International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

COMPARATIVE EVALUATION OF STANDARD AND OPTIMIZED BACKPROPAGATION NEURAL NETWORKS IN RAINFALL PREDICTION

Vertika Shrivastava[1], Dr.Sanjeev Karmakar[2]

[1.] Department of Computer Applications, Bhilai Institute of Technology, Durg, Chhattisgarh, India.

[2.] Department of Computer Applications, Bhilai Institute of Technology, Durg, Chhattisgarh, India.

[1.]mail2vertika@gmail.com, [2.]dr.karmakars@gmail.com

Abstract

Rainfall needs to be predicted with accuracy so that there can be good water resource management, agricultural planning and mitigation of the disasters etc. However, with the messy nature of meteorological data, the simplistic methods of forecasting usually used are not able to replicate the non-linear and complex patterns that these tend to carry. This paper examined optimized BPNNs in rainfall forecasting and compare them with as well as improved BPNN models. Other methods like, Momentum, adaptive learning, learning rate tuning were used to improve the performance of the BPNNs. To guarantee robustness and cross-conditions generality, the models used in the following bigger study were trained and tested on a wide term historical weather data, over a course of 30 years (1990-2023) of a Mahanadi River Basin that is located in the state of Chhattisgarh in India. The training was done with 10 neural network architectures and a model with 10 optimization strategies. They were compared with performances of a baseline (standard BPNN) measured in such terms as Mean Squared Error (MSE), root mean square error (RMSE), and correlation coefficients. The result of the study was proved by the dominating predictive accuracy and speedy computation time using all optimized BPNNs, that is, by available Stochastic Gradient Descent (SGD) with DyNAM and momentum in comparison with the baseline model. These are the improvements in the speed of convergence and reduction in rates of prediction error. The research shows a promising use of confined use of the neural network in rainfall forecasting dragon architecture. The results we obtained promote the adoption of advanced optimisation techniques to train neural networks and therefore do better in terms of reliability and on-time rainfall forecasts. In a further research, we are going to pay attention both to adding more variables of meteorological character and experiment with alternative architecture of neural network to enhance a greater improvement on the forecasts.

Keywords: Rainfall Prediction, Backpropagation Neural Networks (BPNNs), Optimization Techniques, Machine Learning in Meteorology, Forecasting Accuracy

Received: July 28, 2025

1.1. Introduction Background and Motivation

Rainforecasting inarguably is the most significant and valuable task in meteorology and its specific applications are in the agricultural and water resources management as well as natural disaster preparedness. Rainfall forecasting helps rescind the negative impacts of floods, drought, and other weather risks. The computation of rainfall has traditionally been performed based on statistic and physical models. However, despite the fact that such models are helpful, they struggle with accurately describing intricate, nonlinear and chaotic processes of the meteorological process.

In the meantime, the phenomenon of machine learning, in particular, in neural networks has left opportunities to increase the accuracy and reliability of rainfall forecasting systems [133133]. In this paper we examine how optimized neural network architectures, i.e. backpropagation neural networks(BPN), singular methods or hybrids could be utilized to give superior predictive capability in rainfall forecasting. The methods of optimization under consideration are not only supposed to continue with meteorological predictions, but also to be a service to the machine learning literature in general, in that they provide not only a solution to the existing weakness of regular BPN, but also a contribution to the literature and to the future of the field in general.

BPN are also very common when data follows non linear trends. They in their simplest form are however prone to have pitfalls as they are slow to converge and have inaccurate predictions [4]. A lot of optimization methods have been suggested to address these difficulties. They are one of the simplest building blocks of enhancing the efficiency of training and generalization of the models along with the adaptive learning rates, the momentum-based updates, and regularization procedures, etc.

Neural networks are not new in their application in the prediction of rainfall. Learning capacities of multilayer perceptrons (MLPs) and feedforward neural networks (FNNs) to acquire complex patterns in the meteorological data had been demonstrated in a number of pioneering papers [567]. However, these models were given to robustness problems concerning overfitting and increased training time. In a bid to surmount these constraints, subsequent research suggested more advanced architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs) which can model both space and time, respectively.

Although new inventions of them are gradually changing the scene in those bigger models, BPNNs continue to earn their own applicability due to their simplicity in design and survivability. Meanwhile other techniques to enhance the performance of BPNNs have also been proposed, and those include Optimization (e.g. adaptive learning rate that scales the step size in gradient descent during the training period, momentum that smooth out the learning path by folding in the past gradient and regularization methods (L1 and L2 penalties) to prevent over fitting) [8].

Received: July 28, 2025

Despite the fact that such optimization strategies have played a successful role in other areas in the past, little or no research will examine their applicability in terms of rainfall forecasting. The proposed research will address this gap in the body of knowledge by strive to provide a comprehensive evaluation of the effect of different optimization techniques on the performance of BPN when used in rainfall prediction activities.

1.2. The Study Objectives

The general purpose of the current research is to compare the working of optimized BPNNs with normal BPNNs working in rainfall prediction. Its precise intentions are:

[1]. Application of Optimization Strategies: To apply different optimization techniques like, adaptive learning rates, momentum and regularization in the BPNNs training.

[2]. Predictive Accuracy: To check the predictive accuracy of both the regular and improved BPNNs in terms of performance measures like MSE, RMSE and correlation among others.

[3]. Convergence and Efficiency Analysis: To analyze the speed of convergence and effectiveness of computations of the models that are optimized to the standard models.

[4]. Generalization Performance: To evaluate the generalization performance of models based on a wide-ranging dataset that contains a historical weather dataset of various geographic locations.

[5]. Comparison to Traditional Methods: The Optimized BPNN models will be compared to the traditional statistical and physical models developed by prior researchers that are applied in prediction of rainfall.

1.3. Methodology

In order to fulfil these goals, we have used systematic approach which involves collection of data, model usage, training and testing. The important actions involve:

[1]. Data Collection: Gathering of past weather data using the authoritative data sources such as the meteorological stations and satellites, as well. The dataset will comprise variables like temperature, humidity, wind speed and past records of rainfalls.

[2]. Data Preprocessing: The preprocessing of data is to manage the missing values, normalize the inputs, and divide the dataset into training, validation, and test split.

[3]. Model Implementation: Standard and optimized models of BPNNs are to be implemented by utilizing an appropriate programming framework written in Python language. The optimization procedures will be incorporated during the training.

[4]. Training and Validation: The training of the models using the training set as well as the tuning of the hyperparameters using the validation set. The training process will consist in test-tuning the various learning rates, momentum values, and regularization parameters.

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[5]. Evaluation: Performing analysis of performance of the models on the test set according to the determined performance measures. Comparative analysis will be done in order to test the developments which have come in after the optimization techniques.

[6]. Statistical Analysis: Conducting statistical analysis to identify the value of the significant differences that are possibly found between the model performances

1.4. Significance of the Study

The study is important in more ways than one. First, it will aid the existing works that attempt to enhance the accuracy of rainfall prediction, which is crucial to achieve proper management of water resources and prevention of the disaster. Second, this study can help understand which optimization methods work best on BPNNs and can thus be used on any other application in general and meteorology in particular. Third, the contrasts made with the conventional models provide the synoptic view of strengths and weaknesses of the methods based on the neural networks in predicting rainfall.

