ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume 27, Number 3–4, 2024, 336–347

Network Traffic Prediction Performance Using LSTM

Khirota YALDA[1, 2, *], Diyar Jamal HAMAD[1, 2], Nicolae TAPUS[1], and Ibrahim Taner OKUMUS[3]

[1]Dept. Computer Science and Engineering, University POLITEHNICA of Bucharest, Romania
[2]Dept. IT, Erbil Polytechnic University, Iraq,
[3]Dept. Computer Engineering, University Kahramanmaraş Sütçü İmam, Turkey
Email: Kherota.yalda@epu.edu.iq*, diyar.hamad@epu.edu.iq, nicolae.tapus@upb.ro, iokumus@ksu.edu.tr
* Corresponding author

Abstract. As networks expand to support various applications involving text, audio, video, and images, data traffic increases correspondingly. Traffic classification, which identifies the origin of observed traffic, has multiple applications, including dynamic bandwidth allocation, traffic analysis, quality of service, and network security. Traditional network traffic classification methods like deep packet inspection rely on manually creating and maintaining communication profiles for various applications. However, these methods face challenges such as dynamic port changes and encrypted traffic. Machine Learning (ML) classifiers offer effective solutions to these issues, providing accurate network traffic classification. Due to these advancements, deep learning models are now utilized for network traffic classification and prediction. Long Short-Term Memory (LSTM) has emerged as a highly effective deep learning technique for addressing time series prediction challenges. This study aims to analyze the performance of forecasting network traffic using LSTM, with different activation functions and optimizers with Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Coefficient of Determination (R-Squared) parameters as the model evaluation index. This demonstrates how these parameters impact network traffic forecasting performance.

Key-words: Activation function; LSTM; optimization algorithms; traffic prediction.

1. Introduction

Because of the growth in the number of connected devices, and a rise in bandwidth consumption as data quality and quantity increase [1], internet data traffic has significantly exploded. The aforementioned network's resources are crucial, and they must be used for the planned aim.

Understanding the traffic capacity aids in assessing the network's performance and in attaining network optimization [2]. Traffic prediction is essential for network management as it allows administrators to anticipate and prepare for future demands on the network. By analyzing historical data and current trends, network operators can forecast traffic patterns, enabling them to allocate resources effectively, optimize performance, and ensure a consistent user experience. Additionally, traffic prediction plays a crucial role in maintaining Quality of Service levels, detecting network faults or security threats, and meeting service level agreements with customers [3]. It is feasible to make both long-term and short-term predictions. Long-term prediction offers an accurate anticipation of traffic models for assessing upcoming volume demands and enables extra minute arrangement and stronger perceptions. Short-term prediction, which operates on a timescale ranging from milliseconds to minutes, is closely linked to dynamic resource allocation. This approach finds application in tasks such as packet routing, improving QoS procedures, efficiently managing resources and reducing traffic [4]. Traditional methods of traffic prediction typically rely on historical data analysis to forecast traffic patterns. These methods often use techniques like time series analysis, regression analysis, and mathematical modeling to forecast traffic flow depending on factors like the time of day, day of week, and historical traffic patterns. On the other hand, algorithms used in machine learning (ML) techniques are able to recognize patterns and relationships in data without the need for explicit programming. ML models for traffic prediction can handle more complex and dynamic datasets, incorporate a wider range of variables, and adapt to changing conditions more effectively compared to traditional methods. ML methods also have the potential to provide more accurate predictions by continually gaining knowledge from fresh data and making necessary model adjustments [5]. Deep Learning (DL), is a used in the big data analytics process [6]. Multiple DL algorithms, including Convolutional Neural Networks (CNNs), Recurrent Neural Network (RNN), LSTM, Gated Recurrent Units (GRUs), and Prophet can be utilized for forecasting time series data. Many studies in the literature show that LSTM outperforms other DL approaches in time series forecasting. This study aims to determine how parameters like activation functions and optimizers affect LSTM's network traffic prediction performance.

