EAI Endorsed Transactions
on Industrial Networks and Intelligent Systems Research Article EAI.EU

Internet Traffic Prediction Using Recurrent Neural Networks

Mircea Eugen Dodan[1], Quoc-Tuan Vien[1,*] and Tuan T. Nguyen[2]

[1]Faculty of Science and Technology, Middlesex University, United Kingdom. Email: mircea.eugen.dodan@gmail.com; q.vien@mdx.ac.uk.
[2]School of Computing and Mathematical Sciences, University of Greenwich, United Kingdom. Email: Tuan.Nguyen@greenwich.ac.uk.

Abstract

Network traffic prediction (NTP) represents an essential component in planning large-scale networks which are in general unpredictable and must adapt to unforeseen circumstances. In small to medium-size networks, the administrator can anticipate the fluctuations in traffic without the need of using forecasting tools, but in the scenario of large-scale networks where hundreds of new users can be added in a matter of weeks, more efficient forecasting tools are required to avoid congestion and over provisioning. Network and hardware resources are however limited; and hence resource allocation is critical for the NTP with scalable solutions. To this end, in this paper, we propose an efficient NTP by optimizing recurrent neural networks (RNNs) to analyse the traffic patterns that occur inside flow time series, and predict future samples based on the history of the traffic that was used for training. The predicted traffic with the proposed RNNs is compared with the real values that are stored in the database in terms of mean squared error, mean absolute error and categorical cross entropy. Furthermore, the real traffic samples for NTP training are compared with those from other techniques such as auto-regressive moving average (ARIMA) and AdaBoost regressor to validate the effectiveness of the proposed method. It is shown that the proposed RNN achieves a better performance than both the ARIMA and AdaBoost regressor when more samples are employed.

Keywords: Internet traffic prediction; recurrent neural networks; network planning

Received on 09 June 2022, accepted on 28 August 2022, published on 02 September 2022

doi: 10.4108/eetinis.v9i4.1415

1. Introduction

Over the last couple of decades, network traffic has become increasingly more diverse and complex. According to [1], the global IP traffic reached 1.2 zeta Bytes (ZB) of data in 2016 and such trend is growing exponentially. Considering this exponential growth in traffic, it is useful for a network manager to have more efficient tools that would help to make more reliable decisions in planning future expansions of the network and consider a better management of resources including bandwidth allocation for certain flows while at the same time alleviating congestion in the network [2], [3]. The forecasting of traffic is also important in the security field, since unusual patterns in traffic can be detected and compared with the predicted results [4], [5], in case of botnets attacks. There are techniques such as [6] that do not require machine learning or artificial intelligence in order to forecast traffic, but they are not reliable since long-term dependencies are not considered.

Network traffic prediction (NTP) represents a branch of network planning and capacity monitoring, that heavily depends on a set of historical data collected throughout the years. In order to make accurate forecasts about the future characteristics of the flows, multiple factors need to be considered in forecasting future traffic based on previous

[*] Corresponding author. Email: q.vien@mdx.ac.uk

experience. Some of these factors are the statistical characteristics of the Internet traffic which is self-similar and nonlinear in nature [1]. Specifically, the nonlinear nature of the traffic cannot be modelled by either a Poisson or a Gaussian distribution and the long-term dependencies must be modelled with nonlinear predictors.

In order to model highly non-linear time series, recurrent neural networks (RNNs) have been well investigated as a powerful prediction tool. The original RNN was designed by David Rumelhart in 1986 as a secondary method to the already developed autoregressive models.It has gained more popularity since the problem of gradient vanishing and gradient explosion were solved using Gated recurrent units (GRU) and Long-short term memory cells (LSTM) [1], [7]. With the help of LSTM and higher order statistics, the long-term dependencies in a data set could be partially solved for the prediction capabilities of the original RNN.

In this paper, we investigate the optimization of RNN model in order to achieve more accurate mean traffic forecasting in NTP as well as individual flow prediction, by considering the history of the traffic as training input for the neural networks. The main contributions of this paper canbe summarized as follows:

- RNNs are adopted to model the characteristics of Internet traffic flows.
- The impact of changing the dictionary of an RNN is analyzed with the aim of increasing the frequency of certain patterns. Also, the concatenation of multiple LSTM units is investigated to increase the generalization capabilities of the neural network.
- Considering the weekly periodicity and monthly periodicity inside the training data set, their impacts on the number of samples fed into the network are validated along with the delay time whilst training the neural network.
- A heuristic algorithm is proposed to find an optimal RNN architecture for NTP by solving the dissimilarities between the predicted values and the real values. Specifically, the proposed algorithm seeks to minimize the prediction errors including mean squared error, mean absolute error and normalized mean squared error.
- Simulation results are provided for the proposed algorithm and compared with the realtraffic values, as well as other counterpart techniques such as Auto-regressive integrated moving average (ARIMA) model and AdaBoost regressor which is based on decision trees.

The rest of this paper is organized as follows: Section II discusses some of the existing research related to network traffic forecasting. Section III analyses some of the architectural aspects of the proposed RNN for NTP, while also examining the mathematical aspects of the other methodologies that are used for comparison purposes. Section IV presents the simulation results for the proposed scheme and the comparisons with those in counterpart models. Finally, Section V concludes our paper.

2. Related works

This section reviews some of the methodologies and previous works that were carried outon traffic prediction domain by using: i) Linear models with ARIMA, ii) Decision trees using AdaBoost regressor, and iii) Non-linear models with RNNs.

A. Linear Models

Linear models represent one of the most common approaches for network flow predictions and they have been largely examined in the literature. There are multiple types of linear models such as ARIMA and fractal autoregressive integrated moving average (FARIMA). The ARIMA model out of all the linear models provides the best results. According to [8], [9], the ARMA model which is a subset of ARIMA, can be used in the particular scenario where the time series

B. Decision Trees and AdaBoost Regressor

AdaBoost regressor, which is also known as adaptive boosting regressor, is a technique that is based on weak learners such as decision trees, that are grouped together in different types of topologies in order to improve different classification capabilities. The boosting process combines these weak learners, mostly represented by binary decision trees, in order to improve the overall results after multiple iterations. Each decision tree tries to improve the prediction of the previous tree by using a set of weights and by observing where the previous model made mistakes. Oneof the key drawbacks of this method, is the inability to predict the non-linear behaviour of the traffic, while the long-term dependencies are not taken into account, even though the system hasa non-linear behaviour.

C. Non-linear Models and Neural Networks

One simple way to predict future traffic is by using hidden Markov models (HMM) which describes the non-linearity of the system using statistics between the current state of the system and the future possible states. The HMM uses transitioning probabilities and expectations in order to describe the most likely set of hidden states that are going to happen [13], [14]. Even though HMM is a great method of describing non-linear systems, it requires a lot of trainingdata and it does not take into consideration long-term dependencies because it doesn’t have a memory included.

One of the best non-linear techniques that is capable of predicting the non-linear behaviour of the traffic apart from HMM, are neural networks (NNs). The NNs represent a set of inter- connected computational units, which are also called neurons. All these neurons perform simple functions that are easy to understand, but after combining these neurons together in different topologies, the functions that can be performed become complex and nonlinear [15]. Becauseof these nonlinear properties, the NNs are capable to predict Network Traffic behaviour, which is inherently non-linear and also has long-range dependence between different moments in time [1], [2], [15], [16].

In order for an artificial NN (ANN) to work under any given input set of data and to predictthe future behaviour of the time series, the ANN must be trained by using either supervised Learning or unsupervised learning. In the case of supervised learning, the real values that have to be predicted are already available to the programmer. The purpose of the supervised learningis to minimize the error between the real value and the predicted value, by using different typesof metrics such as mean square error (MSE), mean absolute error (MAE), mean relative error (MRE) and relative mean square error (RMSE) [2], or in some cases of the RNNs there is also mean absolute percentage error (MAPE) [17]. According to [15] the most common learning algorithms for supervised learning are resilient propagation, backpropagation and Levenberg- Marquardt.