1.5. Paper Structure

The rest of the paper is organized in the following way:

- Section 2: Related Work: In this section, papers published in this field are examined to provide a review of the current state of the literature related to the topic of the present research, such as the use of optimisation algorithms and neural networks to forecast rainfall.

- Section 3: Methodology: It involves the number of data gathering and data preprocessing, model execution, and model evaluation.

Section 4: Results and Discussion: In this section, the results of the experiment will be provided, i.e. the measures of performance of both the usual and improved models will be given, and the findings will be discussed.

- Section 5: Conclusion and Future Work: In it, the key findings are summarized, the implications of the research are discussed and the ways of further work are outlined.

2. Literature Review

The question of how to improve the accuracy of neural networks is very important. There are several approaches to address this problem, including data augmentation [9] enhancing the mathematical model of neural networks [10], adding the complement neural network [11], and so on. While improving neural network accuracy is a problem that is addressed by all of these techniques, not all models can make use of them. As a result, general methods exist to improve neural network training. One such instance is the optimization of the loss function. The main problem in neural networks is a fall to the global minimum. Initially, the least value was sought using stochastic gradient descent (SGD). [12]. In contemporary neural networks, such a first-order optimizer is still frequently employed and provided a hybrid model for monthly rainfall forecasting in Darjeeling, India, that combines wavelet techniques with artificial neural network (ANN) technology. An analysis was conducted comparing ANN and WNN. They used monthly

Received: July 28, 2025

rainfall, minimum and maximum temperature data, and rainfall at Darjeeling rain gauge station for a period of 74 years, from 1 January 1901 to 1 September 1975. 60% of the data from the first 44 years were utilized for the calibration procedure, while 40% of the data from the next 26 years were used for validation. According to this study, the efficiency index for WNN models was higher than 94%, whereas for ANN models it was just 64%. [13]

Rainfall prediction for the Andhra Pradesh (AP) state (INDIA) was done by *Kavitha Rania et al. (2014)* using artificial neural networks (ANN). The weights of the artificial neural network created for rainfall prediction were trained using a novel heuristic technique called Teaching Learning Based optimization (TLBO). A comparative analysis was conducted between the classical backpropagation learning approach and the modified classical TLBO, or mTLBO. The Indian Institute of Tropical Meteorology (IITM), located in Pune, India, provided the monthly rainfall (mm) data for Coastal Andhra. 1692 monthly measurements from 1871 to 2011 made up the data set. The simulated outcomes demonstrate that, when applied to the studied datasets, ANN-mTLBO outperforms ANN-BP [14]. The work developed an optimised rainfall predictor using Feed Forward -Back Propagation (FFBP) based ANN and Genetic Algorithm (GA) to obtain optimised result. Their objective was to analyse the four-months (June, July, August, and September) of rainfall data of 30 years from 1982-2012 in Goa, India. Promising results were obtained for GA approach than Artificial Neural Network (ANN) alone. [15]

The work employed artificial neural networks (ANNs) with the dynamic nonlinear addiction of growth and decay to input predictors to comprehend the location, the mesoscale motion, the height, and the time of day for freezing stages. The average long-term growth and decay patterns may be accurately replicated by the ANN, allowing their climatology to be analysed for all possible forecasting combinations. It was more difficult to predict in real time due to the low intrinsic predictability of growth and decline, but it significantly improved when persistence, growth and decay, and precipitation intensity were applied to the predictors in the recent past [16].

The work aimed to provide a method of forecasting the frequency and non-occurrence of precipitation in La Trinidad, Benguets, based on a variety of historical meteorological parameters. Five techniques have been employed to develop prediction models for the weather data collection: neural networks, K-Nearest neighbors, Gaussian vector support machines, fine decision trees, and linear discriminants. A poor model selection can no longer increase predictions [17].

A framework for an effective Early Warning System (EWS) using machine learning approaches for extremely brief periods of heavy rainfall was proposed. When it is anticipated that the heavy rain warning criterion will be met, the EWS issues an alarm signal in three hours. The authors created a discretization technique that selectively converts continuous input variables into nominal variables. The meteorological data produced by automated weather stations is pre-processed using principal component analysis and selective discretization. [18] and demonstrated the technique by examining five different classification algorithms and rainfall data from widely used meteorological data sets. Two datasets have been trained and analyzed: one is an

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Australian meteorological dataset, while the other is a non-conclusive dataset of weather data from Delhi, India. [19] By utilizing an Indian dataset the work presented a Multiple Linear Regression (MLR) prediction model. The data inputs contain a wide range of meteorological parameters that are utilized to improve the accuracy of precipitation forecasts. Included among the parameters for validating the proposed model are accuracy, correlation, and mean square error (MSE). [20]

Ahmed, Kamal, et al. used Multi-Model Ensembles (MMEs) to reduce simulation/projection un-certainty of GCM. This paper would categorize the number of op trial number of CMMs that has to be used when it comes to integrating an MME based on the performance evaluation of MMEs that are developed by using machine learning (ML) techniques. The ML-Algorithms developed to carry out the study included artificial neural network (ANN), K-Nearest Neighbour (KNN), Support Vector Machine (SVM), and Relevance Vector Machine (RVM). Thirty-six intercomparison model coupled step 5 gain control model (GCM) was employed to calculate precipitation (P), maximum (Tmax) and low (Tmin) temperatures in Pakistan. The ma-chine learning algorithms were used in this analysis. [21] Basha, CmakZeelan, et al. estimated seasonal rainfall using two of the most popular models, which are non-linear and linear types. The first model is ARIMA model. Precipitation forecast can be made using ANNs like Back Propagation NN, Cascade NN, and Layer Recurrent Network [22].

The most important objective of the research study conducted by Pham, Binh Thai, et al (2020) was in the development and assessment of a certain number of sophisticated models of artificial intelligence (AI), that is to say, the Fuzzy Inference Method built on Adaptive Networks Optimized on the Prediction of Hoa Binh Regular Rainfall within the Province of Vietnam basing upon Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Particle Swarm Optimization (PSOANFIS). To achieve this, they obtained the daily rainfalls as output parameter and made use of this as input parameter, into the modeled meteorological vari-able parameters, solar radiation, wind speed, relative humidity as well as the highest and lowest temperatures [23].

