http://doi.org/10.35784/iapgos.3366 *received: 12.12.2022 | revised: 5.03.2023 | accepted: 13.03.2023 | available online: 31.03.2023*

AN ENHANCED DIFFERENTIAL EVOLUTION ALGORITHM WITH ADAPTIVE WEIGHT BOUNDS FOR EFFICIENT TRAINING OF NEURAL NETWORKS

Saithip Limtrakul, Jeerayut Wetweerapong
Khon Kaen University, Faculty of Science, Department of Mathematics, Khon Kaen, Thailand

Abstract. *Artificial neural networks are essential intelligent tools for various learning tasks. Training them is challenging due to the nature of the data set, many training weights, and their dependency, which gives rise to a complicated high-dimensional error function for minimization. Thus, global optimization methods have become an alternative approach. Many variants of differential evolution (DE) have been applied as training methods to approximate the weights of a neural network. However, empirical studies show that they suffer from generally fixed weight bounds. In this research, we propose an enhanced differential evolution algorithm with adaptive weight bound adjustment (DEAW) for the efficient training of neural networks. The DEAW algorithm uses small initial weight bounds and adaptive adjustment in the mutation process. It gradually extends the bounds when a component of a mutant vector reaches its limits. We also experiment with using several scales of an activation function with the DEAW algorithm. Then, we apply the proposed method with its suitable setting to solve function approximation problems. DEAW can achieve satisfactory results compared to exact solutions.*

Keywords: neural network, differential evolution, training neural network, function approximation

ULEPSZONY ALGORYTM EWOLUCJI RÓŻNICOWEJ Z ADAPTACYJNYMI GRANICAMI WAG DLA EFEKTYWNEGO SZKOLENIA SIECI NEURONOWYCH

Streszczenie. *Sztuczne sieci neuronowe są niezbędnymi inteligentnymi narzędziami do realizacji różnych zadań uczenia się. Ich szkolenie stanowi wyzwanie ze względu na charakter zbioru danych, wiele wag treningowych i ich zależności, co powoduje powstanie skomplikowanej, wielowymiarowej funkcji błędu do minimalizacji. Dlatego alternatywnym podejściem stały się metody optymalizacji globalnej. Wiele wariantów ewolucji różnicowej (DE) zostało zastosowanych jako metody treningowe do aproksymacji wag sieci neuronowej. Jednak badania empiryczne pokazują, że cierpią one z powodu ogólnie ustalonych granic wag. W tym badaniu proponujemy ulepszony algorytm ewolucji różnicowej z adaptacyjnym dopasowaniem granic wag (DEAW) dla efektywnego szkolenia sieci neuronowych. Algorytm DEAW wykorzystuje małe początkowe granice wag i adaptacyjne dostosowanie w procesie mutacji. Stopniowo rozszerza on granice, gdy składowa wektora mutacji osiąga swoje granice. Eksperymentujemy również z wykorzystaniem kilku skal funkcji aktywacji z algorytmem DEAW. Następnie, stosujemy proponowaną metodę z jej odpowiednim ustawieniem do rozwiązywania problemów aproksymacji funkcji. DEAW może osiągnąć zadowalające rezultaty w porównaniu z rozwiązaniami dokładnymi.*

Słowa kluczowe: sieć neuronowa, ewolucja różnicowa, trening sieci neuronowej, aproksymacja funkcji

Introduction

The artificial neural network (ANN) is one of the most popular machine learning techniques and has continuously gained attention in many research fields. The ANN learning procedure imitates the behavior of the nervous system and has multiple characteristics, such as the number of connecting nodes and layers, the initial values of weights, and the types of a transfer function (or activation function). The procedure forwardly transforms input data to output as a mapping function using weights and the transfer function. The network training finds the optimal weights that minimize the error between the actual target and network output on the sample input data. Therefore, the training algorithm which applies an optimization method for minimization strongly affects the performance of ANN.

In the beginning, Back-propagation (BP) [24] was the training method for the multilayer perceptron algorithm. The BP algorithm is a gradient descent optimizer technique that minimizes functions by iteratively moving in the direction of the steepest descent as defined by the negative of the gradient. The BP algorithm has powerful local search capability but has a few drawbacks. Since the objective function of ANN is a multi-modal function [9] that has several local minima, the algorithm can get stuck into a local minimum. Moreover, it possibly has a slow convergence speed depending on the learning parameter values [4, 22–23, 28].

Global search methods are an alternative approach widely studied and applied as training methods for neural networks to overcome the limitation of the traditional gradient-based algorithm. Over the last decade, many researchers have developed global techniques based on natural inspiration that are population-based and capable of parallel calculations. They do not require the derivative computation and can provide the global minimum for the multi-modal objective functions of ANN.

The Differential Evolution (DE) algorithm is one of those popular global approaches. The method is an efficient population-based technique developed by Rainer Storn and Kenneth Price [26] for optimization problems over continuous domains. Many DE variants were proposed based on various modifications of initialization, mutation, crossover, selection, and hybridization schemes [20]. Mezura et al. [16] empirically compared some DE variants to solve global optimization problems. The results showed that the rand/1/bin mutation scheme performs well for multimodal functions. The DE is an attractive optimization tool compared with other evolutionary algorithms due to ease of implementation, robustness, and a few control parameters. The beneficial usages of the DE algorithm for training networks are its ability to reach a global minimum of the objective function, quick convergence, and a small number of parameters of the network settings. Moreover, it is possible to adjust some essential parameters during the process execution. Also, the DE can train the network with non-differentiable transfer functions where the gradient information is unavailable.

In this study, we concern with the learning algorithm of the feed-forward neural network using the DE. We study the capabilities of the network parameters, i.e., weight bounds and scaling of activation function, which affects the performances of DE for finding the solution in terms of search space and output space. The automatic adjustment of the weight bounds in the mutation processing stepand various scale parameters in the activation function are studied.