2. Related Work

Network management, resource allocation, QoS and cyber security protection are the key applications of network traffic prediction. Numerous machine learning-based algorithms for forecasting network traffic have emerged and have been published in the literature [2]. Network traffic analysis is thoroughly examined in the work by Conti et al. in [6]. They looked at a number of algorithms, including k-means, Naive Bayes, and Random Forest. The work examines analysis approaches, validation strategies, and outcomes with a focus on mobile devices. Hua et al. [7] investigated the results of changing the neural network's structure so that neurons' connectivity is random in place of fully linked. The authors concluded that the LSTM's stochastic connectivity can lower computing costs to some scale while retaining acceptable prediction correctness. Additionally, Aloraifan et al. [2] employed bidirectional Long Short-Term Memory (Bi-LSTM) and bidirectional Gated Recurrent Unit (Bi-GRU) models for predicting the network traffic matrix. The comparisons of the suggested techniques were conducted using actual GEANT network traffic statistics. The findings demonstrated that, when compared to other current models, the suggested models offer a significant advance in prediction accuracy. A machine learning-based traffic flow forecasting system for the US city of Bloomington is shown

in [8]. Better traffic prediction is offered by the LSTM algorithm with a least root means square error value. In [9], ARIMA outperformed LSTM in predicting base station traffic using RMSE as the evaluation metric. Despite extensive research, accurately predicting and estimating traffic remains challenging, necessitating effective techniques for handling large datasets.

3. Long Short-Term Memory

Long Short-Term Memory (LSTM) is one of the various types of Recurrent Neural Networks (RNNs), and it can retain and use historical data for forecasting purposes [10]. LSTM is incorporated to tackle the challenges of vanishing and exploding gradients, which are common with simple RNNs. It is well-suited for capturing long-term dependencies inherent in network traffic, as it utilizes chained memory blocks to maintain short-term memory and tends to recall prior actions made in time steps [2]. The memorization of previous stages in an LSTM can be done by including gates along the memory line. Each LSTM node consists of cells that store data streams, with the upper line in each cell serving as a transport line that moves data from the past to the present. Each memory block includes a memory cell and three gates: an input gate (i_t), an output gate (o_t), and a forget gate (f_t). The sigmoidal neural network layers in the gates control data flow, with binary values (0 or 1) indicating whether to let data pass. The forget gate determines which information to discard from the cell state, producing a value between 0 (ignore) and 1 (keep). The input gate selects new data to retain, updating the cell state with new inputs via the $tanh$ layer. Finally, the output gate decides which part of the cell state to output, generating the final outcome value ht. LSTM units can discard irrelevant past information and incorporate valuable new data, effectively capturing temporal aspects in network traffic matrix patterns. The LSTM can be represented by the following equations:

$$\begin{aligned} f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \\ h_t &= o_t \tanh(c_t) \end{aligned} \tag{1}$$

where f_t is the forget gate activation vector, i_t is the input gate activation vector, c_t is the cell state vector, o_t is the output gate activation vector, and h_t is the hidden state vector, at time step t; b_f, b_i, b_c, b_o are the bias vectors for the forget gate, input gate, cell state, and output gate, respectively, W_{xf}, W_{xi}, W_{xc}, W_{xo}, W_{hf}, W_{hi}, W_{hc}, W_{ho}, W_{cf}, W_{ci} and W_{co} are the weights matrices for input, hidden state, and cell state interactions, and σ is the sigmoid activation function.

4. Activation Functions

One of the main elements influencing the networks' performance is their activation function [11]. They are primarily utilized in artificial neural networks to transform input signals into output signals that are fed into the next layer in the stack. The accuracy of a neural network's predictions is determined by the number of layers and, more crucially, the kind of activation function employed. Although a thumb rule indicates that at least two layers should be utilized,

there is no reference that specifies the minimum or maximum number of layers to be used for improved outcomes and accuracy of the neural networks. Additionally, there is no mention of the specific type of activation function that should be used in the literature [12]. The activation functions that are going to be explained and used in this paper are: Sigmoid, Tanh, ReLU.

4.1. Sigmoid activation function

One of the most popular activation functions in artificial neural networks is the sigmoid function, which is a non-linear function:

$$f\left(x_i\right) = \frac{1}{1 + e^{-x_i}} \tag{2}$$

where x_i is the activation function's input.