The authors of the work employed the data of two meteorological station (semi-arid and desert areas of Iraq). To achieve prediction of evaporation, four unique machine learning (ML) models, including classification and regression tree (CART), cascade correlation neural network (CCNNs), gene expression programming (GEP) and support vector ma-chine (SVM) have been generated and modeled by applying different combinations of meteorological variables as input. The findings reveal that sunlight, wind speed, relative humidity, rain and minimum, mean and maximum temperatures provide the best forecasts. [24]

To evaluate the back-propagation model, Budi Raharjo et al. (2021) have been able to optimize the parameters of forecasts of broiler chicken populations in Indonesia provinces. The learning rate (LR) of the backpropagation prediction model is modified by using parameter tuning. A RapidMiner software was used to facilitate the process of analyzing. Data will be applied in two sets, including training data (2017-2018) and testing data (2018-2019). They were used: 1-

Received: July 28, 2025

2-1, 1-25-1 and 1-45-1 Back Propagation models and the LR was (0.1; 0.01; 0.001; 0.2; 0.02; 0.002; 0.3; 0.03; 0.003). The best of the three models used was the 1-45-1 model (LR = 0.3), which gave the high result in terms of Root Mean Squared Error of 0.028 of training data. Testing data in this model had a precision value of 91 % and Root Mean Squared Error of 0.00555. [25].

Reports of the work done by many authors on rainfall forecasting using various methodologies within the principles of Artificial Neural Networks were presented [26]. Adaptive Basis Function Neural Network (ABFNN), Multilayer Perceptron, Rainfall-runoff models, Back-Propagation, Auto-Regressive Moving Average (ARIMA), ANN, K-Nearest Neighborhood (K-NN), Hybrid model (Wavelet-ANN), Hybrid Wavelet-NARX model, Two-stage optimization technique, etc., algorithms/technologies were reviewed. More than 95% of the training and testing papers found accuracy. Because of the non-linear and complicated physical factors affecting the occurrence of rainfall, the rainfall prediction made utilizing ANN approaches was found to be considerably superior to the other techniques like Numerical Weather Prediction (NWP) and Statistical Method. [26]. *The work* sought to optimize ANN's weights in the rainfall classification. For this study's classification, the Gradient Boosting Machine (GBM) was utilized to minimize the error function. To compare the performance of the suggested Optimized ANN classifiers, different classifiers, such as ANN, RF, and NB, were employed along with performance metrics including precision, TPR, FPR, precision, miss rate, and specificity. [27].

3. Methodology

This section ensures a comprehensive and systematic presentation of the methodology used in the study, covering all essential aspects from data collection and preprocessing to model implementation, training, evaluation, and comparative analysis.

3.1. Data Collection

The study area chosen in the present study is Mahanadi River basin. Mahanadi river basin is incorporated within geographical co-ordinates of 80030 to 86050 East longitude and 19020 to 23035 North Latitudes. River Mahanadi is one of the inter-state east flowing rivers of peninsular India. Most parts are tropical monsoon climate in which Mahanadi basin also enjoys. The highest rainfall is normally recorded during the month of July, August and the initial half of the month of September. The average annual precipitation of the basin is 1360mm (16% Coefficient of Variation) of which approximately 86 per cent i.e., 1170 mm takes place within a monsoon season (15 per cent CV) of June to September [28]. The data, gathered and processed in the span of 28 years (between 1990 and 2017 with a resolution of one day a day) show all the features of a chaotic time series. Atmospheric non-linearity causes the existence of irregular and uncertain patterns of rainfall. In order to deal with the inherent disorder of this data in our neural network model we have applied a number of preprocessing procedures, such as data normalization to stabilize the range of data, strategies to deal with missing data points, and others. Complex trends illustrated through visual analysis like plotting the time series, observing autocorrelation functions are hard to put up using the traditional linear models. Thus the

Received: July 28, 2025

backpropagation neural network has the potential of being optimized to work with the chaotic data to enhance rainfall prediction accuracies especially in terms of predicting the non-linear dynamics of such time series.

3.2. Data Preprocessing

Missing data was addressed by techniques such as forward fill, backward fill, and mean imputation. Our Min-Max normalization preprocessed the data so that the data was scaled between 0 and 1 so that the neural network training is efficient. We developed lagged variables (e.g. last earlier rainfall measurements), and choice of constructive meteorological variables (e.g. temperature, humidity).

The amount of collected data was divided into three groups, i.e., training set, validation set, and test set, as follows. 50% of the overall amount of the data was assigned to the training set to train the BPNN that will enable the model to learn the underlying patterns and relations in the data. 40% of the data is assigned to the validation set to tune the hyperparameters, and avoid overfit-ting, by monitoring the performance of the model during the training. The other 10 percent of data was adduced as the test set which tested the final model effectiveness and approximated its generalization capability on unobserved data. The data was divided in a chronological order before being separated. The sum of the observations in each set which were calculated with the selected proportions was determined.

3.3. Model Implementation

In this section, we explain the use of our models of predicting rainfall based on backpropagation neural networks. In our study, we are comparing the predictive capability of the standard backpropagation neural networks (BPNNs) with their optimized version. The process of optimization is carried out to optimize the network architecture, the learning rate and choose adequate activation functions in order to improve the accuracy of predictions.

Our basic models are the standard BPNNs, which are set with standard parameters and structures applied in the rainfall forecasting. We then present other optimization techniques including learning rate based techniques, batch normalization, dropout regularizes and sophisticated gradient descent formalization. These methods are used in a systematic manner in order to note their effect on the performance of the model.

Through this data, we consider historical rainfall data, along with meteorological variables, which have been pre-prepared to maintain the quality and consistency of data. Its implementation requires the division of the dataset into training, validation, and test set to assess the models adequately. To assure robustness as well as overcome overfitting, we use cross-validation.

Model building and training were done with Python and Tensor Flow/Keras and their powerful libraries that enable implementation and optimization of neural networks. The comparison of the models is determined by measuring the performance of the models using Mean Absolute Error (MAE), Mean Squared Error (MSE) and the squared correlation statistics known as R-

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

squared (R 2). This implementation exercise is a foundation designing how we can improve the predictive performance of backpropagation neural network when forecasting rain-fall.

3.3.1 Models under consideration

In present research the performances of the following models have been investigated to obtain the best performing model for rain fall forecasting from the chaotic time series data obtained from weather department:

Model No.	ANN Used	Optimizer Used
1	Feed Forward NN with back propagation algorithm having one hidden layer with two neurons and sigmoid activation function	NIL
2		Gradient Descent (GD) Optimizer
3		Stochastic Gradient Descent (SGD) Optimizer
4		Mini Batch Gradient Descent (MBGD) Optimizer
5		Momentum Optimizer
6		Nesterov Accelerated Gradient (NAG) Optimizer
7		Adaptive Gradient Algorithm (AdaGrad) Optimizer
8		Root Mean Square Propagation (RMSProp) Optimizer
9		Adam Optimizer
10		AdaDelta Optimizer
11		Adamax Optimizer

A brief analysis of all the components of the models is presented in the following paragraphs:

3.3.2 Architecture of the ANN used in the current research

Neural networks Neural networks are based on computational models of the human brain and its structure as well as functioning. They consist of multimode or neurons which coordinate to do complex tasks in concert. Among the numerous types of neural networks, one of the most basic architecture is feed forward neural network (FNN) that is commonly used in most of the pattern recognition and classification issues. The type of algorithm used on our study on the designing of ANN is the kind that is known as the back propagation entailing in the training of the artificial neural networks which are supervised types of algorithms. It has two major steps in its algorithm; the forward and backward passes. The procedure of the BPNN algorithm used in the study work follows as explained below:

Step 1: Initialization

Received: July 28, 2025

Initialize Weights and Biases: Randomly initialize the weights and biases of the network. The initial values are typically small random numbers.