We organize the rest of this paper into the following sections. Section 1 provides the literature reviews concerning training ANN using the DE algorithm and the influences of weight bounds. The DEAW algorithm, an implementation of DE with the weight bound adjustment as a training approach, is described in section 2. Then, we use the DEAW algorithm to solve the function approximation problems. The preliminary and comparison experiments, results, and discussion are presented in Section 3. Finally, section 4 is a conclusion and future work.

1. Literature reviews

1.1. Training neural networks using differential evolution

In 2008, Gao and Liu [10] proposed a modified DE by introducing the mutation operator coupled with Back-propagation (BP) training algorithm to solve the exclusive-OR (XOR) problem and function approximation. The method can reduce the training time, improve the training accuracy, and perform more steadily than the classical BP. Subudhi and Jena [27] solved a non-linear system identification problem using an improved DE with Lavenberg Marquardt (LM) algorithm to train the ANN. The method gives better results in terms of convergence speed and identification errors. Garro et. al. [11] applied the classical DE to ANN for solving the non-linear pattern classification problems. The algorithm can search for optimal synaptic weights by employing a few parameters to tune and perform better than the PSO algorithm on the UCI machine learning benchmark repository [31]. Morse and Stanley [19] also applied the self-adaptive DE algorithm (SDE) to train the ANN. The parameters F and CR of the DE are adaptively adjusted. The SDE gives better results when compared to training ANN by GA algorithm and standard DE on four sets of the UCI dataset. Si et al. [25] presented a DEGL approach to combine global and local mutation steps to create a candidate population vector using the mutation probability adaptation to balance the searchability. They tested the method on four benchmark functions and seven classification problems. The proposed method shows superior performance in less complex networks with a small number of generations compared to the classical DE algorithm. Comparative performances of a neural network trained with variant DE also have been studied. Baioletti et al. [1] tested various combinations of self-adaptive methods (i.e., JDE, JADE, ShaDE, L-ShaDE, MAB-ShaDE, SAMDE), mutation, and crossover operators on some well-known classification problems. The experiments showed that the results from the neural network obtained with DE training are better and more robust than those from back-propagation.

Nevertheless, Piotrowski [21] studied the performance of variant DE such as DEGL, DE-SG, DEGL-Epitr, JADE, SADE, SspDE, Trig-DEGL, Trig-DEGL-Epitr, and the empirical experiments showed that various DE algorithms fell into stagnation during the training of ANN. Moreover, the methods performed poorer than the classical Levenberg-Marquardt algorithm. The stagnation of DE occurs because of the undesired effect of the lack of difference vectors of small magnitude; in other words, the population stops proceeding toward the optimum even though the population diversity remains high. The paper showed that the appropriate range initialization and bound limitation strongly influence speed and efficiency for exploring the solution and are critical to convergence. In that research, they demonstrated that the smaller range and the broader bound gave the best results. By using the set-up, initial the weights with [-1,1] and limit the bound to [-1000,1000], during the beginning stage, the individual disperses rapidly in the decision space, and the magnitude of the difference vector increase, irrespective of DE variant used; however, the improvement slows down during the later stage of the run and leads the algorithm to fall into stagnation. However, several works on training ANN using the DE method did not investigate how to calibrate the weight bound that significantly affects the process of finding the solutions.

1.2. The influence of weight bound

Since the solutions of network training are the synaptic weights in various ranges, DE suffers from the problem of the bound and size of search space [21]. The population-based method needs to search the solution space. It is not only the bounds of the solution that should limit during the initialization but also the diversity of initial weights should vary to cover all over the solution space. Bartlett [2-3] proposed that, by computation theory, the number of training data growing linearly with the number of adjustable parameters leads to good generalization performance. The performance also depends on the bound of the weights rather than the number of the weight connections. The proof also supports heuristics that attempt to keep the weight vectors small during training. Ismailov [13] referred to many previous works that the weights are not necessary a large magnitude. Then the author proposed the sets of weights consisting of a finite number of directions. The result shows that the weights vectors can do well approximation but not always possible. Hahm and Hong [12] mentioned that the weights in the neural networks vary; hence it is too difficult to be applied in applications. In their work, they showed that a network with suitable fixed weights and a sufficient number of neurons can approximate any continuous function over a compact interval.

In 2014, H. Migdady [17] proved the delta rule for hidden and input weights to help understand and explain the behavior of the weight vectors in the feed-forward neural networks with BP. The proof shows that the weight vectors in the neural network are upper bounded, i.e., do not approach infinity. Jesus et al. [14] studied the effects of the initial configuration of weights on the training and activation function of neural networks. They found that the efficiency of the learning process depends on fine-tuning the weight vectors of the networks. In the appropriate case, the networks always converge to the neighborhood of their initial configuration. Cong et al. [6] also analyzed the effect of the initial weights vector in the learning process by BP. The result shows that the process tends to get stuck into the local minima if the weights start from the area which does not contain a global minimum.

Therefore, the weight vectors of the neural networks should be initialized in a small magnitude and near the global minimum. They should be limited by a suitable fixed bound; otherwise, fine-tuned within a small range.

Another essential factor that involves the weights of ANN is the activation function. Many functions have been proposed for different tasks, for example, binary function, radial basis function, and ReLu function, but the most widely used function is the sigmoid function. The sigmoid function consists of 3 parameters that affect the range of the outputs as illustrated in the figure 1.

As shown in figure 1, if input x_1 and x_2 are slightly different values, a small parameter c_1 gives a smoother different output than a large one which provides rapid change. Increasing the parameter c_1 in the ANN training by DE application for approximation and classification, the magnitude of the difference of output vectors will be increased, thus, leading the individual population to explore the search space rapidly. Therefore, we investigate the various values of parameter c_1, called the activation scale.

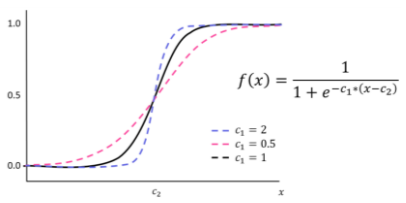


Fig. 1. Basic sigmoid function. The 3 parameters are x, c_1 and c_2. x is an input of the function, c_1 affects the slope of the function and c_2 is a translation of axes

2. Background and methodology

2.1. Feed-forward neural network

In this work, the Multilayer perceptron (MLP) is a general structure of our study. MLP is a feed-forward neural network consisting of multiple layers: an input layer, one or more hidden layers, and an output layer. Each layer has nodes where each node is fully weight interconnected to all nodes in the subsequent layer as shown in figure 2.