There are multiple types of NNs that can be used for traffic forecasting, such as: i) Multi-layer autoencoder which is a feed forward neural networks (FFNN) useful in the scenarios where the entire training data is presented at the beginning of the learning process and there is no needfor recurrence and back-propagation, [2], ii) Convolutional NNs which are mostly used in image processing but it can also be used for traffic classification [18], iii) Modified Elman NNs [19], and iv) RNNs that detects patterns inside time series. Variants of the RNNs were examined in[1], [7], [15], [17], [20]. The reason that the RNNs are so popular is due to the fact that they introduce a recurrent structure that establishes a connection between the current neuron and itself in a feedback loop format [17]. This circular structure acts as a memory that helps each neuron of the RNN to keep track of a state from a previous moment in time. This memory allows the network to influence the current state by using the information from the previous state [17].

This paper will examine how an RNN can predict future traffic samples for NTP, by using a variable sliding window algorithm that takes as input the amount of traffic that happened in the first couple of days and tries to predict the future characteristic based on this training information. Because the sliding window has a limited size, the samples that were predicted in previous time steps are fed back into the window in order to predict future samples.

3. Proposed RNN for NTP

This section investigates how a typical RNN that acts as an encoder – decoder system can work as a prediction mechanism to forecast network traffic. For a better understanding, two other methodologies are used for comparison purposes, including ARIMA and AdaBoost Regressor which are firstly investigated, followed by the proposed RNN model for the NTP.

A. ARIMA Model

ARIMA model is shown to be the best suited for the scenario where the time series presents stationarity between two adjacent moments in time. The stationarity of a data set is important because it indicates if the statistical properties, such as mean, variance and auto- correlation, are constant over small time intervals. The traffic estimation (x_{t_1} for the particular moment in time t_1), in the ARIMA model is strongly correlated with all the previous momentsin time, i.e. $t = t_1–1$, $t = t_1 - 2$, $t = t_1–3$, . . . All the elements of the time series present a linear relation between them. The predicted value (x_{t_1}) is proportional with ($x_{t_{1=0}}$), plus some noise components, which can be described using Gaussian distribution. The dependencies between the different traffic samples can be expressed, by using a polynomial equation with the following form [8], [9]:

$$x_t + \psi_1 x_{t-1} - \psi_2 x_{t-2} - \ldots - \psi_r x_{t-r} = z_t - \theta_1 z_{t-1} - \theta_2 z_{t-2} - \ldots - \theta_s z_{t-s}, \qquad (1)$$

where z_t represents the randomness that is introduced in the traffic following a normal distribution with zero mean and variance of σ^2 [8]. According to [9] the parameters ψ_1, ψ_2, ψ_3, . . . , ψ_r in (1) represent the auto-regressive operators which define the stationarity of the given data points at every point in time, while θ_1, θ_2, θ_3, . . . , θ_s are the moving average operators. It can be noticed that the sample taken at time t is correlated with all other samples taken at $t - 1$, $t - 2$, . . . ,$t - r$. The relation is linear between two adjacent moments in time because the traffic predicted at time $t - 1$ is proportional with that at moment t as follows [8], [9]:

$$x_{t-1} = Bx_t, \qquad (2)$$

$$x_{t-r} = B^r x_t, \qquad (3)$$

where B denotes the lag operator which correlates the sample collected at time t with all otherprevious samples. All the received traffic has to be collected and stored in a database in order to create a functional prediction model. The extended version of (1) can be rewritten in a polynomial format, that also includes the differentiating factor $\Delta = 1 - B$, which is an integral part of the ARIMA model. The factor Δ comes from the difference between two adjacent moments in time ($\Delta x_t = x_t - x_{t-1}$). We can express (1) as [8], [9]:

$$\psi(B)\Delta^d x_t = \theta(B) z_t, \qquad (4)$$

$$\psi(B) = 1 - \psi_1 B - \psi_2 B^2 - \ldots - \psi_r B^r, \qquad (5)$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_s B^s, \qquad (6)$$

where $\psi(B)$ and $\theta(B)$ can be estimated with the Box-Jenkins algorithm, in order to determine the order of the polynomials r, s and also the differentiating factor Δ [8]. The Box-Jenkins approach consists in performing a difference of the time series elements, until all the components of theseries appear to originate from a general stationary process, but there is also another methodology called fitting curve method which is used together with Box-Jenkins algorithm, in which the fittedvalues are subtracted from the original set of data [8]. Finding the order of the polynomials requires the determination of several nonlinear equations by approximating the solutions using numerical techniques. The numerical approximation involves two different techniques such as nonlinear least squares and maximum likelihood forecasting.

The proposed ARIMA and RNN model will take as training input the amount of traffic thatis destined for a particular website or group of websites during a time frame and it is going to predict the future behaviour of the traffic based on that training data. The ARIMA model used, is going to be described by three parameters: p called the lag order, d which is the number oftimes that the raw observations are differenced and finally q which is the size of the moving average window [21]. This linear model will be used only for comparative purposes against the proposed RNN model to evaluate the accuracy of the predictions when the same set of data is given to both techniques.

B. Regression Trees Model and AdaBoost

This subsection discusses how decision trees are used for predictive modelling and how toboost the performance of the trees by using the AdaBoost function. The first type of decision tree is called classification tree, where the target variable can take a discrete set of values andthe tree structure has leaves which represent labels of a specific class, while the branches are conjunctions of features that lead to those labels. The second type of decision tree is calledRegression tree and it is used when the predicted outcome is considered a real number, unlike the Classification tree where the outcome is a class. The tree like structure is maintained in thecase of regression trees, but the value obtained by terminal nodes in the training data is theaverage response that falls within a region of training. In our scenario the real value that has to be predicted is the mean traffic value, which is a real number. Because decision trees learners can create overly complex trees that do not generalize the data well (process called Over-fitting), thereis a need to reduce the complexity of the trees by implementing pruning which reduces the sizeof the decision tree. Another problem is the fact that the decision trees can become unstable due to small variations in the data set, which can be addressed by using a booster such as AdaBoost. AdaBoost is a machine learning algorithm that utilizes multiple weak learners in order to improve the overall performance of the system. The output of the weak learners, which are usually binary decision trees, are taken into consideration in a weighted sum [22]. This weighted sum is used as the result of the AdaBoost and it helps the algorithm to improve the predictions. One of the key components in the case of AdaBoost is the creation of the future binary trees based on the previous stumps. The next binary decision trees are created based on the error of the previous stump, so in the case of AdaBoost the order of creating the stumps matters and influences the final objective of regression.

C. Proposed RNN Model

The proposed RNN model follows a memory-based approach to determine the characteristicsof a time series by keeping track of the previous states. In case of standard feed forward neural Network (FFNN), the input training vector (−X→0) is injected into the FFNN, is not dependent of another input vector (−X→1) which is going to be fed at the next moment in time [24]. However, in case of time series, there is usually a dependency between the vectors inserted for training purposes. The RNN architecture used in this scenario has only one layer of LSTM cells that are interconnected with one another in a series topology. The input layer is used to take training information in order to feed it inside the neural network. The training info is grouped inside a window of size n, where the first $n - 1$ elements of the window represent the pattern that is going to be remembered by the neural network, while the n_th element of the window is the” predicted” value associated with that pattern, during the training process. Both the input layer and the output layer are encoded using one hot encoding, for the RNN to understand the format of the data that is fed inside [26]. The general architecture of the inputs and outputs that form an RNN cell is illustrated in Fig. 1, which consists of:

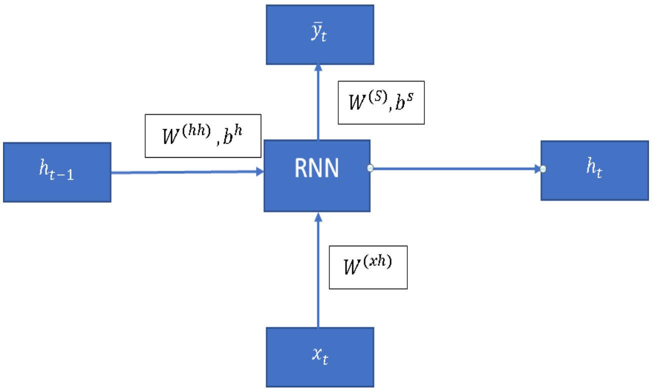

Figure 1. Weights and biases that are fed into the Recurrent Neural Network. [24]

- X_t represents the encoded message that is fed into the network at the moment t. The encoding method includes an embedding technique which produces a vector value. This encoding process is required to convert the original value or set of values into a representation that can be interpreted by the RNN [24].