Set Hyperparameters: Set the learning rate, the number of epochs (iterations), and other hyperparameters like the number of hidden layers and neurons in each layer.

Step 2: Forward Pass

Input Layer: Present the input vector to the input layer of the network.

Hidden Layers: For the hidden layer, compute the input to each neuron as a weighted sum of the outputs from the previous layer plus a bias term. Apply the sigmoid activation function to the computed input to obtain the output of each neuron in the hidden layer.

Output Layer: Compute the input to each neuron in the output layer as a weighted sum of the outputs from the last hidden layer plus a bias term. Apply the activation function to obtain the final output.

Step 3: Compute Loss

Loss Function: Compute the loss (error) by comparing the predicted output with the actual target values using the loss function.

Step 4: Backward Pass (Backpropagation)

Output Layer: Compute the gradient of the loss with respect to the output of each neuron in the output layer. Compute the gradient of the loss with respect to the input of each neuron in the output layer. Update the weights and biases between the last hidden layer and the output layer using the gradients and the learning rate.

Hidden Layers: For the hidden layer, compute the gradient of the loss with respect to the output of each neuron in the hidden layer. Compute the gradient of the loss with respect to the input of each neuron in the hidden layer. Update the weights and biases between the current hidden layer and the previous layer using the gradients and the learning rate.

Step 5: Update Weights

Weight Adjustment: Adjust the weights and biases using the computed gradients to minimize the loss.

Step 6: Iteration

Repeat: Repeat steps 2 to 5 for a predetermined number of epochs or until the loss converges to an acceptable value.

Step 7: Model Evaluation

Validation: After training, evaluate the model using a validation dataset to ensure it generalizes well to unseen data.

Testing: Test the final model on a separate test dataset to assess its performance.

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

3.3.3 Optimization Schemes used in the current research

A. *Gradient Descent (GD) Optimizer:*

Gradient Descent is a first-order iterative optimization algorithm for finding the minimum of a function [31]. Widely used in machine learning, it optimizes the model's parameters by minimizing the cost function, often the loss between predicted and actual values. The core idea is to update the model's parameters in the opposite direction of the gradient of the cost function with respect to the parameters. The update rule for parameter θ is:

$$\theta := \theta - \alpha \nabla J(\theta)$$

where α is the learning rate, controlling the step size, and ∇J(θ) is the gradient of the cost function J(θ). Gradient Descent's effectiveness depends on proper tuning of the learning rate and its variants' parameters, making it a versatile and foundational optimization method in machine learning.

B. *Stochastic Gradient Descent (SGD) Optimizer:*

The Stochastic Gradient Descent (SGD) optimizer is a foundational algorithm widely used for training backpropagation neural networks (BPN) [31]. It is a simplified yet powerful variant of the standard gradient descent method, which updates the model parameters iteratively based on the gradient of the loss function with respect to the parameters. Unlike batch gradient descent, which computes gradients using the entire dataset, SGD updates the parameters using the gradient calculated from a single randomly selected data point (or a small batch of data points) at each iteration. The mathematical formulation of SGD involves adjusting the parameters *θ* at iteration *t* according to the following rule:

$$\theta_{t+1} = \theta_t - \nabla_\theta J(\theta; x_i, y_i)$$

Here, *η* represents the learning rate, a crucial hyperparameter that controls the step size of the updates, and $\nabla_\theta J(\theta; x_i, y_i)$ denotes the gradient of the loss function *J* with respect to the parameters *θ* for the *i-th* training example (x_i, y_i).

C. *Mini-batch Gradient Descent (MBGD) Optimizer:*

Mini-batch Gradient Descent (MBGD) is an optimization algorithm that strikes a balance between the computational efficiency of Stochastic Gradient Descent (SGD) and the stable convergence of Batch Gradient Descent, making it particularly effective for training backpropagation neural networks (BPN) [32]. The fundamental concept behind MBGD is to update the model parameters based on the gradient computed from a small, randomly selected subset (mini-batch) of the training data, rather than using a single data point as in SGD or the entire dataset as in Batch Gradient Descent. This approach allows for more frequent updates than Batch Gradient Descent while reducing the noise inherent in SGD, leading to a more stable and efficient optimization process.

Received: July 28, 2025

The mathematical foundation of MBGD involves calculating the gradient of the loss function J with respect to the model parameters θ using a mini-batch of mmm training examples. The update rule for the parameters at iteration t is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; x_{i:i+m}, y_{i:i+m})$$

where η is the learning rate, and $\nabla_\theta J(\theta; x_{i:i+m}, y_{i:i+m})$ represents the gradient of the loss function with respect to the parameters, averaged over the mini-batch $\{(x_i,y_i),(x_{i+1},y_{i+1}),\ldots,(x_{i+m-1},y_{i+m-1})\}$.

D. *Momentum Optimizer:*

The Momentum optimizer is a popular enhancement to the basic Gradient Descent method, designed to accelerate the convergence of backpropagation neural networks (BPNNs) and alleviate some of the shortcomings of traditional optimization techniques. Proposed by Polyak in 1964 [33], Momentum incorporates a concept analogous to physical momentum, smoothing the updates and directing the optimization process more efficiently. The primary idea is to accumulate a velocity vector in the direction of the gradients of the loss function, which helps to build up speed in directions with consistent gradients and dampens oscillations in directions with noisy or high-frequency gradients. This results in faster convergence, especially in ravines or along directions with steep curvatures.

The mathematical formulation of the Momentum optimizer modifies the standard gradient descent update rule by introducing a momentum term. At iteration t, the update equations for the parameters θ are:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

Here, v_t represents the velocity (momentum) vector at iteration t, γ is the momentum coefficient (typically set between 0.9 and 0.99), η is the learning rate, and $\nabla_\theta J(\theta_t)$ is the gradient of the loss function J with respect to the parameters θ at iteration t. The velocity vector accumulates the gradients over time, weighted by the momentum coefficient, which ensures that the updates consider the historical gradients, thus smoothing the optimization path.

E. *Nesterov Accelerated Gradient (NAG) Optimizer:*

The Nesterov Accelerated Gradient (NAG) optimizer, also known as Nesterov momentum, is an advanced optimization technique for training backpropagation neural networks (BPNNs). It builds upon the traditional momentum method by incorporating a lookahead mechanism, which significantly improves the convergence speed and accuracy of the optimization process. Introduced by Yurii Nesterov [34], NAG refines the idea of momentum by not only considering the past gradients but also taking into account the anticipated future position of the parameters. This anticipatory adjustment allows the optimizer to correct its course more effectively, leading to more precise and stable convergence.

