International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 07 | July 2021 www.irjet.net p-ISSN: 2395-0072

# Optimization of Feedforward Neural Network Training using Modified Local Leader Phase Spider Monkey Optimization

**Daniel Kwegyir[1], Emmanuel Asuming Frimpong[2], Daniel Opoku[3]**

[1-3]*Department of Electrical and Electronic Engineering, College of Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana*

---

**Abstract –** *An optimal trainer for feedforward neural networks (FNNs) is presented. The trainer is an enhanced variant of the spider monkey optimization (SMO) algorithm. The enhanced variant has been named as modified local leader phase spider monkey optimization (MLLP-SMO). The enhancement of the SMO was done by modifying its local leader phase. The modification offers chances to each spider monkey that is selected for update, to update to a better position. The performance of the MLLP-SMO was assessed using six datasets. The datasets relate to fault detection, heart attack, iris species, breast cancer, diabetes and XOR function. The indicators used for the performance assessment are mean squared error (MSE) during training phase and classification accuracy of trained FNN. The MLLP-SMO trainer was compared with four other optimization algorithms. The four other optimization algorithms are the original SMO, particle swarm optimization (PSO), grey wolf optimization (GWO) and genetic algorithm (GA). The results obtained show that the MLLP-SMO performs better than the other algorithms. Its MSEs during training were largely lower than those of the other algorithms and the accuracy of its trained FNNs chiefly higher than the others. The MLLP-SMO is recommended for adoption as an optimal trainer for FNNs.*

***Key Words***: Artificial neural network, Classification, Optimization, Nature inspired algorithm, Spider monkey optimization, Neural network training

## 1. INTRODUCTION

Artificial Neural Network (ANN) is a machine learning tool that models how the human brain processes, visualizes, and recognizes objects [1]. ANN has been used for different applications in diverse fields. In engineering, ANNs have been used in, for example, fault detection, classification and location, transient stability prediction, and load forecasting. Different types of ANNs have been used in literature. Commonly used ones include feedforward neural network [2], recurrent neural network [3], radial basis function neural network [4], and Kohonen self-organizing maps [5].

The performance of ANNs in applications depends on factors such as network architecture (i.e., number of neurons in the input, hidden and output layers), weights of connections between neurons, and biases of neurons. The weights and biases of neurons are determined in the learning (also known as training) process. Learning in ANNs is a process of using optimization algorithms to find a set of weights and biases that best map inputs to outputs [6]. In the learning process, a training algorithm (i.e., trainer) modifies the structural features (i.e., weights and biases) of the ANN for each training iteration to achieve a better performance. The trainer is removed once the ANN has finished learning and is ready to be used for an application such as prediction or classification.

Broadly, there are two approaches to training ANNs. The approaches are supervised learning and unsupervised learning. The supervised learning process enables ANNs to learn the input and output relationships of a training data set while the unsupervised learning process learns from previously undetected patterns in a data set, with no initial outputs [7].

Classical learning methods for training neural networks are back-propagation [8] and Gradient Decent [9]. These methods, which are deterministic, make use of mathematical optimization models to train ANNs. These training methods are simple and fast [6]. However, their performances become poor when initial solutions are not carefully chosen. They also suffer from the problem of local optima entrapment. To improve the classification accuracy of ANNs, high performing optimization techniques are required to produce optimal ANN weights and biases [10]. Accordingly, several optimization techniques have been proposed for enhanced ANN training [11].

The optimization techniques for enhanced ANN training largely employ meta-heuristic optimization algorithms (MHOAs). Generally, the training process of a MHOA starts with an initial random solution which is improved as the training progresses. MHOAs that have been used to train neural networks are ant colony optimization, artificial bee colony optimization, bacterial foraging optimization, bat algorithm, biography-based optimization, bird mating optimization, genetic algorithm, and particle swarm optimization. Others are grey wolf optimization, chemical reaction optimization, cuckoo search optimization, firefly optimization, gravitational search algorithm, invasive weed optimization, krill herd optimization, moth-flame

optimization, social spider optimization and tabu search optimization [11].
These metaheuristic methods largely perform better than deterministic training methods. However, the current performances of these algorithms still leave a lot of room for improvement in optimizing the performance of ANNs. This is because the existing MHOAs do not provide a good trade-off between exploration and exploitation in optimizing the performance of ANNs [11]. Exploration helps in the attainment of global optimum while exploitation aids in achieving local optimum. Therefore, there is the need to explore the use of hybrid approaches (i.e., combine two or more MHOAs) or come up with new MHOAs that have excellent exploitation and exploration attributes. This should provide greater improvement in ANN training [11].

Spider monkey optimization (SMO) is a recently introduced MHOA that has demonstrated enhanced exploration and exploitation ability [12]. The SMO was inspired by the foraging behaviour of spider monkeys [12]. The SMO has been successfully applied to solve complex problems in optimization. It has been shown to outperform artificial bee colony optimization and particle swarm optimization in terms of dependability, effectiveness, and precision [12]. Hence, in this work, the SMO has been explored to optimize the training of ANNs. First, the effectiveness of SMO is further enhanced through a modification of its local leader phase. Second, the enhanced SMO is used as an optimized trainer for a feedforward neural network (FNN). FNNs are one of the most widely used neural network models. The improved SMO is called modified local leader phase spider monkey optimization (MLLP-SMO). The performance of the MLLP-SMO is compared with the original SMO. The MLLP-SMO is further compared with the grey wolf optimizer, particle swarm optimization, and genetic algorithm which are high performing optimization algorithms [6, 11].

The rest of the paper is structured as follows; Section 2 gives a description of the Feedforward neural network (FNN) Section 3 presents the modified local leader phase spider monkey optimization (MLLP-SMO) algorithm. Section 4 presents the MLLP-SMO FNN trainer. In Section 5, the approach used to test the proposed MLLP-SMO trainer, including the datasets used for evaluating the performance of the MLLP-SMO trainer is presented. The evaluation results are presented and discussed in Section 6. Conclusions drawn are presented in Section 7.

## 2. FEEDFORWARD NEURAL NETWORK

Feed-forward neural networks have been widely used for various studies. They are made up of interconnected neurons that are structured in parallel layers. The layers are classified into input layer, hidden layer, and output layer. There are no feedbacks from one layer to another. Depending on the number of parallel layers that exist in the networks, FNNs can be classified as single or multi-layered networks. Single-layered FNNs have only input, and output neurons interconnected together. Multi-layered FNNs have input layer, single or multiple hidden layers, and an output layer [13]. A multi-layered FNN with a single hidden layer, inputs, outputs, biases, and interconnecting weights is illustrated in fig -1.