- $\overline{y}_t$ represents the encoded message that is fed into the network at moment instance t. In the scenario of network traffic prediction, this encoded information represents the predicted internet traffic at time instance t. The representation of this predicted traffic is usually done in the form of a vector.

- $\boldsymbol{h_t}$ in this scenario signifies the memory cell, and the value that is stored inside the cell at time t. This value is dependent on the previous value that was stored inside the cell at a previous moment in time (h_{t-1}). The current state of the memory is also related with the encoded input sample at time t (x_t).

- The RNN cell takes several other inputs such as ($W^{(xh)}$, $W^{(hh)}$, b^h) apart from (x_t, h_{t-1}). All these components represent the weights and biases that are applied to the RNN to facilitate the learning process. As it was mentioned in the description of the parameter h_t, the memory cells are dependent on both the current input as well as the previous state of the memory cell (h_{t-1}). Here, $W^{(xh)}$ represents the amount of contribution (the weight) that (x_t) will have on the memory cell h_t. The other parameter that is taken into consideration is $W^{(hh)}$, which correlates the memory cell at time ($t - 1$) (h_{t-1}) with the memory cell inthe next moment in time t.

The architecture of the RNN that is used for predicting future network traffic is based on an encoder-decoder model, that takes as input the mean value (*mean* ($\overrightarrow{X_0}$)) or the real value of each vector ($\overrightarrow{X_0}$) and propagates the information from the encoder side to the decoder segment as shown in Fig. 2. This type of methodology is also called seq2seq model and it was originally used for machine translation problems. It was proved to be very effective for many applications [24] where the encoder-decoder design requires two different RNNs to be concatenated with one another in a serial method. The training information will be passed throughout the encoder from one neuron to another until it reaches the decoder segment.

In order to improve the encoder-decoder model, the forecasting tool must take into consideration the long-term dependencies between the data points and also the seasonality of the time series that are provided. According to [27] the data traffic that is received and stored by a flow monitor can be categorized into three types of seasonality: i) daily seasonality that correlates Day_n with Day_{n-1} and has the largest contribution; ii) weekly seasonality that correlates Day_n with Day_{n+7}; and iii) monthly seasonality that correlates Day_n with Day_{n+30}.

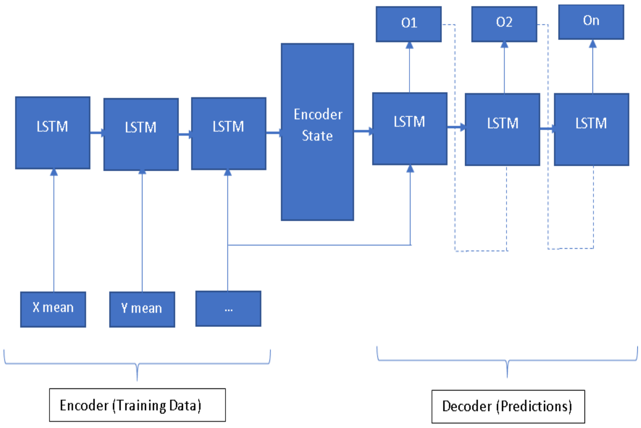

Figure 2. Structure of the encoder-decoder.

D. Evaluating the results

The level of accuracy of the prediction model is established by using various types of error estimation parameters. The purpose of these error estimators is to make a comparison betweenthe real data which is received after performing the experiment and the predicted data which is obtained from mathematical interpretations. In the ideal case the difference between the predicted value and real value should be close to zero, but this scenario cannot be achieved in a real environment. Choosing an appropriate error estimator is therefore vital since not all estimators provide identical outcomes given the same data. In this paper, the following estimators are considered:

- *Mean square error (MSE)*: is the simplest type of error estimator which can be computedby:

$$MSE = E(Y_{t+1} - \overline{Y_{t+1}})^2, \tag{7}$$

Where Y_{t+1} and $\overline{Y_{t+1}}$ denote the real value and the predicted value, respectively.

- *Normalized MSE (NMSE)*: takes into consideration the proportionality with a normalizing factor σ to mitigate the large spikes in error that happen in the scenario when the difference between the predicted value and the real value is very large. The NMSE is thus given by:

$$NMSE = \frac{E(Y_{t+1} - \overline{Y_{t+1}})^2}{\sigma^2} \tag{8}$$

Where σ denotes the standard deviation of the predicted data after the training is complete and n represents the total number of days for which the traffic will be predicted.

- *Mean absolute error (MAE)*: is a common measure of forecasting error in time series analysis and it helps to compare series that have the same scale. The MAE is based on an arithmetic mean, and thus the peek values that do not fit on top of the real values will not have a significant influence on the final measurement. The MAE is expressed as:

$$MAE = \frac{\sum_{t=0}^{n} |Y_{t+1} - \overline{Y_{t+1}}|}{n} \qquad (9)$$

- *Relative error (RE)*: is also called approximation error and it reflects the discrepancy between the exact data that is observed and the approximated data. This type of error is usually represented in percentage by:

$$RE = \frac{\sum_{t=0}^{n} \frac{|Y_{t+1} - \overline{Y_{t+1}}|}{|Y_{t+1}|}}{n} \qquad (10)$$

The model proposed in this paper seeks to maximize the prediction capabilities of the RNN by considering the long-term dependencies between daily seasonality, weekly periodicity and monthly seasonality. The numerical results of the proposed structure will be presented in the next section together with a comparison between the proposed RNN and the other two methods, i.e. ARIMA and AdaBoost.

4. Experimental Results

This section presents the numerical results that are obtained during the experiments, with various forms of RNNs, while also emphasizing the characteristics of each RNN. The experimental results of two other methodologies, ARIMA and AdaBoost regressor, which are trained on the same data set as the RNN, are also described in this section to evaluate the proposed methodology. The training of the RNN, the simulation of the results and the graphical plotting of the predicted values of traffic are implemented using Python programming language.

A. Prepossessing

Before getting into the results of the prediction, it is important to explain the concept of dictionary which represents the entire vocabulary or the entire set of symbols, that is known by the RNN after the training process. The number of individual symbols that are recognizable by the neural network is an important parameter that has to be considered when designing the model. One of the key criteria of choosing the dictionary size (L) is based on how much training data is available to the RNN. If the training data that is going to be fed into the network is quite large and has multiple duplicates of the same symbols, then the vocabulary set can also be left large without adjusting the symbols to a more favourable intermediate value.

One of the methodologies to reduce the size of the dictionary is to replace the real values of the traffic, with approximate values that are the closest to them. This can be achieved by generating equally spaced values in a predetermined range using a step value of L. All the elements of this sequence will have the following format: L, $2 * L$, $3 * L$, $\ldots$, $n * L$ and they represent a multiplicative set. In order to approximate one of the real values with an element that is present inside the multiplicative set, the shortest Euclidean distance must be calculated and the minimum value from that set must be selected. The format of the operation is:

replace_value = min(|real value – multicaplative_set|) (11)

By using this method all the values that occurred uniquely in the database are replaced with approximated values that occur more often. In this way the RNN is more capable of understanding the real patterns that are present inside the training data set and ignore the noisy information that is not important for the training model.

B. The Effect of modifying the dictionary size L

Case1 - L=0: In this case, the mean traffic value is predicted instead of individual traffic value and the parameter L is considered 0, so the dictionary of the RNN is not modified. By putting L = 0, the RNN will not distinguish between the real patterns and the noise that is present inside the data. The other parameters that describe the RNN are as follows:

Number−of−neurons = 150, Number−of−epochs =1000,Batch−size=128,Number−of−samples−for−training = 300. After the data passes the LSTM cells layer, two more layers are added which consists in a rectified linear unit layer (RELU) and a SoftMax layer. The loss function used for training purposes is categorical cross-entropy and it is usually used in the context of comparing two discrete probability distributions with one another and it is well suited for scenarios where classification is required to achieve.

