"Comparison of Some Optimizers in Long Short-Term Memory Networks with an Application to Tigris River Water Imports"

Abstract:

The Long Short-Term Memory (LSTM) RNN architecture was developed to overcome the limitations of traditional RNNs in recognizing long-term dependencies in sequential input. LSTM networks and other neural network topologies rely heavily on optimizers for training. Every iteration of a gradient-based algorithm attempts to approach the minimizer/maximizer cost function by using the gradient's objective function information. Moreover, a comparative analysis of a number of optimizers is presented in this work, including FTRL, Nadam, Adagrad, Adadelta, SGD, RMSprop, and Adagrad. By directly affecting the objective function's convergence, the study emphasizes the crucial role optimizers play in improving gradient descent efficiency. This work fills a gap in the literature on choosing the best optimizers for time-series data by concentrating on water import statistics for the Tigris River. The results offer insightful information about how to choose optimizers wisely to reduce time complexity and increase accuracy. The results of the study show that these optimizers' performance is generally comparable when compared using the root mean square error (RMSE) criterion based on water data. However, Nadam produces RMSE values at 500 epochs that are lower than those of the other methods.

Keyword: Adam Optimizer, Long Short-Term Memory, Nadam Optimizer, SGD Optimizer, RMSprop Optimizer.

1. Introduction

Artificial neural networks (ANNs) are a class of computer systems that mimic and imitate the functions of the human brain to evaluate and handle complicated data in an adaptable way (see Figure 1). They may do massively parallel calculations that require less explicit statistical training, such as mapping, function approximation, classification, and pattern recognition. Furthermore, by employing several training algorithms, ANNs may recognize extremely complicated nonlinear correlations between predictor (independent) and

UNDER PEER REVIEW

outcome (dependent) variables. ANNs can generally be divided into two types based on their network architecture: recurrent and feedforward ANNs, each of which may have multiple subclasses. A popular ANN paradigm for pattern recognition, mapping, classification, and function approximation is feedforward. Every layer is totally connected to every other layer's neuron; there is no connectivity between neurons in the same layer. According to the term, information is sent from the input layer to the output layer using one or more hidden layers (Okut, 2021). For several learning issues involving sequential data, recurrent neural networks (RNN) have surfaced as an effective and scalable (ANN) model. Loops are incorporated into the hidden layer of information in these networks. Due to the multidirectional information flow made possible by these loops, the hidden state represents historical data held at a certain time step. These network types can therefore respond infinitely dynamically to sequential data. RNN is used by numerous apps, including Google's voice search and Apple's Siri. However, issues with vanishing or expanding gradients may arise in both standard and deep RNNs. In an RNN with activation functions, with the addition of more layers, the loss function's gradient approaches zero. If more layers of activation functions are added, the gradient loss function may become closer to 0 (vanish), maintaining the functions in their original form. This puts a halt to additional training and prematurely concludes the process. As a result, parameters only record immediate dependencies; Previous time step data may be lost. Consequently, the model chooses a poor response. Exploding gradients, the opposite problem, can happen because incorrect gradients can be unstable. As a result, errors significantly increase with each time step. Thus, the number of timesteps may be a limiting factor for backpropagation. The problem of vanishing/exploding gradients is addressed by Long Short-Term Memory (LSTM) networks, which were initially shown by (Okut, 2016). Apart from the RNN's hidden state, an LSTM block additionally has memory cells for storing prior knowledge and adds a sequence of gates known as input, output, and forget gates. These gates stop errors from disappearing or blowing up and enable extra modifications (to account for nonlinearity). The solution does not terminate too soon or lose earlier knowledge, and the outcome is more accurately predicted (Hochreiter, & Schmidhuber, 1997). Medical publications have published real-world LSTM uses. For instance, research ((Sher et al.,

UNDER PEER REVIEW

2023); (Yang et al., 2022); (Dogo et al., 2018)) employed various RNN variations for medical record classification and prediction.

The selection of optimizers, which are essential for increasing convergence and enhancing model correctness, has a significant impact on the performance of Long Short-Term Memory (LSTM) networks. Even though LSTM is frequently used in time-series analysis, little study has been done on the best way to choose deep learning optimizers for these networks. This disparity is especially noticeable in water resource management applications, where accurate and efficient models are required due to the complexity of the data.

The use of LSTM to examine Tigris River water import data is the main emphasis of this study. Through the evaluation of several optimizers, the study seeks to shed light on how they affect training time reduction and model performance. Key parameters including batch size, learning rate, and epoch count were carefully chosen to enable robust evaluation, and the dataset was split into 70% for training and 30% for testing.

By providing a thorough examination of optimizer performance, filling a significant gap in the literature, and offering useful recommendations for optimizer selection in LSTM applications, the research findings advance the discipline.