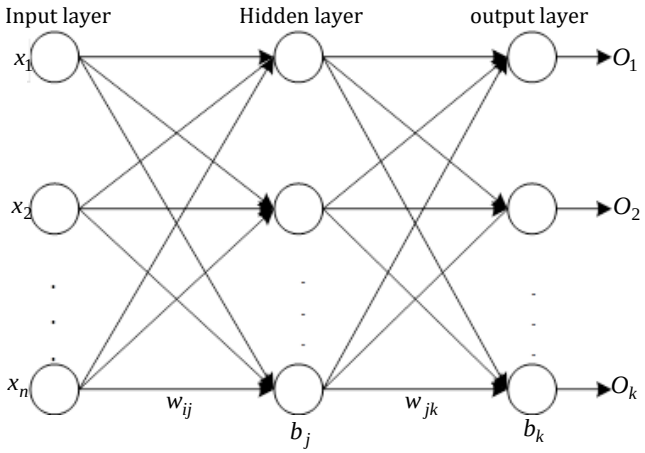


**Fig -1**: Architecture of FNN with one hidden layer

In Fig -1, $(x_1, x_2, \ldots, x_n)$ are the input variables, $w_{ij}$ is the input weights from the *i*th input neuron to the *j*th hidden neuron, $b_j$ is the input bias of the *j*th hidden neuron, $w_{jk}$ is the connection weight of the *j*th hidden neuron to the *k*th output neuron and $b_k$ is the bias of the *k*th output neuron. In the working of the FNN, the weighted sums of the input data are propagated to the hidden layer neurons. The weighted sums of the outputs of the hidden layer neurons are then propagated to the neurons in the output layer. The outputs of the hidden layer neurons are obtained by taking the inputs to the neurons and their biases through an activation (or transfer) function. The weighted sums of the outputs of the hidden layer neurons are propagated to the inputs of the hidden layer neurons. The outputs of the output layer neurons, which are the outputs of the FNN are determined by also applying an activation function to the weighted inputs and biases [14]. Commonly used activation functions are tan-sigmoid (tansig) and log-sigmoid (logsig) transfer functions [15].

The weighted sums of the input neurons, $S_j$, are given by (1) where *n* is the number of input neurons and *h* is the number of hidden layer neurons. The outputs of the

hidden layer neurons, $H_j$, for log-sigmoid activation functions, are given by (2). The weighted sums of the outputs of the hidden layer neurons, $Y_k$, where $m$ is the number of output layer neurons are given by (3). The final outputs of the network, $O_k$, using a log-sigmoid activation functions, are given by (4) [6, 14].

$$S_j = \sum_{i=1}^{n} w_{ij} x_i \quad , \quad i = 1,2,..,n \, , \quad j = 1,2,..,h \tag{1}$$

$$H_j = \log sig(S_j) = \frac{1}{(1+\exp(-(S_j + b_j)))} , \quad j = 1,2,...,h \tag{2}$$

$$Y_k = \sum_{j=1}^{h} w_{jk} H_j \quad , \quad k = 1,2,...,m \tag{3}$$

$$O_k = \log sig(Y_k) = \frac{1}{(1+\exp(-(Y_k + b_k)))} , \quad k = 1,2,...,m \tag{4}$$

It is noted from (4) that the output $O_k$ of the FNN highly depends on the weights and biases. Thus, the optimization of the weights and biases will yield highly accurate outputs. Hence, the focus of any FNN training algorithm should be to optimize the values of the weights and biases.

## 3. PROPOSED MODIFIED LOCAL LEADER PHASE SPIDER MONKEY OPTIMIZATION (MLLP-SMO) ALGORITHM

The SMO algorithm mimics the foraging behaviour of spider monkeys. It follows a fission-fusion concept. The foraging behaviour is basically based on food scarcity or availability which causes the spider monkeys to either split (fission) or combine (fusion). The SMO operates in seven phases. These are initialization, local leader phase, global leader phase, global leader learning phase, local leader learning phase, local leader decision phase and global leader decision phase [16].

In the local leader phase of the SMO algorithm, the update of a spider monkey ( $SM_{ij}$ ) is done by a greedy approach where the fitness of $SM_{ij}$ at a new position is accepted only when it is better than the fitness at the old position. The deficiency of this approach is that $SM_{ij}$ with low fitness but near the global solution is deprived of a chance to update. The effect is that the algorithm could move in a non-optimal direction and skip the true solution. From (5), the update of an $SM_{ij}$ position is highly dependent on β and the impact of a random spider ( $SM_{rj}$ ). β is a uniformly distributed random number in the range of (-1,1). $\alpha$ is a random number generated in the range (0,1).

$$SM_{newij} = SM_{ij} + \alpha \times (LL_{kj} - SM_{ij}) + \beta \times (SM_{rj} - SM_{ij}) \tag{5}$$

Due to the uneven opportunity given to each $SM_{ij}$ in the position update at this phase, there is the possibility that $SM_{ij}s$ with good fitness or all $SM_{ij}s$ chosen by $\alpha > pr$ will not update towards the global optima for a particular iteration. This will cause the algorithm to shift to produce non-optimal results. $pr$ is the perturbation rate and normally ranges from 0.1 to 0.8.

