<table>
<thead>
<tr>
<th>Manuscript Number</th>
<th>NCA-1252R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Title</td>
<td>PREPROCESSING AND PRETRAINING OF MULTI-LAYER FEED FORWARD NEURAL NETWORK MODEL</td>
</tr>
<tr>
<td>Article Type</td>
<td>Original Article</td>
</tr>
<tr>
<td>Keywords</td>
<td>Dimension Reduction, Supervised and Unsupervised Learning, Multi-layer Feed Forward Neural Network, Training, Preprocessing, Pre-training.</td>
</tr>
<tr>
<td>Corresponding Author</td>
<td>Roya - Asadi, Ph.D candidate UM, university of Malaysia KL, MALAYSIA</td>
</tr>
<tr>
<td>Corresponding Author's Institution</td>
<td>UM, university of Malaysia</td>
</tr>
<tr>
<td>First Author</td>
<td>Roya - Asadi, Ph.D candidate</td>
</tr>
<tr>
<td>First Author Secondary Information</td>
<td></td>
</tr>
<tr>
<td>Order of Authors</td>
<td>Roya - Asadi, Ph.D candidate Sameem Abdul Kareem, Ph.D</td>
</tr>
<tr>
<td>Order of Authors Secondary Information</td>
<td></td>
</tr>
</tbody>
</table>
PREPROCESSING AND PRETRAINING OF MULTI-LAYER FEED FORWARD NEURAL NETWORK MODEL

Roya Asadi\textsuperscript{1}, Sameem Abdul Kareem\textsuperscript{2}
\textsuperscript{1,2}Department of Artificial Intelligence, Faculty of computer Science and Information Technology; University of Malaysia
\textsuperscript{1}asadiroya@gmail.com, \textsuperscript{2}sameem@um.edu.my.

Abstract
Multi-layer Feed Forward Neural Network (MFFNN) model obtains its data by learning from the real-world environment. Also MFFNN dynamically recognizes the type of input information through their weights and properties. Learning in MFFNN model is considerably slow because biases and weights have to be updated in each epoch during learning. Currently, data preprocessing and pre-training are the contributing factors in developing efficient techniques for fast MFFNN processing at high accuracy and reduced training time. Finding an effective preprocessing technique is an active area of research. The main problem for supervised MFFNN model lies in finding the suitable weights during training in order to improve training time and achieve higher accuracy. The important issue in the training process of existing MFFNN models is in the initialization of the weights. However, this process is random and creates the paradox of low accuracy and high training time. This study reviews data pre-processing and pre-training techniques in Multi-layer Feed Forward Neural Network. Weights Linear Analysis (WLA) is the newest composition of data preprocessing and pre-training techniques for MFFNN models to increase their classification accuracy and to reduce training time. MFFNN model WLA as a preprocessing and pre-training technique acts as an efficient intelligent classification that can be used for learning the model and is applicable to different domains.

Keywords: Preprocessing, Pre-training, Dimension Reduction, Supervised and Unsupervised Learning, Multi-layer Feed Forward Neural Network, Training.

1. INTRODUCTION

1. Neural networks are suitable for extracting rules, quantitative evaluation of these rules, clustering, self-organization, classification, regression feature evaluation, and dimensionality reduction \cite{22,44}. Learning is the important property of neural networks. Neural networks are able to dynamically learn types of input information based on their weights and properties. During learning, the weights of each value in hidden layers are considered. As the domain becomes smaller and smaller, suitable weights will be obtained through this series of repeated trial and errors after several epochs. Suitable data pre-processing techniques are necessary to find the input values while pre-training techniques are used to find desirable weights that in turn will reduce the training process. This is the essence of the Multi-layer Feed Forward Neural Network (MFFNN), such as the Back Propagation Network (BPN). The combination of data pre-processing and pre-training in MFFNN helps to choose the right input attributes and to generate desirable output with higher efficient results in both speed and accuracy. Neural network models
learn to predict the output based on real world information. Changing the input data or initial conditions can immediately affect the classification accuracy in MFFNN models. Without pre-processing, the classification process is comparatively slow and it may even converge leading to an incomplete classification. Pre-processing and pre-training is applied to improve the accuracy, efficiency, and scalability of the classification [19]. Both processes are required to increase the efficiency in MFFNN [57, 23]. Nonetheless, the main problems in data pre-processing and pre-training include identifying suitable input values and attributes, suitable weights, and considerable processing time [10, 2, 27, 19]. Existing pre-training techniques generate suitable weights for reducing the training process but with the application of random values for initial weights [50, 9]. Pre-processing in the real world environment focuses on some techniques such as data transformation, data reduction, and pre-training. Data transformation and normalization are two important aspects of pre-processing. Data transformation is the coding of values in the input data matrix in order to change them to a reduced form by means of algebraic or statistical formula. Data normalization transforms one input value to another suitable data by distributing and scaling. Data reduction such as dimension reduction and data compression are applied to minimize the loss of information content. MFFNN models are able to identify information of each input based on its weight, hence increasing the processing speed [50]. Pre-training techniques reduce the training process through the preparation of suitable weights.

In this paper, various techniques for data pre-processing and pre-training are reviewed. The advantage and disadvantages of these techniques are also compared.

2. CONCEPTS IN FEED FORWARD NEURAL NETWORKS

McCulloch and Pitts [43] introduced the neural network, which is a creation of computational power through the combination of only a few simple processing units based on the idea of weight sets. Hebb [21] developed the meaning of the first learning rule which was subsequently adopted for the neural network. Haykin [20] completed this idea with a parallel distributed processor. Widrow and Hoff [53] introduced the Mean Square Error (MSE) in the perceptron study which is widely used in ADALINE (ADaptive Linear Neuron) and MADALINE in multi-layer networks. Doszkocs et al. [11] had three prospects of neural networks models, these are, namely: networks based on logical neurons, elementary perceptron and linear networks. Haykin [20] considered the brain feature in which the neural networks obtain knowledge from, comes with different their weights or strengths. These various ideas led to artificial neural networks which consist of neurons in three layers; the input layer, hidden layer and the output layer [20]. Each unit has its weight, and the sum of inputs values is computed for each unit individually. Next, every unit computes an activation function in order to obtain the desired output, before passing it down to the next layer in the feed forward topology. The number of layers and their units in each layer are selectable. The training model works on the basis of network weights and some thresholds. They also must be set to minimize the prediction error made by the network. The concept of learning consists of three majors: supervised learning, unsupervised learning, and reinforcement learning [19]. Supervised training is similar to unsupervised training in the sense that the training sets are provided. The difference between the two is that in supervised training the desired output is provided and the weight matrix is applied based on the difference between the predicted output and the actual output of the neural network. In back-propagation, the delta rule learning uses
gradient-based optimization methods in two basic steps: in the first step the gradient of error function is calculated and the gradient is subsequently employed in the second step. The optimization procedure consists of a relatively large number of small steps, causing the learning to be considerably slow [7]. Optimization problem in supervised learning can be shown as the sum of squared errors between the output activations and the target activations in neural network as well as the minimum weights. Unsupervised learning or self-organized learning finds regularities in the data represented by examples. Most applications in unsupervised learning are estimative problems, such as statistical modeling, compression, filtering, blind source separation, and clustering. In reinforcement learning, item x is usually not given, but is capable of generating certain effects and interactions to the environment. At each point in time, the environment generates an observation for x. The environment is dynamic, hence long-run cost is unknown, but it can be estimated. Neural networks are frequently used in reinforcement learning as part of the overall algorithm. Neural Networks are classified into Feed Forward Neural Network and Recurrent networks (RN). The problem tasks can be divided into five types of distinct application [22]: Classification, Association, Codification, Simulation, and Modeling. In each neuron of the neural networks, input samples are often transformed for computing the output through an activation function. This activation function comes with a specific range of squashing function, known as the identity function, binary step function, sigmoid function, and bipolar sigmoid function as shown in [10]. The back propagation network (BPN) is introduced by Werbos [52] and can be used as a supervised Multi-Layer Perceptrons (MLP) in feed forward network. BPN computes each output value with its sigmoid function of inputs in the forward connection, and computes its error backwards. The algorithm employs weights of each connection in order to reduce the value of the error function. This cycle is repeated until the error becomes much smaller. The user must decide on the network topology with regards to the number of layers and neurons. The Back propagation network training process has several steps as follows:

