MATHEMATICAL MODELING AND COMPUTING, Vol. 12, No. 3, pp. 779–793 (2025)

# The impact of activation functions on LTSM server load prediction accuracy: machine learning approach

Chaban O. M.*, Pukach P. P., Pukach P. Ya., Hladun V. R.

*Lviv Polytechnic National University, Lviv, 79013, Ukraine*
* *Corresponding author: oleksandr.m.chaban@lpnu.ua*

(Received 10 June 2025; Revised 3 August 2025; Accepted 4 August 2025)

The continuously growing number of users and their requests to the server demands substantial resources to ensure fast responses without delays. However, server load is inherently unevenly distributed throughout the day, week, or month. Accurately predicting the required resources and dynamically managing their allocation is crucial, as it can lead to significant cost savings in server maintenance without compromising the user experience. This study investigates the influence of activation function choice on the forecasting accuracy of Long Short-Term Memory (LSTM) neural networks applied to real-world server request data. A dataset of incoming server requests was collected and aggregated into 20-minute intervals over 16 consecutive days. Several activation functions—including ReLU, Swish, and Softplus—were evaluated using mean squared error (MSE) as the primary performance metric. Each model configuration was trained six times to ensure statistical reliability, and the results were taken from one of the most stable runs. The experiments demonstrate that the selection of activation function has a significant impact on prediction accuracy: Swish and ReLU achieved the lowest MSE values, reducing error by up to $6.6-12.3\%$ and $10.5-16.3\%$, respectively, compared to the baseline. Although the sigmoid function yielded the lowest test loss, further analysis revealed that this outcome was misleading: the model systematically underestimated peak loads, resulting in lower error values but poor predictive fidelity with respect to actual server load dynamics. These findings validate the hypothesis that activation function choice is a critical factor in optimizing LSTM-based forecasting models for server load prediction.

**Keywords:** *server load prediction; machine learning; activation function; long short-term memory.*

**2010 MSC:** 68T05 **DOI:** 10.23939/mmc2025.03.779

## 1. Introduction

Modern cloud servers [1] operate in a highly structured and layered manner, splitting resources into progressively smaller and more specialized units [2]. At the highest level, the physical server acts as the foundation, hosting multiple workloads. These workloads are virtualized into nodes, each representing an instance or containerized environment within the server. Nodes, in turn, are further divided into smaller entities depending on the platform. In containerized systems like Kubernetes [3], nodes host Pods, which are the smallest deployable units and often encapsulate one or more containers. Containers then house individual processes or tasks, representing the specific applications or services running on the infrastructure. This hierarchical approach optimizes resource utilization, ensuring flexibility and scalability while maintaining efficient isolation between workloads. By splitting resources into these distinct layers, cloud systems provide the adaptability required for modern dynamic workloads, offering a balance between granularity and control. And there are ongoing efforts which are directed towards further subdivision of units to achieve even greater efficiency.

Most applications initially run on a single node [4]. However, as the number of users and the volume of processed data grow over time, the system load increases. To maintain stable operation and low response times, scaling becomes necessary—adding additional resources to collaboratively process requests and distribute the load efficiently [5].

**Problem statement.** In practice, the situation is more complex. Server load is rarely constant. The number of requests typically fluctuates based on various factors such as the time of day, day of the week, special events, or holidays. Consequently, there is a continuous need to dynamically adjust the allocated resources to match the changing demand.

On the other hand, it is evident that any increase in allocated resources results in higher server hosting costs. Cloud providers such as Amazon Web Services (AWS) [6], Google Cloud Platform (GCP) [7], and Microsoft Azure [8] typically bill users based on resource consumption, including compute time, storage, and network usage. During periods of high request volume, this investment is justified. However, once the request rate declines, idle resources reduce overall cost efficiency.

There are two common scenarios in which such situations may occur. The first is quite typical: the number of requests rises and falls periodically over a defined time interval. In these cases, the volume of released resources is usually modest compared to the peak load. However, due to the regularity of these fluctuations and the continuous monitoring and control of the allocated resources, the cumulative effect can result in significant cost savings.

The second scenario involves unexpected or exceptional events, characterized by sharp, short-term surges in server load. In such cases, it is especially important to promptly deallocate any resources that are no longer needed once the peak subsides, in order to maintain cost efficiency.

However, constant monitoring of resource allocation cannot realistically be performed by a single person — or even a group of people — at least not manually. There is always a risk of human error [9,10]. Even developers who deliberately scale up resources for tasks such as load testing [11] may forget to scale them back down afterward. Given this, it is unreasonable to expect individuals with less technical involvement to manage such adjustments reliably.

That is why it is reasonable for this process to be managed by a mechanism capable of automatically predicting both load increases and decreases. In the case of load increases, additional resources can be allocated in a timely manner to minimize response delays. Conversely, predicting load decreases allows the system to promptly release unused resources and avoid unnecessary maintenance costs.

**The article's objective.** The LSTM model includes numerous parameters that influence its performance, many of which have been extensively studied. However, this study focuses specifically on the impact of the chosen activation function. Accordingly, it concentrates on practical outcomes — namely, the accuracy of server load predictions — and compares these results across different activation functions.

Additionally, a hypothesis is proposed that, alongside other hyperparameters and influencing factors, the type of activation function used in an artificial neuron can also affect the overfitting tendency of a machine learning algorithm, assuming all other system parameters remain unchanged.

The remainder of the paper is organized as follows: the continuation of Section 1 presents the foundational knowledge underpinning the study; Section 2 introduces the dataset, provides its analysis, and describes the materials and methods used; Section 3 discusses the results and presents a comparative analysis; and Section 4 summarizes the study and evaluates the performance of the activation functions in predicting server load.

## 1.1. Time series predictions

Time series analysis encompasses a set of mathematical and statistical methods aimed at identifying the structure of time series and predicting it [12]. These methods include regression analysis [13]. Identifying the structure of a time series is a necessary step in constructing a mathematical model of the phenomenon generating the analyzed series. Predicting future values of a time series based on the analysis of current and historical data is crucial for enhancing decision-making effectiveness.

From a mathematical perspective, a time series is defined as a sequence of scalars or vectors $y(t)$, where $t$ represents a moment in the past. The task of time series prediction is to determine a function $f(x)$ that models the dependence of future time series values on their previous values:

$$y_0(t) = f(y(t-1), y(t-2), \dots, y(t-d)),$$

where the value $d$ is referred to as the time lag. This parameter denotes the number of preceding elements from the dataset that are considered during model training and the prediction of the next value [14].

### 1.2. Long short-term memory

The Long Short-Term Memory (LSTM) model [15,16] is a type of recurrent neural network (RNN) [17] architecture used in deep learning [18] to process sequential data. The primary advantage of this model is its ability to overcome the exploding and vanishing gradient problems, which often occur when learning long-term dependencies, even when the minimal time lags are very long [19].

Mathematically, an artificial neuron is typically represented as a nonlinear function of a single argument — the linear combination of all input signals. An essential component of an artificial neuron is the excitation function, also known as the activation function [20], which defines the type of nonlinearity in the model. As empirically demonstrated in [21], although the activation function is not the primary factor in achieving the highest accuracy in an LSTM network, it still plays a significant role.

When a model has been trained exhaustively on the available training data, it may fail to generalize to new data, resulting in a phenomenon known as overfitting [22]. In such cases, the model often memorizes all the data, including the unavoidable so-called noise in the training set, rather than learning the underlying patterns within the data [23].