The sigmoid function is differentiable, continuous, and bounded in intervals (0,1). Additionally, the sigmoid function has several significant flaws. The range of (0,1) is bound for the sigmoid function. Therefore, the output is always non-negative. Because of this, the activation function is not zero-centered. A wide range of input is bound to a narrow range of values (0,1) by the sigmoid function. Hence, a significant alteration in the input value results in a minor alteration in the output value. Small values are the outcome of this as well. It has the vanishing gradient problem as a result of having small gradient values. The tanh function was created to combine the zero-centered nature and advantages of the sigmoid function [13].

4.2. Tanh activation function

The $\tanh$ function, also known as the Tangent Hyperbolic Function, is a variation of the σ Function. While both functions are nonlinear, the $\tanh$ function differs by being zero-centered, mapping real-valued numbers into the range $[-1, 1]$. Despite having a stronger gradient than the σ, the $\tanh$ function also faces the issue of "vanishing gradients." The $[-1, 1]$ output range leads to a mean of outputs near zero, which helps concentrate the data and facilitate learning. The general expression of this activation function is

$$\tanh\left(x_i\right) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} \tag{3}$$

where e is the base of the natural algorithm, and x_i is the input to the function.

4.3. ReLU activation function

Neural networks commonly use the nonlinear activation function known as the rectified liner unit or ReLU [12]. The "expansion and disappearance" issue with the σ and $\tanh$ functions is resolved by the application of the ReLU activation function:

$$ReLU\left(x\right) = \ x^{+} = max\left(0, x\right) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The formula (4) states that if the input x is positive, the output equals x; otherwise, it is zero [12].

The ReLU function offers the advantage of non-simultaneous stimulation of neurons, leading to more efficient performance compared to other functions by avoiding the activation of all neurons at once. This can lead to more efficient computation and memory usage since many neurons remain inactive during the forward pass and result in faster training times [12]. Despite its benefits, there are some drawbacks worth considering. Zero is produced as the output of the ReLU function for any input that is less than zero. Consequently, the input components with negative weighted sums are unable to contribute to the process as a whole. ReLU can thereby deactivate a significant portion of a network, rendering it fragile. The neurons that ReLU deactivates are referred to as dead neurons, and this issue is also known as the dying ReLU problem [13].

5. Optimization Algorithms for LSTM Model

Optimization algorithms are the foundation for a machine's capacity for observational learning [13]. They attempt to minimize the loss function by calculating gradients. Learning can be used with different kinds of optimization algorithms in different contexts [14].

5.1. Adaptive moment estimation (Adam)

Adam maintains a constant learning rate for all weight updates throughout training while using adaptive rates for each parameter. It balances the learning rate for each weight by estimating the first and second moments of the gradient, combining features of SGD and RMSprop. The algorithm requires the first and second moment variables, m and v. At each time step t, the biased estimates for these moments are updated following gradient computation [15]:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \\ \widehat{m_t} &\leftarrow \frac{m_t}{1-\beta_1^t} \\ \widehat{v_t} &\leftarrow \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \in} \end{aligned} \tag{5}$$

The gradient at a given time t is represented by g_t, and its first and second moments are mt and v_t, respectively. The hyperparameters β_1 and β_2 regulate the decay rates of the moment estimates. The learning rate is α, and the tiny constant epsilon is used to prevent division by zero. One of its considerable advantages lies in terms of training speed.

5.2. Adaptive gradient algorithm (AdaGrad)

Throughout the learning phase, the AdaGrad algorithm suggests using previous data to modify the learning rate for every parameter. Upgrading the algorithm's convergence and forecast correctness is the aim of this adaptation. One drawback of the AdaGrad algorithm is that it slows down learning when the gradients' total of squares increases. The AdaGrad algorithm is defined by the following equation [16]:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{6}$$

where $\theta_{t,i}$ is the value of parameter i at time step t, $g_{t,i}$ is the objective function's gradient with respect to parameter i at time step t, $G_{t,ii}$ is the sum of the squares of the gradients with respect to parameter i up to time step t, η is the learning rate (a hyperparameter), and ϵ is a small constant (usually added for numerical stability to avoid division by zero) [15].