- **Normalized the input values**
  The input values X are normalized between 0.0 and 1.0 using the min and max technique because standard sigmoid is being used as the activation function for speed up training process.

- **Initialize the weights**
  The weights W are initialized to some random numbers between 0 and 1 (or -1 to 1 or -0.5 to 0.5) for all input values. Each unit has a bias associated with it.

- **Propagating the input forward**
  For each hidden or output layer unit j, the \( I_j \) net input of unit \( j \) is computed with respect to the previous layer \( i \), \( I_j = \sum_i W_{ij}O_i + \theta_j \) where \( O_i \) is the output of unit \( j \), \( \theta_j \) is the threshold of unit \( j \) and \( W_{ij} \) is the weight between units \( i \) and \( j \) as well as the output (sigmoid) function of each unit \( j \), that is, \( W_{ij} = O_{ij} = 1/(1+e^{-\theta_i}) \). After the net input of each unit is computed, each connected input to the unit is multiplied by its corresponding weights and summed. For each unit, a bias is initialized as \( \theta \) the unit threshold.

- **Back-propagating the errors**
  The error \( Err_j = O_j(1-O_j)(T_j-O_j) \) is computed for each unit \( j \) of the output layer where \( T_j \) is the computed output and \( O_j \) is the actual output of unit \( j \), so the error \( Err_j = O_j(1-O_j) \sum_k \).
Err_k W_{jk} is computed for each unit j of the hidden layer with respect to the next higher layer k where W_{jk} is the weight between unit j and unit k.

- **Updating weights and biases**
  The above steps are repeated for each input vector in the training set with updated weights and bias coefficients until the error become acceptably low.

\[
\Delta W_{ij} = (L)Err_j O_i \quad \text{Weight increment}
\]

\[
W_{ij} = W_{ij} + \Delta W_{ij} \quad \text{Weight updating}
\]

\[
\Delta \theta_{ij} = (L)Err_j \quad \text{Bias increment}
\]

\[
\theta_{ij} = \theta_{ij} + \Delta \theta_{ij} \quad \text{Bias updating}
\]

L represents the Learning Rate, which is a real number constant between 0.0 and 1.0. Δpoints indicates the single value for changing. Weights and biases are updated for all of the samples in the training set. This strategy is called epoch updating. All the steps can be summarized in the back-propagation network algorithm and the flow chart is shown in Figure 1 [19].

---

BPN (D, Weights, Biases; CL)
Input: Database D, database of input values; Initialize all weights and biases in network;
Output: CL, Classified List of Database D;

Begin

While termination condition is not satisfied { 

//Propagating the inputs forward:
For all training sample X in samples { 

For all hidden or output layer unit j { 

  // Compute the net input of unit j with respect to the previous layer, i
  \[ I_j = \sum_i W_{ij} O_i + \theta_{ij} \]

  // Compute the output of each unit j
  \[ O_i = 1/(1+e^{-I_j}) \]

  // Back propagate the errors:
  For all unit j in the output layer
    // Compute the error
    \[ Err_j = O_j(1-O_j)(T_j-O_j) \]
    For all unit j in the hidden layers, from the last to the first hidden layer
      // Compute the error with respect to the next higher layer, k
      \[ Err_j = O_j(1-O_j) \sum_k Err_k W_{jk} \]
      For all weight W_{ij} in network { 
        // Weight increment
        \[ \Delta W_{ij} = (L)Err_j O_i \]
        // Weight updating
        \[ W_{ij} = W_{ij} + \Delta W_{ij} \]
        For all bias \theta_j in network { 
          // Bias increment
          \[ \Delta \theta_{ij} = (L)Err_j \]
          // Bias updating
          \[ \theta_{ij} = \theta_{ij} + \Delta \theta_{ij} \]
        }
      }
    }
  }
}

Figure 1: BPN algorithm [19]
Training in BPN stops when one of the following happens: when all $\Delta W_{ij}$ in the previous epochs are very small, the percentage of samples are misclassified in the previous epoch is below the threshold, or a predetermined number of epochs has expired. In training the BPN, the input pattern is divided into two sets: a training set and a test set. The error on the training set shows how well the network has learned to fit the data. The test set performance shows how well the network generalizes. A network with fewer weights may not be perfect to model the underlying function. For example, a network with no hidden layers is really a model with a simple linear function.

If the training set error is very low as compared to the test set error, network has over-fitted the data. Over-fitting the data during the training process is essentially over-learning. When this happens, the training will decrease the number of hidden units and/or hidden layers [19]. The time complexity of the BPN depends mainly on the number of weighted functions in the hidden layers $f_h$ and the number of iterations $L$, hence the time complexity of BPN is $O(Lf_h)$.

3. DATA PRE-PROCESSING TECHNIQUES OF FEED FORWARD NEURAL NETWORK

Learning in neural networks will be faster and the performance will be better when the input values are pre-processed. Some important data pre-processing aspects are as follows:

*Scaling*: To equalise the importance of the variables. The attribute values are tailored so that they fall within a specified range. Considering average function of each attribute values and computing every attribute ratio to this average is a very simple $\text{Min (Minimum)}$ and $\text{Max (Maximum)}$ technique in normalization.

*Transformation*: To normalize distribution of input values.

*Trend*: To recognize current status of the variable by the most recently available value for the variable; to recognize the volatility of the variable over time by using standard deviation or dividing them by the absolute value of the mean of the points in the data series; to recognize the direction of the input value. One simple way is using the percentage of changing. Normalization is necessary for comparison across data series with different scales.

*Seasonality*: The seasonal or postponement aspect of some types of data which is another aspect of trend analysis. In this case, the input value must be considered cyclically.