Underfitting is the opposite of overfitting and occurs when a model fails to capture the underlying variability of the data [24]. This issue is particularly evident in scenarios such as modeling nonlinear data with a linear model [25].

One method to address overfitting is the dropout approach, a regularization technique for artificial neural networks that also significantly accelerates the training process [26]. The essence of this method lies in randomly selecting a subnetwork from the overall network during training and performing the training specifically on that subnetwork. After training the selected subnetwork, a new subnetwork is randomly chosen, and the training continues. The neurons for each subnetwork are selected randomly based on a probability known as the dropout rate. Neurons that undergo more training are assigned greater weights in the final network [27].

### 1.3. Rationale for selecting LSTM

While recent advancements in time series forecasting have introduced a variety of specialized tools and frameworks — such as Facebook Prophet [28], Amazon DeepAR [29], and the Darts library [30] — this study deliberately focuses on the use of Long Short-Term Memory (LSTM) networks. The rationale 1for this choice is twofold.

First, the primary objective of this research is to investigate the impact of different activation functions on prediction performance. LSTM models provide full access to internal architectural components, including activation layers, which allows for controlled experimentation. In contrast, many modern frameworks encapsulate their internal mechanisms, thereby restricting such modifications and making them unsuitable for activation-level analysis [28, 29].

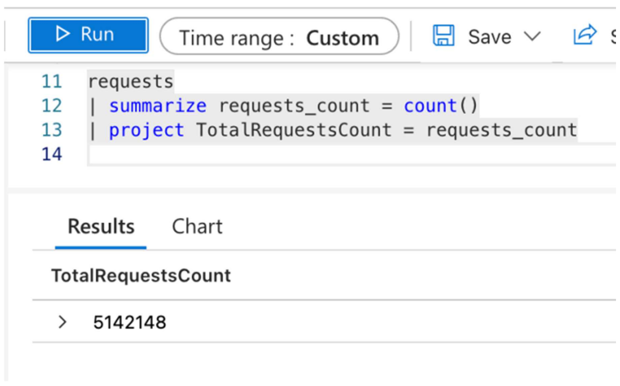
Second, LSTM networks are particularly effective for time series data that exhibit both short- and long-term dependencies, as is the case with server load patterns [15]. These patterns include periodic user activity and sudden usage spikes, which LSTM architectures are well-equipped to capture.

Furthermore, LSTM models remain widely used in server-side forecasting tasks within cloud environments. Their flexibility and transparency make them a natural choice for research focused on internal model behavior rather than purely predictive accuracy.

## 2. Materials and methods

### 2.1. Input data and its analysis

The dataset was sourced from the request monitoring tool within the Microsoft Azure Portal [31], which is connected to the server of a mobile application. The application's user base is distributed across the European region, with a significant portion located in the United Kingdom. Requests were collected during the period from December 8, 2024, at 06:00 to December 25, 2024, at 06:00, encompassing a total of 17 full days. The total number of requests to the server during this period was 5 142 148, as shown in Figure 1.



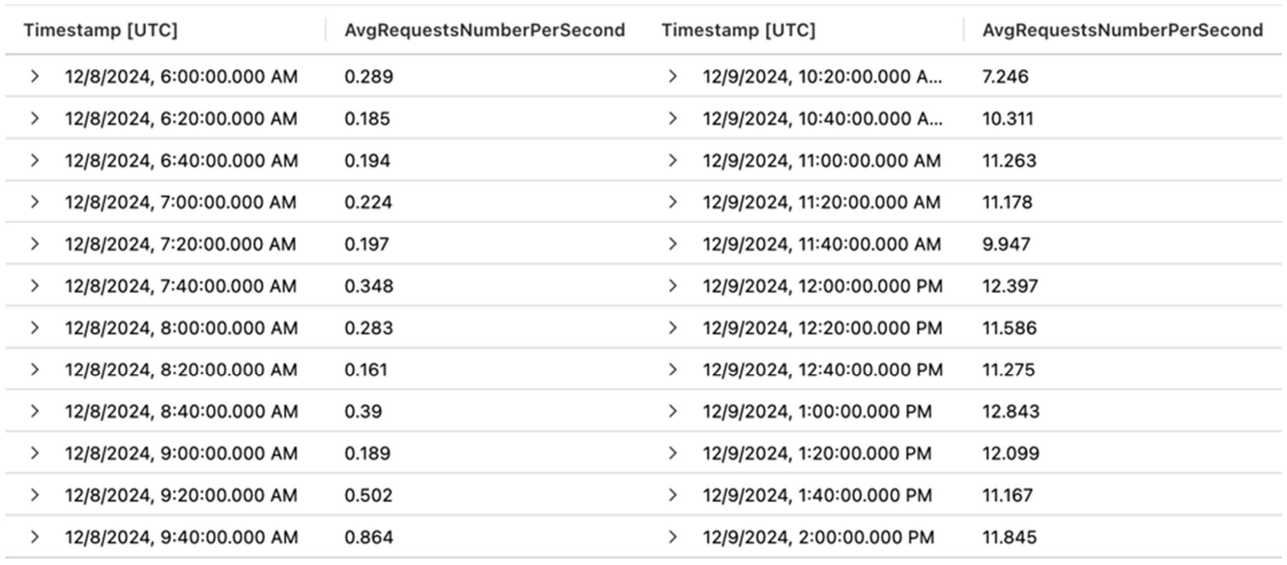
```
requests
| summarize requests_count = count()
| project TotalRequestsCount = requests_count
```

| TotalRequestsCount |
|---|
| 5142148 |

**Fig. 1.** Total number of requests to the server during the period.

The uniqueness of the dataset lies in its coverage of two complete weeks starting from a Sunday, with the final three days marking the beginning of a third week. This is particularly notable, as the third week includes December 25 — a bank holiday in almost all European countries. It is important to note that the dataset does not encompass the entire day of December 25, as it concludes at 06:00 on that date. Nonetheless, it presents a unique opportunity to analyze a dataset that partially includes such a significant event.

The raw data were processed to include a timestamp at 20-minute intervals throughout the period, along with a value representing the average number of requests received by the server per second during each corresponding interval. As a result of this transformation, a total of 1 224 records corresponding to 20-minute intervals were obtained. An example of such records is shown in Figure 2.