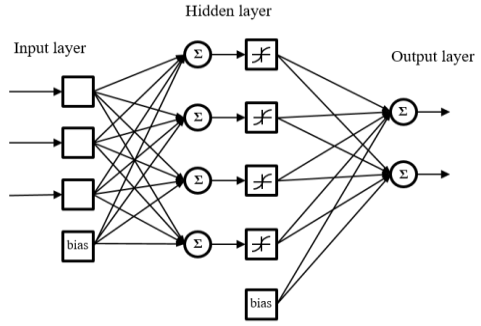


Fig. 2. Feed-forward neural network model

An MLP transforms the inputs into the outputs through the non-linear function; called the transfer function or activation function, expressed by:

$$x_h = f_1([x_i],[w_{i,h}]) = f_1\left(\sum_{i=1}^{d}(x_i * w_{i,h})\right) \quad (1)$$

$$x_o = f_2([x_h],[w_{h,o}]) = f_2\left(\sum_{h=1}^{nh}(x_h * w_{h,o})\right) \quad (2)$$

where f_1, f_2 are the activation functions of the hidden nodes and the output nodes, respectively. x_h and x_o are the output of each hidden node h and output node o. $w_{i,h}$ is a connecting weight between the input i and the hidden node h while $w_{h,o}$ connects between the hidden node h and the output o. In general, the activation function applied to the hidden node is the sigmoid function and that for the output is usually a linear function. The functions are expressed as:

Sigmoid function

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \quad (3)$$

Linear function

$$\text{linear}(x) = x \quad (4)$$

where x is a value or a vector.

Since the ANN learns the mathematical model by training the network, the training rule is essential for a learning algorithm. The updating rules determine how connection weights are changed. The learning algorithm finds the optimal weights by minimizing the objective function defined by the distance error between the target value and the output from the network. Let the weights vector be $W = [W_1, W_2]$, where $W_1 = [w_{i,h}]$ and $W_2 = [w_{h,o}]$. The ANN find the optimal solutions by minimizing the sum of error:

$$E(W) = \sum_{d\in D}\left(t_d - F(x_d, W)\right)^2 \quad (5)$$

where D is a set of training data and d is the index of each training data. The target and output values are t_d and $F(x_d, W)$, respectively. The output $F(x_d, W)$ can be calculated by

$$F(x_d, W) = f_2(f_1(x_d, W_1), W_2) \quad (6)$$

2.2. Differential evolution algorithm

Differential evolution algorithm is a population-based optimization method derived from the Genetic algorithm (GA). The method consists of three operations, i.e., mutation, crossover, and selection for generating new candidate solutions. Unlike GA, the weight called 'Scaling Factor: F' multiplies the difference between the randomized two population vectors to add to the third one and obtain the mutant vector. The DE crossover operator uses a crossover probability; named 'Crossover Rate: CR' in the range of [0,1] to combine the mutant vector with the target vector to create a trial vector. In selection, the trial candidate vector replaces the target vector if it gives a superior solution; otherwise, the original candidate remains unchanged. Many variant DE methods are different in the step of mutation and crossover [1, 7, 8]. In this work, we apply the basic mutation scheme called DE/rand/1/bin to avoid the effect of other improvements to the DEAW performance. The following steps describe the DE algorithm.

- Step I: (Initialization) Randomly initial population vectors $x_i\ ;\ i = 1, \dots, NP$
- Step II: (Mutation) Generate a mutant vector v_i by adding weighted difference between two population vectors to a third one. The equation can be expressed as:
$$v_i = x_{r1} + F * (x_{r2} - x_{r3}) \quad (7)$$
where x_{r1}, x_{r2}, x_{r3} are 3 randomized distinct vectors and different from x_i, and F is a scaling factor in the range [0.5,1].
- Step III: (Crossover) Create a trial vector u_i from x_i and v_i by
$$IC = rand(1, ndim) \quad (8)$$
$$u_{i,j} = \begin{cases} v_{i,j} & if \quad \text{rand}(0,1) \le CR \text{ or } j = IC \\ x_{i,j} & if \quad \text{rand}(0,1) > CR \end{cases} \quad (9)$$
where (i, j) is an element j^{th} of vector i and CR in [0,1] is a crossover rate.
- Step IV: (Selection) Choose the vector for the next generation by comparing the objective function values of x_i and u_i:
$$x_i^{new} = \begin{cases} x_i & if \quad f(x_i) < f(u_i) \\ u_i & if \quad otherwise \end{cases} \quad (10)$$
Repeat Step II–IV until reaching the termination conditions.

2.3. Training neural network by using differential evolution

Applying DE to train the network, the vectors of connecting weights in ANN are used as the individual population vector of the DE algorithm, i.e., a set of population vectors $W = \{W_i\}$, where $i = 1, \dots, NP$ and NP is the number of populations. Let $X = \{x_d\}, T = \{t_d\}$ where $d = 1, \dots, ndata$ are the set of training and target data vectors, respectively. $F(\cdot)$ is the composite of activation functions in the network and $F(x_d, W_i)$ is an output corresponding to x_d and W_i. In each iteration, the current weight vectors of ANN are obtained from the DE approach by reducing the error between the target T and the output obtained from $F(X, W_i)$ in the selection process.

As described in section II, the range of the weights affects the approximated result of the activation function. Thus, the performance of the DE algorithm used for ANN training significantly depends on the initialization range and the bounds. The ANN training algorithm should initialize the weights in a small bound and extend it later.