To enhance the SMO , each $SM_{ij}$ chosen by $\alpha > pr$ to update, is given a chance in the search space of the local leader phase, according to the fitness of their old position, to update to a better position. The number of chances offered to an $SM_{ij}$ to update is defined according to a fitness proportionate selection in genetic algorithm [17] and the total number of $SM_{ij}s$ in the search space ( $Y$ ). The number of chances given to each $SM_{ij}$ for the next iteration is defined by (6).

$$No.\ of\ chances\ of\ SM_{ij} = \frac{fit(SM_{ij(old)})}{\sum_i^N fit(SM_{ij(old)})} \times Y \tag{6}$$

Where $fit(SM_{ij(old)})$ is the fitness of $SM_{ij}$ in its old position and N is the number of $SM_{ij}s$. If after the number of chances, an $SM_{ij}$ does not update to a better position, then it is set to its old position. The pseudocode for the proposed MLLP algorithm for the position update is presented as follows:

**for** each member $SM_{ij} \in k^{th}$ group
  **for** each $j \in \{1,2,..D\}$ **do**
    **if** $\alpha(0,1) > pr$ **then**
      $f_{ij} = SM_{ij} + \alpha(0,1) \times (LL_{kj} - SM_{ij})$
      $g_{ij} = (SM_{rj} - SM_{ij})$
  **While** (chances to update has not elapsed), **do**
    $SM_{chances(ij)} = f_{ij} + \beta(-1,1) \times g_{ij}$
  **if** $fit(SM_{chance(ij)}) > fit(SM_{ij})$

$SM_{new(ij)} = SM_{chance(ij)}$
**break**
**else if** (chances elapsed)
$SM_{new(ij)} = SM_{ij}$
**break**
**end if**
**end while**
**end if**
**end for**
**end for**

To use the MLLP algorithm to update the position of a spider monkey, $SM_{ij}$, to obtain a new position, $SM_{new(ij)}$, a random number denoted by $\alpha$ is generated within the limits of 0 and 1. This random number is then compared to the perturbation rate (*pr*). If $\alpha$ is greater than *pr*, the $SM_{ij}$ is selected to update its position in the $j^{th}$ dimension. First, the effect of the local leader's position ($LL_{kj}$) on $SM_{ij}$ is checked by $f_{ij}$, as defined in the pseudocode. The effect of the random spider ($SM_{rj}$) on $SM_{ij}$ is also checked by $g_{ij}$. The $SM_{ij}$ is then given some chances with a value defined according to (6) to update to a better position in the search space. For each position generated under the chances given ($SM_{chances(ij)}$), its fitness is checked and compared to the fitness of the old position ($SM_{ij}$) of the spider monkey. The spider monkey is assigned the new position ($SM_{new(ij)}$) if the fitness value at the new position is better than the fitness at the old position. The position update is then completed irrespective of whether the number of chances is elapsed. On the other hand, if a better position is not found in the search space after the number of chances offered a spider monkey is elapsed, its position is not updated but set to the old position ($SM_{ij}$). This process is repeated for all spider monkeys in the search space.

In the proposed position update algorithm, $f_{ij}$ checks for how far the $SM_{ij}$ is drawn towards the local leader ($LL_{kj}$) in the $j^{th}$ dimension, whilst $g_{ij}$ checks the influence of the random spider $SM_{rj}$ in the $j^{th}$ dimension. The number of chances as defined in (6) balances $f_{ij}$ and $g_{ij}$ by giving proportionate opportunity to each $SM_{ij}$ to update by $\alpha > pr$, according to its old fitness value and maintains the stochastic nature of the local leader phase. Figure 1 is a flowchart that shows the implementation of the MLLP-SMO.

The implementation of the MLLP-SMO algorithm is summarized as follows:

Step 1: Initialize population, local leader limit, global leader limit, maximum number of groups, and perturbation rate.

Step 2: Evaluate population. Here, the fitness of each spider monkey in the population is determined using their initial positions on the objective function of the problem being solved.

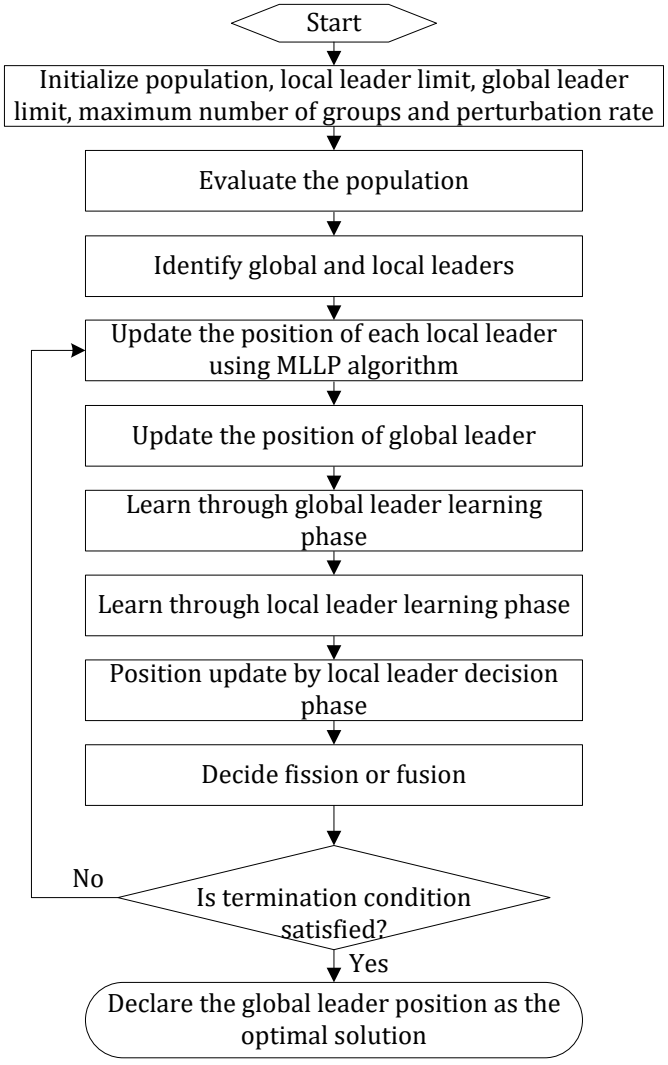


**Fig -2**: Implementation of MLLP-SMO algorithm

Step 3: Identify local and global leaders. This involves the selection of spider monkeys with best fitness in each group as local leaders and the individual with best fitness amongst all the groups as global leader.

Step 4: Update the position of each spider monkey using the MLLP algorithm. Spider monkeys with $\alpha$ greater than the perturbation rate update their $j^{th}$ dimension position towards the positions of their local leaders and a random spider monkey's position.

Step 5: Update the position of global leader using global leader phase update rule. Spider monkeys with fitness probability greater than the perturbation rate update their positions in the $j^{th}$ dimension.

Step 6: Learn through global leader learning phase. Here, spider monkey with best fitness after global leader phase is selected as global leader and its update is checked against the global leader limit.

Step 7: Learn through local leader learning phase. This entails selecting a spider monkey with the best fitness as the local leader and checking its update against the local leader limit count.