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The mathematical formulation of NAG involves a two-step process at each iteration *t*. First, it calculates a temporary update using the current velocity vector, effectively peeking ahead to the future position of the parameters. Then, it computes the gradient at this lookahead position and adjusts the parameters accordingly. The equations for NAG are:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$

$$\theta_{t+1} = \theta_t - v_t$$

Here, θ represents the parameters, η is the learning rate, γ is the momentum coefficient, v_t is the velocity (momentum) vector, and $\nabla_\theta J(\theta)$ is the gradient of the loss function *J* with respect to the parameters θ. The key innovation is the gradient computation at the lookahead position $\theta_t - \gamma v_{t-1}$, which provides a more informed update direction.

F. AdaGrad (Adaptive Gradient Algorithm) Optimizer:

The AdaGrad (Adaptive Gradient Algorithm) optimizer is an influential adaptive learning rate method specifically designed to improve the training process of backpropagation neural networks (BPN). Introduced by Duchi, Hazan, and Singer in 2011 [35], AdaGrad adjusts the learning rate dynamically for each parameter based on the historical gradient information, allowing it to perform well in scenarios with sparse data and high-dimensional parameter spaces. The core idea behind AdaGrad is to scale the learning rates inversely proportional to the square root of the accumulated squared gradients, which provides a per-parameter learning rate adjustment. This enables more substantial updates for infrequent parameters and smaller updates for frequent ones, effectively balancing the learning process.

Mathematically, AdaGrad modifies the standard gradient descent update rule by incorporating a term that adapts the learning rate for each parameter. The update equations for AdaGrad are as follows:

$$g_t = \nabla_\theta J(\theta_t)$$

$$G_t = G_{t-1} + g_t \odot g_t$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Here, g_t represents the gradient of the loss function *J* with respect to the parameters θ at iteration *t*, G_t is the sum of the squares of all past gradients for each parameter, η is the global learning rate, and ϵ is a small constant to prevent division by zero. The element-wise operations, denoted by $\odot$, ensure that each parameter θ_i is updated independently, based on its accumulated gradient history.

G. RMSProp (Root Mean Square Propagation) optimizer:

The RMSProp (Root Mean Square Propagation) optimizer is an adaptive learning rate method designed to improve the efficiency and performance of training backpropagation neural

Received: July 28, 2025

networks (BPN). Proposed by Geoffrey Hinton in a lecture in 2012 [36], RMSProp addresses the key limitation of the AdaGrad optimizer, specifically the issue of rapidly decreasing learning rates which can hinder the training process over time. RMSProp modifies AdaGrad by introducing a mechanism to maintain a moving average of the squared gradients, which helps to stabilize the learning rate and ensures more consistent and sustained progress during training.

Mathematically, RMSProp adjusts the learning rate based on a moving average of squared gradients. At each iteration *t*, the gradient g_t of the loss function *J* with respect to the parameters *θ* is computed. The update rules for RMSProp are as follows:

$$E[g^2]t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E|g^2|_{t+\epsilon}}} g_t$$

In these equations, $E[g^2]_t$ represents the exponentially decaying average of past squared gradients, γ is the decay rate (usually set between 0.9 and 0.99), η is the learning rate, and ϵ is a small constant added to prevent division by zero. The decay rate γ controls how much the past gradients influence the current average, effectively balancing short-term and long-term gradient information.

H. *Adam Optimizer:*

The Adam (Adaptive Moment Estimation) optimizer is a popular algorithm for training backpropagation neural networks (BPNNs), integrating the advantages of two other extensions of stochastic gradient descent (SGD), namely AdaGrad and RMSProp. Introduced by Diederik Kingma and Jimmy Ba in 2014 [29], Adam combines the benefits of adaptive learning rate methods to achieve faster convergence while maintaining robust performance. The mathematical foundation of Adam involves updating the network weights based on estimates of first and second moments of the gradients. Specifically, at each iteration *t*, Adam computes the exponentially decaying average of past gradients m_t (akin to momentum) and the exponentially decaying average of past squared gradients v_t (which models the variance of the gradients). These estimates are then corrected for their initialization biases, resulting in bias-corrected first ($\widehat{m_t}$) and second moment estimates ($\widehat{v_t}$). The weight update rule is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t}$$

where η is the learning rate, ϵ is a small constant to prevent division by zero, and θ_t represents the parameters at iteration *t*.

I. *AdaDelta Optimizer:*

The AdaDelta optimizer is a sophisticated optimization algorithm designed to enhance the training of backpropagation neural networks (BPNNs) by addressing some of the limitations inherent in earlier adaptive learning rate methods like AdaGrad. Proposed by Matthew Zeiler in 2012 [30], AdaDelta aims to alleviate the problem of monotonically decreasing learning rates in AdaGrad, which can lead to premature convergence and impede the optimizer's ability to

Received: July 28, 2025

explore new regions of the parameter space. The mathematical foundation of AdaDelta revolves around two main concepts: accumulating gradient updates over time and adapting the learning rates based on a moving window of gradient updates rather than an ever-increasing sum.

In AdaDelta, the update rule for the parameters θ at iteration t is influenced by an exponentially decaying average of squared gradients, denoted as $E[g^2]_t$, and an exponentially decaying average of squared parameter updates, $E[\Delta\theta^2]_t$. These averages are computed as follows:

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

$$E[\Delta\theta^2]_{t-1} + (1-\rho)(\Delta\theta_t)^2$$

where ρ is a decay constant typically set close to 1 (e.g., 0.95). The update for the parameters is then given by:

$$\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Here, ϵ is a small constant to prevent division by zero. This approach ensures that the learning rate is dynamically adjusted based on the historical context of the gradients and parameter updates, facilitating more stable and efficient training.

J. <u>Adamax Optimizer:</u>

Adamax, an optimization algorithm developed as a variant of the Adam optimizer, is designed to leverage the infinity norm (also known as the max norm) rather than the L_2 norm, making it particularly effective for training backpropagation neural networks (BPNNs) in certain contexts. Proposed by Diederik Kingma and Jimmy Ba in 2014 [29], Adamax adapts the same underlying principles of adaptive learning rates and momentum found in Adam, but with key modifications that improve its performance and stability under specific conditions. The mathematical foundation of Adamax builds upon the core idea of the adaptive moment estimation (Adam) algorithm, yet it introduces changes to the way moments are computed and applied.

In Adamax, the update rule for the network parameters θ at iteration t involves three main steps: calculating the exponentially decaying averages of past gradients (first moment estimate, m_t), computing the exponentially decaying averages of past absolute gradients (infinity norm, u_t), and applying these estimates to adjust the parameters. The equations are as follows:

i. *First moment estimate (momentum term):*

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

where g_t is the gradient at time step t, and β_1 is the exponential decay rate for the first moment estimate.

ii. *Infinity norm (second moment estimate):*

$$u_t = \max(\beta_2 u_{t-1}, |g_t|)$$

Received: July 28, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

where β_2 is the exponential decay rate for the second moment estimate.

iii. *Parameter update rule:*

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_1} m_t$$

Here, η is the learning rate. Unlike Adam, which divides by the root mean square of past gradients, Adamax uses the infinity norm, leading to a more stable and less oscillatory update process.