In our study, the DEAW algorithm initializes both ranges and bounds of the weights by small values at the beginning. Then, the algorithm adjusts the bound corresponding to the mutant vectors and the number of the current iterations in the process. If a component value of a mutant vector is greater than the upper bound or less than the lower bound, then the bound is extended. As a result, the search space gradually broads, and the magnitude of the difference vector also slightly increases. In practice, it is difficult to determine where the range is satisfied unless several empirical tests.

The DEAW algorithm applies an adaptive process to the mutation step for automatic bound adjustment. The amount of extending rate depends on the current iteration. This strategy increases the rapid exploration ability of the algorithm at the beginning stage and decreases it at the later stage. The equation for adjustment is expressed as:

$$ext\_rate = \frac{1}{iter} \quad (11)$$

where $iter$ is the number of current iteration. Algorithm in the figure 3 shows our enhanced DE with adaptive weights bound for training ANN. Also, table 1 represents all notations.

(Initialization)
$W \leftarrow W_i, i = 1, \ldots, NP$
initial $Fbest, W_{best}$
Do until reaching termination conditions:
while $i \leq NP$ **do**
  (Mutation)
  $r1, r2, r3 \leftarrow random \in [1, NP]$
  $r1 \neq r2 \neq r3 \neq i$
  $V_i \leftarrow W_{r1} + F \times (W_{r2} + W_{r3})$
  (Bound adjustment)
  loop1:
  if $j \leq ndim$ **then**
    if $V_{i,j} > UB$ **then**
      $UB \leftarrow UB + ext\_rate$
      $V_{i,j} \leftarrow$ random $\in [LB, UB]$
    end if
    if $V_{i,j} < LB$ **then**
      $LB \leftarrow LB - ext\_rate$
      $V_{i,j} \leftarrow$ random $\in [LB, UB]$
    end if
  end if
  $j \leftarrow j + 1$
  goto *loop1*.
  (Crossover)
  $IC \leftarrow random \in [1, ndim]$
  loop2:
  if $j \leq ndim$ **then**
    if *rand(0,1)*$\leq$ CR || $j = IC$ **then**
      $U_{i,j} \leftarrow V_{i,j}$
    else
      $U_{i,j} \leftarrow W_{i,j}$
    end if
  end if
  $j \leftarrow j + 1$
  goto *loop2*.
  (Selection)
  $Err_{U_i} \leftarrow 0$
  loop3:
  if $d \leq ndata$ **then**
    $Err_{U_i} \leftarrow Err_{U_i} + (t_d - F(x_d, U_i))^2$
  end if
  $d \leftarrow d + 1$
  goto *loop3*.

  $Err_{U_i} \leftarrow \frac{1}{ndata} \times Err_{U_i}$
  if $Err_{U_i} < Err_{W_i}$ **then**
    $W_i \leftarrow U_i$
    $Err_{W_i} \leftarrow Err_{U_i}$
  end if
  if $Err_{U_i} < Fbest$ **then**
    $Fbest \leftarrow Err_{U_i}$
    $W_{best} \leftarrow U_i$
  end if
  $i \leftarrow i + 1$
end while

Fig. 3. DEAW algorithm

Table 1. Notation

Notation	Detail
NP	population size
$ndata$	number of training data
$ndim$	dimension of the weight vector where $ndim = [(indim + nbias) \times nhidden] + [(nhidden + nbias) \times outdim]$
$indim$	dimension of the inputs
$outdim$	dimension of the outputs
$nhidden$	number of hidden nodes
$nbias$	number of bias nodes
W	a set of weights vectors; $W = \{W_i ; i = 1, \ldots, NP\}$
X	a set of input data; $X = \{x_d ; d = 1, \ldots, ndata\}$
T	a set of targets corresponding to X; $T = \{t_d ; d = 1, \ldots, ndata\}$
UB, LB	upper and lower bounds, respectively
F	a scaling factor
CR	a crossover rate
ext_rate	extending rate
$F(x, W)$	the output of ANN for (x, W)
$Fbest$	minimum error obtained by current best weight vector
W_{best}	the current optimal weight vector

3. Experimental results and discussion

To verify the efficiency of our proposed method and control parameter values, we apply the DEAW algorithm to train the feed-forward neural network for solving function approximation problems. The details of the experiments are described in the following subsections.

3.1. Experimental setup

In this work, we apply the DE as a learning algorithm to feed-forward neural networks with one hidden layer. The number of nodes in the input layer depends on each problem and includes one bias. The number of hidden nodes is varied appropriately depending on the complexity of the problem and also includes one bias. The algorithm applies the sigmoid function to the hidden nodes while the linear function to the output nodes.

Each experimental study performs 30 runs. We use a trimmed mean method for calculating the average values of the results to eliminate the outliers. In our tests, we trim 15% of the results, i.e., five runs for both best and worst cases. The maximum iteration depends on the training dataset and the complexity of the problem. The configuration of DE is simple. The algorithm uses a basic scheme called DE/rand/1/bin and general control parameters, $F = 0.5$, $CR = 0.9$. The population size is 50 for all experiments. More individual settings for each study are described in the following subsections.

3.2. Function approximation problems

To observe the performance of our proposed enhancement, we first apply the DEAW algorithm to the 1D function approximation. Two testing functions are described as:

$$f_1(x) = \sin(2\pi x) \times (4\pi x) \quad (12)$$

$$f_2(x) = \sin(2\pi x) \times \sin(3\pi x) \times \sin(5\pi x) \quad (13)$$

The data of each function is uniformly generated on $x \in [0,1]$, 100 data for training and 199 testing data including the training data. As illustrated in figure 4, both functions are multimodal with 6 and 10 local maxima/minima, respectively.