*Categories*: To recognize the right category of each input values and to give one special code to the input values.

*Thermometer coding*: To measure the rank of the input values and the differences between them by using ordinal variables.

*Circular discontinuity*: Some input values are fundamentally circular and maybe two different situations point to one value. The input values with different situations can be encoded.
3.1 DATA PRE-PROCESSING FUNCTIONS

Several powerful data pre-processing functions on ordinal basis of improving efficiency will be discussed as the latest methods in this study [10, 19]. These are mathematical and statistical functions to scale, filter, and pre-process the data.

- Data cleaning

In real world, data is incomplete, noisy and inconsistent. There are some basic techniques for data cleaning as follow [19].

**Missing values:** For solving this problem, there are several methods such as ignoring the tuple, filling in the missing value manually, using a global constant to fill in the missing value, using the attribute mean to fill in the missing value, using the attribute mean for all samples belonging to the same class as the given tuple and using the most probable value to fill in the missing value.

**Noisy data:** Noise is a random error or variance in a measured variable. There are some techniques for smoothing data such as; binning, in which the attribute values are first sorted and then partitioned into buckets or bins. Then, they are smoothed by bin means, bin median, or bin boundaries.

**Inconsistent data:** This problem can be solved by using external references.

Data cleaning has different techniques to fill missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies. Therefore, data cleaning is not just dataset updating by using good data [19, 33, 32, 45]. Kimbal [32] described six phases for data cleaning: elementizing, standardizing, verifying, matching, house holding and documenting. Important issues in data mining are duplication and redundancy of data [16]. Some of the statistical functions for data cleaning are mean, standard deviation, range [3]. These techniques are fast and easy. Another technique for data cleaning is pattern-based techniques. This method uses combined techniques such as partitioning, classification and clustering [33, 45]. Association rules with high accuracy can also be applied for data cleaning. Association rules can use data of different types and give more information about the relationships between data values [40, 41].

- Data integration

A database with multiple data sources and data cubes has problems of data duplication and redundancy of. These databases need an integrated access to multiple data sources. Database integration utilizes two main techniques: schema integration and data integration [6, 19].

The correlations between attributes of a dataset can show the problems of data redundancy and duplication [6, 19, 17]. In statistics, the correlation is $\rho_{ij}$ based on the standard deviations $\sigma_i$, $\sigma_j$ and covariance $\text{Cov}_{ij}$. The covariance function returns the average deviation for each pair of attributes. If the result of $\rho$ is greater than zero, the attributes have a positive correlation. If the result of $\rho$ is zero, the attributes are independent. If the result of $\rho$ is less than zero, the attributes have negative correlation.

Data value conflicts are another problem of database with multiple data sources [6, 19, 17]. In the real world scenario, attribute values from different sources may differ because of different
representation, scaling, or encoding such as different metric units. The functions of the mean and standard deviation can solve this problem. \( \mu \) is the mean of values for each tuple, and \( \sigma \) is the standard deviation each tuple. \( Z \) is a standard normalized value for attribute \( X \) and is computed based on the formula below.

\[
Z = \frac{(X - \mu)}{\sigma}
\]

Solving the problem of redundancy can help to improve speed and accuracy of the data mining process.

- **Data transformation**

Data transformation can involve items of smoothing, aggregation, normalization and data reduction:

**Smoothing**: To remove noise; such as, binning, regression and clustering;

**Aggregation**: Summarization or aggregation of operations is applied to the data;

**Generalization**: To replace low level (primitive record) data by higher level concepts through the use of concept hierarchies;

**Normalization**: To scale attribute data and to fall within a small specified range. Normalization techniques are the same as Min and Max normalization, Z-score normalization. In the work by Neal et al. [46] on prediction of the gold market, the preprocessing experiment included scaling the input data. Min and max technique is used in Standard Back Propagation Network (SBPN) to transform and to scale the input values between 0 and 1 if the activation function uses the standard sigmoid, and -1 to 1 for accelerating process [19]. The technique of using Log (input value) is similar to the min and max in the range (0, 1). Another similar method is the Sin (Radian (input value)) between -\( \pi \) to +\( \pi \) where Radian (input value) must be between 0 to \( \pi \). The classification process will be faster given a symmetric data. When input value is large, it will be difficult to recognize differences between the hidden layer neurons of two near large values, because their difference will be very subtle [37]. Brown et al. [5] analyzed the prediction of desired output by using \( \tanh \) on the hidden layer for speed-up convergence. They showed that when the input values of a two-layer network with linear output function is between \(-\alpha\) to \(\alpha\), the learning is faster than when it is in the range of 0 to \(\alpha\), specially when using \( \tanh \) as the activation function in three hidden layers. The main disadvantage of the min and max technique is the lack of a special and unique class for each data [37];

**Attribute construction**: To construct and add new attributes from the given attributes set to help the mining process.

- **Data reduction**

To reduce volume of data representation and transform date to the same analytical results:

**Data cube aggregation**: Aggregation of operations and applying them to the data in the construction of a data cube. A data cube is a three or higher dimensional matrix of values.
Data compression: Encoding the data and reducing the dataset size for data compression. If the original data can be reconstructed from the compressed data without any loss of information, it is called lossless or we can reconstruct an approximation of the original data, it is called lossy.

Numerosity reduction: To replace the data by alternative, smaller data representations such as parametric models or non parametric models such as regression and log-linear models, clustering, histograms, sampling.

Discretization and concept hierarchy generation: To divide the range of attributes from continuous to intervals. Some classification algorithms works on categorical (non-numerical) attributes. Data discretization is useful for automatic generation of concept hierarchies based on the number of distinct values of each attribute. The attributes with the most distinct values is placed at the lowest level of the hierarchy.

Dimension data reduction: Dimension data reduction method projects high dimensional data matrix to lower dimensional sub-matrix for effective data pre-processing. There are two types of reduction, which are supervised and unsupervised dimension reduction. The type of reduction is based on the relationship of the dimension reduction to the dataset itself or to an integrated known feature. In supervised dimension reduction, suitable sub-matrices are selected based on their scores, prediction accuracy, the selection of the number of necessary attributes, and computing the weights with a supervised classification model. Unsupervised dimension reduction maps high dimension matrix to lower dimension and creates new low dimension matrix considering just the data points.

Dimension reduction techniques are also divided into linear and nonlinear methods based on the consideration of various relations between parameters. In the real world, data is nonlinear; hence only nonlinear techniques are able to handle them. Linear techniques consider linear subsets of the high dimensional space, while nonlinear techniques assume a more complex subset of the high dimensional space [50, 23]. Jolliffe [26] explained one powerful method in dimension reduction, namely, the Principal Component Analysis (PCA) [27, 50, 8, 39]. PCA is a classical multivariate data analysis method that is useful in linear feature extraction and data compression. If the dimension of the input vectors is large, the vector components are highly correlated (redundant). In this situation, it is useful to reduce the dimension of the input vectors.