3.3.4 Performance Metrics used for the evaluation of the Models

Evaluating the performance of an optimized Backpropagation Neural Network (BPN) involves various metrics that assess how well the model performs on training and unseen data. The choice of metrics depends on the specific problem being addressed (e.g., classification, regression). Here are some common performance evaluation metrics along with their mathematical background:

A. *Accuracy*

Accuracy measures the proportion of correct predictions over the total number of predictions, commonly used in classification tasks.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

B. *Precision, Recall, and F1-Score*

These metrics are crucial for imbalanced classification problems:

- **Precision** (Positive Predictive Value):

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall** (Sensitivity or True Positive Rate):

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- **F1-Score** (Harmonic Mean of Precision and Recall):

$$F1 - Score = 2X \frac{Precision\ X\ Recall}{True\ Positives + False\ Negatives}$$

Where,

True Positive (TP): True positive is a measure that presents a correct classification of the model with a positive class.

True negative (TN): A true negative is the instances when the model is right in predicting the negative class.

Received: July 28, 2025

False Positive (FP): False positive is a model error that results when a particular instance of the negative class was falsely predicted by the model as an instance of the positive class. This can also be termed as Type I error.

False Negative (FN): A false negative is a situation whereby the model used makes a wrong prediction that the instance is of the negative type instead of saying that it belongs to the positive category. This is also called Type II error.

C. Confusion Matrix

A confusion matrix provides a detailed breakdown of correct and incorrect classifications, typically in a 2x2 matrix for binary classification, showing True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

D. Mean Squared Error (MSE)

MSE is widely used in regression tasks to measure the average squared difference between actual and predicted values.

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

Where y_i is the actual value and $\hat{y}$ is the predicted value.

E. Root Mean Squared Error (RMSE)

RMSE is the square root of MSE, providing error in the same units as the target variable.

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

F. Mean Absolute Error (MAE)

MAE measures the average absolute difference between actual and predicted values.

$$MAE = \frac{1}{n}\sum_{i-1}^{n}|y_i - \hat{y}_i|$$

G. R-squared (Coefficient of Determination)

R-squared evaluates the proportion of variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum_{i-1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i-1}^{n}(y_i - \bar{y})^2}$$

Where $\bar{y}$ is the mean of the actual values.

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

4. Results and Discussion

4.1 Experiment Design

The following table describes the salient features of the Optimized BPN Models with optimizers used and their analysis:

Optimizer Used	Hidden Layer	Activation Function	Architectural Complexity	Justification
No Optimizer, i.e., Standard BPN	One hidden layer with two neurons	Logistic (sigmoid) activation function	Simple architecture with one hidden layer and minimal neurons	The basic feedforward neural network without optimization is chosen for simplicity and ease of implementation. It serves as a baseline model for comparison with optimized versions
Gradient Descent (GD)-OBPN Model	1 hidden layer with 2 neurons.	Logistic (sigmoid) activation function	Simple architecture with one hidden layer and minimal neurons	GD is chosen for its simplicity and efficiency in updating weights based on individual training examples. It works well for large datasets and can be used as a baseline optimization method
(SGD)-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Simple architecture with one hidden layer and minimal neurons	Chosen for its simplicity and efficiency in updating weights based on individual training examples. It works well for large datasets and can be used as a baseline optimization method
Mini-Batch Gradient Descent-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Similar to SGD, with a simple architecture	Mini-batch gradient descent strikes a balance between the efficiency of stochastic gradient descent and the stability of batch gradient descent. It processes small batches of data at a time, which can lead to faster convergence and better generalization.

Received: July 28, 2025

Momentum-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Comparable to other architectures, with a simple design	Selected for its ability to accelerate gradient descent in the relevant direction and dampen oscillations. It helps overcome local minima and leads to faster convergence
(NAG)-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Similar to other architectures, with a simple design	Chosen for its ability to provide faster convergence compared to vanilla momentum. It uses a more accurate estimate of the gradient by looking ahead in the direction of the momentum update
AdaGrad - OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Similar to other architectures, with a simple design	The learning rate was chosen because of its capability of adapting to the parameters. It updates large parameters infrequently and frequent parameters in small size, and so it is adaptable to sparse data
RMSprop-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Comparable to other architectures, with a simple design	Considered for its ability to handle the problem of vanishing and exploding gradients. It scales the learning rates differently for each parameter based on the magnitude of recent gradients, leading to more stable training
Adam-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Simple architecture similar to the no optimization case	It was selected because of its efficacy in processing sparse gradients and rate adaptation of each individual parameter. It assists in the more intense training of deep neural networks than elementary back-propagation.

Received: July 28, 2025

Adadelta-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Similar to the other cases, with a simple architecture	Selected for its ability to handle noisy gradients and adapt learning rates dynamically. It is suitable for training neural networks in scenarios where the gradient updates are noisy or sparse.
Adamax-OBPN Model	One hidden layer with two neurons	Logistic (sigmoid) activation function	Comparable to other architectures, with a simple design	Considered for its robustness to hyperparameter choices and good performance on both convex and non-convex optimization problems. It offers an efficient alternative to Adam optimizer

4.2 Data Sets:

	1	2	3	4	5	6	7	9	10	12
Count	3758	3753	3762	3754	3761	4132	4135	4133	3757	3733
Mean	7.884	7.343	7.497	6.499	9.513	168.572	325.624	182.732	42.329	3.912
Std	21.257	17.114	18.626	17.213	24.383	112.709	152.219	118.259	58.529	15.578
Min	0	0	0	0	0	0	0	0	0	0
25%	0	0	0	0	0	92.601	226.149	99.506	0	0
s50%	0	0	0	0	0	149.050	311.200	164.200	21	0
75%	0.300	5.250	4.000	3.350	7.200	219.825	404.000	241.951	58.6	0

Received: July 28, 2025

	1	2	3	4	5	6	7	9	10	12
Max	284.600	246.200	198.700	407.900	385.000	1071.000	1654.200	1137.360	523	357

4.3 Training Procedure:

1. *Data Splitting:*

- The dataset was split into training and testing sets using the *train_test_split* function from *sklearn.model_selection*.
- The *test_size* parameter was set to 0.2, indicating a 80-20 split between training and testing data.
- Optionally, cross-validation or bootstrapping techniques could be applied to further assess model performance. However, in the provided code, only a single train-test split is utilized.

2. *Training the BPN:*

- The BPN model was initialized and trained using the MLP Classifier class from *sklearn.neural_network.*
- Various optimization techniques were employed, including SGD, AdaGrad, RMSprop, etc., by specifying the appropriate solver parameter in the MLPClassifier.
- The model was fitted to the training data using the fit method, passing the scaled training features (*X_train_scaled*) and the encoded training labels (*y_train*) as input.