Step 8: Position update by local leader decision phase. For this, spider monkeys belonging to a particular local group whose local leader is not updating up to the local leader limit are given opportunity to update through random initialization.

Step 9: Decide fission or fusion using global leader decision phase. The global leader either divides the spider monkeys into groups if the stopping criteria (i.e., maximum number of groups) is not met or fuse them into a single group if otherwise.

Step 10: If termination condition (i.e., maximum number of groups is reached or minimum error specified is attained) is satisfied, declare the global leader position as the optimal solution. Otherwise, go back to Step 4.

## 4. PROPOSED MLLP-SMO TRAINER

The training of neural networks entails the continuous mapping of input datasets to the output datasets to find the optimal set of weights and biases within a minimum number of iterations. The determination of optimal weights and biases produces trained ANNs with high classification accuracy [18].

Performances of trainers, during and after training, are assessed using error functions. The lower the error, the better the performance. Commonly used error functions are mean square error (MSE), sum of squared error (SSE) and root mean square error (RMSE). In this work, the mean square error function was used to assess the training performance. Consequently, the objective function for training the FNN with the MLLP-SMO is to minimize the MSE of each training iteration. The MSE is given by (7).

$$MSE = \sum_{j=1}^{n} \frac{\sum_{i=1}^{m} \left(a_i^j - d_i^j\right)^2}{n} \quad (7)$$

where $n$ is the total number of training samples, $m$ is the number of output samples, $a_i^j$ is the actual output of the $i$ input data point from the $j$ training sample and $d_i^j$ is the desired output of the $i$ input data point from the $j$ training sample. The weights ($w$) and biases ($b$) supplied to the MLLP-SMO as variables to be optimized.

Fig -3 is a flowchart that summarizes the training process of the FNN using the proposed MLLP-SMO trainer. The process is further outlined as follows:

1. Divide input and output datasets into training, validation, and testing data.
2. Supply the training and validation datasets to the feed-forward neural network (FNN).
3. Initialize population size (i.e., number of spider monkeys), local leader limit, global leader limit, maximum group, and perturbation rate of the MLLP-SMO.
4. Generate initial weights and biases using the MLLP-SMO and supply to feed-forward neural network to begin the search and training process.
5. Train the FNN network with the received weights and biases on the training and validation datasets.
6. Check the mean square error (MSE) between each input training sets and respective output training sets.
7. If stopping criteria or minimum MSE is not attained, search through solution space of MLLP-SMO for better weights and biases for further training.
8. If stopping criteria or minimum MSE is attained, output the FNN with the optimal weights and biases.

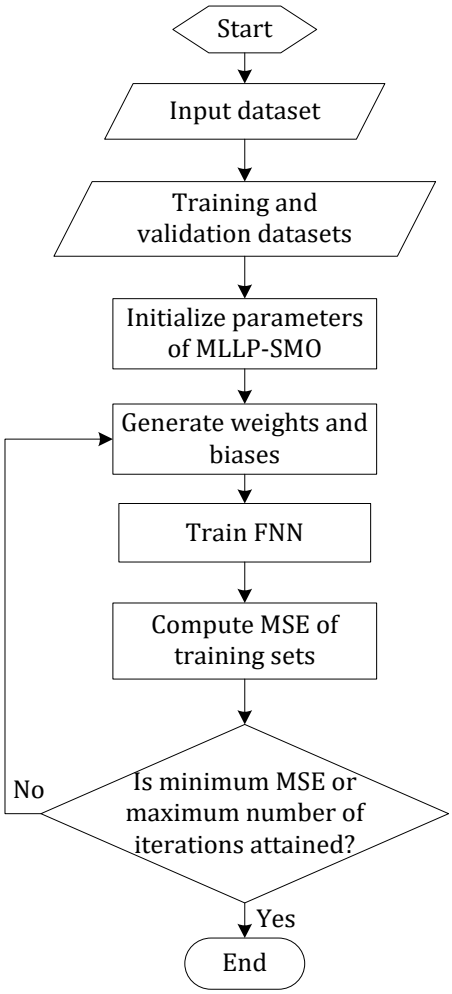


**Fig -3**: Flowchart for FNN training with MLLP-SMO

## 5. APPROACH USED TO TEST PROPOSED TRAINER

The performance of the proposed MLLP-SMO trainer was evaluated using widely used datasets. The performance of the MLLP-SMO was then compared with the original SMO and some good performing trainers in literature. The other trainers that the MLLP-SMO was compared with are particle-swarm optimization (PSO), grey wolf optimizer (GWO), and genetic algorithm (GA). The same datasets were used for all the trainers in both the training and testing phases. All training and testing were done using a windows operating system-based computer having the following specifications: Intel (R) Core TM i7-10750H with CPU of 2.60 GHz and 16.0GB RAM. Sub-section 5.1 presents the parameters used for the MLLP-SMO, GWO, PSO, GA and SMO. The datasets used for training and testing are presented in sub-section 5.2.

### 5.1 Parameters used for trainers

Table -1 presents the parameters used for the MLLP-SMO, SMO, GWO, GA and PSO trainers. The parameters for GWO, GA and PSO were taken from [6]. For each trainer, the initial weights and biases for the FNN were randomized in the range of (-10,10). Due to the randomized selection of weights and biases, each algorithm was run five separate times. For each trainer, the weights and biases for the run that produced the least MSE become the optimized weights and biases for the associated FNN.

### 5.2 Datasets for FNN training and testing

The trainers (i.e., MLLP-SMO, SMO, GWO, GA and PSO) trained the FNN using 6 detection/classification datasets from the Kaggle repository [19]. The datasets are fault detection dataset [20] heart attack analysis and prediction dataset [21], iris species dataset [22], Pima Indians diabetes dataset [23], breast cancer Wisconsin (diagnostic) data set [24], and 4-bit XOR. The dataset for fault detection has 6 input variables and 2 output classes. The 6 input variables are three-phase currents and voltages. The 2 classes of outputs are no-fault condition and fault condition. The heart attack analysis and prediction datasets contain 13 input variables (or attributes) and two output classes. The input variables are age, sex, constrictive pericarditis, chest pain type, resting blood pressure, serum cholestoral, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise induced angina, ST depression induced by exercise relative to rest, slope of the peak exercise ST segment, number of major vessels colored by flourosopy, and Thallium Stress. The two output classes which give a diagnosis of heart disease (i.e., angiographic disease status) are: less than 50% diameter narrowing and greater than 50% diameter narrowing. The iris dataset is an