UNDER PEER REVIEW

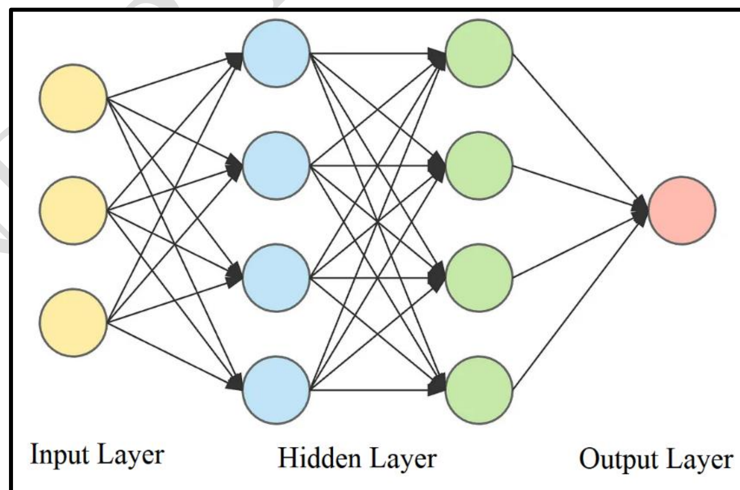

Figure 1. The structure of artificial neural networks.

2. Literature Review.

A specific kind of recurrent neural network (RNN) called Long Short-Term Memory (LSTM) networks is made to efficiently model sequential input. For time-series analysis, LSTM has gained popularity by resolving the drawbacks of conventional RNNs, especially the vanishing gradient issue. Applications needing temporal dependency modeling, such water resource management, can benefit from its long-term information retention. (Waqas, & Humphries, 2024)

Various studies have shown that optimizers in Long Short-Term Memory (LSTM) networks perform significantly differently. Other hyperparameters, such as the number of LSTM layers, were found to have less of an impact on the classification accuracy of LSTM models in the context of remote sensing applications, compared to the optimizer selection (Sher et al., 2023). More specifically, the Stochastic Gradient Descent (SGD) optimizer was found to be the most exceptional technique for LSTM models in the prediction of electricity usage. Furthermore, TAFEO, an improved equalization optimizer technique, outperformed existing intelligent optimization algorithms in enhancing the accuracy of LSTM models for reservoir identification (Yang et al., 2022). Dogo et al. provided a clear structural configuration for a "Convolutional Neural Network (ConvNet)". Seven of the most popular first-order stochastic gradient-based optimization techniques were compared and reviewed. that they tested the optimization algorithms in terms of convergence rate, precision, and error function using three publicly available image classification datasets that were selected at random. The following algorithms are tested: Adam, RMSProp, AdaDelta, (AdaGrad), Nadam, and Adamax. SGD with momentum (SGDm), with nesterov + momentum (SGDm+n), and vanilla SGD (vSGD) are also tested. Three datasets—"Cats and Dogs," "Natural Images," and "Fashion Mnist"—have been utilised. Compared to the other optimization procedures, the average experimental results indicated that "Nadam" achieved different levels of efficiency for each dataset (Dogo et al., 2018). Fatima stated thst the best optimizer for the neural network model. Using four unconnected datasets, they analyzed some of the most well-known optimization methods to see which optimizer offers the deepest A REVIEW OF COMPARISON OF GRADIENT DESCENT ALGORITHM BASED OPTIMIZATION TECHNIQUES. The most accurate, dependable, and performant neural network is PJAEE, 18 (4) (2021) 2723. Owing to its superior overall performance across all four Deep Neural Network models, to obtain the best accuracy, the

UNDER PEER REVIEW

Adam optimization strategy may almost always be applied to any classification model (Fatima, 2022). It was also determined that RMSprop was a reasonable alternative in three of the four datasets. Additionally, they noticed that the Adadelta and SGD optimizers had trouble producing sufficient results in.

3. Long Short-Term Memory

Two major shortcomings in vanilla RNN (see FIGURE 2, (Farrag, & Elattar, 2021)) were intended to be addressed by LSTM. Long-term reliance in RNNs is the first, and it is not advised for many applications. To put it simply, even though the past states' timestamps are extremely far from the present timestamp, the RNN cell's current state is nevertheless influenced by them. In many applications, like speech recognition and auto-translation, this is not reasonable. The second gradient appears and explodes. In order to adjust the weights during each training cycle, RNN learning methods rely on obtaining updates proportionate to the partial derivative of the loss (error) function concerning the current weight (Van Houdt et al., 2020). The issue is that this gradient will be vanishingly small, which will essentially prohibit the learning process from occurring by preventing the weight from increasing its value. LSTM introduces the gates concept to solve these issues. In an LSTM, there are three distinct types of gates: input, output, and forget. The sigmoid function is applied to the current input and the hidden state via the forget gate in the subsequent step to ascertain each of their respective contributions.