4.4 Performance Metrics

At last, considering such performance indicators as average outcomes in training and testing sets, precision, recall, and F 1 score, this combination no optimization may be put in the following order: Gradient Descent Optimizer, then No Optimization. However, according to performance against MAE and MAPE, the values of which should be as low as possible, this ranking can be moved a bit. Therefore, No Optimization will be the best and the rest can be ordered by priority as: Momentum optimizer, then, the Nesterov, AdaGrad and RMSprop. ADAM is at the higher position than these optimizers. Next Adadelta and Adamax come and then Gradient Descent Optimizer and SGD. In such a way, in performance, i.e., in their average values achieved in various subsets, the data shown in Table 2 completely differ in the case of the optimizers and their performance measured by such points of evaluation and estimation as MAE or MAPE. In this way, the Gradient Descent No Optimization alternative offered the best training precision out of all these alternatives but at the same time, it displayed the greatest MAE. Meanwhile, AdaGrad and RMSprop were found to be perfect on the principles of MAE and MAPE and the next option in the list was ADAM with quite reasonable results.

Received: July 28, 2025

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

4.5 Experimental Setup

Experiments were done in a desktop computer that had an Intel Core i7 CPU, 16 GB RAM and a GeForce GTX graphics card. It was applied to the Python 3.8. NumPy, pandas, scikit-learn, matplotlib libraries were used to operate with the data, construct models, obtain an optimization, and visualize the results. Considering the fact that the size of the dataset is not very large and that the models have quite a dense structure, training and searching the neural network did not impose a serious strain on the computing capacity. The training was done through a very large number of iterations on epochs with the training of the model on gradients and the model coefficients via optimization algorithms. Experiments were finished in a rather short period of time.

4.6 Validation and Testing

In order to prove and verify the optimized model of BPN, the data was divided into the training and the test set. The model was then trained to minimize the training set with various optimization methods, which include SGD, AdaGrad, RMSprop among others. The testing was done on the testing set and the performance evaluation metrics as accuracy, F1 score, recall, precision, MAE and MAPE were then computed. The result of the experiment provided a conclusion to the effect that the model outper-formed the baseline methods or other defined neural network architectures hence competitive performance, to the effectiveness of the model to the forecasting work.

4.7 Results

The results obtained through the experiments performed with the above discussed detail are presented in the following table.

Sr. No	**Model**	**Train Accuracy**	**Train Precision**	**Train Recall**	**Train F1 Score**	**Test Accuracy**	**Test Precision**	**Test Recall**	**Test F1 Score**	**Mean Absolute Error (MAE)**	**Mean Absolute Percentage Error (MAPE)**
1	No Optimization	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	15.659	0.135
2	Gradient Descent Optimizer	0.974	0.974	0.974	0.974	0.848	0.848	0.848	0.848	15.860	1.129e+16
3	SGD	0.974	0.974	0.974	0.974	0.848	0.848	0.848	0.848	15.860	1.129e+16
4	Mini Batch	0.943	0.943	0.943	0.943	0.854	0.854	0.854	0.854	15.193	8.217e+15

Received: July 28, 2025

International Journal of Applied Mathematics

Volume 38 No. 2s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

5	Momen-tum	0.877	0.877	0.877	0.877	0.864	0.864	0.864	0.864	15.755	4.353e+14
6	Nesterov	0.877	0.877	0.877	0.877	0.864	0.864	0.864	0.864	15.755	4.353e+14
7	AdaGrad	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	9.038	0.912
8	RMSprop Opti-mizer	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	15.659	0.135
9	ADAM	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	15.659	0.135
10	Adadelta	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	15.659	0.135
11	Adamax	0.876	0.876	0.876	0.876	0.865	0.865	0.865	0.865	15.659	0.135

4.8 Discussions

- *No optimization:* The model performance shows moderate levels of accuracy and other performance metrics. There is a generally good fit in the model.
- Gradient *Descent optimizer (GD):* GD optimizer enhances the training accuracy and precision significantly above the benchmark. Nonetheless, due to the high MAPE on the test, the model likely fell into overfitting. Similarly, GD also has an improvement in the training accuracy and precision. However, the MAPE for the test also neutralizes the model to overfitting.
- Stochastic Gradient Descent (SGD): SGD optimizer enhances the training accuracy and precision significantly above the benchmark. Nonetheless, due to the high MAPE on the test, the model likely fell into overfitting. Similarly, SGD also has an improvement in the training accuracy and precision. However, the MAPE for the test also neutralizes the model to overfitting.
- *Mini Batch Gradient Descent optimizer*: Mini Batch also demonstrated excellent performance in the training section with high accuracy and precision. However, the test MAPE level indicates the model either underfits or overfits.
- *Momentum*: The performance of the base model is consistent with the momentum optimizer.
- *Nesterov Accelerated Gradient Optimizer*: NAG optimizer performance is similar to momentum, with fewer improvements and stipulated by test MAPE indicating overfitting.
- *AdaGrad*: The optimizer has fewer generalization errors evidenced by test MPAE.

Received: July 28, 2025

- *RMSprop optimizer*: The optimizer had a good performance with regards to test MAPE.
- *ADAM*: the ADAM optimizer also has reduced test MAPE and is similar to RMSprop and AdaGrad.
- *Adadelta*: Shows that adadelta limit the number of generalization errors.
- *Adamax*: Adamax is also similar to other optimizers and generally better than the base model; thus, the reduced MAPE portrays no generalization errors.

5. Conclusions and Future Work

5.1 Summary of Findings

This research has demonstrated the potential of optimized backpropagation neural networks (BPNNs) in improving the accuracy and efficiency of rainfall prediction. By implementing various optimization techniques such as adaptive learning rates, momentum, and regularization, we observed significant enhancements in predictive performance compared to standard BPNNs. Our findings highlight the importance of optimizing neural network training processes to capture complex, nonlinear patterns in meteorological data more effectively.

5.2 Future Work

While the results of this study are promising, several areas for future research and development remain:

Integration of Additional Meteorological Variables: Future studies could incorporate a broader range of meteorological variables, such as atmospheric pressure, solar radiation, and soil moisture levels. These additional inputs could help improve the predictive accuracy of the models by providing a more comprehensive representation of the factors influencing rainfall.

Exploration of Advanced Neural Network Architectures: Beyond BPNNs, exploring advanced neural network architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) could be beneficial. CNNs can be particularly effective for spatial data, while RNNs and Long Short-Term Memory (LSTM) networks are well-suited for temporal data, potentially enhancing the model's ability to capture complex spatiotemporal patterns.

Hybrid Modeling Approaches: Combining neural networks with traditional statistical and physical models could lead to hybrid approaches that leverage the strengths of each method. For instance, integrating neural networks with autoregressive models or ensemble methods might enhance predictive performance and robustness.

Real-time Rainfall Prediction Systems: Developing real-time rainfall prediction systems that continuously update predictions as new data becomes available would be a practical extension of this research. Implementing such systems in operational settings could provide timely forecasts to support decision-making in agriculture, water management, and disaster response.

Received: July 28, 2025

Volume 38 No. 2s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Impact of Climate Change: Considering the impacts of climate change on rainfall patterns is another important area for future research. Modeling how changing climatic conditions affect rainfall predictions can help improve the resilience and adaptability of forecasting systems.