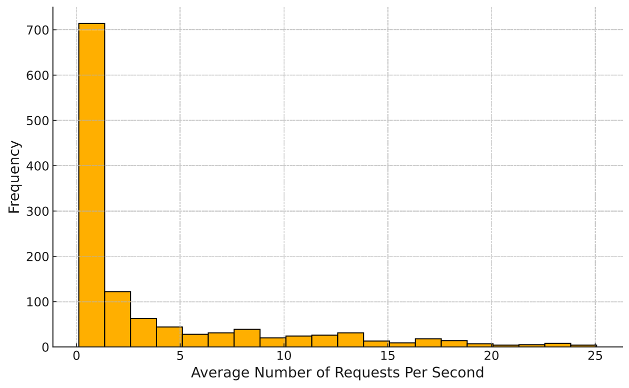


| Timestamp [UTC] | AvgRequestsNumberPerSecond | Timestamp [UTC] | AvgRequestsNumberPerSecond |
|---|---|---|---|
| 12/8/2024, 6:00:00.000 AM | 0.289 | 12/9/2024, 10:20:00.000 A... | 7.246 |
| 12/8/2024, 6:20:00.000 AM | 0.185 | 12/9/2024, 10:40:00.000 A... | 10.311 |
| 12/8/2024, 6:40:00.000 AM | 0.194 | 12/9/2024, 11:00:00.000 AM | 11.263 |
| 12/8/2024, 7:00:00.000 AM | 0.224 | 12/9/2024, 11:20:00.000 AM | 11.178 |
| 12/8/2024, 7:20:00.000 AM | 0.197 | 12/9/2024, 11:40:00.000 AM | 9.947 |
| 12/8/2024, 7:40:00.000 AM | 0.348 | 12/9/2024, 12:00:00.000 PM | 12.397 |
| 12/8/2024, 8:00:00.000 AM | 0.283 | 12/9/2024, 12:20:00.000 PM | 11.586 |
| 12/8/2024, 8:20:00.000 AM | 0.161 | 12/9/2024, 12:40:00.000 PM | 11.275 |
| 12/8/2024, 8:40:00.000 AM | 0.39 | 12/9/2024, 1:00:00.000 PM | 12.843 |
| 12/8/2024, 9:00:00.000 AM | 0.189 | 12/9/2024, 1:20:00.000 PM | 12.099 |
| 12/8/2024, 9:20:00.000 AM | 0.502 | 12/9/2024, 1:40:00.000 PM | 11.167 |
| 12/8/2024, 9:40:00.000 AM | 0.864 | 12/9/2024, 2:00:00.000 PM | 11.845 |

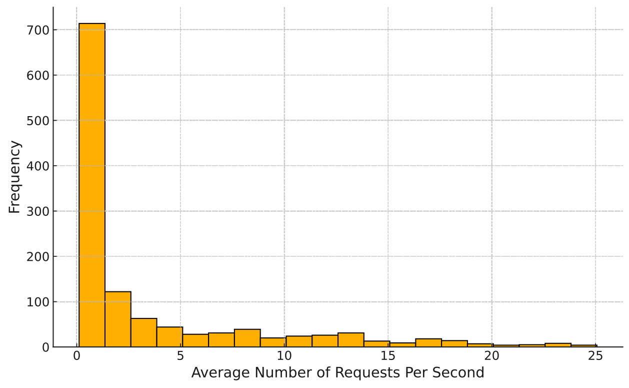
**Fig. 2.** Examples of transformed to 20-minute intervals data.

Figure 3*a* illustrates a histogram of the average number of requests per second across the intervals in the dataset. The vast majority of intervals — a total of 856 — have an average request rate below 3 requests per second, with the overall mean being 3.501 requests per second. The maximum recorded value is 25.065 requests per second, observed on December 20, 2024, between 14:00 and 14:20. This was the only interval that exceeded an average of 25 requests per second. Furthermore, 368 intervals recorded an average rate greater than 3 requests per second, 166 intervals exceeded 10 requests per second, and 71 intervals surpassed 15 requests per second.

Figure 3*b* demonstrates the variation in the average request rate across 20-minute intervals throughout the day. As expected, there is minimal activity during the night, from 23:00 to 06:00, with an average of 0.314 requests per second across these intervals. A slight increase in activity is observed from 06:00 to 07:00 and again from 18:00 to 23:00, covering the early morning and evening hours. The average request rate rises to 0.878 requests per second during these periods.



*a*



*b*

**Fig. 3.** Dataset analysis: (***a***) Frequency of the average number of requests per second, (***b***) Average request rate by time intervals.

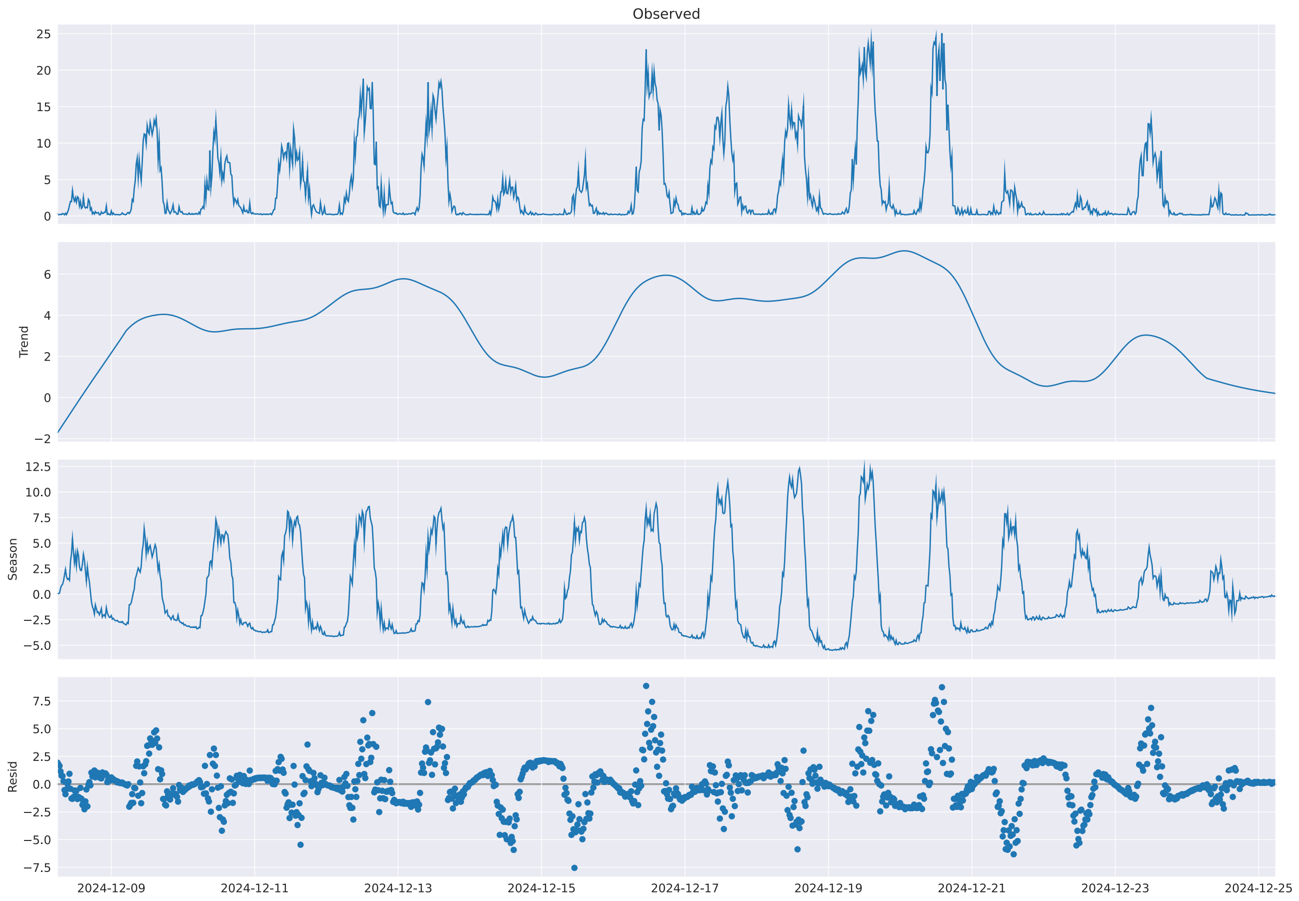


**Fig. 4.** Time series decomposition.

A rapid increase in activity begins at 07:00, reaching its first and highest peak during the 11:00–11:20 interval, with an average of 10.948 requests per second. Following this peak, a slight decline is observed, reaching a low at the 12:40–13:00 interval, with an average of 8.999 requests per second, likely attributable to lunchtime. Activity then gradually rises to a second peak during the 14:20–14:40 and 14:40–15:00 intervals, with averages of 10.418 and 10.362 requests per second, respectively. After this, a sharp decline is observed over the next three hours. The period from 07:00 to 18:00 has an average request rate of 7.059 requests per second, while the period of highest activity, from 10:00 to 15:40, averages 9.717 requests per second.