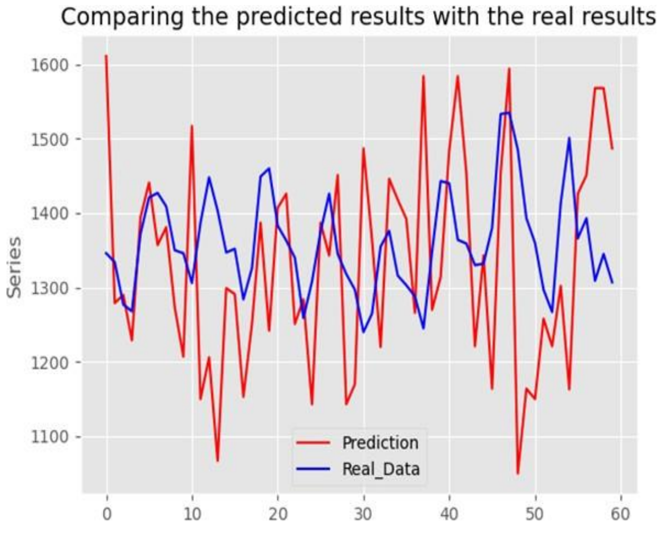

Figure 3. Prediction of mean traffic for 60 days with L=0

As it can be seen in Fig. 3, the RNN is not capable of accurately predicting the future 60 samples. The output is noisy and it doesn't capture the true nature of the training data samples.

Case2 - L=10: In this second case, we examine the same situation in which the mean traffic value is predicted instead of individual traffic value, but the L values is 10. By putting L = 10, the RNN will distinguish between the real patterns and the noise and the prediction becomes more accurate. All the other parameters remain the same, including the two more layers that are added at the end.

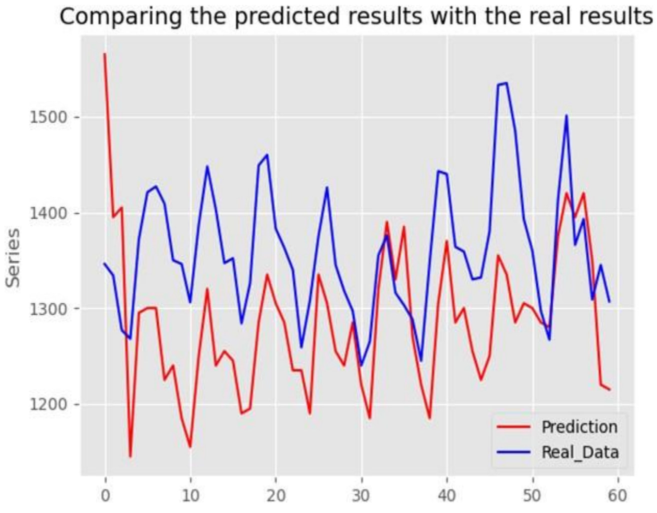

Figure 4. Prediction of mean traffic for 60 days with L=10

As it can be seen in Fig. 4, the RNN is capable of predicting the general trend of the next 60 data points and it can distinguish between relevant data and noisy data, but it is not capable of predicting the exact value without some significant error.

Case3 - L=25: The last case examines the same situation, but the L value is 25. All the other parameters remain the same, including the two more layers that are added at the end.

By increasing the value of L to 25, all the small variations that are present in the training data set, shall be approximated with the value of 1200, while only a small set of them are approximated with 1250. Because of this reduced dictionary, the RNN is not capable of predicting the small variations present inside the data sequence. As it can be seen in Fig. 5 by using this type of training data the neural network does not get the general trend of the data given and it completely misses the mean value of the real traffic that is provided.

From cases 1, 2 and 3 we can observe how important preprocessing of the data is for accurate predictions and how the size of the dictionary can influence how the data is interpreted by the neural network. By increasing the parameter L to large values, the small variations inside the data are lost, but if L it is set to 0 all the noise inside the data remains and the prediction becomes chaotic and unpredictable. That is why it is important to adjust the L parameter according to the data given and the small variations present inside the training data.

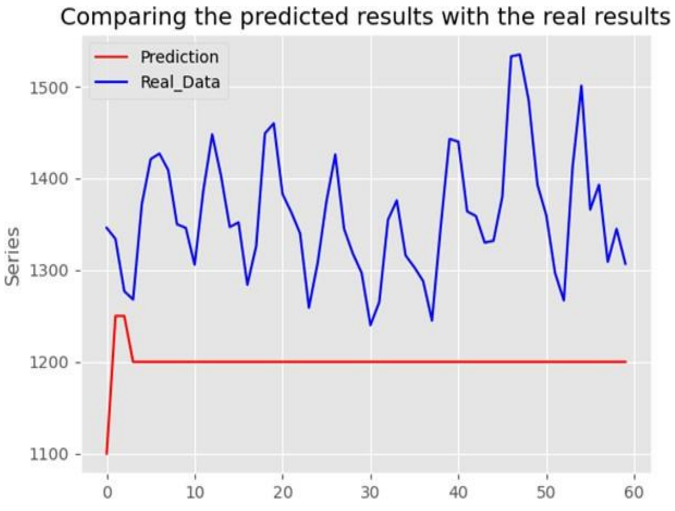

Figure 5. Prediction of mean traffic for 60 days with L=25

C. The Effect of modifying the learning rate and β factor of the Adam optimizer

The learning rate is a hyper-parameter that controls how much the gradient will adjust, based on the estimated error and how much the weights are changed. Choosing the learning rate can be a difficult process, because a learning rate that is too small can result in long periods of training and sometimes the optimization process might get stuck in a local minimum value [28]. If the learning rate is too large, the neural network might become unstable during the training process and it may result in sub-optimal results. The learning rate is usually defined as a number between 0 and 1, but the default value that is used in Python programming language is 0.001. There are also optimizers such as Adam, that have adaptive learning rates based either on the momentum value, or by changing the learning rate automatically based on the number of epochs and the loss function. The other parameter β is also called the exponential decay rate for the first moment estimate and the default value is 0.9. The purpose of this β is to control the exponential decaying rates of the averages that describe the gradient descent function [29], [30].

In the following tested scenarios the other variables that describe the neural network are also modified from the previous studied case, such as: the number of neurons used by the recurrent neural network and the number of epochs in which the training takes place. Throughout this case, the number of neurons and epochs will be constant and only the learning rate and β will be changed in order to examine the effects of these parameters.

Case1 - Learning−Rate=0.0025 and β = 0.99: In this case we predict mean traffic value instead of individual traffic value and the parameter L is considered 10, while the default learning rete is changed from 0.001 to 0.0025 and the β is changed to 0.99. The other parameters are configured as follows: *Number−of−neurons = 350, Number−of−epochs=1500,Batch−size=110,Number−of−samples−for−training = 300, Learning−rate = 0.0025.* There is no RELU function added after the LSTM cells, but the SoftMax is still kept. The loss function used for training purposes in this scenario is categorical cross-entropy which was also used in the previous case to make the required classifications.

After the prediction is complete the loss of the categorical cross-entropy function goes down from loss = 5.1 to loss = 0.6. The value of MSE = 15698.2 and the value of the mean absolute error is MAE = 98.1.

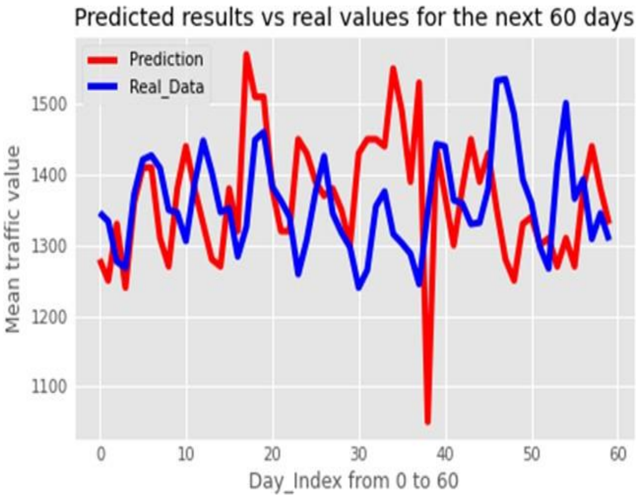

Figure 6. Prediction of mean traffic for learning rate LR=0.0025 and β=0.99.

Case2 - Learning-Rate=0.04 and β = 0.9999: In this second scenario we predict the mean traffic value, in a similar way with case1 by keeping all the parameters the same, except*Learning-rate* = 0.04 and β = 0.9999.