5.3 Concluding Remarks

This study has highlighted the effectiveness of optimization techniques in enhancing the performance of BPNNs for rainfall prediction. The promising results suggest that continued advancements in neural network methodologies and their application to meteorological data can significantly improve our ability to forecast rainfall accurately. By addressing the outlined future research directions, the scientific community can develop more robust, accurate, and practical rainfall prediction models, ultimately contributing to better management of water resources and disaster preparedness.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper

References

[1] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural Networks, vol. 2, pp. 359–366, 1989.

[2] K. C. Luk, J. E. Ball, and A. Sharma, “A study of optimal model lag and spatial inputs to artificial neural network for rainfall forecasting,” J. Hydrol., vol. 227, pp. 56–65, 2000.

[3] K. C. Luk, J. E. Ball, and A. Sharma, “An application of artificial neural networks for rainfall forecasting,” Math. Comput. Model., vol. 33, pp. 683–693, 2001.

[4] W. Suparta and A. A. Samah, “Rainfall prediction by using ANFIS times series technique in South Tangerang, Indonesia,” Geodesy Geodyn., vol. 11, no. 6, pp. 411–417, 2020.

[5] A. Jain, A. Kumar, M. K. Tiwari, and A. Verma, “Neural network-based approach for short term rainfall prediction using hydro-meteorological data,” Water Resour. Manag., vol. 32, no. 2, pp. 509–522, 2018.

[6] V. Ramasubramanian and P. Victor, “Rainfall Prediction Using Artificial Neural Network,” Int. J. Comput. Appl., vol. 975, p. 8887, 2019.

[7] M. A. Wani and D. Jangid, “Rainfall Prediction using Artificial Neural Network,” Int. J. Recent Technol. Eng., vol. 8, no. 5, pp. 1747–1751, 2020.

[8] M. G. M. Abdolrasol et al., “Artificial Neural Networks Based Optimization Techniques: A Review,” Electronics, vol. 10, p. 2689, 2021.

[9] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” J. Big Data, vol. 6, p. 60, 2019.

[10] K. Qian et al., “Modeling neuron growth using isogeometric collocation based phase field method,” Sci. Rep., vol. 12, p. 8120, 2022.

[11] Y. Liu et al., “Multimodal MRI Volumetric Data Fusion with Convolutional Neural Networks,” IEEE Trans. Instrum. Meas., vol. 71, pp. 1–15, 2022.

Received: July 28, 2025

[12] Q. Li, D. Xiong, and M. Shang, “Adjusted stochastic gradient descent for latent factor analysis,” Inf. Sci., vol. 588, pp. 196–213, 2022.

[13] R. V. Ramana, B. Krishna, S. R. Kumar, and N. G. Pandey, “Monthly rainfall prediction using wavelet neural network analysis,” Water Resour. Manag., vol. 27, no. 10, pp. 3697–3711, 2013.

[14] B. K. Rania, K. Srinivasa, and A. Govardhan, “Rainfall Prediction with TLBO Optimized ANN,” J. Sci. Ind. Res., vol. 73, pp. 643–647, 2014.

[15] R. Arya and M. L. Pai, “Rainfall Prediction Using an Optimised Genetic-Artificial Neural Network Model,” Int. J. Pure Appl. Math., vol. 119, pp. 669–678, 2018.

[16] L. Foresti et al., “Using a 10-year radar archive for nowcasting precipitation growth and decay: A probabilistic machine learning approach,” Weather Forecasting, vol. 34, no. 5, pp. 1547–1569, 2019.

[17] R. E. N. Macabiog and J. C. Dela Cruz, “Rainfall Predictive Approach for La Trinidad, Benguet using Machine Learning Classification,” in Proc. IEEE HNICEM, 2019.

[18] S.-H. Moon et al., “Application of machine learning to an early warning system for very short-term heavy rainfall,” J. Hydrol., vol. 568, pp. 1042–1054, 2019.

[19] B. Quinn and E. Abdelfattah, “Machine Learning Meteorologist Can Predict Rain,” in Proc. IEEE UEMCON, 2019.

[20] R. K. Grace and B. Suganya, “Machine Learning based Rainfall Prediction,” in Proc. IEEE ICACCS, 2020.

[21] K. Ahmed et al., “Multi-model ensemble predictions of precipitation and temperature using machine learning algorithms,” Atmos. Res., vol. 236, p. 104806, 2020.

[22] C. Basha et al., “Rainfall Prediction using Machine Learning & Deep Learning Techniques,” in Proc. IEEE ICESC, 2020.

[23] A. Tamilmani and M. Sughasiny, “Development of advanced artificial intelligence models for daily rainfall prediction,” Atmos. Res., vol. 237, p. 104845, 2020.

[24] Z. M. Yaseen et al., “Prediction of evaporation in arid and semi-arid regions: a comparative study using different machine learning models,” Eng. Appl. Comput. Fluid Mech., vol. 14, no. 1, pp. 70–89, 2020.

[25] B. Raharjo et al., “Optimization Forecasting Using Back-Propagation Algorithm,” J. Appl. Eng. Sci., vol. 19, no. 4, pp. 1083–1089, 2021.

[26] A. Tamilmani and M. Sughasiny, “Optimized Artificial Neural Network Classifier for the Prediction of Rainfall,” Int. J. Comput. Intell. Control, vol. 13, no. 2, pp. 377–387, 2021.

[27] N. Thakur, S. Karmakar, and S. Soni, “Rainfall Forecasting Using Various Artificial Neural Network Techniques – A Review,” Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 7, no. 3, pp. 506–526, 2021.

[28] V. Kumar, S. K. Jain, and Y. Singh, “Analysis of long-term rainfall trends in India,” Hydrol. Sci. J., vol. 55, no. 4, pp. 484–496, 2010.

[29] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv preprint arXiv:1412.6980, 2014.

Received: July 28, 2025

[30] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” arXiv preprint arXiv:1212.5701, 2012.

[31] D. Zou, Y. Cao, D. Zhou, and Q. Gu, “Stochastic Gradient Descent Optimizers Over-Parameterized Deep ReLU Networks,” arXiv preprint arXiv:1811.08888, 2018.

[32] H. Wang, H. Luo, and Y. Wang, “MBGDT: Robust Mini-Batch Gradient Descent,” arXiv:2206.07139 [cs.LG], 2022.

[33] B. T. Polyak, “Some Methods of Speeding Up the Convergence of Iteration Methods,” Zh. Vych. Mat., vol. 4, no. 5, pp. 791–603, 1964.

[34] Y. Nesterov, “A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$,” Sov. Math. Dokl., vol. 27, pp. 372–376, 1983.

[35] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Sub gradient Methods for Online Learning and Stochastic Optimization,” J. Mach. Learn. Res., vol. 12, pp. 2121–2159, 2011.

[36] L. Alzubaidi et al., “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions,” J. Big Data, vol. 8, p. 53, 2021.

Received: July 28, 2025