Schultz, M. (2019). Predicting price changes in Ethereum. International Journal on Computer Science and Engineering (IJCSE) ISSN: 0975-3397 Vol. 7 No. 4 Apr 2015.

[3] Livieris, I. E., Kotsilieris, T., Stavroyiannis, S., & Pintelas, P. (2019). Forecasting stock price index movement using a constrained deep neural network training algorithm. Intelligent Decision Technologies. 1-14.

[4] Karakoyun, E. S., & Cibikdiken, A. O. (2018). Comparison of ARIMA Time Series Model and LSTM Deep Learning Algorithm for Bitcoin Price Forecasting. In The 13th Multidisciplinary Academic Conference in Prague 2018 (The 13th MAC 2018) (pp. 171-180).

[5] McNally, S., Roche, J., & Caton, S. (2018). Predicting the price of Bitcoin using Machine Learning. In 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP) (pp. 339-343). IEEE.

[6] Bakar, N. A., & Rosbi, S. (2017). Autoregressive integrated moving average (ARIMA) model for forecasting cryptocurrency exchange rate in high volatility environment: A new insight of bitcoin transaction. International Journal of Advanced Engineering Research and Science, 4(11).

[7] Derbentsev, V., Datsenko, N., Stepanenko, O., & Bezkorovainyi, V. (2019). Forecasting cryptocurrency prices time series using machine learning approach. In SHS Web of Conferences (Vol. 65, p. 02001). EDP Sciences.

[8] Ciresan, Dan; Meier, U.; Schmidhuber, J. (June 2012). "Multi-column deep neural networks for image classification". 2012 IEEE Conference on Computer Vision and Pattern Recognition: 3642–3649.

[9] Krizhevsky, A., Sutskever, I., & Hinton, G.(2012). "ImageNet Classification with Deep Convolutional Neural Networks" (PDF). NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada.

[10] "Google's AlphaGo AI wins three-match series against the world's best Go player". TechCrunch. 25 May 2017.

[11] Trimborn, S. & Huardle, W. K. (2016). Crix an index for blockchain based currencies, SFB 649 Econmic Risk, revise and resubmit Journal for Empirical Finances 2016-021.

[12] R. Law and N. Au, *A neural network model to forecast Japanese demand for travel to Hong Kong*, Tourism Management, 20, 89-97, 1999.

[13] Wang, Q., & Sun, X. (1996). Enhanced artificial neural network model for Chinese economic forecasting. In *Proceedings of the international conference on management science and the economic development of China* (Vol. 1, pp. 30-6).

[14] Zhou, S. K., Greenspan, H., & Shen, D. (Eds.). (2017). *Deep learning for medical image analysis*. Academic Press.

[15] Deng, L., & Liu, Y. (Eds.). (2018). *Deep Learning in Natural Language Processing*. Springer.

[16] Hemanth, D. J., & Estrela, V. V. (Eds.). (2017). *Deep learning for image processing applications* (Vol. 31). IOS Press.

[17] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222-2232.

[18] Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. 2014. arXiv:1412.6980v9

[19] Bottou, Léon; Bousquet, Olivier (2012). "The Tradeoffs of Large Scale Learning". In Sra, Suvrit; Nowozin, Sebastian; Wright, Stephen J. (eds.). Optimization for Machine Learning.

[20]George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[21] Nguyen Hoang An and Duong Tuan Anh. Comparison of strategies for multi-step-ahead prediction of time series using neural network. *In International Conference on Advanced Computing and Applications (ACOMP)*, pages 142–149. IEEE, 2015.

[22] Vincent, L. E., & Thome, N. (2019). Shape and time distortion loss for training deep time series forecasting models. In *Advances in Neural Information Processing Systems* (pp. 4191-4203).

[23] Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2015). *Introduction to time series analysis and forecasting*. John Wiley & Sons.

[24] Marcellino, M., Stock, J. H., & Watson, M. W. (2006). A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series. *Journal of econometrics*, *135*(1-2), 499-526.