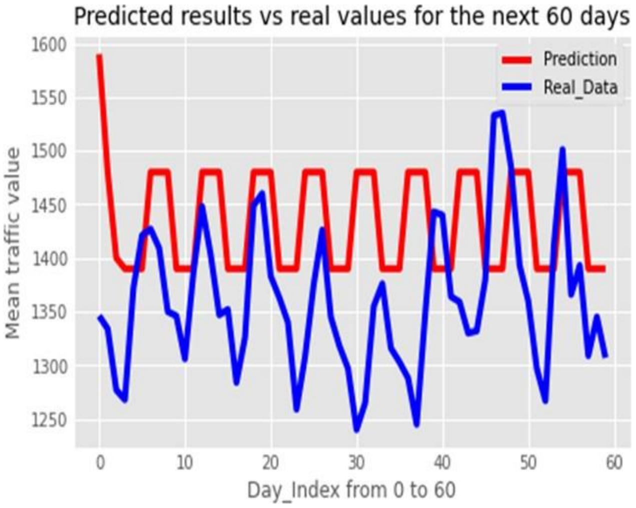

Figure 7. Prediction of mean traffic for learning rate LR=0.04 and β=0.9999.

The loss of the categorical cross-entropy function goes down from loss = 5.1 to loss = 3.46after training process is complete, which represents the worst performance out of all the scenarios which were examined. The MSE = 17824 and MAE = 97.2. We can see an improvement in terms of MAE compared to case1, but because of the large learning rate, the Adam optimizer is not capable of reaching a point of minimum and the algorithm will keep fluctuating.

D. RNN and ARIMA prediction results

Comparison between the RNN model and the ARIMA model - 300 samples for training:

The RNN model considered in this case, is the one which includes the weekly periodicity butnot monthly periodicity and has 500 neurons and the SoftMax layer at the end. The batch size inthis case is 128, the number of epochs is 1250 and the learning rate is 0.0016. The error of thepredictors are as follows: $MSE\text{-}RNN = 14728$, $MAE\text{-}RNN = 94$, $NMSE\text{-}RNN = 177$, while the errors for the $ARIMA$ are $MSE\text{-}ARIMA = 20997$, $MAE\text{-}ARIMA = 104$ and $NMSE\text{-}ARIMA = 199.2$. The RNN manages to capture the general trend of the real data better than ARIMA in terms of MSE and MAE, but the predicted values of RNN have asmaller dispersion than in the case of ARIMA, so the $NMSE\text{-}ARIMA$ is larger than the$NMSE\text{-}RNN$. The variables (p,d,q) chosen for the ARIMA in this case are (8,0,1) so the ARIMA has the auto regressive component equal to 8, the integration component equal to 0 and the moving average component equal to 1.

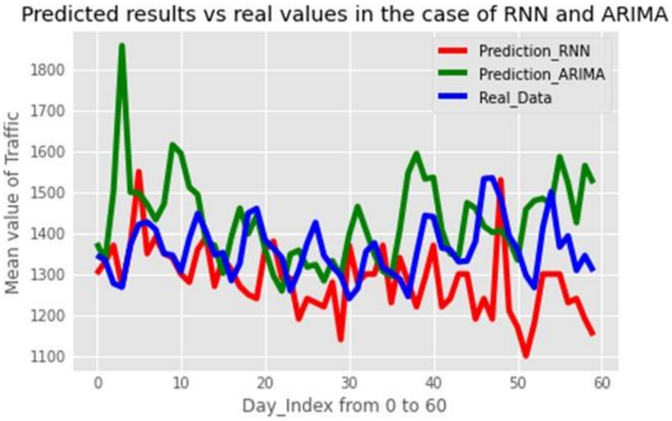

Figure 8. RNN predictor and ARIMA predictor for 300 training samples.

Comparison between the RNN model and the ARIMA model - 400 samples for training:

The RNN structure used in this case is identical with the one from the previous comparison, butin this scenario the next 60 points that have to be predicted are different from the training data setthat was fed initially. The learning rate is kept to 0.0001 and the number of epochs is 1250. The results of the error terms such as the mean squared error, mean absolute error and normalized mean squared error have the following values: $MSE\text{-}RNN = 59689$, $MAE\text{-}RNN = 179.38$ and $NMSE\text{-}RNN = 284.26$, while the $MSE\text{-}ARIMA = 196221.6$, $MAE\text{-}ARIMA = 343$, $NMSE\text{-}ARIMA = 576$. In this case the RNN manages to reduce all the error parameters byalmost half compared with the results achieved with ARIMA.

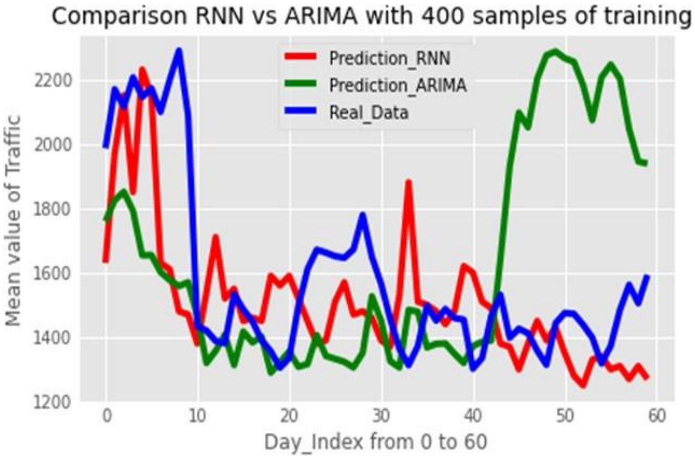

Figure 9. RNN predictor and ARIMA predictor for 400 training samples.

E. RNN and AdaBoost prediction results

Comparison between the RNN model and the AdaBoost model - 300 samples for training:

In this scenario the RNN model considered is the same one used in the case of ARIMA,and it includes the weekly periodicity but not monthly periodicity and has 500 neurons and the SoftMax layer.

The AdaBoost in this case has 10000 decision trees and a learning rate of 0.01, while the RNN has 500 neurons, batch size = 128, epochs = 1250 and a learning rate of 0.0016. As it can be seen in Fig. 10, the AdaBoost outperforms the RNN model, by capturing the general trend inside the data more accurately, but this can also be seen in the error terms where AdaBoost manages to score very small error values: MSE−RNN = 14728, MAE−RNN = 94, and NMSE−RNN = 177, while MSE−AdaBoost = 1943, MAE−AdaBoost = 30 and NMSE−AdaBoost = 36. The reason for which the AdaBoost manages to score better than the RNN, is because the AdaBoost is well suited for scenarios when data presents some sort of periodicity inside.

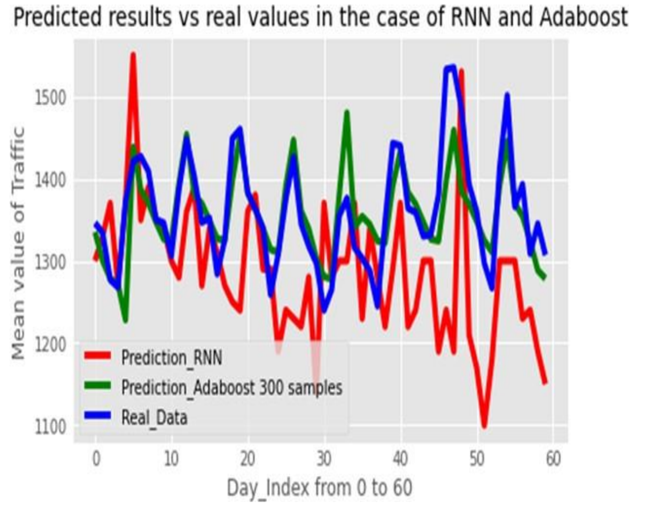

Figure 10. RNN predictor and AdaBoost predictor for 300 training samples.

Comparison between the RNN model and the AdaBoost model - 400 samples for training:

This second scenario uses the same models for RNN and AdaBoost as in the previous case, but this time 400 samples that do not present stationarity or periodicity are fed into both models.

In this second case (see Fig. 11) the RNN outperforms the AdaBoost and this can also be seen in the error terms: MSE−RNN = 59689, MAE−RNN = 179, NMSE−RNN = 284.26 while the error terms of AdaBoost are MSE−AdaBoost = 116545, MAE−AdaBoost = 221, NMSE−AdaBoost = 1837. Because the testing data presents a high level of non-stationarity the AdaBoost cannot converge to a small value of error and the classifications made by the weak learners cannot classify the future traffic accurately.