**Table -1**: Parameters used for trainers.

| Algorithm | Parameter | Value |
|---|---|---|
| MLLP-SMO | Maximum number of groups | 5 |
| | Number of spider monkeys | 100 |
| | Local leader limit | 50 |
| | Global leader limit | 150 |
| | Perturbation rate | 0.1 |
| | Max. no. of iterations | 250 |
| SMO | Maximum number of groups | 5 |
| | Number of spider monkeys | 100 |
| | Local leader limit | 50 |
| | Global leader limit | 150 |
| | Perturbation rate | 0.1 |
| | Max. no. of iterations | 250 |
| GWO | α | Linearly decreased from 2 to 0 |
| | Population size | |

| | | |
|---|---|---|
| | Max. no. of generations | 200<br>250 |
| GA | Type<br>Selection<br>Crossover<br><br>Mutation<br><br>Population size<br>Max. no. of generations | MATLAB optimizer<br>Roulette wheel<br>Single point (probability=1)<br>Uniform (probability=0.01)<br>200<br>250 |
| PSO | Topology<br>Cognitive constant<br>Social constant<br>Population size<br>Inertia constant<br>Maximum number of iterations | Fully-connected<br>1<br>1<br>200<br>0.3<br>250 |

example of discriminant analysis and classification. The dataset has 4 input variables namely sepal length, sepal width, petal length and petal width, all in centimeters. The dataset classifies iris plants into 3 species namely, setosa, versicolor and verginica. The breast cancer dataset contains 30 input variables and two output classes. The output classes are benign and malignant. The 30 input variables were assessed to be excessive so the 'ranksearch' tool in Weka software (a machine learning tool) was used to select the features (variables) whose usage will give the best performance [25]. Consequently, the following five attributes were selected as input variables: radius mean, texture mean, perimeter mean, area mean, and smoothness mean. The XOR was selected to represent a non-linear dataset. It has four attributes with two outputs classes of 0 and 1. The last dataset is the Pima Indians diabetes dataset. This dataset
has 8 input variables and 2 output classes. The input variables are number of pregnancies had, glucose level, blood pressure, skin thickness, insulin level, body mass index, diabetes pedigree function and age. The two output classes are "patient does not have diabetes" and "patient has diabetes". Each of these datasets provided a unique level of difficulty for the classification.

### 5.3 Architecture of FNN

For each FNN, the number of neurons in the input layer equaled the number of input variables (i.e., attributes) for each dataset. With regards to the number of neurons in the hidden layer, the optimal number is determined on a trial-and-error basis using varied number of neurons. However, a number that is equal to 2N+1, where N is the number of neurons in the input layer, is commonly used in literature [6]. Each FNN had 1 neuron in the output layer which outputted the varied classifications of outputs of the datasets.

The structure of the FNN for each dataset is shown in Table -2. The FNN for fault prediction had 6 neurons (1 each for the 6 input variables) in the input layer, 10 neurons in the hidden layer and 1 neuron in the output layer. The FNN for indicating potential heart attack had 13 neurons (1 each for the 13 input variables) in the input layer, 27 neurons in the hidden layer, and 1 neuron in the output layer. The FNN for identifying iris species had 4 neurons (1 each for the 4 input variables) in the input layer, 9 neurons in the input layer and 1 neuron in the input layer. The FNN for breast cancer detection had 5 neurons (1 each for the 5 input variables) in the input layer, 11 neurons in the hidden layer and 1 neuron in the output layer. The FNN for XOR classification had 4 neurons (1 each for the 4 input variables) in the input layer, 9 neurons in the hidden layer and 1 neuron in the output layer. The FNN for detecting diabetes in Pima Indians had 8 neurons in the input layer, 17 neurons in the hidden layer and 1 neuron in the output layer.

The FNN for fault detection was trained such that the output neuron produced an output of either 0 or 1. An output of 0 indicated a no-fault condition whereas an output of 1 communicated the presence of a fault. The FNN for classifying heart attack datasets, was trained to produce an output of 0 if there is less than 50% diameter narrowing and 1 if there is greater than 50% diameter narrowing. The FNN for the iris species classification was trained to give outputs of 1, 2 and 3 for setosa species, versicolor species and verginica species, respectively. Regarding the breast cancer status classification, the FNN was trained to output 0 for a benign condition and 1 for a malignant cancer. For the FNN that will serve as an XOR classifier, the FNN was trained to produce outputs of 0s' and 1s in line with XOR outputs.

The following divisions of the datasets were used for training, validation, and testing. For the dataset on fault detection, 700 sets were used for training, 300 for validation and 633 for testing. Regarding the heart attack dataset, 162 was used for training, 50 for validation and 91 for testing. As regards the iris species dataset, 100 was used for training, 50 for validation and 150 for testing. For the breast cancer dataset, 298 was used for training, 100 for validation and 171 for testing. Relating to the XOR dataset, 10 was used for training, 4 for validation and 16 for testing. Lastly, for the Pima Indians diabetes dataset, 438 was used for training, 100 for validation and 230 for testing.

**Table -2**: Architecture of FNN for datasets

| Datasets | No. of input variables | FNN architecture |
|---|---|---|
| Fault detection | 6 | 6-10-1 |
| Heart attack | 13 | 13-27-1 |

| | | |
|---|---|---|
| Iris species | 4 | 4-9-1 |
| Breast cancer | 5 | 5-11-1 |
| 4-bits XOR | 4 | 4-9-1 |
| Pima Indians diabetes | 8 | 8-17-1 |

## 6. RESULTS AND ANALYSIS

This section presents the performance of the modified local leader phase -spider monkey optimization (MLLP-SMO) trainer and compares it with the performances of the spider monkey optimization (SMO) trainer, the genetic algorithm (GA) trainer, the grey wolf optimizer (GWO) trainer, and the particle swarm optimization (PSO) trainer. The performances were assessed using the mean squared error (MSE), percentage classification accuracy and boxplots (i.e. data fitting). The results have been presented under 6 sub-sections in line with the datasets used.