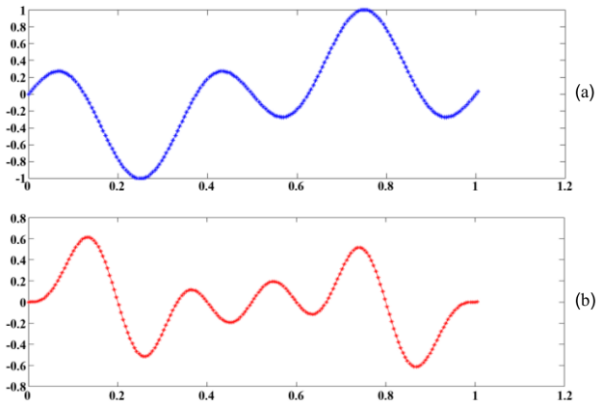


Fig. 4. Illustrated graphs of the functions: (a) f_1 and (b) f_2 on [0,1]

This study concerns the effects on the ANN performance caused by the weight bound and the activation scale. Both the initial and limit of weight bound are essential for the efficiency of the performance of ANN. Thus, we conducted two experiments. First, we observed the effects of different combination of fixed limit bounds with varied activation scales. Fig. 5(a) shows the values of the Sigmoid function generated on the various interval ranges. The graphs generated on the limit bounds have the similar shape but different output ranges. A small bound gives small output ranges, whereas a wide bound gives a larger one. However, as seen in the figure, the outputs are almost not changed where $x < -10$ and $x > 10$. Thus, we only investigated the limit bound settings, $[-1,1], [-10,10], [-20,20]$ and $[-30,30]$ for our experiments.

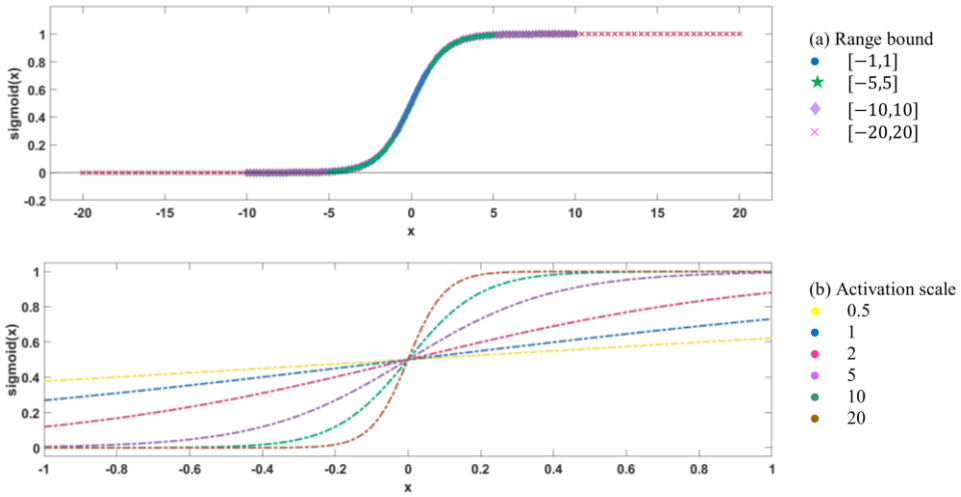


Fig. 5. (a) The outputs from Sigmoid function as an activation function with scale =1. (b) Each range of bound also gives the same shapes but different output ranges. The outputs obtained by various activation scales express similar shapes but different slopes

Fig. 5(b) shows the output obtained by varying the scale of activation function. The outputs give similar shapes but different slopes. Large scales show extreme slope while small scales show slight slope. In this study, we then use 1, 10, 20, and 30 as observation scales. Note that a general ANN usually uses [-1,1] for the bound and 1 for the activation scale. Second, we initialized a small range of weight bound, i.e $[-2,2]$ or $[-10,10]$ and then the range is gradually extended during the mutation process. For the ANN structure, we applied a structure 2-10-1 for this experiment. The algorithm runs for 50000 iterations for function f_1 and 150000 for f_2, or the mean error (VTR) reaches 0.5e-3. Tables 2 and 3 illustrate the results of the training and testing. The best scale for each bound represented in bold.

3.2.1. The effects of varying ranges of weight bound and activation scales

As shown in tables 2 and 3, a standard configuration, limit weight bound as $[-1,1]$ and activation scale as 1, gives unsatisfied results, i.e. the errors are greater than the VTR. In table 2, the results of the f_1 approximation obtained by limiting the bound to $[-1,1]$ and $[-10,10]$ with scale 1 show unsuccessful converge to the VTR. Nevertheless, increasing of activation scale can improve the accuracy but has to trade-off with high standard deviation values, i.e., unreliable results. For the limit bounds $[-20,20]$ and $[-30,30]$, the results show that scale 1 gives acceptable errors. Then, increasing the activation scale increases a small error rate.

Similarly, the results obtained from the second function, f_2, are presented in table 3. The bound limit to $[-1,1]$ with the scale 1 of the activation function also gives an unsatisfied error, but increasing the scale shows better performance. For the ranges $[-10,10]$, $[-20,20]$, and $[-30,30]$, increasing the activation scale to 10 gives the best result, then the efficiency decreases slightly. These experiments express the satisfying results using a suitable fixed limit range of bound with a regular activation scale 1. But, if the bound is improper, increasing the scale to the large one improves the accuracy of the results by trading off a higher standard deviation (SD).

Table 2. Training and testing result of f_1 with different fixed limit bounds and various activation scales