PEER REVIEW

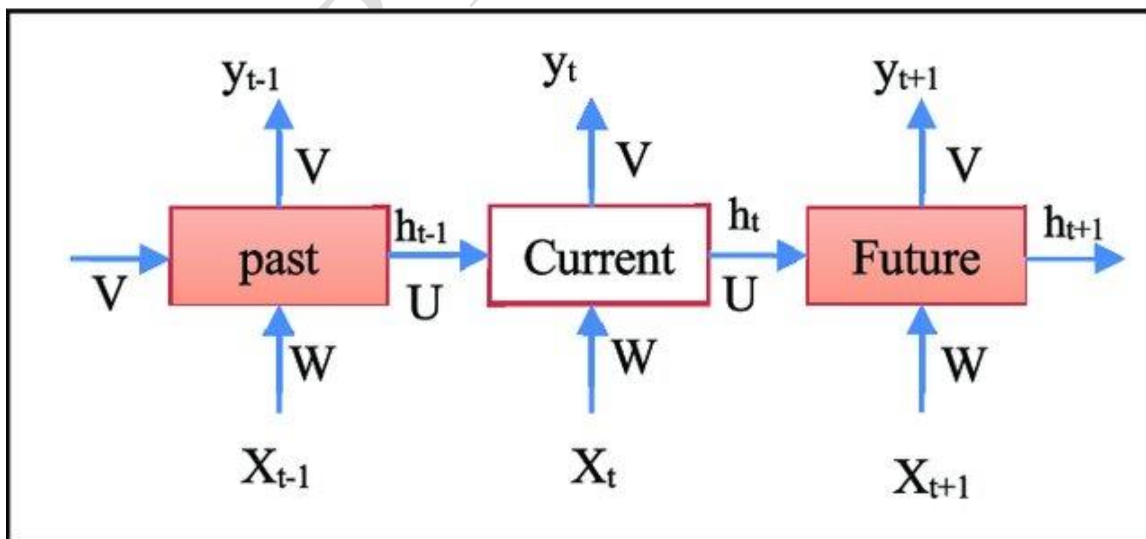

Figure 2. States of vanilla RNN unit in work.

condition of the LSTM cell, as expressed by equation (1). Thus, it is essential to resolving the issue of long-term dependency. The input gate combines the use of tach and sigmoid. Equations (2,3) illustrate how functions are applied to the input and hidden state to calculate the cell's current state. Equations (4,5) illustrate how the output gate determines the cell's subsequent hidden state using the output of the preceding two gates.The dataflow and relationships between the various gates across the LSTM cell are depicted in Figure 3 (Wen et al., 2017). As can be seen, an LSTM cell requires significantly more computations and has many more parameters than typical artificial neural structure.

$$i_t = \sigma\left(W_{\chi i \chi t} + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i\right) \quad (1)$$

$$f_t = \sigma\left(W_{\chi f \chi t} + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f\right) \quad (2)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tan h\,(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

$$o_t = \sigma\left(W_{\chi o}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o\right) \quad (4)$$

$$h_t = o_t \odot \tanh(C_t) \quad (5)$$

where:

i_t: Represents the activation of the input gate at time step t.

f_t: Represents the activation of the forget gate at time step t.

C_t: Represents the cell state at time step t.

o_t: Represents the activation of the output gate at time step t.

h_t: Represents the hidden state (output) at time step t.

$W_{xi}, W_{xf}, W_{xo}, W_{xc}$: Weight matrices associated with the inputs for the input gate, forget gate, output gate, and cell state, respectively.

$W_{hi}, W_{hf}, W_{ho}, W_{hc}$: Weight matrices associated with the hidden state h_{t-1} for the input gate, forget gate, output gate, and cell state, respectively.

W_{Ci}, W_{Cf}, W_{Co},: Weight matrices associated with the previous cell state C_{t-1} for the input gate, forget gate, and output gate, respectively.

b_i, b_f, b_o, b_c: Bias terms for the input gate, forget gate, output gate, and cell state, respectively.

σ: The activation function, often a sigmoid function.

tanh: The hyperbolic tangent activation function, used for updating continuous values.

UNDER PEER REVIEW

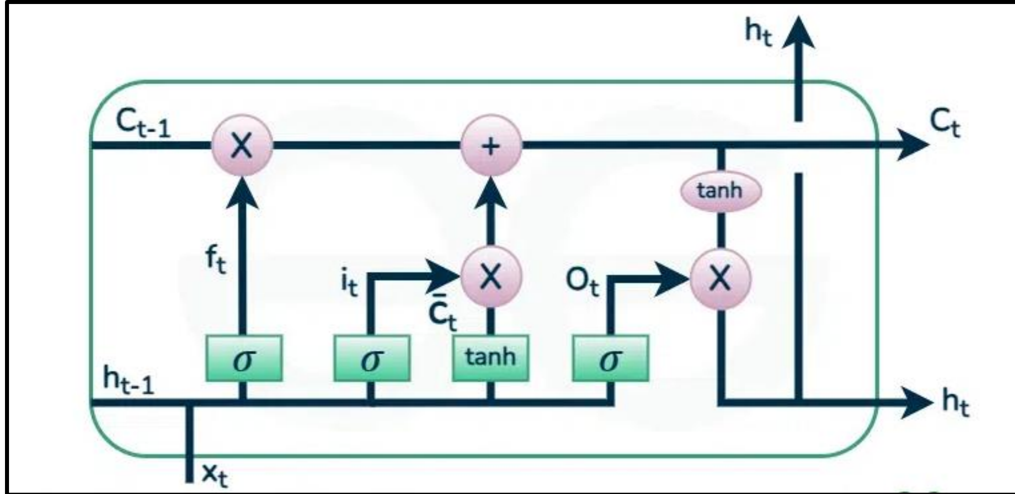

Figure 3. The Long Short-Term Memory (LSTM) cell.