5.3. Stochastic Gradient Descent (SGD) algorithm

An SGD algorithm optimizes large-scale deep learning models efficiently by using random samples instead of the entire dataset for each cycle. The term "stochastic" refers to this random sampling approach. After each training phase, SGD updates the network structure to find the global minimum. It approximates the gradient for a randomly selected batch, reducing error. This involves repeatedly passing through batches and randomly shuffling the dataset [17]:

$$\theta_{t+1} = \theta_t - \alpha g_t \tag{7}$$

where θ represents the model's parameters (weights), and α stands for the learning rate, a small positive constant. At each iteration t, one instance (x_t, y_t) is randomly selected from the dataset. The gradient g_t of the loss with respect to the parameters is computed using this example. The negative sign in the update rule directs adjustments opposite to the gradient's direction, aimed at minimizing the loss function. This cycle continues until the parameters converge to an optimal value (where the gradient approaches zero) or for a predetermined number of iterations.

The optimization problem considered in this paper involves training the LSTM model to minimize the error between predicted and actual network traffic values. This is achieved by minimizing a loss function that quantifies the difference between the model's predictions and the actual observations in the training dataset. To accomplish this, various optimization algorithms were employed to iteratively adjust the model parameters, with the goal of reducing prediction error and enhancing model accuracy and performance. Specifically, the LSTM model for network traffic prediction was trained using the default settings of Adam, AdaGrad, and SGD optimizers in TensorFlow. These optimizers aimed to minimize loss functions such as MAE, MAPE, MSE, RMSE, and R-squared. Adam, with its adaptive learning rates and momentum, facilitated faster convergence and superior performance, especially with the ReLU activation function, leading to lower error metrics. AdaGrad, which adjusts learning rates based on historical gradients, was effective for sparse data but exhibited slower convergence. SGD, while simple, updated parameters based on the gradient of the loss and showed slower convergence. The results highlight Adam's superior performance due to its effective handling of sparse gradients and bias correction, making it a robust choice for predictive modeling tasks. Consistent random seeding across multiple runs ensured result stability, further demonstrating Adam's reliability compared to the more variable performance of Sigmoid-AdaGrad combinations.

6. Execution and Results

One of the most significant problems in networks is traffic prediction. The development of computing technologies has made it possible to forecast traffic flow and congestion. This study

predicts traffic flow using LSTM machine learning methods. The metrics MAE, MAPE, MSE, RMSE, and R-Squared are used to assess the suggested system's performance.

6.1. Dataset

The internet traffic dataset (in bits) from an ISP (`https://www.kaggle.com/datasets/yjimmy/network-traffic-dataset/`) is used for forecasting and analysis. The data set contains of the Traffic aggregated in the academic network backbone of the United Kingdom.This study will use two normalized data columns—traffic and time—to forecast traffic. Data normalization, a preprocessing method, scales data to ensure each feature contributes equally, often projecting values into a range like [0,1]. Normalization is crucial when variables have significantly different ranges [18].

6.2. The tools used

The statistical measures used to assess each model's network traffic forecasting are shown below.

Mean Absolute Error (MAE): Mean absolute error (MAE) quantifies the average magnitude of differences between real data and predictions, indicating the level of precision of a model [19]:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{8}$$

Mean Absolute Percentage Error (MAPE): MAPE assesses prediction accuracy by indicating the average percentage difference between predicted and actual values, regardless of whether the estimate was too high or too low. A MAPE of zero would imply perfect accuracy in predictions since the sum of positive and negative percentage errors equals zero. A forecast is considered reasonably accurate if its MAPE is below 5%. MAPE between 10% and 25% indicates poor but acceptable accuracy, while a MAPE exceeding 25% signifies extremely low accuracy, rendering the forecast unacceptable [20]:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - x_i|}{|y_i|}100\right) \tag{9}$$

This calculates the percentage error for each observation.