	Train	Test	Train	Test	Train	Test	Train	Test
Bound	[-1,1]							
Act. Scale	1		10		20		30	
Mean	2.1379E-01	2.1509E-01	8.7346E-02	8.6543E-02	3.4065E-02	3.4196E-02	**4.9487E-03**	**4.8475E-03**
SD	1.0520E-04	1.0724E-04	2.2854E-03	2.3133E-03	4.4727E-03	4.5838E-03	3.4863E-03	3.6537E-03
%SD	0.0492	0.0499	2.6165	2.6730	13.1299	13.4044	70.4473	75.3722
S/F	0/30		0/30		0/30		0/30	
Bound	[-10,10]							
Act. Scale	1		10		20		30	
Mean	4.7658E-02	4.7415E-02	1.3538E-04	1.3512E-04	1.3054E-04	1.3103E-04	**1.2804E-04**	**1.2847E-04**
SD	2.6946E-02	2.6802E-02	8.4570E-05	8.2669E-05	6.8728E-05	6.7929E-05	5.8379E-05	5.8734E-05
%SD	56.5402	56.5268	62.4674	61.1826	52.6479	51.8407	45.5928	45.7201
S/F	0/30		25/5		25/5		25/5	
Bound	[-20,20]							
Act. Scale	1		10		20		30	
Mean	**1.2673E-04**	**1.2489E-04**	1.6685E-04	1.6779E-04	1.4308E-04	1.4447E-04	2.3952E-04	2.4720E-04
SD	3.7550E-05	3.495E-05	5.7644E-05	5.8589E-05	5.2012E-05	5.2410E-05	1.7268E-04	1.8891E-04
%SD	29.6301	27.988.	34.5478	34.9179	36.3523	36.2783	72.0959	76.4200
S/F	30/0		30/0		26/4		22/8	
Bound	[-30,30]							
Act. Scale	1		10		20		30	
Mean	**1.2911E-04**	**1.2750E-04**	1.5695E-04	1.5805E-04	1.4834E-04	1.4948E-04	1.3211E-04	1.2271E-04
SD	4.2856E-05	4.3647E-05	6.567E-05	6.5685E-05	7.2725E-05	7.4669E-05	5.639E-05	4.1808E-05
%SD	33.1950	34.2331	41.5588	41.5588	49.0256	49.9537	42.4949	34.0699
S/F	27/3		30/0		29/1		27/3	

Table 3. Training and testing results of f_2 with different fixed limit bounds and various activation scales

	Train	Test	Train	Test	Train	Test	Train	Test
Bound	[-1,1]							
Act. Scale	1		10		20		30	
Mean	8.5423E-02	8.6.63E-02	5.8407E-02	5.8800E-02	2.4274E-02	2.4286E-02	**9.0211E-03**	**9.0736E-03**
SD	4.1820E-05	4.4285E-05	3.0044E-04	3.3011E-02	5.7640E-04	6.1962E-04	1.3748E-04	1.5214E-04
%SD	0.0490	0.0515	0.5144	0.5614	2.3745	2.5513	1.5239	1.6767
S/F	0/30		0/30		0/30		0/30	
Bound	[-10,10]							
Act. Scale	1		10		20		30	
Mean	3.0107E-02	3.0446E-02	**1.8191E-04**	**1.7828E-04**	2.8801E-04	2.8779E-04	2.4614E-04	2.4672E-04
SD	2.6566E-03	2.7404E-03	1.0940E-04	1.0852E-04	2.3135E-04	2.3233E-04	1.5117E-04	1.5087E-04
%SD	8.617	9.0009	60.1378	60.8799	80.3295	80.7274	61.4153	61.1510
S/F	0/30		27/3		23/7		24/6	
Bound	[-20,20]							
Act. Scale	1		10		20		30	
Mean	3.9421E-03	3.9733E-03	**3.4243E-04**	**3.4050E-04**	4.3055E-04	4.3112E-04	4.0058E-04	3.9842E-04
SD	3.7996E-03	3.8353E-03	3.3185E-04	3.3096E-04	4.3222E-04	4.3536E-04	5.1463E-04	5.1377E-04
%SD	96.3855	96.5287	96.9113	97.2004	100.3894	100.9824	128.4717	128.9524
S/F	11/19		22/8		20/30		22/8	
Bound	[-30,30]							
Act. Scale	1		10		20		30	
Mean	1.4220E-03	1.4105E-03	**7.9663E-04**	**7.9655E-04**	1.3334E-03	1.3472E-03	1.1072E-03	1.1647E-03
SD	9.0660E-04	9.2014E-04	9.7159E-04	9.8190E-04	1.7056E-03	1.7215E-03	1.3743E-03	1.4455E-03
%SD	63.7547	65.2372	121.9622	123.2691	127.9126	127.7852	124.1269	124.1072
S/F	10/20		18/12		16/14		16/14	

3.2.2. The efficiency of extendable weight bound

A. The performance on 1D-function approximation

As shown in experiment 1, proper limit bounds are different for f_1 and f_2. Our DEAW algorithm with extendable weight bound modification is applied to demonstrate the efficiency of adjustable bound. Since the most appropriate range for both functions is $[-20,20]$, the initial margin of the bound is set to $[-2,2]$ and $[-10,10]$ and also combined with various activation scales. The structure of neural networks and termination conditions are the same as in experiment 1. The results of the method applied to f_1 and f_2 are shown in tables 4 and 5, respectively. The best scale for each bound represented in bold.

Table 4 shows that by setting the initial bound to [-2,2] with the scale 1, the error cannot reach the VTR while success in the setting [-10,10]. Both initial bounds give the absolute values of weight components in the interval [10,20], which is similar to the observation in experimental result 1. Larger activation scales combined with the DEAW algorithm provide better results for a small initial range while giving a few differences for a broader initialization, but the standard deviation becomes higher if the scale exceeds. The final bound is also the same as obtained by Experiment 1, i.e. $[-20,20]$. In addition, the number of success cases using the DEAW algorithm increases, and the error rates decrease.

From table 5, the results also give similar performance. The narrow initial bound provides better accuracy when the activation scales increase; however, an exceedingly large scale raises the high standard deviation. The final absolute values of the weight components lie in [20,30], the same as experiment 1.

Moreover, the DEAW algorithm also performs better in the number of success cases and the error rates compared to previous experiments. Hence, the results of both experiments show that increasing the scale of activation function can improve the efficiency of unsuitable fixed limit bound; however, it trades off higher standard deviation where the scale exceeds. In addition, gradually and adaptively adjusting the bound of the weights leads to better performance and more efficient results than the fixed limit bound.

B. The performance on 2D-function approximation

To verify the performance on 2D-function approximation problem, we applied the DEAW method to approximate a complex sine function expressed as:

$$f_3(x_1, x_2) = 0.5(sin(5x_1) + sin(5x_2)) \quad (14)$$

where the data is uniformly generated on $x_1, x_2 \in [-1,1]$, 100 data for training and 199 testing data including the training data. For the ANN structure, we used a structure 2-10-1 with activation scale 1 for this experiment and set the iterations to 10000.