.

3.1 Gradient descent optimization

One of the most widely used optimization methods is gradient descent, which is also the method most frequently used to optimize neural networks. Gradient descent is a method for by updating the parameters in the opposite direction as the cost function's gradient, ∇θJ(θ), with regard to the parameters, one can minimize a cost function J(θ) with the model parameter θ (Ruder, 2016). To reach a local minimum, the number of steps we take is determined by the learning rate. Gradient descent comes in three flavors, each with a different amount of data we employ in order to determine the object function's gradient. The three methods are mini-batch gradient descent, batch gradient descent, and stochastic gradient descent.gradient descent in the stochastic and mini-batch modes. Note that good convergence is not guaranteed by a standard mini-batch gradient descent. presents two problems that require attention and a few hurdles. First, picking the right course of study might be challenging. Learning rate schedules are designed to adjust how quickly students learn while receiving instruction. Minimizing non-convex error functions is the second goal for neural networks. Dauphin et al. claim that saddle points, not the local minimal, are the true cause of the issue. We don't go into the algorithms in this section; instead, we provide a quick overview of different gradient descent optimizers to help you understand how to change the learning rate (Gong et al., 2020).

.

3.1.1 Adagrad Optimizer

UNDER PEER REVIEW

An algorithm for gradient-based optimization called Adagrad adjusts the learning rate according to the parameters. Adagrad significantly enhanced SGD, and Dean et al. utilized it at Google to train massive neural networks Additionally, Adagrad was utilized by Pennington et al. to train Word Inclusion. At time step t, the gradient of the object function, denoted by gi,j, needs to be set in relation to parameter θi (Dean et al., 2012).

$$g_{t,i} = \nabla\theta J(\theta) \quad (6)$$

The SGD update for every parameter θi at every time step t therefore becomes

$$\theta_{t+1}, i = \theta_{i,j} - \alpha . gt, i \quad (7)$$

Using the previously computed gradients for each parameter θi, Adagrad adjusts the general learning rate in its update algorithm at each time step t:

$$\theta_{t+1}, i = \theta_{i,j} - \frac{\alpha}{\sqrt{G_{t,ii} + \epsilon}} . gt,\ \ i \quad (8)$$

Each diagonal element i, where i is the total of the squares of the gradients with respect to θi up to time step t, and Gt ∈ Rd×d is a diagonal matrix with a smoothing term that avoids division by zero. Adagrad's default learning rate is 0.01.

3.1.2 Adadelta Optimizer

Adadelta is a supplement to Adagrad that monitors a decreasing learning rate. The running average E[g 2]t, time-averaged. Then, the only variables influencing step t are the previous average and the current gradient (Zeiler, 2012). The vector of Adadelta as per the formula:

$$E[g^2] = \gamma E[g^2]t - 1 + (1 - \gamma) g_t^2 \quad (9)$$

We determine that γ should be roughly equal to the momentum term, or 0.9. For clarity, we will now recast our vanilla SGD update in terms of the parameter update vector Δθt:

$$\Delta\theta_t = -\alpha . g_{t,i} \quad (10)$$

$$\Delta\theta_{t+1} = \theta_t + \delta\theta_t \quad (11)$$

Consequently, the shape of the Adagrad parameter update vector that we previously acquired is as follows:

UNDER PEER REVIEW

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot g_t \quad (12)$$

The following shorthand criteria can be used in place of the gradient's RMSE criterion, which acts as the denominator:

$$\Delta\theta_t = -\frac{\alpha}{RMSE[g]_t} \quad (13)$$

The authors state that the update's units, which ought to correspond with the parameter's hypothetical units, are different from those in SGD, Momentum, or Adagrad. First, we define another exponentially declining average, but this time it comes from squared parameter updates instead of squared gradients, in order to do this. Compute the average, but this time use squared parameter updates rather than squared gradients.

$$E[\Delta\theta^2]_t = \gamma E\Delta\theta^2{}_{t-1} + (1-\gamma)\Delta\theta^2{}_t \quad (14)$$

Here is the RMSE of the parameter updates:

$$RMSE[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \quad (15)$$

he Adadelta update rule is ultimately produced by substituting the RMS of parameter updates for the learning rate in the preceding update rule:

$$\Delta\theta_t = \frac{RMSE[\Delta\theta]}{RMSE[g]_t} \quad (16)$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \quad (17)$$

A default learning rate is eliminated from the update rule by Adadelta, so we don't even need to define one.

3.1.3 RMSprop Optimizer

Geoff Hinton proposes the adaptive learning rate approach, or RMSprop. RMSprop and Adadelta are two solutions designed to address Adagrad's declining learning rates. MSprop has to be updated.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \quad (18)$$

$$\Delta\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t}} g_t \quad (19)$$

Additionally, RMSprop divides the learning rate by the average of squared gradients, which decays exponentially. Hinton suggests that γ be set to 0.9. Furthermore, the learning rate n should have an acceptable default value of 0.001.