After performing the decomposition of the time series [32], the result is presented in Figure 4 and consists of four components. The first subplot displays the original time series, where weekdays with significantly higher loads are clearly visible. Five consecutive high spikes alternate with two smaller ones, representing weekdays and weekends, respectively. Notably, the final visible spike is even smaller than a typical weekend spike, reflecting the day before the bank holiday and capturing the reduced activity typical of that period.

The second subplot highlights the trend component. This pattern mirrors the original series, with the load increasing during weekdays and forming noticeable dips over the weekends. The region surrounding the bank holiday exhibits a similar pattern to the weekends, emphasizing the reduced activity during that period.

The third subplot illustrates the seasonal component. The same weekday–weekend pattern is observed here. Additionally, the widest amplitude occurs on December 18 and 19 — the last two days of a standard working week. This observation aligns with the highest peaks in the first subplot and corresponds to the strongest upward trend in the second subplot.

The fourth subplot presents the residual component, which represents the variations not captured by the trend or seasonal patterns.

## 2.2. Data preparation

To enable a more efficient and thorough analysis of the dependencies influencing the predicted value, the prepared dataset is reformatted to account for the time lag factor.

Schematically, this process can be viewed as a straightforward transformation of the prepared dataset $D$, converting it from a scalar format to a vector format. Let the dataset be defined as:

$$D = (X, Y),$$

where

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Then, the reformatted dataset $D'$, adjusted for the time lag value, is structured as follows:

$$D' = (X', Y'),$$

where

$$X' = \begin{pmatrix} (x_1, x_2, \ldots, x_d) \\ (x_2, x_3, \ldots, x_{d+1}) \\ \vdots \\ (x_{n-d}, x_{n-d+1}, \ldots, x_n) \end{pmatrix}, \quad Y' = \begin{pmatrix} y_d \\ y_{d+1} \\ \vdots \\ y_n \end{pmatrix}.$$

Thus, the algorithm can be supplied with relationships not only of the type "input value – output value" but also of the type "series of input values – output value".

By employing this approach, a portion of the input data — equal to the size of the time lag $d$ — is discarded. However, in practice, this portion is relatively small compared to the overall dataset, and the resulting improvements in the model's prediction quality are significant and outweigh the trade-off.

The time lag value was set to represent 12 hours, divided into 20-minute intervals, resulting in $d = 36$.

As shown in Figure 5, the consolidated vector dataset is divided into three parts: training data, validation data, and test data, in a ratio of 11:2:3. The neural network is trained using the training data, while the validation data is used exclusively to calculate the model's prediction error at each training epoch. The test data is reserved for conducting the final evaluation of the trained models.

Due to the nature of the real data used — which possesses its own specific characteristics — the training set differs from the validation and test sets. It is larger and, unavoidably, includes weekends, making it distinct from the other subsets.

Thus, at each iteration of the machine learning algorithm, it is possible to compute both the prediction error for the training data and the prediction error for the validation data, which the model

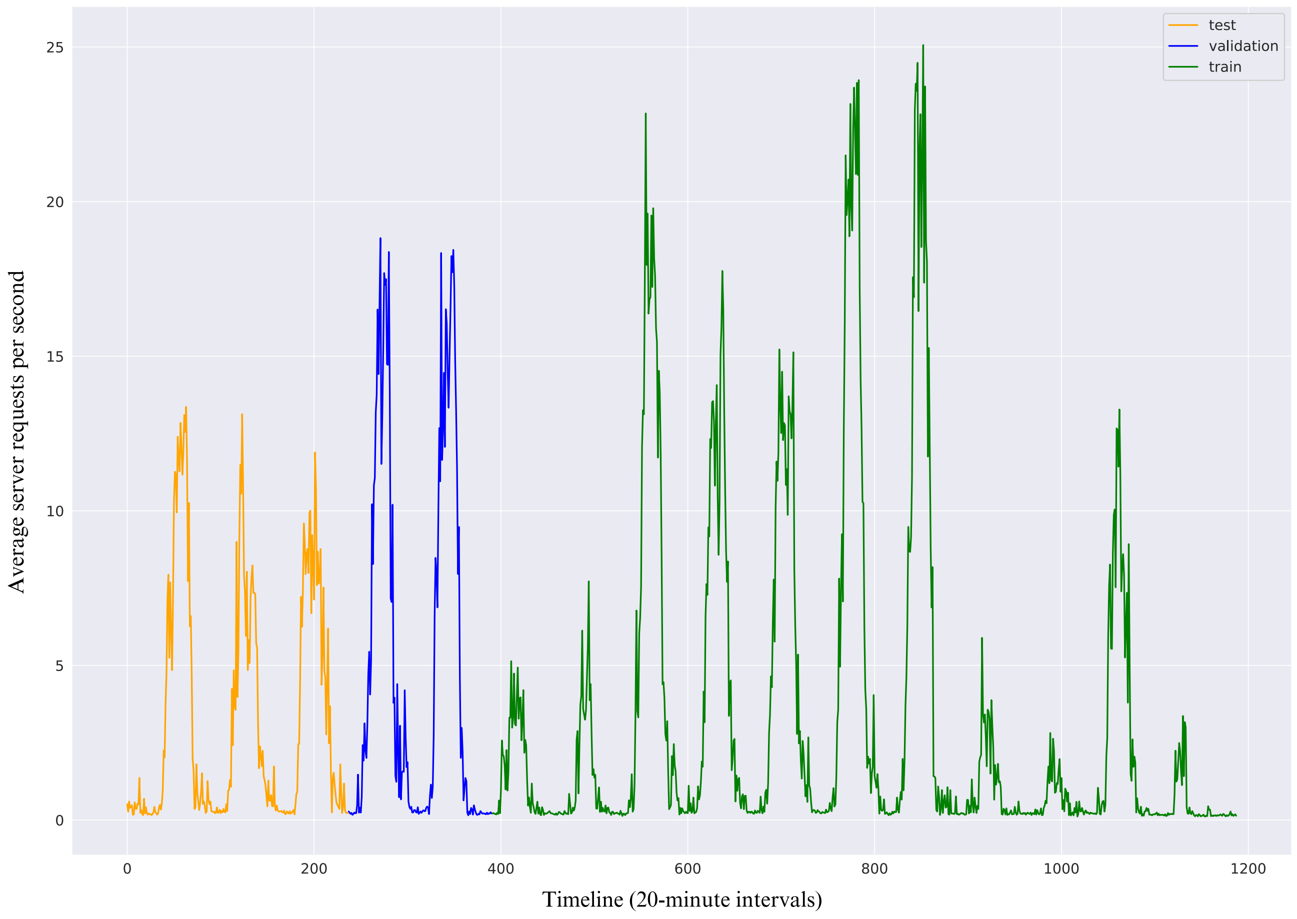


**Fig. 5.** Data split into training, validation, and test parts.

has not encountered during training. This approach facilitates the construction of an extended training history graph. Such a graph enables the monitoring of potential overfitting by assessing whether the training and validation errors align or diverge during the training process.

To rigorously quantify these errors, the mean squared error (MSE) [33] is employed as the evaluation metric. The MSE is defined by the following formula:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2,$$

and provides a numerical measure of the deviation between the predicted outcomes and the actual target values.