Mean Squared Error (MSE): This measure computes the squared average of the error between the actual and forecasted data. If the MSE value is low, the model's accuracy is higher [19]. The best value for MSE = 0 and the worst value = $+\infty$ [20]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{10}$$

Root mean square error (RMSE): One often used statistic to assess a predictive model's accuracy is the Root Mean Square Error (RMSE). It calculates the average error magnitude between the observed and anticipated values [19]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{11}$$

R-Squared: Another name for R-Squared is the coefficient of determination which is often denoted as R^2. The R-squared value ranges from 0 to 1, with a negative result indicating the model's inability to fit the data. A higher value closer to 1 signifies a better fit of the model [21]:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{\sum_{i=1}^{n}(y_i - \overline{x})^2} \tag{12}$$

To obtain the $\overline{x}$ the following equation can be used:

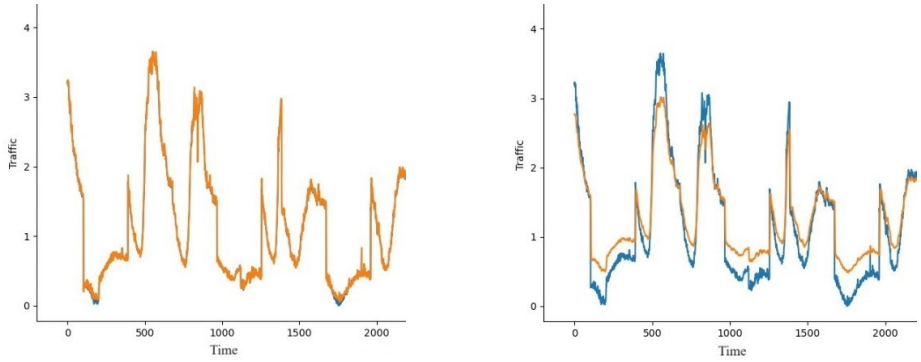
$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{13}$$

where n is the number of observations (data points); y_i is the actual (observed) value for the ith data point; x_i is the predicted value for the ith data point. $\overline{x}$ is the mean of the observed values.

6.3. Experiments

LSTM, a machine learning forecasting tool, is employed using Python. The Keras library, known for its ability to execute high-performance models, is utilized for machine learning predictions. These metrics, including MAE, MAPE, MSE, RMSE, and R-Squared, will be used to anticipate future traffic and evaluate the performance of the model. MAE, MSE, and RMSE are quantified using identical units as the data, typically referred to as 'bit.' On the other hand, MAPE measures in percentage, and R^2 is a dimensionless measure, not associated with any specific unit of measurement. Python packages such as Keras, Pandas, TensorFlow, and Matplotlib are employed. A high-level library called Keras is built atop TensorFlow. Keras provides an API similar to scikit-learn for creating neural networks. TensorFlow, an open-source library primarily used for managing complicated calculations in deep learning, is employed. Deep learning networks are trained using TensorFlow. Pandas can be used for data wrangling and exploratory analysis [8]. Matplotlib is a comprehensive Python visualization toolbox for interactive graphics, static and animated. Import the required libraries, such as Keras and TensorFlow, in order to implement the LSTM algorithm. When using the σ or $\tanh$ activation functions, LSTM is sensitive to the input data's scale. Therefore, if the data is not already normalized, it's important to rescale it to a range between 0 and 1. The MinMaxScaler() function was used to scale the data to a range between 0 and 1. Then, divided the dataset into sets for testing and training. 80% of the data is used for training, while the remaining 20% is used for testing. Utilize the numpy.reshape() function to transform the input data for testing and training into a three-dimensional feature. In this representation, the first dimension represents the batch size, the second dimension represents the time-steps, and the third dimension represents the number of units in a single input sequence. A few modules from Keras are imported to build the LSTM model. The Sequential module initializes the neural network, while the Dense module adds a layer of neural networks with dense connections. The LSTM layer is configured with an output space dimensionality of 64, 32, and 16 units, and the input_shape parameter is set to the shape of the training set. A comprehensive investigation evaluated the performance of Long Short-Term Memory (LSTM) networks under various activation functions (ReLU, Tanh, Sigmoid) and optimization algorithms (Adam, AdaGrad, SGD). This resulted in nine distinct experimental configurations. The LSTM model utilized in this work consists of three LSTM hidden layers and one dense output layer. The model runs with a batch size of 1 and for 100 epochs and uses 32465 trainable parameters. The constructed model was evaluated using the testing data. The matplotlib package is used to

plot the results. Time is represented on the X-axis, while Traffic is represented on the Y-axis. The graphs show how well the model's predictions align with the actual values, providing a visual representation of the LSTM model's performance.