UNDER PEER REVIEW

3.1.4 Adam Optimizer

Furthermore, Adam is used to calculate adaptive learning rates for each parameter. Adam keeps track of both the exponentially declining average of previous gradients (mt) and the exponentially declining average of previous squared gradients (vt), like Adadelta and RMSprop. momentum is the process of obtaining gradients at timestep t with regard to the stochastic object (Kingma, & Ba, 2014).

$$g_t \leftarrow \nabla\theta f_t(\theta_{t-1}) \quad (20)$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad (21)$$

$$u_t = \beta_2 u_{t-1} + (1-\beta_2) g_t^2 \quad (22)$$

The estimations of the values mt and ut, respectively, stand for the mean (first moment) and uncentered variance (second moment) of the gradients. The writers of Adam note that mt and vt are skewed towards zero since they are initialized as vectors of 0s. This bias is particularly noticeable in the early time steps and at low decay rates (i.e., when β1 and β2 are relatively close to 1). To counter these biases, they compute bias-corrected first and second moment estimations:

$$m_t^{new} = \frac{m_t}{1-\beta_1^t} \quad (23)$$

$$u_t^{new} = \frac{u_t}{1-\beta_2^t} \quad (24)$$

The parameters are then updated using these to create the Adam update rule, just as we've seen with Adadelta and RMSprop:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{u^{new} + \epsilon}} m_t^{new} \quad (25)$$

The authors suggest that the default values for β1 and β2 be 0.9 and 0.999, respectively. Empirically, they show that Adam outperforms other adaptive learning-method algorithms and does well in real-world circumstances.

3.1.5 Adamax

UNDER PEER REVIEW

Adamax is a variant of Adam that is based on the infinite norm . The Adamax optimizer is found in Section 7 of the Adam paper. Each weight scale in Adam has an update algorithm that scales its gradients, which are inversely connected to a (scaled) L 2 norm of each gradient's distinct past and current. A L p norm-based update rule is a generalization of the L 2 norm-based update rule. These versions get numerically unstable for large p. But in the unique scenario where we let p to go to ∞, an unexpectedly straightforward and reliable solution appears (Duchi et al., 2011). Updated first moment estimate that is biased:

$$m_t \leftarrow \beta_1 . m_{t-1} + (1 - \beta_1) \quad (26)$$

Revise the infinite norm with exponential weight:

$$u_t \leftarrow max(\beta_2 . u_{t-1}, |g_t|) \quad (27)$$

Modify the parameters:

$$\theta_t \leftarrow \theta_{t-1} - \left(\frac{\alpha}{1 - \beta_1^t}\right) . m_t / u_t \quad (28)$$

3.1.6 Nadam Optimizer

The Nesterov Adam optimizer is another term for Nadam. Just as Adam is basically an RMSprop with momentum, Nadam is basically Adam RMSprop with Nesterov momentum (Kim & Kim, 2017). From

$$g_t^{\wedge} \leftarrow \frac{g_t}{1 - \Pi_i^t = 1\beta_{1i}} \quad (29)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (30)$$

$$m_t^{\wedge} = \frac{m_t}{1 - \prod_{i=1}^{t+1} \beta_{1i}} \quad (31)$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2 \quad (32)$$

$$u_t^{\wedge} = \frac{u_t}{1 - \beta_2^t} \quad (33)$$

Additionally, to the update of the momentum vector for the upcoming timestep β_{t+1}mct, the vector m also includes the gradient update for the present timesteps gbt.

$$m_{t-1}^{\wedge} \leftarrow (1 - \beta_1) . g_t^{\wedge} + \beta_{t+1} . m_t^{\wedge} \quad (34)$$

Next, alter Nadam's parameter as follows:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{u_t^{\wedge} + \epsilon}} m_{t-1}^{-} \quad (35)$$

UNDER PEER REVIEW

Nadam's default settings are α = 0.002, β1 = 0.9, and β2 = 0.999.

4. Application

In this study, time series data representing Tigris River Water Imports were collected from the website of the Authority of Statistics and Geographic Information System (ASGIS) in Iraq. The data spans from 1980 to 2021 and includes 168 quarterly observations, detailing the water flow in million cubic meters. Understanding the patterns and trends of water resource availability is crucial for sustainable development, policy planning, and efficient water management in Iraq. As the main supplier of water for domestic, industrial, and agricultural purposes, the Tigris River is vital to the nation's economy and way of life. By identifying seasonal fluctuations, long-term patterns, and possible anomalies, this dataset analysis helps policymakers make well-informed decisions regarding the management and allocation of resources. Given the growing problems with water scarcity in the area, it is crucial to employ predictive models, such LSTM, on this dataset to get insights into future patterns of water import.

The `auto.arima` function from the `forecast` package in R was used to fit an Autoregressive Integrated Moving Average (ARIMA) model. The results indicated that the appropriate model for this data is ARIMA(3,0,0) with RMSE equal to (19.145).

The network was implemented using the statistical programming language R, leveraging two well-known machine learning packages in Python, TensorFlow and Keras. The network was trained using 70% of the data for training and 30% for testing with specific parameters, including a learning rate of 0.001, a batch size of 32, 25 units in the LSTM layer, and varying epochs (10, 50, 100, 250, and 500). A seed value of 123 was set for reproducibility. The activation function used was Rectified Linear Unit (ReLU), and various optimizers were explored as: Adam, SGD, RMSprop, Adagrad, Adadelta, Adamax, Nadam, and FTRL ((Arnold et al., 2024); (Rimal, Y., & Sharma, 2024);).