PCA technique has three effects [47, 39], it orthogonalizes the components of the input vectors so they are uncorrelated with each other; it orders the resulting orthogonal components (principal components) so that those with largest variation come first; it eliminates components that contribute the least to the variation in the data set. Next, the input vectors are normalized and zero mean and unity variance are computed before the mean and standard deviation method is employed [10]. The basic assumption is that most information in classification of high dimensional matrix has a large variation.

PCA computation maximizes the variance in the process environment for standardized linear process. The advantages of PCA are three-fold. First, it is an optimal linear scheme for compressing a set of high dimensional vectors into a set of lower dimensional vectors and reconstructing the original set. PCA is often used for reducing the dimensional input values into
two or three dimensions. Second, dimensionality reduction is possible through computation of the highest variance in the components of the input feature vector, without performing any transformation on the input environment. The input values are analyzed within their own input environment, and the transformations results are deterministic as well as independent from initial conditions. Third, compression and decompression are easy to perform [35] because calculation is based on matrix multiplication. The time complexity of PCA is $O(p^2n)+O(p^3)$. The disadvantage of PCA is that, PCA is not able to find non-linear relationship within the input values; therefore that kind of data will be lost. This is essentially losing the input values during training.

3.2 Using soft computing for data flexibility

Soft computing is an association of methodologies. Soft computing methodologies such as fuzzy sets, genetic algorithms, rough sets and neural network are most widely applied in the data mining [44]. Hybrids of neural networks have been used in a variety of data mining tasks [4], clustering and dimensionality reduction.

Fuzzy set by Zadeh [54] provide natural models for dealing with uncertain process and is suitable for incomplete data, noisy and fast processing. It can be used in clustering, association rules and functional dependencies and summarization. The elements of the fuzzy sets have degrees of membership in the range of $[0, 1]$; 0 for not being a member and 1 for being member. Membership values from 0 to 1 are generated by a membership function. Fuzzy logic is a hyperset of the schematic Boolean logic with extensions to provide for general information. Neuro-fuzzy is combination of the human-like reasoning style of fuzzy systems with the learning of neural networks and IF-THEN fuzzy rules set. Some of the most efficient soft computing rule generation methods are neuro-fuzzy systems. Neuro-fuzzy is a combination of features of neural networks and fuzzy sets in generating natural rules, handling data and highly nonlinear decision limits and querying unknown data to reach a decision [1, 34, 48].

Neural networks and rough sets are used for classification and rule generation [57]. A rough set has been described by Zdzisław [55] and it is based on discovering redundancies, dependencies in datasets and is suitable for dimension reduction of large datasets. A rough set is a formal estimation of a schematic set in terms of a pair of sets which give the lower and the upper estimation of the original set. In other variations of rough set, the estimating sets may be fuzzy sets. Using rough set theory is suitable for solving the problem of noisy data too. The rough set approach is based on discrete-valued attributes and equivalence classes within the given training data. Genetic algorithms as described by Holland [24] are used for various optimization and search processes, like query optimization and template selection. Genetic algorithms do not use any gradient information. The rules of genetic algorithms can be represented by a string of bits. Genetic algorithms can be applied together with regression association rules models. George and Srikanth [18] applied a fuzzy-genetic integration where genetic algorithms are applied to determine suitable data summary. Kiem and Phuc [29] developed a rough-neuro-genetic hybridization for discovering clustering of large databases.

4. PRE-TRAINING TECHNIQUES OF FEED FORWARD NEURAL NETWORK

Initialization of weights is the first critical step in the training process of supervised Multi-layer Feed Forward Neural Network (SMFFNN) models such as the back-propagation networks
Training can be accelerated through a good initialization of weights during pre-training. To date, random numbers are used to initialize the weights [50, 9]. The number of epochs in the training process depends on the initial weights. In the BPN, correct weights results in successful training. Otherwise, the BPN may not obtain an acceptable range of results and may halt during training. Training of the BPN includes the choice of the activation function in hidden layers for weights computing. Usually, the initialization of the weights in the pre-training of the BPN is random. Most papers do not report the evaluation of the speed and accuracy, only some comments about initializing of weights, network topology such as the number of layers and unknown practical nodes, if any. In turn, processing time depends on the initial values of the weights and biases, the learning rate, as well as the network topology [56, 16]. In the following sections, the latest main methods of pre-training are discussed.

4.1 Min and max

Zhang et al. [56] and Fernández-Redondo and Hernández-Espinosa [13] discussed several initial weight initialization techniques. In the Min and Max, weight is random number in the special range.

- Kim and Ra [30] introduced a minimum bound for initialization of weights by using the following formula:

\[(L / N \text{ input})^{1/2} = |W| ; L \text{ is the learning step. Fernández-Redondo and Hernández-Espinosa [13] proposed an upper bound of 0.1 plus a lower bound.}\]

- Ho-Sub et al. [25] classified the input values in three groups, whereby the weights of the most important input were initialized at [0.5, 1], the weights with the least important input were initialized at [0, 0.5], and the rest were initialized at [0, 1]. The first two groups contain about one quarter of the total number of input values and the third group contains about one half.

- Keeni et al. [28] introduced the idea for initializing the weight range within the domain of [−0.77; 0.77] with fixed variance of 0.2. The experiment achieved the best mean performance for multi-layer perceptrons with only one hidden layer.

- Currently, the Min and Max technique using a standard BPN has initial random weights in the range of [-0.05, 0.05] [13]. Also, Fernández-Redondo and Hernández-Espinosa [13] presented a BPN with the weights initialized using the technique of Ho-Sub et al. in two different ranges of [0, 0.5] and [0.5, 1] [25]. Moreover, Fernández-Redondo and Hernández-Espinosa [13] presented a BPN with the weights initialized using the technique of Kim and Ra [30] with an upper bound of 0.1 plus a lower bound.

Experimental results by Fernández-Redondo and Hernández-Espinosa presented the BPN with weights initialized within the range of [−0.05, 0.05], implemented on the HEART and BUPA datasets using the UCI repository [13]. Generalization performance was measured by computing the correct percentage against the test set. In the HEART or Heart Disease datasets, the mean value of the correct percentage in the test set is 81.2±1.1 with 220±80 epochs. In the BUPA or Liver Disorders dataset, the mean value of the correct percentage in the test set is 59.4±1.0 with 1300±400 epochs. Also, experimental results by Fernández-Redondo and Hernández-Espinosa also presented the BPN with weights initialized using the technique of Ho-Sub et al., in the two different ranges of [0, 0.5] and [0.5, 1], implemented on the HEART and BUPA datasets [13, 25]. In the HEART or Heart Disease datasets, the mean value of the correct percentage in the test set
is $80.9\pm0.8$ with $500\pm200$ epochs. In the BUPA or Liver Disorders, the mean value of the correct percentage in the test set is $61.3\pm1.3$ with $2100\pm500$ epochs. Fernández-Redondo and Hernández-Espinosa also used a BPN with weights initialized using the technique of Kim, et al. with an upper bound 0.1, implemented on the HEART and BUPA datasets[13, 30]. In the HEART or Heart Disease datasets, the mean value of the correct percentage in the test set was $81.7\pm1.1$ with $220\pm80$ epochs. In BUPA or Liver Disorders, the mean value of the correct percentage against the test set was $60.0\pm0.9$ with $1300\pm300$ epochs. These results are comparable to the results obtained using the pre-training method of Statistically Controlled Activation Weight Initialization (SCAWI) [12]. Nonetheless, the disadvantage of the Min and Max method is in the need to initialize random numbers which creates critical problem during training.