### 6.1 Fault detection dataset

Table -3 presents the MSEs obtained during training and the classification accuracies during testing, with regards to the dataset for fault detection. It is noted from the Table -3 that the proposed MLLP-SMO produced the least MSE during training. The FNNs trained by the MLLP-SMO and the GWO were the best performing classifiers, with 100% accuracy. These were followed by the GA and SMO trained FNNs. The FNN trained by the PSO exhibited the least classification accuracy. The boxplots for the classifications are shown in Fig -4. It is noted from the figure that the boxplots for the MLLP-SMO and GWO trained FNNs do not have any outliers

Table -3: Performance of trainers for fault detection dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 0.1633 | 76 |
| MLLP-SMO | 0.0125 | 100 |
| GA | 0.1670 | 80 |
| GWO | 0.0182 | 100 |
| PSO | 0.1951 | 75 |

whereas the plots for the GA and SMO trained FNNs, which did not perform well, have several outliers. For the PSO trained FNN, which performed poorly, the first quartile (1Q), third quartile (3Q), mean and median were each 0.6298 giving an inter-quartile range (IQR) of 0.

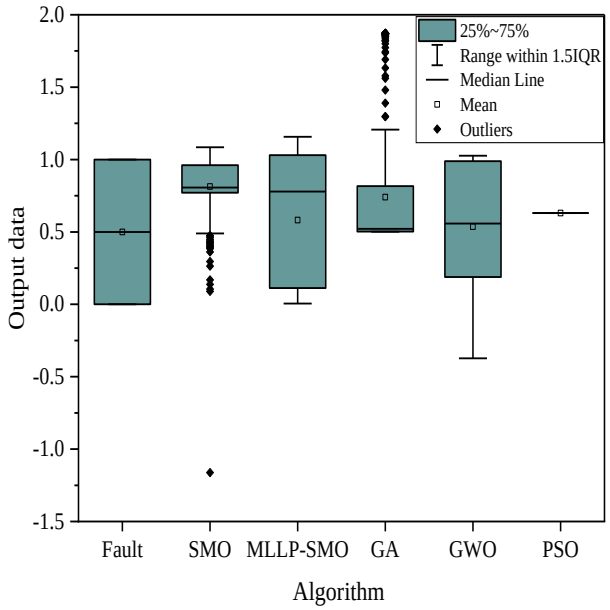


**Fig -4**: Boxplots of FNN outputs for fault detection dataset

### 6.2 Iris species dataset

The MSEs obtained during training and the percentage accuracies of the trained classifies during testing, for the iris species dataset are presented in Table -4. It is noted from the Table that the MLLP-SMO trainer had the least MSE of 0.0103 during training. The SMO trainer had the highest MSE of 0.5578. With regards to the classification accuracy during testing, all the FNNs performed very well. The MLLP-SMO and the GWO equally had the highest classification accuracy of 99%. The SMO had the lowest performance accuracy. Further details of the classification accuracy are presented in the boxplots shown as fig -5. It is observed from fig -5 that the boxplots for the various FNNs closely matche that for the dataset. Although none of the classifiers produced outputs that were outliers, none of them achieved 100% accuracy because, for example, some outputs that were expected to be 1s were misclassified as 2s.

**Table -4**: Performance of trainers for Iris species dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 0.5578 | 94 |
| MLLP-SMO | 0.0103 | 99 |
| GA | 0.1030 | 95 |
| GWO | 0.0271 | 99 |
| PSO | 0.0361 | 97 |

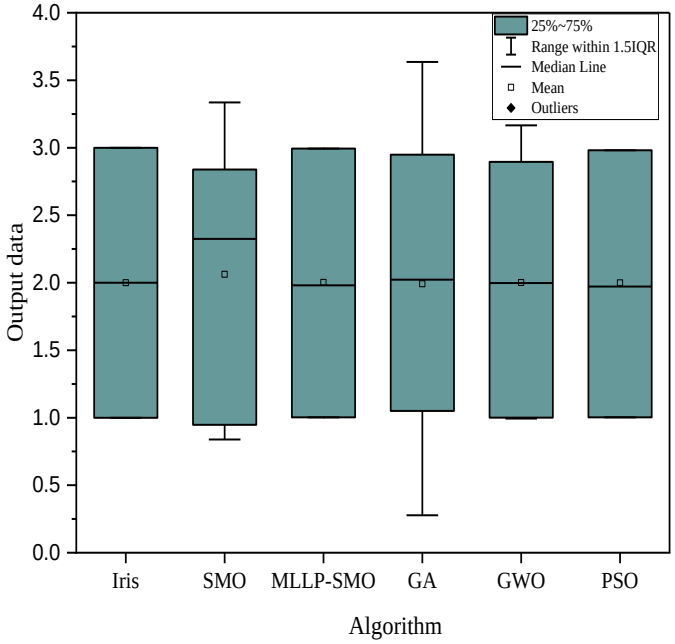


**Fig -5**: Box plot for Iris datasets

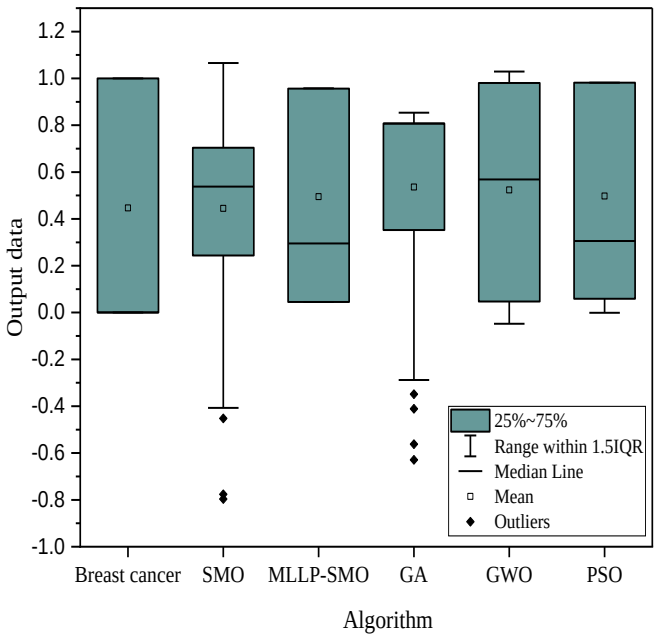


**Fig -6**: Boxplot for breast cancer datasets

### 6.3 Breast cancer dataset

For the breast cancer dataset, the MSEs obtained during training and the classification accuracies are presented in Table -5.