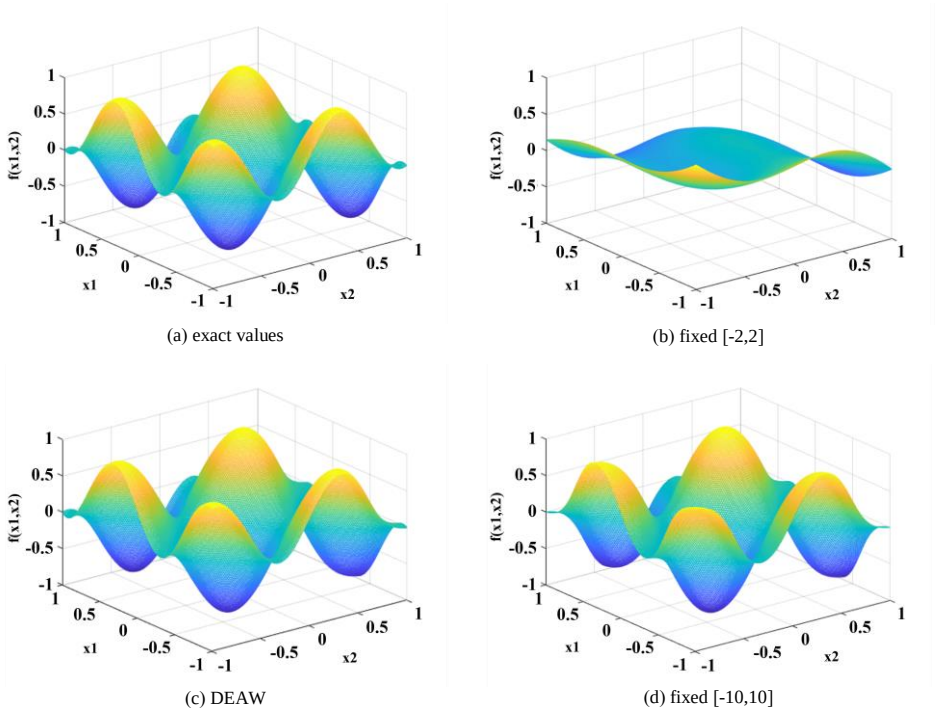
The tests are divided into 2 steps. First, we applied the range [-2,2] as a fixed bound and adjustable bound with the activation scale 1. Then, we observed the final range of adaptive bound and used it as an appropriated fixed bound. The results of 2D-function approximation are presented in the table 6 and also illustrated in figure 6.

Table 4. Training and testing results of f_1 by using small initial bound with extendable capability and various scales of activation function

	Train	Test	Train	Test	Train	Test	Train	Test
Ini. Bound	[-2,2]							
Act. Scale	1		10		20		30	
Mean	7.1854E-02	7.1033E-02	1.0772E-04	1.0749E-04	**9.9939E-05**	**1.0027E-04**	**9.9954E-05**	**1.0008E-04**
SD	7.0817E-03	7.0683E-03	2.3063E-05	2.3675E-05	4.7903E-08	5.2718E-07	2.1096E-08	7.9188E-07
%SD	9.8556	9.9508	21.4108	22..244	0.0479	0.5257	0.0211	0.7912
Best range	-		[0,10] , (10,20]		**[0,10] , (10,20]**		**[0,10] , (10,20]**	
S/F	0/30		30/0		30/0		30/0	
Ini. Bound	[-10,10]							
Act. Scale	1		10		20		30	
Mean	**9.9921E-05**	**9.7.36E-05**	1.3479E-04	1.3562E-04	1.6926E-04	1.7040E-04	1.0465E-04	1.1441E-04
SD	9.1541E-08	2.6754E-06	5.1366E-05	5.2019E-05	1.3662E-04	1.3827E-04	1.4954E-05	4.2545E-05
%SD	0.0916	2.7572	38.1071	38.3574	80.7149	81.1425	14.2903	37.1848
Best range	**(10,20] , (20,30]**		(10,20] , (20,30]		(10,20] , (20,30]		(10,20] , (20,30]	
S/F	30/0		30/0		24/6		27/3	

Table 5. Training and testing results of f_2 by using small initial bound with extendable capability and various scales of activation function

	Train	Test	Train	Test	Train	Test	Train	Test
Ini. Bound	[-2,2]							
Act. Scale	1		10		20		30	
Mean	2.7737E-02	2.7386E-02	2.3435E-04	2.3510E-04	**1.0715E-04**	**1.0778E-04**	1.6960E-04	1.7072E-04
SD	2.8388E-03	2.7984E-03	1.8049E-04	1.8150E-04	1.9345E-05	1.9620E-05	1.8635E-04	1.8790E-04
%SD	10.2349	10.2184	77.0168	77.2002	18.0548	18.2037	109.8780	110.0575
Best range	-		(10,20] , (20,30]		**(10,20] , (20,30]**		(10,20] , (20,30]	
S/F	0/30		24/6		27/3		23/7	
Ini. Bound	[-10,10]							
Act. Scale	1		10		20		30	
Mean	1.7726E-03	1.7853E-03	**3.0116E-04**	**2.9888E-04**	3.6320E-04	3.6522E-04	3.3510E-04	3.3838E-04
SD	2.1968E-03	2.2189E-03	1.0539E-04	1.0301E-04	4.7594E-04	4.7803E-04	3.1736E-04	3.2125E-04
%SD	123.9316	124.2888	34.9953	34.4642	131.0420	130.8873	94.7069	94.9380
Best range	(20,30] , (30,50]		**(30,50] , (20,30]**		(30,50] , (20,30]		(30,50] , (20,30]	
S/F	12/18		24/6		21/9		19/11	



(a) exact values (b) fixed [-2,2]

(c) DEAW (d) fixed [-10,10]

Fig. 6. Illustrated Graphs of Complex-sine function f_3: (a) exact values, (b) approximated values with fixed bound [-2,2], (c) approximated values obtained by the DEAW algorithm and (d) approximated valued with fixed suitable bound [-10,10]