### 4.2 SCAWI

Drago and Ridella [12] introduced a method called Statistically Controlled Activation Weight Initialization (SCAWI). They used the meaning of paralyzed neuron percentage (PNP) and conceptualized on testing the number of times a neuron lies in a completed situation with acceptable error. The formula below was designed for initializing the weights $W$, whereby $V$ is the mean square value of the input and $r_{ij}$ is a random number uniformly distributed in the range $[-1, +1]$.

$$W_{ij}^{\text{input}} = 1.3 / (1+N \text{ input}.V^2)^{1/2} \cdot r_{ij}$$

Fernandez-Redondo and Hernandez-Espinosa [14] and Funahash [15] improved this method seeking better result:

$$W_{ij}^{\text{hidden}} = 1.3 / (1+0.3 \cdot N \text{ hidden})^{1/2} \cdot r_{ij}$$

Fernández-Redondo and Hernández-Espinosa [13] implemented the BPN using the SCAWI method on the HEART and BUPA datasets from the UCI repository. In the HEART or Heart Disease dataset, the mean value of the correct percentage in the test set was $80.8.2\pm1.0$ with $200\pm100$ epochs. In the BUPA or Liver Disorders, the mean value of the correct percentage in the test set is $60.9\pm0.9$ with $1600\pm400$ epochs. Comparing the results with the pre-training method of the Min and Max using the same dataset, the speed of the BPN using the Min and the Max pre-training method was better than the speed of the BPN by using SCAWI. However, similar to the Min and Max method, the SCAWI also uses random numbers to feed the formula; and hence have the same training problem as the Min and Max.

### 4.3 DPT

Li et al., [38] described the Delta Pre-Training (DPT) technique as a different weights initialization technique. The core of the DPT is in using the Delta rule instead of using random numbers, after this phase, the SMFFNN model training process is carried out to complete network training. First, the multi-layer model is partitioned at the hidden layer into two simple perceptrons models. The weights are initialized with zero values by using the Delta rule in two perceptrons.

Experimental results by Fernández-Redondo and Hernández-Espinosa [13] presented the BPN with weights initialized using the DPT technique, implemented on the HEART and BUPA datasets from the UCI repository. In the HEART or the Heart Disease datasets, the mean value of the correct percentage in the test set is $78.4\pm1.9$ with $1\pm0$ epochs. In the BUPA or Liver
Disorders, the mean value of the correct percentage in the test set is 58.6±1.4 with 1±0 epochs. The disadvantage of this technique is the initialization of the zero value that is not based on computing real weights.

4.4 Shimodaira technique
Shimodaira [49] introduced one pre-training technique based on geometrical considerations. In the algorithm shown below, $W_i$ is the weights from $n$ units in the lower layer to the unit number $i$.

- Calculate $b=|f^{-1}(I-e) - f^{-1}(-e)|$; where $f$ is the transfer function and $e$ is a parameter which was appropriated the value of 0.1 in the reference.
- Calculate $W^* = b/(2)^{1/2} \times k \times n$; where $k$ is the parameter.
- Calculate $\gamma \leq a_i \leq \gamma$; where $\gamma$ is parameter and generate a uniform random number $a_i$ in this range.
- Calculate $W_i = W^* \times (a_i + I)$; repeat steps 2 and 3 $n$ times to calculate the $n$ weights.
- The threshold $w_0$ is zero.

Experimental results by Fernández-Redondo and Hernández-Espinosa [13] presented the BPN with weights initialized using the Shimodaira [49] technique, implemented on the HEART and BUPA datasets from the UCI repository. In HEART or Heart Disease datasets, the mean value of the correct percentage in the test set is 81.7±1.1 with 290±130 epochs. In the BUPA or Liver Disorders, the mean value of the correct percentage in the test set is 60.8±1.4 with 1400±300 epochs. The disadvantage of this technique is the initialization zero value that is not based on computing real weights.

4.5 Multi-layer auto-encoder networks
Multi-layer encoders are feed-forward neural networks for training odd number of hidden layers [50, 23, 9]. Generally, a type of back propagation networks is used in auto encoders. The feed forward neural network trains to minimize the mean squared error between the input and the output by using the sigmoid function. A high-dimensional matrix may be reduced into a low-dimensional matrix through the extraction of node values in the middle hidden layer. In addition, the auto-encoder/auto-Associative neural networks are neural networks that are trained to recall their inputs. When the neural network uses the linear neuron and activation functions, the auto-encoder processes are similar to the PCA [36]. The Sigmoid activation function allows the auto-encoder network to train a nonlinear mapping between the high-dimensional and low-dimensional data matrix. After the pre-training phase, the model is “unfolded” to encode and decode the initial weights.

Back-propagation neural network advances the global fine-tuning phase through the auto-encoder to fine-tune the weights for optimization. In a high number of multi-layer auto-encoders connections, back propagation network approach is considerably slow. The Restricted Boltzmann Machines (RBM) is able to train efficiently using an unsupervised learning procedure. RBM is a two-layer network with visual and hidden nodes, and suitable for an ensemble of binary vectors (i.e. images). The single layer of hidden units in RBM are not connected to each other and have undirected, symmetrical connections to a layer of visible units.
All visible and hidden unit configurations have energy. The auto-encoder network is unrolled and fine-tuned by a supervised model of the BPN in a standard way. Since the 1980s, BPN employing deep auto-encoders with chosen initial weights were close enough to a good result.

Experiment by Lanckriet et al. [36] on SPECTF Heart dataset reported the performance accuracy of 75.1% with 14 epochs and an error rate of 0.25. Nonetheless, its accuracy is reportedly lower than the SBPN. The performance accuracy of the SBPN using the Min and Max pre-training method is 79.0% with 25 epochs and 0.21 error rate using the same dataset. Meanwhile, the performance accuracy of the BPN using PCA on the SPECT Heart is 73.3% with 14 epochs and 0.27 error rate. From this result, the speed is better than the SBPN results but not its accuracy percentage. The performance accuracy of the SBPN by using the Min and Max pre-training method on the SPECTF Heart is 87.0% with 25 epochs and 0.13 error rate. Martin et al., [42] implemented a Deep encoder network pre-trained with the RBMs (DNet) to achieve the goal of a large-margin K-NN (K=5) classification on USPS dataset from http://www.cs.toronto.edu/roweis/data/usps all.mat/ and the MINIST dataset from http://yann.lecun.com/exdb/mnist/. Experimental results on the USPS and MNIST handwritten digits show that the DNet-KNN is powerful in both classifications. The error rate of the classification on USPS dataset is 1.138 for DNet-KNN (K=5) where the error rate of the K-NN (K=5) is 4.83. The error rate of classification on the MINIST dataset is 0.94 for DNet-KNN (K=5) where the error rate of K-NN (K=5) is 3.05. The main disadvantages of this method are using random number for weight initialization and due to the high number of multi-layer auto-encoders connections in the training process, resulting in slow performance.