**Table -5**: Performance of trainers for breast cancer dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 0.6940 | 89 |
| MLLP-SMO | 0.2041 | 94 |
| GA | 0.7582 | 84 |
| GWO | 0.2169 | 93 |
| PSO | 0.1942 | 95 |

It can be observed from the Table that the PSO had the least MSE during training followed by the MLLP-SMO. The trainer that had the highest MSE was the GA. With regards to the accuracies of the trained FNNs, the FNN trained by the PSO had the highest accuracy of 95%. This was followed by the FNN trained by the MLLP-SMO with an accuracy of 94%. The FNN trained by the GA was the least performing classifier, with an accuracy of 84%. The boxplots for the outputs of the classifiers are shown in fig -6. It is noted from fig. -6 that the FNNs trained by the GA and SMO had several outliers, which accounted for their rather poor performances. The plots for the high performing FNNs which were trained by the PSO, MLLP-SMO and GWO closely matches that of the dataset.

### 6.4 XOR dataset

Table -6 shows the MSEs computed for the trainers during the FNN training. The Table also shows the classification accuracies of the FNNs trained by the various trainers. The MLLP-SMO had an exceptionally low MSE of 0.004 and was followed by the GWO with a low MSE of 0.0812. The PSO had the highest MSE of 0.6475. Both the FNNs trained by the MLLP-SMO and GWO had classification accuracies of 100% and were therefore the best performing classifiers. These were followed by the FNN trained by the GA which had a classification accuracy of 87%. The classifiers trained by the SMO and the PSO were equally poor performing with each having an accuracy of 75%. The boxplots of the various classifiers are presented in fig -7.

**Table -6**: Performance of trainers for XOR dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 0.5830 | 75 |
| MLLP-SMO | 0.0004 | 100 |
| GA | 0.4733 | 87 |
| GWO | 0.0812 | 100 |
| PSO | 0.6475 | 75 |

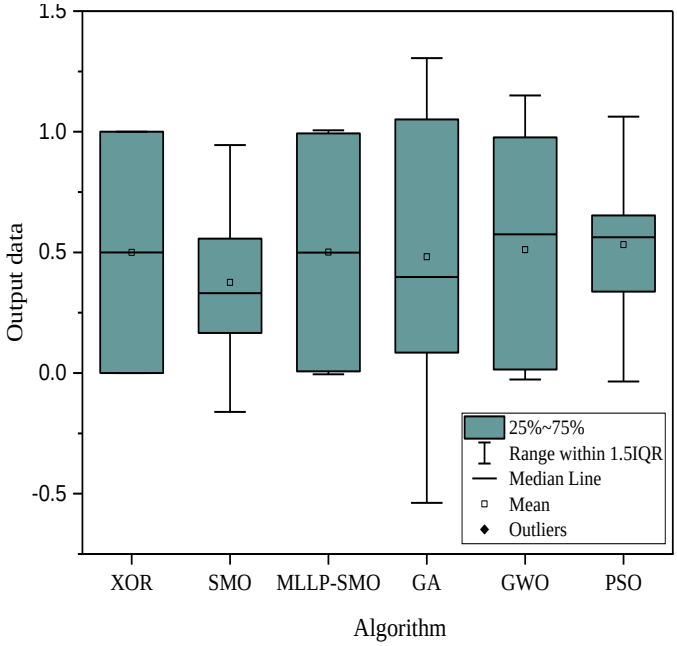


**Fig -7**: Boxplot for XOR datasets

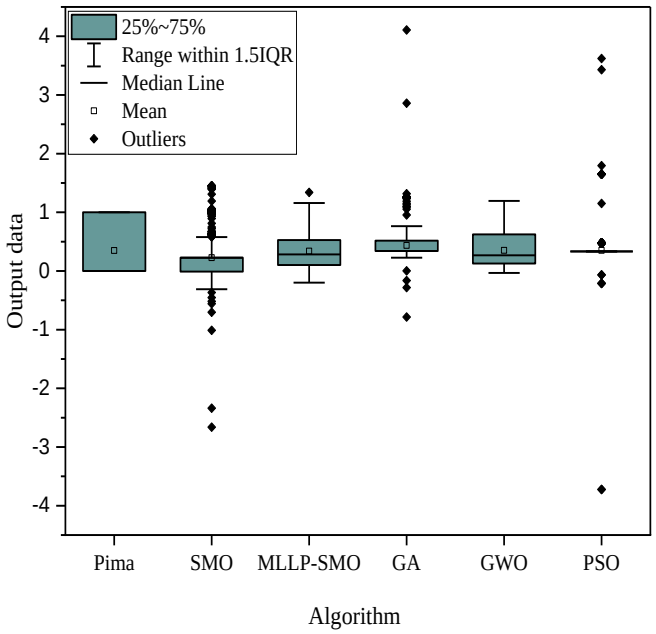


**Fig -8**: Boxplot for Pima datasets

### 6.5 Pima Indians diabetes dataset

The MSEs obtained during the trainings are presented in Table -7. The Table also shows the classification accuracies of the FNNs trained by the various algorithms. From Table -7, the MSEs are generally high. This is due to the complex nature of the dataset. The MLLP-SMO trainer however had the lowest MSE of 0.6023 and was followed by the GWO with an MSE of 0.7571. The PSO had the highest MSE of 1.3525. The generally high MSEs translated into poor performing FNNs. All the classifiers had equally poor accuracies of 65%. The boxplots for the outputs of the classifiers are shown in fig -8. The plots show several outliers for the classifiers trained by the SMO, GA and PSO. The MLLP-SMO had the least number of outliers with the GWO producing no outlier. Although the GWO produced no outlier classification, that did not translate into higher classification accuracy. This was because several of the outputs were misclassified. Some desired outputs of 1 were misclassified as 0. Similar misclassifications were recorded for all the other classifiers.

**Table -7**: Performance of trainers for Pima Indians dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 1.1037 | 65 |
| MLLP-SMO | 0.6023 | 65 |
| GA | 1.0640 | 65 |
| GWO | 0.7571 | 65 |
| PSO | 1.3525 | 65 |

### 6.6 Heart attack dataset

Table -8 shows the MSEs obtained during training and the accuracies of the classifiers during testing, for the heart attack dataset. It is noted from the Table that the MLLP-SMO had the least MSE, followed by the GWO. The trainer that had the highest MSE was the GA. With regards to the classification accuracy, none of the classifiers could attain 100% accuracy.