(a) ReLU-Adam (b) Sigmoid-AdaGrad

Fig. 1. Representation of True Values (red colour) vs Predictions - Test Set (blue colour).

Fig. 1 reflects the outcomes of the best and worst scenarios of the True Values vs Predictions of the Test Set. The results comparing the three types of activation functions and the three optimizers, considering various network sizes in each layer of the LSTM for the tested values in this paper, are illustrated in Table 1 which provides quantitative measures of the model's prediction accuracy. It gives specific numerical values that indicate how much error there is between the predicted values and the actual values. From the results, it is evident that based on all metrics, the ReLU activation function with the Adam optimizer performs better. For MAE, MAPE, MSE, and RMSE, smaller values indicate better model performance, as they reflect smaller errors between predicted and actual values. For R^2, larger values are generally desirable because they indicate that the model provides a better fit to the data.

Table 1. The SPR and external angle for various parameters

Metrics	AdaGrad	ReLu Adam	SGD	AdaGrad	Sigmoid Adam	SGD	AdaGrad	Tanh Adam	SGD
MAE	0.07023	0.03448	0.04264	0.12701	0.04440	0.03783	0.06781	0.03603	0.04919
MAPE	15.96914	6.95164	8.66401	42.92128	10.32788	7.10015	13.88704	8.28768	11.4347
MSE	0.02003	0.00619	0.00754	0.02718	0.00719	0.00708	0.01918	0.00621	0.00987
RMSE	0.14154	0.07866	0.08686	0.16485	0.08480	0.08411	0.13849	0.07882	0.09935
R^2	0.97649	0.99274	0.99114	0.96810	0.99156	0.99170	0.97749	0.99271	0.98841

The initialization of algorithms typically involves setting initial values for various parameters that govern their behavior during training. These default values are commonly used in practice and are suitable for a wide range of applications. For example AdaGrad initializes the sum of squared gradients $(G_{t,ii})$ to zero for each parameter. In this implementation, the algorithms where initialized with default or commonly used values, to ensure they were suitable for optimizing the parameters of the LSTM network for traffic prediction. In this implementation, the default

settings for these algorithms were used as provided by popular machine learning libraries such as TensorFlow. Calling functions such as Adam(), AdaGrad(), and SGD() without specifying parameters results in the algorithms using their default values. Similarly, default activation functions were employed, integrated directly into the LSTM layers. When training neural networks, selecting appropriate parameter values significantly impacts model performance and convergence speed. For the Adam optimizer, a common starting learning rate (α) is 0.001, adjustable to 0.01 or 0.0001 if convergence is slow. Beta1 (β_1) and Beta2 (β_2), controlling moment estimates, are typically 0.9 and 0.999, respectively, with (ϵ) at 1e-8 to avoid division errors. AdaGrad uses a default learning rate (η) of 0.01, adjustable between 0.1 and 0.0001. Stochastic Gradient Descent (SGD) starts with a learning rate (α) of 0.01, with suggestions to explore 0.1, 0.001, or 0.0001 for optimal training. Fine-tuning the learning rate is crucial, while β_1, β_2, and ϵ generally remain stable. In the execution of LSTM models for network traffic prediction, the reliability of the results was ensured by repeating each model's execution multiple times. Employing a controlled environment with consistent random seeding aimed to mitigate any potential variability introduced by random initialization. Remarkably, across all experiments utilizing different activation functions paired with various optimizers, the results exhibited remarkable stability, with negligible changes observed between successive executions. However, it is noteworthy that the Sigmoid activation function combined with the AdaGrad optimizer yielded remarkable alternation in results with each execution. The observed instability in results can be attributed to several factors. First off, the vanishing gradient issue affects the Sigmoid activation function, especially in deep neural networks. This problem arises when the Sigmoid function saturates at high values, producing gradients that are almost zero. This can cause training to diverge or stagnate and prevent effective learning. Second, the AdaGrad optimizer modifies model parameter learning rates according to their past gradients, which may result in unnecessarily large updates for features that appear infrequently. The Adam, AdaGrad, and SGD optimizers each update model parameters differently, leading to varying prediction values. SGD (Stochastic Gradient Descent) adjusts each model parameter separately according to its own learning rate. AdaGrad, an optimization algorithm for adaptive learning rate adjustment, scales the learning rate for each parameter based on historical gradients, giving more weight to parameters with infrequent updates and larger updates to less frequently occurring parameters. Adam updates model parameters by adjusting the learning rate in accordance with both the size and direction of past updates. These distinct methods of updating parameters result in the model producing different prediction values.