5. WEIGHT LINEAR ANALYSIS (WLA)

The proposed technique of MFFNN is incorporated as a combination of data pre-processing and a new pre-training technique for simplifying the training process. Weights Linear Analysis (WLA) is a technique for reducing the training process with an application of new, high accuracy and speed model of Multi-layer Feed Forward Neural Network (MFFNN). The key ideas of the WLA model are: (1) to recognize high deviations in input values matrix from the global mean; similar to the PCA; and (2) to use the meaning of the vector torque formula during pre-training for the MFFNN, instead of the energy formula in the multi-layer auto-encoders of RBM [50, 9]. Figure 3 shows the diagram of the proposed WLA and MFFNN technique.
Figure 3 also illustrates the main issue in WLA, which is the use of input values for both preprocessing and pre-training. Weight linear analysis as a preprocessing technique simply uses the input values matrix without applying any random numbers. Also, WLA is the pre-processing technique to accelerate the classification in the MFFNN models. In WLA, high deviations from the global mean in the input values matrix may cause more scores similar to PCA, and the meaning of vectors torque is the core idea of using WLA in the MFFNN model.

Weight Linear Analysis consists of four components:

*The first component of the WLA is Input values.* The input values matrix can be of any numeric type, range and measurement unit. The rows of the matrix are instances of the dataset and the columns of the matrix are the attributes of the dataset.

*The second component of the WLA is the Vertical evaluation and normalizing of the input values.* In this phase, a data preprocessing technique, which is, the Min and Max technique is used for normalizing the input values and generating the vector values. The Input matrix of values consists of every single value with individual measurement unit type and range. The main concession of the proposed method is that no missing values exist and every value is acceptable. For this propose, other preprocessing techniques such as data cleaning are useful. Hence, the special value and worth of each data is evaluated as the ratio to the entire dataset of matrix.

*The third component of the WLA is pre-training.* In improving the pre-training performance, real weights are initialized. Pre-training of WLA contains of horizontal evaluation by computing the standard normalized values for every instance. Horizontal evaluation applies normalized input values from the data preprocessing component and computes the standard normalized values.
The first distribution of the standard normalized values is computed. $\mu$ is the mean of values for the vectors of each row, and $\sigma$ is standard deviation of values for the vectors of each row. $Z_{nm}$ is a standard normalized value and is computed based on the formula below.

$$Z_{nm} = (C_{nm} - \mu_m) / \sigma_m$$

$Z_{nm}$ shows the distance of $C_{nm}$ to mean its rows. Global mean is the center of vectors torques. The arms of value vectors are computed based on definition of deviation and distribution of standard normalization and show the weights of attributes.

**Computing the weight of each input value by using arms of values vectors:** The distances of the vector values from the global mean of matrix are computed. These distances show the arms or weights of attributes. The weights are the arms in the vectors torques and the distance from the global mean of the input values matrix. This definition of the weights is based on the statistical and mathematical definition of deviation and distribution of standard normalization and vector torque. $W_m$ is equivalent to $|Z_{11}|+|Z_{22}|+|Z_{..}|+|Z_{nm}|) / n$. $|Z_{nm}|$ is the absolute of the normal value $Z_{nm}$. Hence, the weight selection is not based on randomization. The weights may have thresholds but must be managed in the hidden layer of the MFFNN using the following equation.

$$W_m = (|Z_{11}|+|Z_{22}|+|Z_{..}|+|Z_{nm}|) / n$$

The matrix of the input values consists of every value with every measurement unit type and range. Through WLA in the MFFNN, each normalized value vector creates one vector torque ratio to the global mean of the matrix. The vectors are evaluated together and will finally reach the equilibrium. Figure 4 shows an example of the action given four vector torques, whereby all vectors create their own vector torques.

![Figure 4: The action of four vector torques](image)

$C_a$, $C_b$, $C_c$, $C_d$ are the vectors of values while $D_a$, $D_b$, $D_c$, $D_d$ are the arms of the vector torques of values. These arms are based on their distances from the global mean point of matrix. The vector torque of $C_a$ is $C_a \times D_a$, the vector torque of $C_b$ is $C_b \times D_b$, the vector torque of $C_c$ is $C_c \times D_c$ and the vector torque of $C_d$ is $C_d \times D_d$. In this research, the physical and mathematical meaning of vector torque is used, for the classification of instances.

For example, if vector $C_k$ is highly correlated to $C_a$, through the addition vector torques, will create a noise and the correlation is then 1. When the correlation is 1, this means that the two attributes are indeed one attribute and duplication exists in the input values matrix. $C_k$ which is highly correlated to correlation with $C_a$, moves closer to the location of vector $C_a$ with the
addition of vector torques, where the global mean moves to the new location of the torque. Hence after equivalence, the global mean take place as one constant at the special point of the axis of vector torques. The weight distributed between $C_a$ and $C_k$ is shown in Figure 5.

![Figure 5: The action of four vector torques after the addition of a new vector](image)

The fourth component of WLA is the dimension reduction of data. In this phase, there are normalized values and potential real weights. The weights show deviations of the input values matrix from the global mean, similar to the PCA. WLA with the potential of having real weights recognizes the high dimensional data matrix for effective data preprocessing. The unnecessary attributes have weak weights and can be pruned. The suitable submatrix of necessary attributes can be selected based on their potential weights. So WLA can map high dimension matrix to lower dimension matrix. The strong weight causes high variance. If the dimensions of the input vectors are large, the components of the vectors are highly correlated (redundant). WLA can solve this problem in two ways. First, after equivalence, the global mean takes place as one constant at the special point of axis of the vector torques and the weights are distributed between the vectors. Then, WLA solves redundancy through dimension reduction. This phase of the WLA can be performed in the hidden layer during pruning.

The output of data pre-processing is normalized values which use to generate real weights in pre-training section. Normalized values and real weights will use by supervised MFFNN model to classify the dataset.