After implementing the program, the results were obtained and are presented in Table 1.

Table 1: Model performance of LSTM evaluated based on optimizers and epochs, measured by RMSE

Epochs

UNDER PEER REVIEW

Optimizers	10	50	100	250	500
Adam	15.605	10.108	7.113	6.693	**6.688**
SGD	16.996	16.860	16.614	15.646	**14.127**
RMSprop	15.291	10.452	7.192	7.360	**6.702**
Adagrad	16.862	16.501	16.188	15.609	**14.962**
Adadelta	17.036	17.028	17.016	16.978	**16.910**
Adamax	16.208	13.422	10.799	7.083	**6.890**
Nadam	15.716	10.059	7.042	6.686	**6.608**
Ftrl	**16.574**	16.661	16.721	16.837	16.957

The RMSE results for various epochs and optimizers are presented in Table 1, revealing the following insights:

Comparison with ARIMA model:

The RMSE value of ARIMA (3,0,0) model was 19.145, which is higher than all values obtained by LSTM network with optimizers like Adam, Nadam, and RMSprop at a larger number of epochs.

Adam:

Adam appears to perform excellently across all epochs, with the RMSE gradually decreasing as the number of epochs increases, until it reaches its lowest value at 500 epochs (6.688). The final RMSE value for the Adam model is among the lowest in the table, making it among the best performing optimizers.

SGD:

SGD shows gradual improvement in RMSE, but overall performance is lower than Adam, with RMSE decreasing slowly to 14.127 at 500 epochs. Despite continuous improvement, SGD still underperforms Adam and other optimizers.

RMSprop:

The table shows a significant improvement in RMSE between 10 and 100 epochs, but it becomes more stable after 250 epochs, reaching 6.702 at 500 epochs. Although RMSprop is not the best, it comes close to Adam.

Adagrad:

UNDER PEER REVIEW

Adagrad shows a slight improvement in RMSE with increasing epochs, but the RMSE values remain high compared to other optimizers, reaching 14.962 at 500 epochs.

Adadelta:

The RMSE of Adadelta appears to remain relatively constant across all epochs, indicating no significant improvement as the number of epochs increases. Adadelta shows no real advantage over other optimizers, as its RMSE remains much higher than Adam.

Adamax:

Adamax shows a significant improvement in RMSE with increasing epochs, reaching 6.890 at 500 epochs, which is very close to Adam's performance. It shows strong performance, especially at a higher number of epochs.

Nadam:

Similar to Adam, Nadam shows excellent performance with a very low RMSE at 500 epochs (6.608). In fact, Nadam outperforms Adam in several cases, making it a particularly strong choice.

Ftrl:

Ftrl shows very slight improvement with increasing epochs, but its performance is still poor with an RMSE of 16.957 at 500 epochs.

Based on the analysis, **Adam** and **Nadam** are the most effective optimizers for this LSTM neural network, consistently achieving the lowest RMSE values across different training epochs, with Nadam slightly outperforming Adam in several cases. **RMSprop** also showed strong performance, coming close to Adam and Nadam. On the other hand, **SGD**, **Adagrad**, **Adadelta**, and **Ftrl** performed less effectively, achieving significantly higher RMSE values, suggesting that they may not be the best choices for this particular application.

The LSTM model performed well with Adam, Nadam, RMSProp, and Adamax optimizers, while it performed poorly with SGD, Adagrad, and Adadelta, according to our study results, which are in line with many previous studies. For example, Sher et al. (2023) showed that Adam, Nadam, and RMSProp were among the best, while SGD, Adagrad, and Adadelta performed unsatisfactorily, which is consistent with our results. Similarly, Dogo et al. (2018) found that Nadam was the most effective, while AdaDelta performed the worst, which is consistent with our results. Another study by Fatima (2020) showed that

UNDER PEER REVIEW

Adam, Nadam, RMSProp, and Adamax were the best at improving the performance of the LSTM model, which is largely consistent with our results. Thus, our results are largely consistent with the results from previous studies, which strengthens the reliability and effectiveness of Adam and Nadam optimizers.

Figure 4 illustrates the behavior of loss with changing epochs, depicting a continuous improvement as epochs increase. The Nadam optimizer's loss behavior across 500 epochs is depicted in the graph. During the first 50 epochs, there is a noticeable drop in loss, indicating how effectively Nadam optimizes the model. The deterioration slows down between 50 and 250 epochs, suggesting that the optimizer was able to improve the model's performance. The loss stabilizes at a relatively low value after 250 epochs, demonstrating Nadam's resilience and dependability in producing the best training outcomes. This behavior supports Nadam's efficacy for LSTM models and is consistent with the low RMSE values it obtained, as indicated in Table 1.