6.4. Factors contributing to Adam's superior performance

Adam, AdaGrad, and SGD are three different optimization algorithms commonly used in training machine learning models. There are several elements that can affect an algorithm's efficacy, including the dataset's type, the model architecture, and the problem at hand. Here are some reasons why Adam may be considered more effective than AdaGrad and SGD in certain scenarios:

Adaptive Learning Rates: Adam incorporates adaptive learning rates for each parameter. It calculates individual learning rates for each parameter by considering both the past gradient and the second moment of the gradients. This adaptability can be beneficial when dealing with features that have different scales.

Momentum: Adam includes a momentum term that helps smooth out the parameter updates. This can be advantageous in scenarios where the optimization landscape is noisy or has frequent

changes in direction. The momentum term helps the optimizer to continue moving in the correct direction, even if individual gradients fluctuate.

Bias Correction: Adam performs bias correction for the first and second moments to mitigate the effects of initialization bias. This can contribute to more stable and accurate updates, especially in the early stages of training.

Effective Handling of Sparse Gradients: Adam uses the moving average of both the first and second moments, which can be more effective in handling sparse gradients. Because of Adam's quicker convergence and resilience to various difficulties, in general it performs better than other adaptive learning rate algorithms. As a result, deep learning primarily uses this approach as its default.

7. Conclusions

One of the main aspects that require monitoring is traffic. This paper focuses on traffic flow prediction using an internet traffic dataset (in bits) from an ISP. Three optimization algorithms were utilized to effectively train the LSTM model for forecasting traffic flow. The integration of these algorithms was carried out within a Python environment using Jupyter Notebook. These optimization algorithms played a crucial role in minimizing the loss function during the training process by iteratively adjusting the model parameters. The default settings for each optimization algorithm were employed. The LSTM models were trained for 100 epochs with a batch size of 1. To determine the most effective activation function for traffic flow prediction, the algorithm's activation functions are evaluated based on performance metrics. the experiments provide valuable insights into the effectiveness of different activation functions and optimization algorithms for LSTM networks. The results demonstrated high prediction accuracy and convergence for both training and testing datasets, validating the efficacy of the optimization algorithms in solving the network traffic prediction problem. The findings of the study illustrate that the ReLU activation function yields smaller metric values (0.034 MAE, 6.443 MAPE, 0.006 MSE, 0.079 RMSE and 0.992 R^2) when utilizing the Adam optimizer. This suggests that the ReLU activation function, and the Adam optimizer, outperforms others in terms of network traffic data prediction. It is noteworthy that even successful forecasting (90% accuracy or higher) only indicates general accuracy in predicting traffic, precise traffic volumes are not determined. As future work, there is an intention to broaden the exploration of additional machine learning approaches. However, one encountered issue was the lack of publicly available datasets for evaluating machine learning approaches.

References

[1] I. SALEH and H. JI, *Network traffic images: A deep learning approach to the challenge of Internet traffic classification*, Proceedings of the 2020 Annual Computing and Communication Workshop and Conference, Las Vegas, NV, USA, 2020, pp. 329–334.

[2] D. ALORAIFAN, I. AHMAD, and E. ALRASHED, *Deep learning based network traffic matrix prediction*, International Journal of Intelligent Networks **2**, 2021, pp. 46–56.

[3] D. ANDREOLETTI, S. TROIA, F. MUSUMECI, S. GIORDANO, G. MAIER, and M. TORNATORE, *Network traffic prediction based on diffusion convolutional recurrent neural networks*, Proceedings of 2019 IEEE Conference on Computer Communications Workshops, Paris, France, 2019, pp. 246–251.