## 2.3. Methods and functions used

The LSTM model was implemented using the PyTorch [34,35] library. The model was configured with the following parameters:

— The main block consists of 64 hidden LSTM neurons.
— A dropout rate of 0.2 was applied during training.
— The number of training epochs was set to 500.

The following activation functions were analyzed to assess their impact on the prediction accuracy.

The **hyperbolic tangent activation function** [36]

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

is characterized by its ability to amplify weak signals more significantly than strong ones. This occurs because regions corresponding to strong signals lie on the flatter sections of the function.

This function serves as the default activation function in LSTM models [37,38]. It has also inspired the development of modified variants, such as the hyperbolic tangent exponential linear unit (TeLU) activation function [39], which combines the linear characteristics of traditional activation functions with the non-linear properties of exponential and hyperbolic tangent functions. This hybrid approach offers a balance that enhances learning efficiency while mitigating gradient-related issues.

The **linear activation function**

$$f(x) = t\,x$$

is known for its transfer mechanism and is typically used in neurons that form the input layer of multi-layer neural networks. Variations of the linear function can also be employed, such as the semi-linear function or the step function [40], which can be expressed by the following formula:

$$f(x) = \begin{cases} 0, & x \leqslant 0, \\ 1, & x \geqslant 1, \\ x, & \text{else}. \end{cases}$$

The **sigmoid activation function** [41]

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

was previously among the most commonly used types of transfer functions. Sigmoid-type functions gained popularity due to their ability to overcome the limitations of threshold activation functions, which restricted neuron outputs to binary values and confined neural networks to classification tasks. By enabling a smooth transition from binary to analog outputs, these functions significantly expanded the scope of neural network applications.

The **ReLU (Rectified Linear Unit) activation function** [42]

$$f(x) = \max(0, x)$$

introduces non-linearity into the network, enabling it to learn complex patterns. It is computationally efficient and helps mitigate the vanishing gradient problem, a common issue associated with other activation functions.

The **Swish activation function** [43] is a smooth, non-monotonic function, also referred to as a self-gated activation function. It is a hybrid function, proposed as a combination of the sigmoid and ReLU activation functions. Its success has sparked interest in developing novel modifications, such as P-Swish [44] and T-Swish [45]. The Swish function is defined as:

$$f(x) = x \cdot \sigma(\beta x), \tag{1}$$

where $\sigma(\beta x)$ is the sigmoid function, and $\beta$ is a tunable parameter that determines the "sharpness" of the Swish function. Therefore, Equation (1) can also be expressed as:

$$f(x) = \frac{x}{1+\ e^{-\beta x}}.$$

The **Softplus activation function** [46]

$$f(x) = \ln(1+e^{x})$$

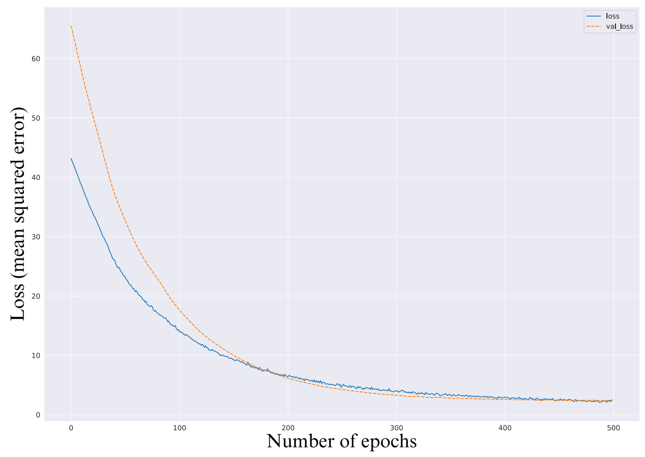
is a smooth approximation of the ReLU activation function. Unlike ReLU, it is continuously differentiable, which can facilitate optimization in neural networks.

## 3. Results

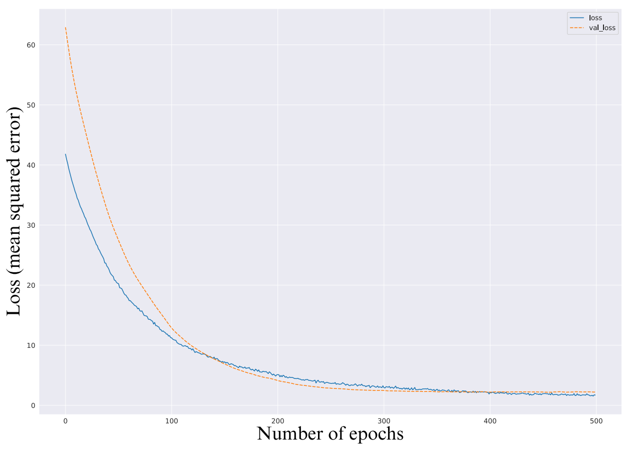
The training progress graphs for models with different activation functions are shown in Figure 6, illustrating the validation error at each epoch. As observed from the figure, the best results after 500 epochs are achieved by the hyperbolic tangent and Softplus activation functions.

The linear activation function is the only one where the validation loss is slightly lower than the training loss. If the difference were larger, it would be a clear indication of overfitting. However, in this case, it merely suggests that, starting from the 400th epoch, the model with the linear activation function began adapting to the data and exhibited early signs of overfitting.

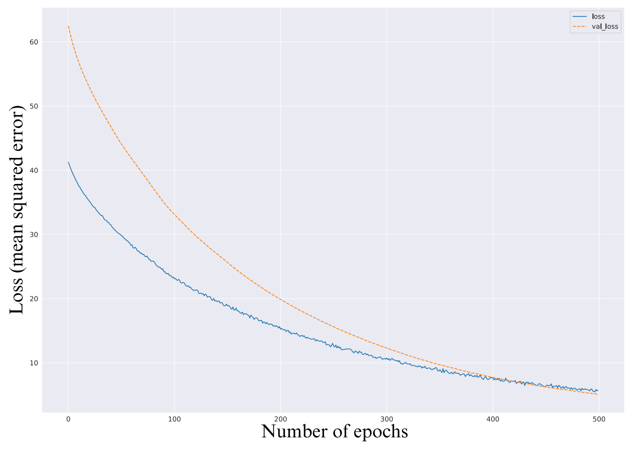
The remaining activation functions — sigmoid, ReLU, and Swish — demonstrated comparable performance, with the training loss being slightly higher than the validation loss. However, the difference is small enough to indicate that the learning process remains stable and is not a cause for concern.