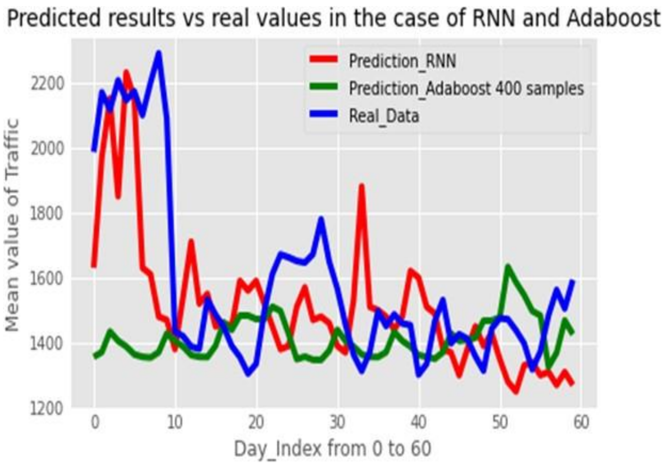

Figure 11. RNN predictor and AdaBoost predictor for 400 training samples.

In this second case (see Fig. 11) the RNN outperforms the AdaBoost and this can also be seen in the error terms: MSE−RNN = 59689, MAE−RNN = 179, NMSE−RNN = 284.26 while the error terms of AdaBoost are MSE−AdaBoost = 116545, MAE−AdaBoost = 221, NMSE−AdaBoost = 1837.

Because the testing data presents a high level of non-stationarity the AdaBoost cannot converge to a small value of error and the classifications made by the weak learners cannot classify the future traffic accurately.

F. RNN, ARIMA and AdaBoost prediction results for individual flows

Comparison between the RNN model and the ARIMA model for individual flows - 300 samples for training:

In this scenario unlike the previous ones, we predict the amount of times a particular webpage was accessed during a time span of 60 days, by feeding the training data only from a single flow. In the previous scenarios which were analysed, the mean value of traffic was examined without considering a particular destination website. The ARIMA model considered in this case, has the following values for (p, d, q): (4, 0, 1), but similar results are obtained with (8, 0, 1). The RNN model used in this scenario is similarly set with 500 neurons, a learning rate of 0.0016 and all the other parameters are the same.

The RNN model as it can be seen in Fig. 12 manages to capture the general trend more accurately than the ARIMA model, because the ARIMA totally overestimates the amount of traffic between the time interval of day 40 and day 50. The values of the error functions are: MSE−RNN = 1115, MAE−RNN = 13.2, NMSE−RNN = 84, while for the ARIMA MSE−ARIMA = 5862, MAE−ARIMA = 30, NMSE−ARIMA = 86. The error values in the case of

RNN can be seen that are smaller than the errors of ARIMA.

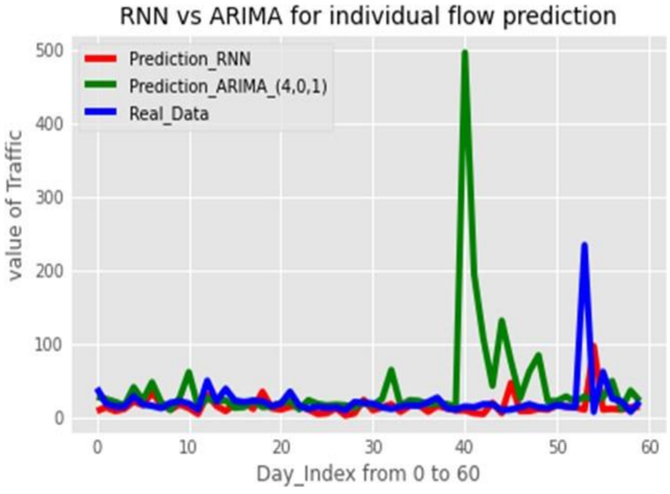

Figure 12. RNN predictor and AdaBoost predictor for 300 training samples-single flow.

Comparison between the RNN model and the AdaBoost model for individual flows - 300 samples for training:

The AdaBoost and RNN models are similar to those used as in the previous cases, but we use single flow traffic for training purposes instead of mean traffic value.

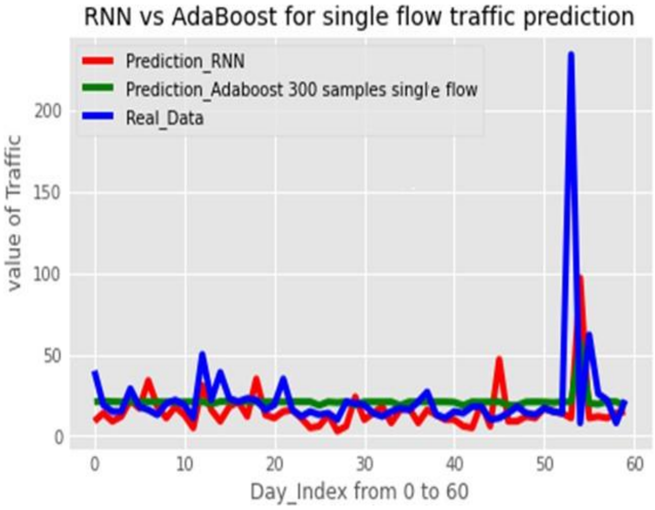

Figure 13. RNN predictor and AdaBoost predictor for 300 training samples single flow.

Even if the values of the MSE and MAE are smaller than in the case of RNN, the neural network manages to capture the small transitions more accurately than the AdaBoost, thing which can also be seen in NMSE criterion. We can see in Fig. 13 that the AdaBoost approximates all the small transitions with values in between (23 − 26) packets, while the RNN tries to capture even the small transitions. The values of the errors are as follows: AdaBoost−MSE = 886, AdaBoost−MAE = 10.7, AdaBoost−NMSE = 182, while the error values of RNN are: MSE−RNN = 1115, MAE−RNN = 13, NMSE−RNN = 84.

G. Comparison of the performance using MSE, MAE, NMSE and Relative error

This section takes all the MSE terms described in previous sections and compares the results.

Performance evaluation of RNN, ARIMA and AdaBoost models for mean traffic values 300 samples for training: The values of error previously obtained in the other subsections, indicate the capability of each model to predict future samples based on previous 300 days of training data.

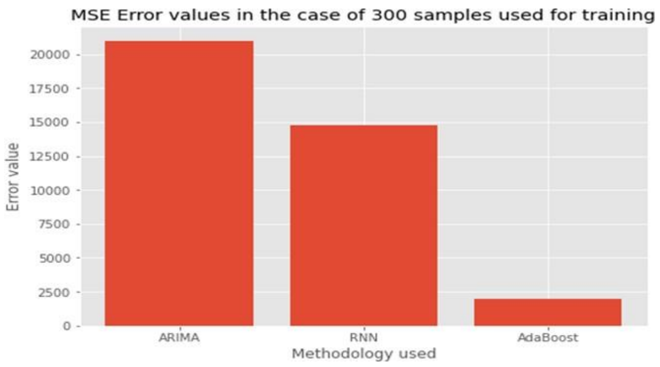

Figure 14. Comparing the MSE between RNN, ARIMA, and AdaBoost for 300 samples.

Figure 14 shows that AdaBoost has the smallest MSE and it has the most accurate prediction, when stationary samples must be predicted.

Performance evaluation of RNN, ARIMA and AdaBoost models for mean traffic values - 400 samples for training: This section evaluates the performance of the models, by using the results obtained in the previous subsections where 400 non-stationary data-points were used for training purposes.

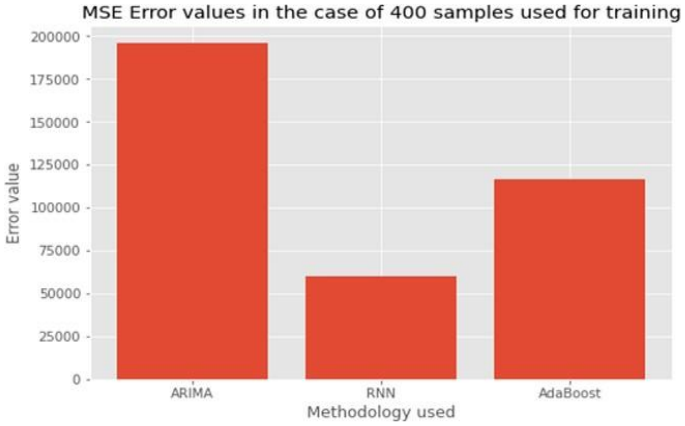

Figure 15. Comparing the MSE between RNN, ARIMA, and AdaBoost for 400 samples.