[4] R. M. RAFEEQ, *Developing machine learning based framework for network traffic prediction*, Eurasian Journal of Engineering and Technology, **4**(1), 2022, pp. 100–106.

[5] D. ALEKSEEVA, N. STEPANOV, A. VEPREV, A. SHARAPOVA, E. S. LOHAN, and A. OMETOV, *Comparison of machine learning techniques applied to traffic prediction of real wireless network*, IEEE Access **9**, 2021, pp. 495–514.

[6] M. CONTI, Q. Q. LI, A. MARAGNO, and R. SPOLAOR, *The dark side (-channel) of mobile devices: a survey on network traffic analysis*, IEEE Communications Surveys and Tutorials **20**(4), 2018, pp. 2658–2713.

[7] Y. HUA, Z. ZHAO, Z. LIU, X. CHEN, R. LI, and H. ZHANG, *Traffic prediction based on random connectivity in deep learning with long short-term memory*, Proceedings of 2018 Vehicular Technology Conference (VTC-Fall), Chicago, IL, USA, 2018, pp. 1–6.

[8] J.CYNTHIA, G. PRIYA, C. SAMUEL, M. SUGUNA, J. SENTHIL and S. JEBARAJ, *Traffic flow forecasting using machine learning techniques*, Webology **18**(4), 2021, pp. 1512–1526.

[9] H. SHI, *Mobile network traffic prediction based on machine learning*, Proceedings of the 2022 International Conference on Economic Management and Cultural Industry, Online, Atlantis Press, 2023, pp. 1691–1698.

[10] A. MOGHAR and M. HAMICHE, *Stock market prediction using LSTM recurrent neural network*, Procedia Computer Science **170**, pp. 1168–1173, 2020.

[11] J. FENG and S. LU, *Performance analysis of various activation functions in artificial neural networks*, IOP Conference Series: Journal of Physics **1237**(2), 2019, paper 22030.

[12] S. SHARMA, S. SHARMA, and A. ATHAIYA, *Activation functions in neural networks*, International Journal of Engineering Applied Sciences and Technology **4**(12), 2020, pp. 310–316.

[13] L. DATTA, *A survey on activation functions and their relation with Xavier and He normal initialization*, arxiv, 2020, arXiv:2004.06632.

[14] R. ZAHEER and H. SHAZIYA, *A study of the optimization algorithms in deep learning*, Proceedings of 2019 International Conference on Inventive Systems and Control, Coimbatore, India, 2019, pp. 536–539.

[15] D. SOYDANER, *A comparison of optimization algorithms for deep learning*, International Journal of Pattern Recognition and Artificial Intelligence **34**(13), 2020, paper 2052013.

[16] A. MUSTAPHA, L. MOHAMED, and K. ALI, *Comparative study of optimization techniques in deep learning: application in the ophthalmology field*, IOP Conference Series: Journal of Physics **1743**(1), 2021, paper 012002.

[17] S. H. HAJI and A. M. ABDULAZEEZ, *Comparison of optimization techniques based on gradient descent algorithm: a review*, PalArch's Journal of Archaeology of Egypt **18**(4), 2021, pp. 2715–2743.

[18] J. S. P. FONG and K. WONG TING YAN, *Data normalization*, in Information Systems Reengineering, Integration and Normalization, Springer International Publishing, Cham, 2021, pp. 287–316.

[19] S. PRAJAM, C. WECHTAISONG, and A. A. KHAN, *Applying machine learning approaches for network traffic forecasting*, Indian Journal of Computer Science and Engineering **13**(2), 2022, pp. 324–335.

[20] D. CHICCO, M. J. WARRENS, and G. JURMAN, *The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE, and RMSE in regression analysis evaluation*, PeerJ Computer Science **7**, 2021, paper e623.

[21] A. G. PRIYA VARSHINI, K. ANITHA KUMARI, D. JANANI, and S. SOUNDARIYA, *Comparative analysis of machine learning and deep learning algorithms for software effort estimation*, IOP Conference Series: Journal of Physics **1767**, 2021, paper 012019.