UNDER PEER REVIEW

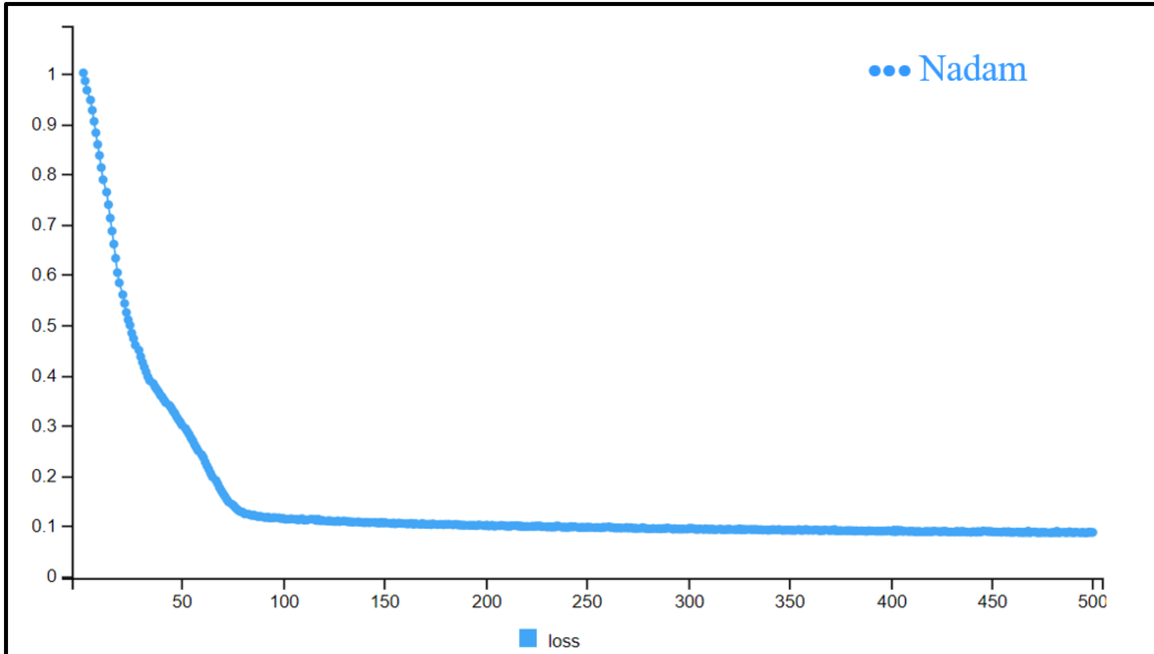

Figure 4. Mean Squared Error (MSE) of the loss function for the training data across varying epochs.

Figure 5 illustrates the estimation behavior for both the training and testing datasets. The graph shows the difference between the predictions made by the LSTM model using the Nadam optimizer (red line) and the actual water imports from the Tigris River (blue line). The training (1980–2010) and testing (2010–2020) phases are separated by a dashed line on the x-axis, which displays the time period (1980–2020). The accuracy of the model and Nadam's efficacy are demonstrated by the close correspondence between the actual and anticipated values. Nadam's dependability in time-series forecasting is confirmed by the little variations that occur at high peaks and falls, but overall, the predictions hold up well throughout both periods.

UNDER PEER REVIEW

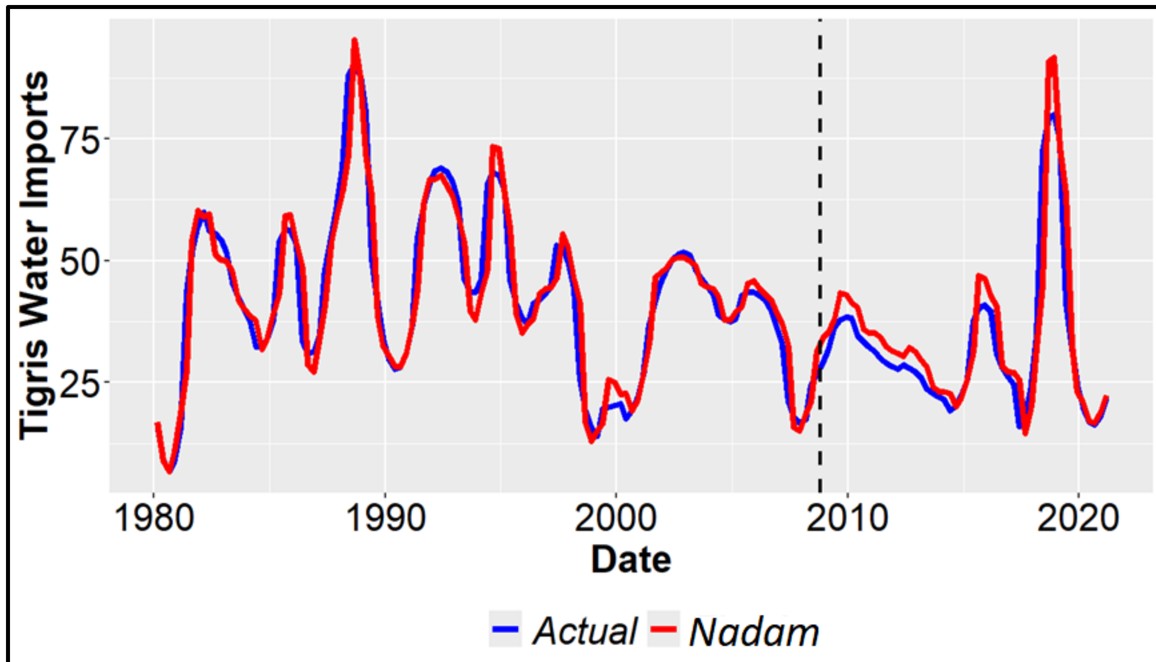

Figure 5. Actual and estimated Tigris River water imports.