Performance evaluation of RNN, ARIMA and AdaBoost models for single flow traffic values 300 samples for training:
This last comparison takes the MSE values obtained for single flow prediction. As it can be seen in Fig. 16 the RNN and AdaBoost obtain similar error values,even if in Fig. 13, we can see that the RNN captures the small fluctuations more accurately. Both models capture the general trend quite well, but none of them was capable of predicting the spike which happened in day 52.

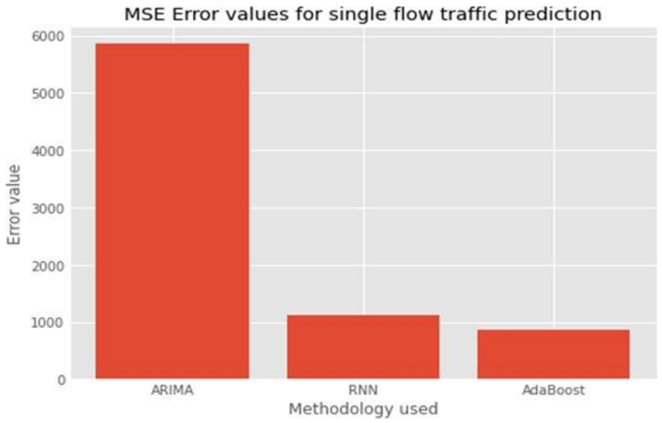

Figure 16. Comparing the MSE between RNN, ARIMA, and AdaBoost for single flow

Performance evaluation of RNN, ARIMA and AdaBoost models for relative error values when 300 samples are used for training:

This section is similar with the previous section, but it takes the relative error terms given by equation (10) and compares the results.

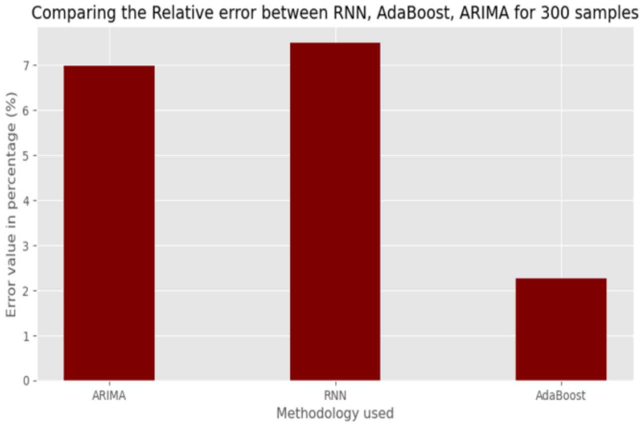

Figure 17. Comparing the Relative error between RNN, AdaBoost, ARIMA for 300 samples.

As shown in Fig. 17, the relative error values obtained for RNN, ARIMA and AdaBoost are: Relative_Error_RNN= 7.49, Relative_Error_ARIMA =6.98 and finally AdaBoost has Relative_Error_AdaBoost = 2.27.

Performance evaluation of RNN, ARIMA and AdaBoost models for relative error values when 400 samples are used for training:

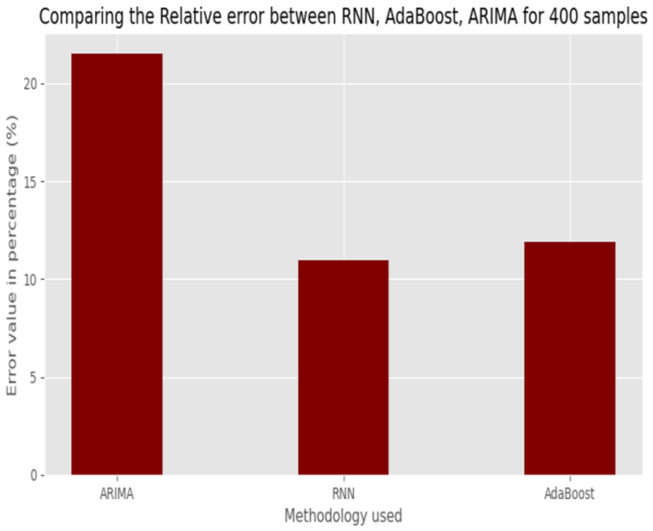

Figure 18. Comparing the Relative error between RNN, AdaBoost, ARIMA for 400 samples.

Figure 18 plots the relative error values obtained for RNN, ARIMA and AdaBoost as follows: Relative_Error_RNN= 10.96, Relative_Error_ARIMA = 21.48 and finally AdaBoost has Relative_Error_AdaBoost = 11.89. As it can be seen the recurrent neural network methodology performs best in this scenario in terms of relative error and it captures the general trend of the real data much more accurately, as it can also be seen in Fig. 11. The AdaBoost does not capture the initial peak traffic that happens between day 0 and day 10, even though the other 50 days are predicted reasonably well. Comparing between the scenario where 400 samples are given for training and the scenario where 300 samples are given for training, it can be seen that AdaBoost performs well when the data displays periodicity, but RNN performs very well even in scenarios where periodicity is not present inside the data.

Performance evaluation using Relative error of RNN, ARIMA and AdaBoost models for single flow traffic values using 300 samples of training:

Considering single flow traffic prediction, Fig. 19 shows the relative error values obtained for RNN, ARIMA and AdaBoost as follows: Relative_Error_RNN= 59.8, Relative_Error_ARIMA = 178.89 and finally AdaBoost has Relative_Error_AdaBoost = 44.38. As it can be seen the ARIMA model has a large error of 175 percent, but this error is mainly caused by the value predicted in day 40 as it can be seen in Fig. 12. Unlike ARIMA and RNN the AdaBoost performs better in terms of Relative error, but the AdaBoost does not capture the small variations in the data and it interprets all the days between 0 and 40 almost as a constant value, as it can be seen in Fig. 13.

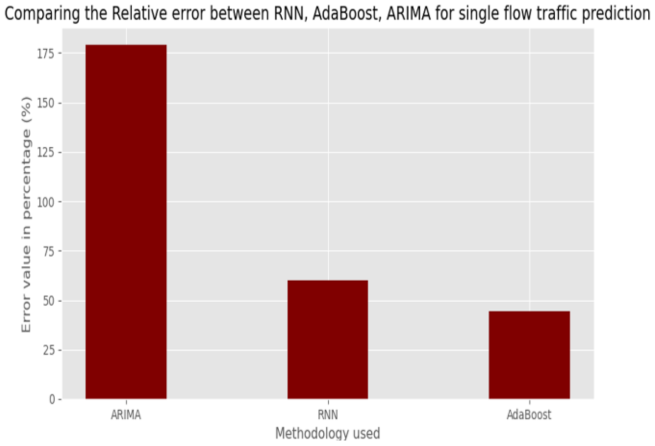

Figure 19. Comparing the Relative error between RNN, AdaBoost, ARIMA for 300 samples, single flow

Considering single flow traffic prediction, Fig. 19 shows the relative error values obtained for RNN, ARIMA and AdaBoost as follows: Relative_Error_RNN= 59.8, Relative_Error_ARIMA = 178.89 and finally AdaBoost has Relative_Error_AdaBoost = 44.38. As it can be seen the ARIMA model has a large error of 175 percent, but this error is mainly caused by the value predicted in day 40 as it can be seen in Fig. 12. Unlike ARIMA and RNN the AdaBoost performs better in terms of Relative error, but the AdaBoost does not capture the small variations in the data and it interprets all the days between 0 and 40 almost as a constant value, as it can be seen in Fig. 13.

5. Conclusions

Network traffic forecasting using artificial intelligence is a research area that is relevant for both current and future medium to largescale networks, because of the various advantages that offers, such as better capacity planning, improved quality of service and better intrusion detection systems.