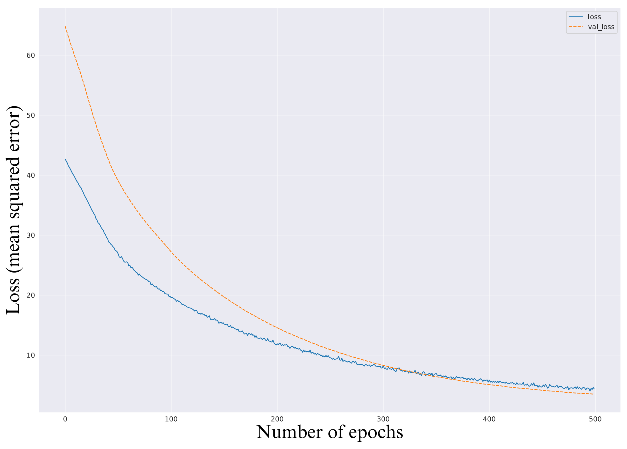
***a*** (hyperbolic tangent)



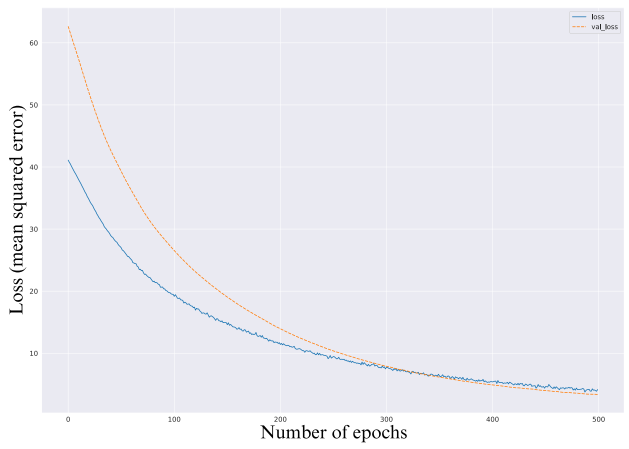
***b*** (linear)



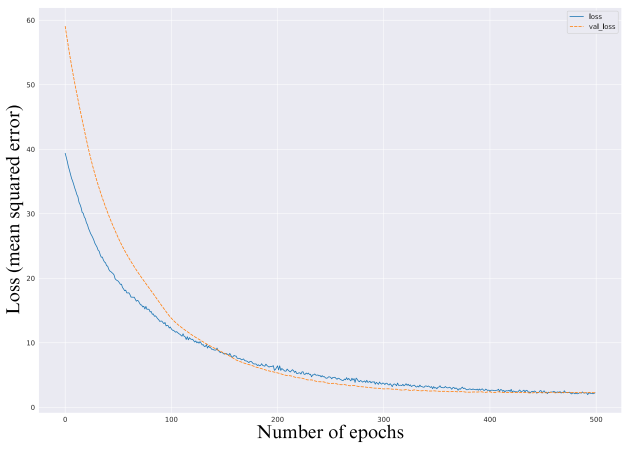
***c*** (sigmoid)



***d*** (ReLU)



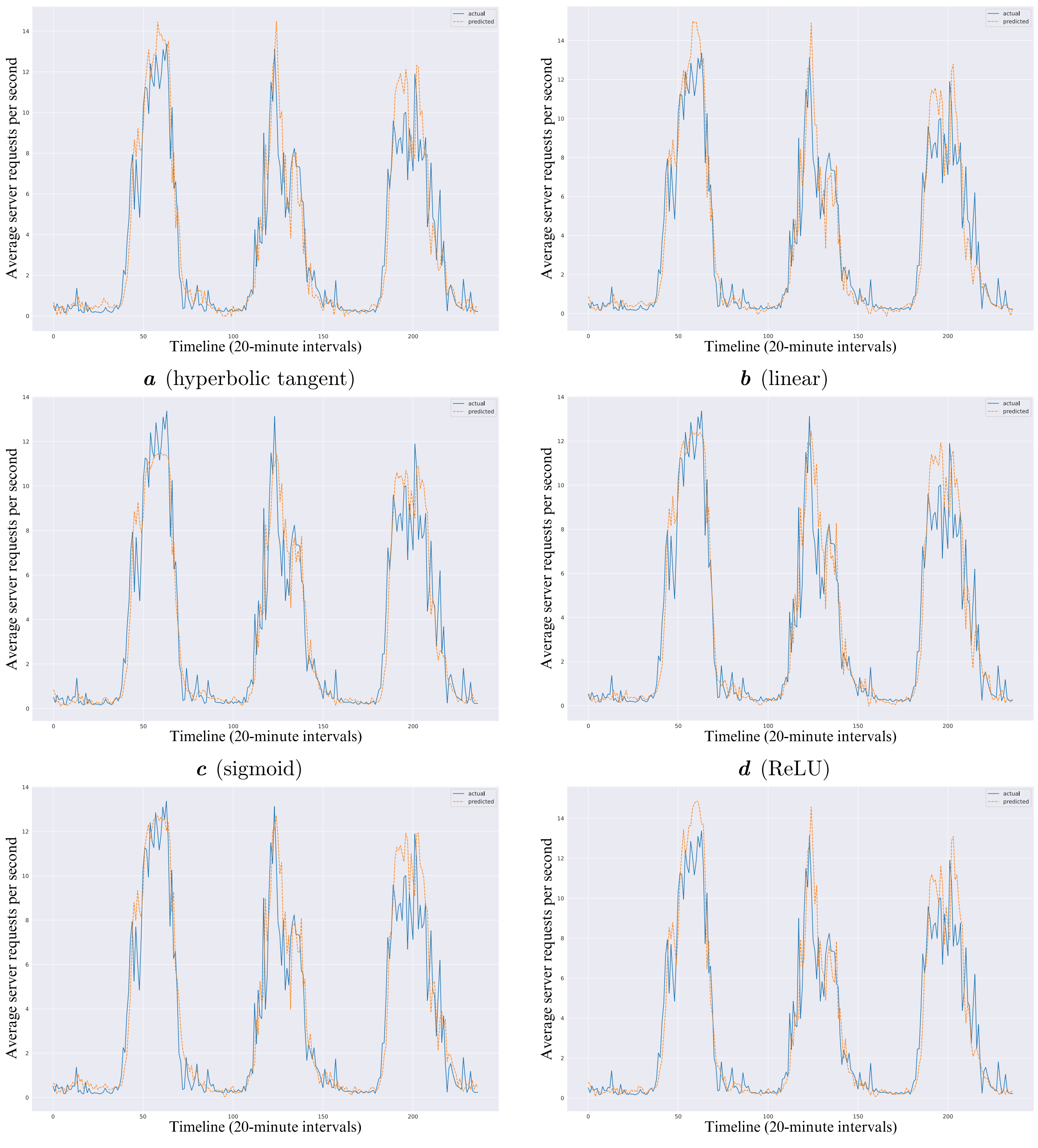
***e*** (Swish)



***f*** (Softplus)

**Fig. 6.** Training progress of the model with different activation functions.

Figure 7 illustrates server load predictions over a three-day period using models with different activation functions. Before analyzing these predictions, it is essential to establish the criteria for evaluating model performance. The primary objective of these models is to dynamically adjust server



***a*** (hyperbolic tangent) ***b*** (linear)

***c*** (sigmoid) ***d*** (ReLU)

***e*** (Swish) ***f*** (Softplus)

**Fig. 7.** The model's prediction with different activation functions.

resource allocation to ensure cost-efficient utilization. Consequently, if a model significantly overestimates server load in scenarios where no such load exists, its evaluation will be negatively impacted due to the potential for unnecessary resource allocation.

However, the foremost priority is maintaining the server's responsiveness. Users must not experience delays in response times due to optimization efforts. Therefore, models that slightly overestimate the actual load are preferred over those that underestimate it, as the latter could result in insufficient resource allocation and degraded user experiences.

On the first day, which featured an extended period of high load, the Swish activation function performed the best, capturing nearly all spikes with high accuracy. Its closest competitor was the ReLU activation function, which slightly underestimated the final spike. The hyperbolic tangent, linear, and Softplus activation functions exhibited suboptimal performance due to overestimations, with accuracy declining in the order listed. Notably, the hyperbolic tangent function produced less pronounced overestimations while still successfully identifying all spikes. In contrast, the sigmoid activation function performed poorly, significantly underestimating the load and failing to predict any high-load spikes.

The second day presented a different scenario, characterized by a narrow medium spike followed by a very sharp high spike and a subsequent decline to a broad medium load. The first spike was missed by all activation functions, although the predictions made by the Swish and ReLU functions were the closest to the actual data.