[25] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *ACM SIGIR Conference on Research &Development in Information Retrieval*, pages 95–104. ACM, 2018.

[26] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W Cottrell. Adual-stage attention-based recurrent neural network for time series prediction. *In International Joint Conference on Artificial Intelligence (IJCAI), pages 2627–2633*. AAAI Press, 2017.

[27] An, N. H., Anh, D. T. (2015, November). Comparison of strategies for multi-step-ahead prediction of time series using neural network. In *2015 International Conference on Advanced Computing and Applications (ACOMP)* (pp. 142-149). IEEE.

[28] Sagona-Stophel, Katherine. "Bitcoin 101 white paper" (PDF). Archived from the original (PDF) on 13 August 2016. Retrieved 11 July 2016

[29] Trimborn, S. and Hardle, W. K. (2016). Crix an index for blockchain based currencies, SFB 649 Econmic Risk, revise and resubmit Journal for Empirical Finances 2016-021.

[30] Manu, K. S. (2019). Forecasting Bitcoin Prices: ARIMA and Seasonal Decomposition Approach.

[31] Bakar, N. A., Rosbi, S. (2017). Autoregressive integrated moving average (ARIMA) model for forecasting cryptocurrency exchange rate in high volatility environment: A new insight of bitcoin transaction. International Journal of Advanced Engineering Research and Science, 4(11).

[32] Siami-Namini, S., Tavakoli, N., Namin, A. S. (2018, December). A Comparison of ARIMA and LSTM in Forecasting Time Series. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 1394-1401). IEEE.

[33] Ji, S., Kim, J., & Im, H. (2019). A Comparative Study of Bitcoin Price Prediction Using Deep Learning. *Mathematics*, *7*(10), 898.

[34] Shintate, T., & Pichl, L. (2019). Trend prediction classification for high frequency bitcoin time series with deep learning. *Journal of Risk and Financial Management*, *12*(1), 17.

[35] Lahmiri, S., & Bekiros, S. (2019). Cryptocurrency forecasting with deep learning chaotic neural networks. *Chaos, Solitons & Fractals*, *118*, 35-40.

[36] Amjad, M., & Shah, D. (2017, February). Trading bitcoin and online time series prediction. In *NIPS 2016 Time Series Workshop* (pp. 1-15).

[37] A. Gulli and S. Pal. Deep Learning with Keras. Packt Publishing Ltd, 2017.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg. Scikit-learn: Machine learning in python. Journal of Machine Learning Research, 12(Oct):2825–2830, 2011.

[39] Tianfeng Chai and Roland R Draxler. Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[40] Ji, S., Kim, J., & Im, H. (2019). A Comparative Study of Bitcoin Price Prediction Using Deep Learning. *Mathematics*, *7*(10), 898.

[41] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

[42] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222-2232.

[43] Choi, J. Y., & Lee, B. (2018). Combining LSTM network ensemble via adaptive weighting for improved time series forecasting. *Mathematical Problems in Engineering*, *2018*.

[44] Livieris, I. E. & Pintelas, P. (2008). A survey on algorithms for training artificial neural networks. *Department of Mathematics, University of Patras. Tech. Rep*.

[45] Livieris, I. E., Stavroyiannis, S., Pintelas, E., & Pintelas, P. (2020). A novel validation framework to enhance deep learning models in time-series forecasting. *Neural Computing and Applications*, 1-19.

[46] Livieris, I. E., Pintelas, E., Kiriakidou, N., & Stavroyiannis, S. (2020). An advanced deep learning model for short-term forecasting US natural gas price and movement. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 165-176). Springer, Cham.

[47] Livieris, I. E., Pintelas, E., Stavroyiannis, S., & Pintelas, P. (2020). Ensemble Deep Learning Models for Forecasting Cryptocurrency Time-Series. *Algorithms*, *13*(5), 121.