In this research, the impact of various factors, such as approximating the training values that are learned by the RNN, modifying the intermediate layer functions between the LSTM cells andthe SoftMax output, introducing weekly periodicity and monthly periodicity into the training dataset, modifying the learning rate and exponential decay rate of the neural network to see how thepredictions are affected. A heuristic algorithm was proposed to create a recurrent neural network model that uses a sliding window algorithm in order to separate the training data samples that are fed into the network from the results that have to be achieved after the prediction is complete. The simulation results showed that the proposed scheme is capable of predicting future traffic values with good accuracy. In terms of error estimation values, the RNN minimizes the mean square error, mean absolute error and the normalized mean squared error better than the ARIMA model in both scenarios where 300 samples are used for training and also in the case when 400 samples are used for training. The second algorithm that was used as a comparison with the proposed model, is the AdaBoost regressor which is a robust algorithm capable of predicting the data that presents periodicity inside of it. The results obtained using AdaBoost regressor, when 300 samples were used for training purposes, achieved better error results than the proposed RNN model while also capturing the general trend more accurately. On the other hand, the results achieved when 400 samples were used for training, achieved far worst results than the proposed neural network.

6. References

[1] N. Ramakrishnan and T. Soni, ”Network Traffic Prediction Using Recurrent Neural Networks,” 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, 2018, pp. 187-193, doi: 10.1109/ICMLA.2018.00035.

[2] W. Wang et al., ”A network traffic flow prediction with deep learning approach for large-scale metropolitan area network,” NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, 2018, pp. 1-9, doi: 10.1109/NOMS.2018.8406252.

[3] S. Troia, R. Alvizu, Y. Zhou, G. Maier and A. Pattavina, ”Deep Learning-Based Traffic Prediction for Network Optimization,” 2018 20th International Conference on Transparent Optical Networks (ICTON), Bucharest, 2018, pp. 1-4, doi: 10.1109/ICTON.2018.8473978.

[4] T. Ding, A. AlEroud and G. Karabatis, ”Multi-granular aggregation of network flows for security analysis,” 2015 IEEE International Conference on Intelligence and Security Informatics (ISI), Baltimore, MD, 2015, pp. 173-175, doi: 10.1109/ISI.2015.7165965.

[5] G. Vormayr, T. Zseby and J. Fabini, ”Botnet Communication Patterns,” in IEEE Communications Surveys Tutorials, vol. 19, no. 4, pp. 2768-2796, Fourthquarter 2017, doi: 10.1109/COMST.2017.2749442.

[6] F. Shen, W. Zhang and P. Chang, ”An Engineering Approach to Prediction of Network Traffic Based on Time- Series Model,” 2009 International Joint Conference on Artificial Intelligence, Hainan Island, 2009, pp. 432-435, doi: 10.1109/JCAI.2009.104.

[7] D. H. Hagos, P. E. Engelstad, A. Yazidi and Ø. Kure, ”Recurrent Neural Network-Based Prediction of TCP Transmission States from Passive Measurements,” 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, 2018, pp. 1-10, doi: 10.1109/NCA.2018.8548064.

[8] H.Zare Moayedi and M. A. Masnadi-Shirazi *Arima model:”Arima model for network traffic prediction and anomaly detection”*. In *2008 International Symposium on Information Technology, Kuala Lumpur, 2008, pp. 1-6, doi: 10.1109/IT-SIM.2008.4631947*

[9] Y. Yu, J. Wang, M. Song and J. Song, ”Network Traffic Prediction and Result Analysis Based on Seasonal ARIMA and Correlation Coefficient,” 2010 International Conference on Intelligent System Design and Engineering Application, Changsha, 2010, pp. 980-983, doi: 10.1109/ISDEA.2010.335.

[10] T. H. H. Aldhyani and M. R. Joshi, ”Integration of time series models with soft clustering to enhance network traffic

forecasting,” 2016 Second International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), Kolkata, 2016, pp. 212-214, doi: 10.1109/ICRCICN.2016.7813658.

[11] Y. Song, M. Liu, S. Tang and X. Mao, ”Time series matrix factorization prediction of internet traffic matrices,” 37th Annual IEEE Conference on Local Computer Networks, Clearwater, FL, 2012, pp. 284-287, doi: 10.1109/LCN.2012.6423629.

[12] B. Yu, G. Graciani, A. Nascimento and J. Hu, ”Cost-adaptive Neural Networks for Peak Volume Prediction with EMM Filtering,” 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 4208-4213, doi: 10.1109/BigData47090.2019.9006188.

[13] Zhitang Chen, Jiayao Wen and Yanhui Geng, ”Predicting future traffic using Hidden Markov Models,” 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016, pp. 1-6, doi: 10.1109/ICNP.2016.7785328.

[14] Joao Paulo Coelho, Tatiana M. Pinho, Jose Boaventura-Cunha, “Hidden Markov Models, Theory and Implementation using MATLAB”, 2019 by Taylor Francis Group, LLC, CRC Press, Version Date: 20190401, International Standard Book Number-13: 978-0-367-20349-8 (Hardback).

[15] J. Rodrigues, A. Nogueira and P. Salvador, ”Improving the Traffic Prediction Capability of Neural Networks Using Sliding Window and Multi-task Learning Mechanisms,” 2010 2nd International Conference on Evolving Internet, Valcencia,2010, pp. 1-8, doi: 10.1109/INTERNET.2010.11.

[16] G. Feng, ”Network Traffic Prediction Based on Neural Network,” 2015 International Conference on Intelligent Transporta- tion, Big Data and Smart City, Halong Bay, 2015, pp. 527-530, doi: 10.1109/ICITBS.2015.136.

[17] Q. Zhuo, Q. Li, H. Yan and Y. Qi, ”Long short-term memory neural network for network traffic prediction,” 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, 2017, pp. 1-6, doi: 10.1109/ISKE.2017.8258815.

[18] T. Ko, S. M. Raza, D. T. Binh, M. Kim and H. Choo, ”Network Prediction with Traffic Gradient Classification using Convolutional Neural Networks,” 2020 14th International Conference on Ubiquitous Information Management and Communication (IMCOM), Taichung, Taiwan, 2020, pp. 1-4, doi: 10.1109/IMCOM48794.2020.9001712.

[19] X. Wang, C. Zhang and S. Zhang, ”Modified Elman neural network and its application to network traffic prediction,” 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, Hangzhou, 2012, pp. 629-633, doi: 10.1109/CCIS.2012.6664250.

[20] J. Skupa and J. Safarik, ”Survey of traffic prediction methods for dynamic routing in overlay networks,” 2017 IEEE 14th International Scientific Conference on Informatics, Poprad, 2017, pp. 339-343, doi: 10.1109/INFORMATICS.2017.8327271.

[21] Jason Brownlee, “Introduction to Time Series Forecasting with Python - How to Prepare Data and Develop Models to Predict the Future” https://machinelearningmastery.com/make-sample-forecasts-arima-python/, https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

[22] Jason Brownlee, “ XGBoost With Python”,” Gradient Boosted Trees with XGBoost and scikit-learn” Edition: v1.14 https://machinelearningmastery.com/adaboost-ensemble-in-python/

[23] S. Wu and H. Nagahashi, ”Parameterized AdaBoost: Introducing a Parameter to Speed Up the Training of Real AdaBoost,” in IEEE Signal Processing Letters, vol. 21, no. 6, pp. 687-691, June 2014, doi: 10.1109/LSP.2014.2313570.

[24] Simeon Kostadinov, “Recurrent Neural Networks with Python Quick Start Guide”, November 2018, Published by Packt Publishing Ltd., 35 Livery Street Birmingham

[25] Jason Brownlee, “Long Short-Term Memory Networks with Python – Develop Sequence Prediction Models With Deep Learning”, Copyright 2017 Jason Brownlee. All Rights Reserved, Edition: v1.0

[26] Dhruvil Shah, “Exploring the Next Word Predictor! – Different approaches for building the Next Word Predictor”, May 8 2020, https://towardsdatascience.com/exploring-the-next-word-predictor-5e22aeb85d8f

[27] https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43795

[28] Jason Brownlee, “Master Machine Learning Algorithms, Discover How They Work and Implement Them From Scratch”

[29] Diederik P. Kingma, Jimmy Lei Ba, ”ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION” Published as a conference paper at ICLR 2015 https://arxiv.org/pdf/1412.6980.pdf

[30] Jason Brownlee, ”Better Deep Learning - Train Faster, Reduce Overfitting, and Make Better Predictions”, https://machinelearningmastery.com/better-deep-learning/