**Table -8**: Performance of trainers for heart attack dataset

| Algorithm | MSE (Training) | Accuracy (%) (Testing) |
|---|---|---|
| SMO | 0.5257 | 84 |
| MLLP-SMO | 0.3399 | 89 |
| GA | 0.9709 | 67 |
| GWO | 0.3790 | 87 |
| PSO | 0.4805 | 85 |

This is due to the complex nature of the heart attack dataset. Notwithstanding this, the MLLP-SMO trained FNN produced the best performance, with a percentage accuracy of 89%. The FNN trained using GWO followed with a classification accuracy of 87% and then the FNN trained by the PSO which had a classification accuracy of 85%. The GA trained FNN had the lowest accuracy of 67%.

## 7. CONCLUSIONS

The training of feedforward neural networks (FNN) has been enhanced using a modified local leader phase spider monkey optimization (MLLP-SMO) algorithm. The MLLP-

SMO has been compared with four other algorithms. The MLLP-SMO trainer had the least MSE in 5 out of the 6 datasets used for performance assessment. It shared the top spot with the GWO, on trained FNN classifier accuracy, in 3 out of the 6 datasets. The MLLP-SMO classifier was the best performing classifier in 1 out of the 6 datasets. Its classifier performed equally with the other classifiers in 1 out of the 6 datasets. The classifier trained by the MLLP-SMO was however the second best performing, after the PSO, in 1 out of the 6 datasets. Thus, the MLLP-SMO outperformed all the other optimization algorithms as a trainer for feedforward neural networks. The best performance of the MLLP-SMO is due to its higher ability to avoid local optima entrapment in training of the FNN to produce optimal weights and biases. Thus, compared with the other algorithms, the MLLP-SMO has better exploitation and exploration abilities. The use of the proposed MLLP-SMO trainer for training FNNs will yield higher classification accuracies than the other algorithms.

## REFERENCES

[1] R. Matsumura, K. Harada, Y. Domae, and W. Wan, "Learning based industrial bin-picking trained with approximate physics simulator," Adv. Intell. Syst. Comput., vol. 867, pp. 786–798, 2019.

[2] H. H. Abdallah, M. Chtourou and T. Guesmi and A. Ouali, "Feedforward neural network-based transient stability analysis of electric power systems", European Transactions on Electrical Power, vol. 16, no. 6, pp. 577 – 590, 2006

[3] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking Neural Networks and online learning: An overview and perspectives," Neural Networks, vol. 121, pp. 88–100, 2020.

[4] J. A. Bullinaria, "Radial Basis Function Networks: Algorithms," Neural Comput. Lect. 14, pp. 1–12, 2015.

[5] T. Kohonen, "Essentials of the self-organizing map", Neural Networks, vol. 37, pp. 52-65, 2013.

[6] S. Mirjalili, "How effective is the Grey Wolf optimizer in training multi-layer perceptrons," Applied Intelligence, vol. 43, no. 1, pp. 150–161, 2015.

[7] R. Sathya, and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification", International Journal of Advanced Research in Artificial Intelligence, vol. 2, no. 2, pp. 41-80, 2013.

[8] M. Buscema, "Back Propagation Neural Networks", Substance Use & Misuse, vol. 33, no. 2, pp. 233-270, 1998.

[9] F. Dkhichi and B. Oukarfi, "Neural Network Training By Gradient Descent Algorithms : Application on the Solar," International Journal of Innovative Research in Science Engineering and Technology, vol. 3, no. 8, pp. 15696–15702, 2014.

[10] G. Wang, L. Guo, A. H. Gandomi, G. Hao, and H. Wang, "Chaotic Krill Herd algorithm," Inf. Sci. (Ny)., vol. 274, pp. 17–34, 2014.

[11] D. Devikanniga, K. Vetrivel, and N. Badrinath, "Review of meta-heuristic optimization based artificial neural networks and its applications," J. Phys. Conf. Ser., vol. 1362, no. 1, 2019.

[12] H. Sharma, G. Hazrati, and J. C. Bansal, "Spider monkey optimization algorithm," Stud. Comput. Intell., vol. 779, no. January, pp. 43–59, 2019.

[13] M. H. Sazli, "A brief review of feed-forward neural networks", Commun. Fac. Sci. Univ. Ank. Series A2-A3, vol. 50, no. 1, pp 11-17, 2006.

[14] E. A. Frimpong, P. Y. Okyere and J. Asumadu, "On-line determination of transient stability status using multilayer perceptron neural network", Journal of Electrical Engineering, vol. 69, no. 1, pp. 58–64, 2018.

[15] M. H. Beale, M. T. Hagan, H. B. Demuth, Neural Network ToolboxTM, User Guide, MATLAB, R2016b, pp. 3-2 - 3-5, 2016.

[16] H. Sharma, G. Hazrati, and J.C. Bansal, "Spider monkey optimization algorithm, Evolutionary and Swarm Intelligence Algorithms", Studies in Computational Intelligence, 779, 43–59, 2019.

[17] A.Younes, A. Elkamel, and S. Areibi, "Genetic algorithms in chemical engineering: a tutorial", Book Chapter on Evolutionary Computations in Chemical Engineering, World scientific book, pp. 1-32, 2008.

[18] "Genetic Algorithms - Parent Selection – Tutorialspoint". [Online].https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm. [Accessed: 30-Apr-2021].

[19] https://www.kaggle.com/datasets, [Accessed: 30-Apr-2021].

[20] "Fault detection and classification dataset", [online] https://www.kaggle.com/esathyaprakash/electrical-fault-detection-and-classification [Accessed: 7-Jul-2021].

[21] "Heart Attack Analysis & Prediction Dataset", [Online]. https://www.kaggle.com/rashikrahmanpritom/heart-attack-analysis-prediction-dataset. [Accessed: 14-May-2021].

[22] "Iris Species" [Online]. https://www.kaggle.com/uciml/iris. [Accessed: 14-May-2021].

[23] "Pima Indians Diabetes Database" [Online]. https://www.kaggle.com/uciml/pima-indians-diabetes-database. [Accessed: 14-May-2021].

[24] "Breast Cancer Wisconsin (Diagnostic) Data Set" [Online]. Available: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. [Accessed: 14-May-2021].

[25] https://sourceforge.net/projects/weka/files/weka-3-6-windows-x64/3.6.9/ [Accessed: 16-Dec-2020].