### 5.1 Algorithm of Weight Linear Analysis

The algorithm of WLA is shown in Figure 6.

```
WLA (D; L, W)
Input: Database D, database of input values;
Output: Matrix L, Normalized database of D; W, real weights;

Begin

//1- Data pre-processing component
//Computing vertical evaluation: In this phase, the input values are translated.
   Let row number: n;
   Let column number: m;
   Let copy of database D in Matrix nxm of L;
   For all columns of Matrix L m do
```
For all rows of Matrix \( L \) \( n \) do 
\[
L(n,m) = L(n,m) / \text{Average}(\text{column } m);
\]
Matrix \( \text{LTemp} \) = copy of Matrix \( L \); 
}

//2- Pre-training component
//Computing horizontal evaluation: In second phase of procedure, the weight of each input value is computed.
//Computing standard normalized values in row:
Let \( \mu_n \) = Mean of values vectors in row;
Let \( \sigma_n \) = Standard deviation of values vectors in row;
For all columns of Matrix \( L \) \( m \) do
  For all rows of Matrix \( L \) \( n \) do
    \( \text{LTemp}(n,m) = (\text{LTemp}(n,m) - \mu_n) / \sigma_n; \)
  // Computing arms of values vectors (Weights):
  For all columns of Matrix \( L \) \( m \) do
    \( W_m = (\text{Average of Absolute} (\text{LTemp( column } m)) ; \)
Return Matrix \( L \), real weights \( W \)

Figure 6: The algorithm of WLA

5.2 MFFNN with Weight Linear Analysis
The outputs of WLA technique are normalized input values and real weights. MFFNN will process based on the algebraic consequence of the vectors torques. The vectors torque \( T_{nm} = C_{nm} \times W_m \) are the basis of the physical meaning of torque. Each torque \( T \) shows a real worth and threshold of each value between whole values in the matrix. The algebraic consequence of vectors torques \( \hat{S}_n \) is equivalent to \( T_{n1} + T_{n2} + \ldots + T_{nm} \). In each row of input values matrix, \( \hat{S}_n \) is computed. The output will be classified based on \( \hat{S}_i \). Recall that BPN uses a sigmoid activation function to transform the actual output between the domain \([0, 1]\) and to compute the error by using the derivative of the logistic function \( O_j (1 - O_j) \) and \( (T_j - O_j) \) in order to compare the computed output \( O_j \) with the true output \( T_j \). The True output forms the basis of the class labels in a given training dataset.

In the case of the WLA, real weights are computed and the MFFNN the desired output is computed by using a binary step function as the activation function instead of the sigmoid function. Also there is no need to compute the error and the derivative of the logistic function \( O_j (1 - O_j) \) and \( (T_j - O_j) \) for the purpose of comparing the computed output with the true output. The output \( \hat{S}_n \) are sorted and two stacks are created based on the true output with the class label of 0 and 1. The Binary step function is applied to both stacks, serving as a threshold, and, generates the desired output of 0 or 1. In essence, understanding this model is very easy; MFFNN applies WLA similar to the simple neural network shown in Figure 7. Figure 7 illustrates the feature of the model.
The number of layers, nodes, weights, and thresholds in MFFNN using WLA pre-processing is logically clear without the presence of any random elements. MFFNN will classify input data by using the output of WLA, whereby there is one input layer with several input nodes, one hidden layer, and one output layer with one node. The hidden layer contains the weighted function $\sum_i W_{ij} I_i$ and $W_{ij}$ are real weights from the pre-training. The hidden layer is necessary for pruning or considering management options for weights optimization. The hidden layer consists of two nodes. One node has the condition of weights being smaller than or equal average of weights, another node has the condition of weights being bigger than average of weights.

In pruning, the input values with weak and low weights can be omitted because they have a weak effect on the desired output. In management strategies, the input values with weights bigger than the average weights perform effectively on the desired output, therefore they can be optimized. The output node in the output layer contains $\sum_j W_{\mu j} I_j$ where $W_{\mu j}=1$ for computing the desired output. Here, the BPN is run for only one epoch during the training process without the need to compute bias and error in the hidden layer. In evaluating the test set and predicting the class label, the model use the weights and thresholds from training phase, and the class label of each instance can be predicted by the binary step function as illustrated in Figure 2.2. The algorithm of the MFFNN with WLA is shown in Figure 8.

**MFFNN (D; CL)**

Input: Database D, database of input values;
Output: CL, Classified Database D based on credit-rate;
Begin
{
//Propagating the inputs forward:
Call WLA (D; L, W);
For all sample X in the training samples
{
For output layer unit j
\[ I_j = \sum_i W_{ij}.I_i; \]
\[ \text{Let Sort } (I_j) = \text{Sorted list of all } I_j; \]
// Create Stack0 and Stack1 for definition of thresholds
Let Stack0: Stack of \( I_j \) with condition of class label 0;
Let Stack1: Stack of \( I_j \) with condition of class label 1;
// Back propagate the threshold and Binary step function
For unit j in the output layer
Apply Binary-step-function (Input: Threshold, Output: CL);
}
Return CL
}

Figure 8: MFFNN algorithm with WLA

To illustrate the semantic and logic of the MFFNN model with WLA, the problem of the Exclusive-OR (XOR) is considered. Table 1 illustrates the XOR problem as follows:
\[ X_1 \oplus X_2 = \text{XOR}(X) \]

<table>
<thead>
<tr>
<th>Attribute 1</th>
<th>Attribute 2</th>
<th>Output of XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Based on Table 1, there are two logical attributes and four instances. Attribute characteristic is binary 0, 1. Usually XOR is used by multi-layers artificial neural networks. The analysis of the XOR problem is illustrated in Table 2, together with its features and class label.

<table>
<thead>
<tr>
<th>Instance 1</th>
<th>Attribute 1</th>
<th>Attribute 2</th>
<th>Class label of XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance 2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Instance 3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Instance 4</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Learning of MFFNN using WLA takes one epoch without computing the sigmoid function, training cycle, mean square error, and updating weights. Real weights are obtained through WLA and the MFFNN model applies them to compute thresholds and the binary step function for generating the desired output as well as predicting the class label for XOR. The real weights of attribute 1 and attribute 2 are the same (0.5) by using WLA because the correlation between the values of attribute 1 and attribute 2 is zero (\( \rho = 0 \)). In this case, the output of the model shows the...
error is 0 and the outputs are the same class labels. Figure 9 illustrates the result of the MFFNN by using the WLA implementation.

![MFFNN Diagram](image)

The problem of XOR is implemented using the WLA and the MFFNN. The experimental results are compared with the results from the Standard BPN, Improved BPN and the PCA+BPN. Table 3 shows the comparison in terms of speed and accuracy of the XOR problem.

**Table 3: Classification of the XOR problem**

<table>
<thead>
<tr>
<th>Classification model</th>
<th>Number of epoch</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFNN with WLA</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Improved BPN</td>
<td>3167</td>
<td>0.0001</td>
</tr>
<tr>
<td>Standard BPN</td>
<td>7678</td>
<td>0.0001</td>
</tr>
<tr>
<td>PCA+BPN</td>
<td>200</td>
<td>0.0002</td>
</tr>
</tbody>
</table>

The result of MFFNN with WLA is better than others.

The Experiment on the SPECTF Heart and SPECT Heart datasets is shown the accuracy of SBPN using Min and Max pre-training method with considering initial random weights in range of [-0.77, 0.77] on SPECTF Heart is 79% with 25 epochs and 0.21 error rate and on SPECTF Heart is 87% with 25 epochs and 0.13 error rate. Table 4 shows comparison of classification accuracy on SPECTF Heart dataset.