Similar to the first day, the hyperbolic tangent, linear, and Softplus activation functions overestimated the load. However, due to the sharpness of the spike, the impact of this overestimation was reduced. Both Swish and ReLU delivered highly accurate predictions, with minor underestimations, and Swish exhibited a slight edge in precision. The sigmoid activation function again significantly underestimated the load but successfully mirrored the sharpness of the spike. Notably, all activation functions — except linear — incorrectly anticipated a second, slightly lower spike.

In the latter part of the day, the ReLU activation function performed best, closely followed by Swish, hyperbolic tangent, and even sigmoid, which delivered better-than-expected performance.

A comparison of the first two days for the sigmoid activation function reveals that its predictions plateaued at a certain level. Moreover, the prediction pattern on the first day appears highly atypical when compared to other graphs, which exhibit relatively sharp peaks. In contrast, the sigmoid function's prediction for that day resembles a rounded curve, broad enough to span the entire duration of the high load. This behavior suggests that the sigmoid function imposes a fixed upper limit that it cannot exceed. Given this limitation and the presence of a prolonged high load, the resulting broad, rounded prediction appears consistent with the function's characteristics.

The third day was notable for its load pattern, which extended over a longer duration, resulting in a broader but comparatively lower graph. This characteristic led to significant overestimations by all activation functions without exception. Among them, the sigmoid activation function produced the closest predictions to the actual data. However, this outcome should be regarded as fortuitous, as the unusually low load during peak hours coincided with the function's typical tendency toward strong underestimation.

Despite the overall poor performance, the linear and Softplus activation functions can be considered the better performers on this day. The hyperbolic tangent function, on the other hand, delivered the worst results. Toward the end of the peak period, two sharp spikes occurred, but none of the trained models successfully captured them.

Table 1 provides a comparison of the training, validation, and test losses for models with different activation functions evaluated in this study. The training and validation losses are recorded at the final, 500th epoch.

**Table 1.** Training, validation and test losses for models.

| Model's activation function | Training loss | Validation loss | Test loss |
|---|---|---|---|
| Hyperbolic tangent | 2.456501 | 2.444833 | 2.144749 |
| Linear | 1.719406 | 2.211547 | 2.209427 |
| Sigmoid | 5.725735 | 5.097187 | 1.708496 |
| ReLU | 4.357741 | 3.481589 | 1.933583 |
| Swish | 4.151990 | 3.352038 | 2.012375 |
| Softplus | 2.251868 | 2.274745 | 2.260747 |

The linear activation function is the only one that exhibits a significant difference between the training and validation losses, with the training loss being lower — potentially indicating overfitting of the model. Furthermore, the hyperbolic tangent and Softplus activation functions demonstrate the smallest differences between training and validation losses. This observation aligns with the results shown in Figures 6$b$, 6$a$, and 6$f$, respectively.

At first glance, it may seem surprising that the sigmoid activation function exhibits the smallest test loss, potentially leading to incorrect conclusions. However, loss alone should not be the sole criterion for evaluating model performance. It is evident that the predictions on the third day heavily influenced this metric. As discussed in the analysis of Figure 7$c$, the sigmoid function's tendency to underestimate resulted in smaller errors for that day, while other activation functions incurred significantly higher errors due to their attempts to accurately capture the actual spikes.

Additionally, the first day's performance likely contributed to the sigmoid function's lower test loss. Instead of attempting to accurately predict spikes, the model flattened its predictions toward the average load, failing to capture actual variations. Although this behavior reduced the overall loss metric, it does not indicate good performance, as the primary objective is to predict load patterns — particularly spikes — rather than merely minimize the loss. Therefore, despite the lower test loss, the performance of the sigmoid function is considered poor.

Analyzing the test loss for the other activation functions reveals a noticeable improvement for both ReLU and Swish, each showing a considerable reduction. The hyperbolic tangent function also demonstrated a lower test loss; however, the difference between its test and validation losses is relatively modest.

The test loss for the linear activation function is nearly equivalent to its validation loss, showing no improvement. Both values remain higher than the training loss, once again suggesting overfitting in the model.

Interestingly, the Softplus activation function exhibited relatively consistent loss values across the training, validation, and test sets.

## 4. Conclusions

This study presents a comparative analysis of prediction models based on recurrent neural networks that employ different activation functions for artificial neurons. The results include representative prediction examples and training process graphs that compare training and validation errors. The findings support the hypothesis that the choice of activation function significantly influences prediction accuracy, even when all other model parameters remain fixed.

As part of this study, a flexible evaluation framework was developed for LSTM-based models, enabling the systematic comparison of various activation functions and supporting future investigations of new activation function classes. Data collection, preprocessing, and statistical analysis were conducted manually to maintain full control over data integrity and consistency. A key contribution of this work is the introduction of a real-world dataset of server requests, which captures realistic load fluctuations and may serve as a benchmark for future studies. Considerable effort was devoted to the data acquisition and preparation stages to ensure the reliability of the experimental results.

The Swish and ReLU activation functions demonstrated excellent performance, occasionally yielding near-perfect predictions of server load, thus proving to be highly suitable for this task. In most cases, Swish exhibited a slight but consistent advantage in accuracy over ReLU.

To further investigate the prediction accuracy of server load using models with different activation functions, the following steps are planned:

— Enable models to account for the distinct characteristics of various types of days (e.g., weekdays, weekends, holidays) during training and prediction.
— Incorporate explainable AI (XAI) techniques to interpret the internal decision-making processes of LSTM models employing different activation functions, with a particular focus on feature relevance and temporal sensitivity during high-load periods.

[1] Saraswat M., Tripathi R. C. Cloud computing: Comparison and analysis of cloud service providers–AWS, Microsoft, and Google. 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART). 281–285 (2020).

[2] Pahl C., Brogi A., Soldani J., Jamshidi P. Cloud container technologies: A state-of-the-art review. IEEE Transactions on Cloud Computing. **7** (3), 677–692 (2019).

[3] The Kubernetes Authors, Kubernetes, Kubernetes (2024). `https://kubernetes.io`.

[4] Pukach P., Hladun V. Using dynamic neural networks for server load prediction. Computational Linguistics and Intelligent Systems (CoLInS), Workshop. Vol. II. 157–160 (2018). `https://colins.in.ua/wp-content/uploads/2018/07/colins_2018_157_160.pdf`.

[5] Shivakumar S. K. Architecting High Performing, Scalable and Available Enterprise Web Applications. Amsterdam, Netherlands, Morgan Kaufmann (2015).

[6] Amazon Web Services, Inc., Amazon Elastic Compute Cloud (EC2). Overview of Amazon Web Services, Aug. 27, 2024, pp. 1–3. `https://aws.amazon.com/ec2/`.

[7] Bisong E. An overview of Google Cloud Platform services. Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners. Berkeley, CA, USA, Apress (2019).

[8] Collier M., Shahan R. Microsoft Azure Essentials: Fundamentals of Azure. Redmond, WA, USA, Microsoft Press (2016).

[9] Ranković T., Šiljić F., Tomić J., Sladić G., Simić M. Misconfiguration Prevention and Error Cause Detection for Distributed-Cloud Applications. 2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY). 000297–000302 (2024).

[10] Nobles C. Investigating cloud computing misconfiguration errors using the human factors analysis and classification system. Scientific Bulletin. **27** (1), 59–66 (2022).

[11] Menasce D. A. Load testing of Web sites. IEEE Internet Computing. **6** (4), 70–74 (2002).