The researchers notice that the estimation behavior is close to the real one, which indicates that the results are good.

In summary, this study demonstrates the effectiveness of using data to predict Tigris River water imports through an LSTM neural network model, especially when the model is optimized using algorithms such as Adam, Nadam, RMSProp, and Adamax. The results indicate that these optimizers provide accurate predictions and closely follow the actual trends of the data, making them suitable options for modeling complex temporal data such as water imports. This study is not only in line with previous research, but also contributes to understanding best practices for improving LSTM models in hydrological forecasting.

5. Conclusion

In this study, the LSTM neural network model was applied to the time series data of Tigris River water imports, with the aim of accurately predicting water import levels over time. The results showed that using optimizers such as Adam, Nadam, RMSprop, and Adamax was the most effective in improving the model performance, as these optimizers provided the lowest RMSE values across different training cycles (epochs), reflecting the model's ability to capture complex patterns in temporal data.

UNDER PEER REVIEW

Compared to the traditional ARIMA model, the LSTM model showed superior performance using the aforementioned optimizers, as the RMSE values were significantly lower, enhancing the effectiveness of LSTM in dealing with complex temporal data. On the other hand, optimizers such as SGD, Adagrad, and Adadelta showed less effective performance, indicating that these optimizers may not be the optimal choice for similar applications.

The results of this study are consistent with many previous studies, which enhances the reliability of using Adam, Nadam, RMSprop, and Adamax optimizers in improving LSTM models. In addition, the study highlights the importance of choosing appropriate enhancers to improve the performance of predictive models, especially in applications that require high accuracy in temporal predictions.

In the future, further research could include exploring the impact of more enhancers or incorporating other deep learning techniques to improve the accuracy of predictions. The impact of increasing data size and diversity on the performance of the LSTM model could also be studied to improve its ability to generalize and predict more accurately in different conditions.

Disclaimer (Artificial intelligence)

Author(s) hereby declare that NO generative AI technologies such as Large Language Models (ChatGPT, COPILOT, etc.) and text-to-image generators have been used during the writing or editing of this manuscript.

UNDER PEER REVIEW

REFERENCES:

Arnold, C., Biedebach, L., Küpfer, A., & Neunhoeffer, M. (2024). The role of hyperparameters in machine learning models and how to tune them. *Political Science Research and Methods*, *12*(4), 841-848.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. (2012). Large scale distributed deep networks. *Advances in neural information processing systems*, *25*.

Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., & Aigbavboa, C. O. (2018, December). A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 international conference on computational techniques, electronics and mechanical systems (CTEMS)* (pp. 92-99). IEEE.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Fatima, N. (2020). Enhancing performance of a deep neural network: A comparative analysis of optimization algorithms.

Farrag, T. A., & Elattar, E. E. (2021). Optimized deep stacked long short-term memory network for long-term load forecasting. *IEEE Access*, *9*, 68511-68522.

Gong, D., Zhang, Z., Shi, Q., van den Hengel, A., Shen, C., & Zhang, Y. (2020). Learning deep gradient descent optimization for image deconvolution. *IEEE transactions on neural networks and learning systems*, *31*(12), 5468-5482.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

Kim, J., & Kim, H. (2017, February). An effective intrusion detection classifier using long short-term memory with gradient descent optimization. In *2017 International Conference on Platform Technology and Service (PlatCon)* (pp. 1-6). IEEE.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Okut, H. (2016). Bayesian regularized neural networks for small n big p data. *Artificial neural networks-models and applications*, *28*.

Okut, H. (2021). Deep learning for subtyping and prediction of diseases: Long-short term memory. *Deep Learning Applications*.

Rimal, Y., & Sharma, N. (2024). Hyperparameter optimization: a comparative machine learning model analysis for enhanced heart disease prediction accuracy. *Multimedia Tools and Applications*, *83*(18), 55091-55107.

Sher, M., Minallah, N., Ahmad, T., & Khan, W. (2023). Hyperparameters analysis of long short-term memory architecture for crop classification. *International Journal of Electrical and Computer Engineering (Ijece)*.

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

Waqas, M., & Humphries, U. W. (2024). A critical review of RNN and LSTM variants in hydrological time series predictions. *MethodsX*, 102946.

Wen, W., He, Y., Rajbhandari, S., Zhang, M., Wang, W., Liu, F., ... & Li, H. (2017). Learning intrinsic sparse structures within long short-term memory. *arXiv preprint arXiv:1709.05027*.

Yang, F., Xia, K., Fan, S., & Zhang, Z. (2022). Equalization Optimizer-Based LSTM Application in Reservoir Identification. *Computational Intelligence and Neuroscience*, *2022*.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

UNDER PEER REVIEW