**Table 4: Comparison of classification accuracy on SPECTF Heart dataset**

<table>
<thead>
<tr>
<th>The classification methods</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFNN by using WLA</td>
<td>94.0%</td>
</tr>
<tr>
<td>MFFNN by using WLA with dimension reduction</td>
<td>85%</td>
</tr>
<tr>
<td>SBPN</td>
<td>79.0%</td>
</tr>
<tr>
<td>BPN by using PCA</td>
<td>75.1%</td>
</tr>
<tr>
<td>K-NN (K=1)</td>
<td>72.1%</td>
</tr>
<tr>
<td>CLIP3</td>
<td>77.0%</td>
</tr>
<tr>
<td>CLIP4</td>
<td>77.0%</td>
</tr>
</tbody>
</table>
The accuracy of CLIP3 and CLIP4 is 77.0% while the accuracy of 1-NN is 72.1% \cite{31, 49}. The accuracy of the BPN using PCA is 75.1%, SBPN is 79.0%, and the MFFNN using WLA yielded the highest accuracy of 94.0. The accuracy of the MFFNN using WLA with dimension reduction is 85%. The performance accuracy of MFFNN using WLA is better than the other methods because it uses real weights and thresholds, and it does not work on random initializations like the other methods. Table 5 shows the comparison of the training speed across the MFFNN+WLA method on the SPECTF HEART dataset.

Table 5: Comparison of the training speed on the SPECTF Heart dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Epoch</th>
<th>CPU time (second)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFNN using WLA</td>
<td>1</td>
<td>0.061</td>
<td>0.11</td>
</tr>
<tr>
<td>SBPN</td>
<td>25</td>
<td>4.98</td>
<td>0.21</td>
</tr>
<tr>
<td>BPN by using PCA</td>
<td>14</td>
<td>1.6</td>
<td>0.25</td>
</tr>
</tbody>
</table>

The training of the standard BPN on the SPECTF Heart dataset required 25 epochs with 4.98 second of CPU time and error rate of 0.21. The BPN using PCA on the SPECTF Heart took 14 epochs, 1.6 second of CPU time and an error rate of 0.21. The MFFNN using WLA only took one epoch with 0.061 second and an error rate of 0.11, which shows a significant improvement in speed as compared against other techniques.

The MFFNN using WLA pre-processing achieved higher performance speed because the training process only required one epoch. The time complexity of the WLA is $O(pn)$. The time complexity of the WLA technique depends on the number of attributes $p$ and the number of instances $n$.

Table 6 shows the comparison of the classification accuracy on the SPECT Heart dataset.

Table 6: Comparison of classification accuracy on SPECT Heart dataset

<table>
<thead>
<tr>
<th>The classification methods</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFNN by using WLA</td>
<td>92.0%</td>
</tr>
<tr>
<td>MFFNN by using WLA With dimension reduction</td>
<td>87%</td>
</tr>
<tr>
<td>SBPN</td>
<td>87.0%</td>
</tr>
<tr>
<td>BPN by using PCA</td>
<td>73.3%</td>
</tr>
<tr>
<td>K-NN (K=1)</td>
<td>80.2%</td>
</tr>
<tr>
<td>CLIP3</td>
<td>84.0%</td>
</tr>
<tr>
<td>CLIP4</td>
<td>90.4%</td>
</tr>
</tbody>
</table>

The accuracy of the CLIP3 is 84.0% while the CLIP4 is 90.4%. The accuracy of the 1-NN is 80.2% \cite{31, 49}. The accuracy of the BPN using PCA is 73.3%, the SBPN is 87.0%, while that of the MFFNN using WLA has the highest accuracy of all, which is 92.0%. The accuracy of the MFFNN using the WLA with dimension reduction is 87%.

The Comparison of the training speed across the techniques on the SPECT Heart dataset is shown in Table 7 as follows:
<table>
<thead>
<tr>
<th>Method</th>
<th>Epoch</th>
<th>CPU time (second)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFNN by using WLA</td>
<td>1</td>
<td>0.036</td>
<td>0.13</td>
</tr>
<tr>
<td>SBPN</td>
<td>25</td>
<td>2.92</td>
<td>0.15</td>
</tr>
<tr>
<td>BPN by using PCA</td>
<td>14</td>
<td>1.08</td>
<td>0.27</td>
</tr>
</tbody>
</table>

The SBPN method converged in 25 epochs at 2.92 second CPU times and an error rate of 0.15 on the SPECT Heart training dataset. The BPN using PCA required 14 epochs with 1.08 second CPU time and an error rate of 0.27. The MFFNN using WLA converged in only one epoch with 0.036 second CPU time and an error rate of 0.13, which is the highest processing speed as compared to other techniques.

6. CONCLUSION

This study focuses on data pre-processing and pre-training techniques of the Multi-layer Feed Forward Neural Network models. WLA is the newest technique and is a combination of data preprocessing and pre-training that prepares normalized input values and real weights for the MFFNN models. MFFNN models generate real thresholds by using outputs of WLA. MFFNN model classifies the dataset based on these real thresholds. Data pre-processing and pre-training techniques cause an increase in the classification accuracy and a reduction in the training time of the MFFNN model.

REFERENCES


THE AUTHORS

Roya Asadi received the Bachelor degree in Computer Software engineering from Electronics and Computer Engineering Faculty, Shahid Beheshti University and Computer Faculty of Data Processing Iran Co. (IBM), Tehran, Iran. She obtained Master of Computer science in Database System from UPM university of Malaysia. She is a research student of Ph.D. of Computer Science in Artificial Intelligence (Neural Network) in University of Malaya. Her professional working experience includes 12 years of service as Senior Planning Expert 1. Her interests are in Data Mining, Neural Network modeling, Intelligent Systems, Medical Informatics.

Assoc. Prof. Datin. Dr. Sameem Abdul Kareem: received the B.Sc. in Mathematics (Hons) from University of Malaya, in 1986, the M.Sc. in Computing from the University of Wales, Cardiff, UK, in 1992, and the Ph.D. in computer science from University of Malaya, Malaysia, in 2002. She is currently an Associate Professor of the Department of Artificial Intelligence, Faculty of Computer Science and Information Technology, University Of Malaya. Her research interests include Medical Informatics, Information Retrieval, Data Mining and Intelligent Techniques. She has published over 80 journal and conference papers.
Reviewer #2: The paper is interesting and well written but it might be better classified as Review Article or case/application. The paper has 2 self-citation (reference 49 and reference 50) which are not validated by this Editorial Manager system. Therefore, this work reported could be redundant.

Dear Administrator of NCA Journal

Sincerely thanks for your advices and attention;

1- This article is reviewing of PREPROCESSING AND PRETRAINING TECHNIQUES OF MULTI-LAYER FEED FORWARD NEURAL NETWORK MODELS.

We edited and checked the structure of this paper based on the structure of a review paper as you mentioned.

2- The self-citations are removed.

So thanks,

Best regards;

Roya Asadi, PhD. Candidate,
Department: Artificial Intelligence - Neural Network.
Faculty of Computer Science and Information Technology
University of Malaya, 50603 Kuala Lumpur, Malaysia
E-mail: asadiroya@gmail.com