[12] Hamilton J. D. Time Series Analysis. Princeton University Press (2020).

[13] Skiera B., Reiner J., Albers S. Regression Analysis. Handbook of Market Research. 299–327 (2022).

[14] Brockwell P. J., Davis R. A. Introduction to Time Series and Forecasting. New York, USA, Springer (2002).

[15] Hochreiter S., Schmidhuber J. Long short-term memory. Neural Computation. **9** (8), 1735–1780 (1997).

[16] Zhu X., Sobhani P., Guo H. Long short-term memory over recursive structures. ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning. 1604–1612 (2025).

[17] Mienye I. D., Swart T. G., Obaido G. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. Information. **15** (9), 517 (2024).

[18] Mathew A., Amudha P., Sivakumari S. Deep learning techniques: An overview. Advanced Machine Learning Technologies and Applications. 599–608 (2021).

[19] Van Houdt G., Mosquera C., Nápoles G. A review on the long short-term memory model. Artificial Intelligence Review. **53** (8), 5929–5955 (2020).

[20] Zell A. Simulation neuronaler Netze. Bonn, Germany, Addison-Wesley (1994).

[21] Sewak M., Sahay S. K., Rathore H. Assessment of the relative importance of different hyper-parameters of LSTM for an IDS. 2020 IEEE Region 10 Conference (TENCON). 414–419 (2020).

[22] Santos C. F. G. D., Papa J. P. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. ACM Computing Surveys. **54** (10s), 213 (2022).

[23] Ying X. An overview of overfitting and its solutions. Journal of Physics: Conference Series. **1168** (2), 022022 (2019).

[24] Jabbar H. K., Khan R. Z. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). Computer Science, Communication & Instrumental Devices. 163–170 (2014).

[25] Everitt B. S., Skrondal A. The Cambridge Dictionary of Statistics. Cambridge, Cambridge University Press (2010).

[26] Liang X., Wu L., Li J., Wang Y., Meng Q., Qin T., Chen W., Zhang M., Liu T.-Y. R-Drop: Regularized dropout for neural networks. 35th Conference on Neural Information Processing Systems (NeurIPS '2021). 10890–10905 (2021).

[27] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. **15** (1), 1929–1958 (2014).

[28] Taylor S. J., Letham B. Forecasting at Scale. The American Statistician. **72** (1), 37–45 (2018).

[29] Salinas D., Flunkert V., Gasthaus J., Januschowski T. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting. **36** (3), 1181–1191 (2020).

[30] Herzen J., Lässig F., Piazzetta S. G., Neuer T. et al. Darts: User-Friendly Modern Machine Learning for Time Series. Journal of Machine Learning Research. **23** (124), 1–6 (2022).

[31] Wilder B. Cloud Architecture Patterns: Using Microsoft Azure. CA, USA, O'Reilly Media (2012).

[32] Dagum E. B. Time series modeling and decomposition. Statistica. **70** (4), 433–457 (2010).

[33] Hodson T. O., Over T. M., Foks S. S. Mean squared error, deconstructed. Journal of Advances in Modeling Earth Systems. **13** (12), e2021MS002681 (2021).

[34] Ketkar N., Moolayil J. Introduction to PyTorch. Deep Learning with Python. Berkeley (2021).

[35] Imambi S., Prakash K. B., Kanagachidambaresan G. R. PyTorch. In Programming with TensorFlow, K. B. Prakash and G. R. Kanagachidambaresan, Eds. Cham, Switzerland, Springer, 87–104 (2021).

[36] Zamanlooy B., Mirhassani M. Efficient VLSI implementation of neural networks with hyperbolic tangent activation function. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. **22** (1), 39–48 (2014).

[37] Ali M. H. E., Abdel-Raman A. B., Badry E. A. Developing novel activation functions based deep learning LSTM for classification. IEEE Access. **10**, 97259–97275 (2022).

[38] Farzad A., Mashayekhi H., Hassanpour H. A comparative performance analysis of different activation functions in LSTM networks for classification. Neural Computing and Applications. **31** (7), 2507–2521 (2019).

[39] Fernandez A., Mali A. Stable and robust deep learning by hyperbolic tangent exponential linear unit (TeLU). Preprint arXiv:2402.02790 (2024).

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

[41] Narayan S. The generalized sigmoid activation function: Competitive supervised learning. Information Sciences. **99** (1–2), 69–82 (1997).

[42] Schmidt-Hieber J. Nonparametric regression using deep neural networks with ReLU activation function. The Annals of Statistics. **48** (4), 1875–1897 (2020).

[43] Ramachandran P., Zoph B., Le Q. V. Searching for activation functions. Preprint arXiv:1710.05941 (2017).

[44] Mercioni M. A., Holban S. P-Swish: Activation function with learnable parameters based on Swish activation function in deep learning. 2020 International Symposium on Electronics and Telecommunications (ISETC). 1–4 (2020).

[45] Javid I., Ghazali R., Syed I., Husaini N. A., Zulqarnain M. Developing novel T-Swish activation function in deep learning. 2022 International Conference on IT and Industrial Technologies (ICIT). 1–7 (2022).

[46] Nair V., Hinton G. E. Rectified linear units improve restricted Boltzmann machines. ICML'10: Proceedings of the 27th International Conference on International Conference on Machine Learning. 807–814 (2010).

# Вплив активаційних функцій на точність прогнозування серверного навантаження за допомогою нейронних мереж типу LSTM

Чабан О. М., Пукач П. П., Пукач П. Я., Гладун В. Р.

*Національний університет "Львівська політехніка", вул. С. Бандери, 12, 79013, м. Львів, Україна*

Постійне зростання кількості користувачів і їхніх запитів до серверів вимагає значних обчислювальних ресурсів для забезпечення швидкої обробки без затримок. Водночас навантаження на сервери є природно нерівномірним упродовж доби, тижня або місяця. Точне прогнозування необхідних ресурсів і динамічне управління їх розподілом є надзвичайно важливими, оскільки це може суттєво знизити витрати на обслуговування серверної інфраструктури без погіршення якості користувацького досвіду. У цьому дослідженні розглянуто вплив вибору активаційної функції на точність прогнозування за допомогою нейронних мереж типу Long Short-Term Memory (LSTM), застосованих до реальних даних запитів до серверів. Зібраний набір даних містить інформацію про вхідні запити, агреговану в інтервали по 20 хвилин протягом 16 діб. Було досліджено кілька активаційних функцій — зокрема ReLU, Swish, та Softplus — з використанням середньоквадратичної помилки (MSE) як основного показника ефективності. Кожну конфігурацію моделі навчали шість разів для забезпечення статистичної надійності, а результати бралися з одного з найбільш стабільних запусків. Отримані експериментальні результати свідчать, що вибір активаційної функції суттєво впливає на точність прогнозування: функції Swish і ReLU продемонстрували найменші значення MSE, зменшивши похибку на 6.6 – 12.3% та 10.5 – 16.3% відповідно порівняно з базовим варіантом. Хоча сигмоїдна функція забезпечила найнижче тестове значення втрат, подальший аналіз виявив хибність цього результату: модель систематично недооцінювала пікове навантаження, що призводило до занижених значень помилки, але водночас — до низької достовірності прогнозу динаміки реального навантаження. Отримані результати підтверджують гіпотезу про те, що вибір активаційної функції є критичним чинником для оптимізації моделей прогнозування навантаження серверів на основі LSTM.

**Ключові слова:** *прогнозування навантаження на сервер; машинне навчання; активаційні функції; нейронні мережі типу LSTM.*