Table 6. Training and testing results of Complex sine function using fixed and extendable bound

Complex sine function f_3						
Ini. Bound	[-2,2]		[-2,2]		[-10,10]	
capability	extendable		fixed		fixed	
Act. Scale	1		1		1	
No. hidden	10		10		10	
	Train	Test	Train	Test	Train	Test
Mean MSE	5.520 E-04	7.870 E-04	1.521 E-01	1.554 E-01	1.706 E-05	4.403 E-04
SD	0.00058	0.00053	0.00368	0.00352	0.00001	0.00008
Avg. range	UB	LB	-		-	
	9.12	-9.53				

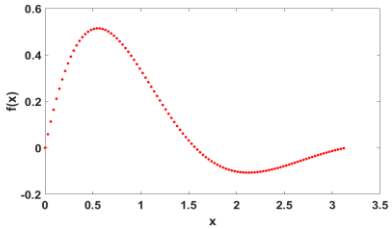
As shown in table 6, the ANN with fixed bound [-2,2] cannot properly approximate the function, as illustrated in figure 6(b). By using the DEAW method, on the other hands, the error rate reaches to 5.520e-4 which can represent the function as shown in figure 6(c). The average of final bound obtained from the DEAW algorithm is in the range of [-10,10]. Applying this appropriated range gives the best performance.

3.2.3. The comparison experiments

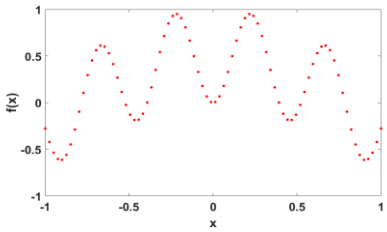
A. Comparison on 1D-function approximation problems

Hereafter, we utilized the small initial bound, $[-2,2]$, and the activation scale, 10, to demonstrate the efficiency of the DEAW algorithm by applying it to other three 1-D functions used in [15, 18, 29-30]. Figure 7 (a), (b), (c) express the details of each function and table 7, 8, 9 show the comparison results with those work.

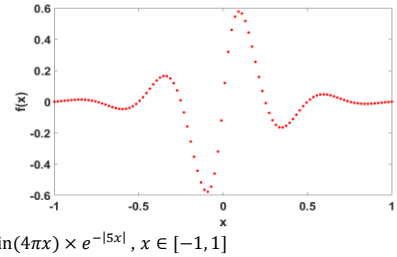
The results of the DEAW algorithm compared with the performance of PSO-based algorithms [18, 29] are shown in table 7. The compared methods are tested on f_4 using the weight bound [0,1] and varying numbers of hidden nodes. A small number of hidden nodes using PSO-BP gave the worst results. Their works presented the best MSE around 4e-5 using 7 hidden nodes, whereas the DEAW reached the VTR at 1e-5 using only 3 hidden-layer nodes with an adaptive weight-bound strategy.



(a) $f_4(x) = \sin(2x) \times e^{-x}$, $x \in [0, \pi]$
Training: identical sampling interval of 0.03 from $[0, \pi]$
Testing: identical sampling interval of 0.1 from $[0{,}02, \pi]$



(b) $f_5(x) = \sin(8x) \times \sin(6x)$, $x \in [-1,1]$
Training: uniformly sampled 80 points
Testing: uniformly sampled 160 points including testing points



(c) $f_6(x) = \sin(4\pi x) \times e^{-|5x|}$, $x \in [-1, 1]$
Training: uniformly sampled 100 points
Testing: uniformly sampled 200 points including testing points

Fig. 7. Illustrated graphs of the 3 functions and details of each dataset

For PSOGSA comparison, the results show that the DEAW method gives lower MSE. The difference results of both 3 and 7 hidden nodes provides the two-tailed P values less than 0.0001. By conventional criteria, this is considered to be statistically significant with 95% confidence interval.

Next, we applied the DEAW algorithm to compare with the performance in [15] using NN-based on the PSO learning method. The PSO initializes the weights in the fixed range $[-1,1]$. After running to 2000 iterations, see table. 8, RMSE of the PSO method reaches 0.29 ± 0.02, while the DEAW method also gives a better result at 0.23 ± 0.03. The t-test gives the two-tailed P value less than 0.0001. This difference is considered to be statistically significant with 95% confidence interval. However, both experiments are not satisfied the appropriate approximation, as illustrated in figure 8(a). The DEAW method then continues by running 30000 iterations with ten hidden nodes. As a result, the DEAW can reach the RMSE 1e-2, as illustrated in figure 8(b).

In the last experiment, we compared our method with the work based on the Radial basis function neural network (RBFN) and wavelet neural network (WNN) [30]. In that work, the researcher measured the accuracy by using the normalized square root mean square (N_e) calculated by the following equation:

$$N_e = \frac{1}{\sigma_y}\sqrt{\sum_{i=1}^{n}(y_i - f_i)^2} \tag{15}$$

where y and f are the target and output values of the network, respectively. n is the total number of testing samples, and σ_y is the standard deviation of the target values. A small N_e indicates high accuracy. This experiment, the VTR setting is 1e-6.

As shown in the figure 9 the results indicate that our proposed method gives better approximated values than RBFN (upper-green) and expresses appreciated results as WNN (upper-magenta).

Table 7. The comparison of PSO based method and our DEAW method on f_4

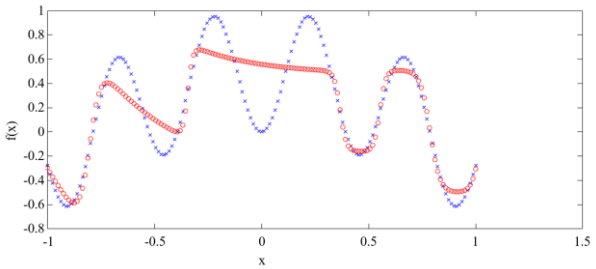
	PSO-BP [30]		PSOGSA [18]		DEAW	
Input/Output	1/1		1/1		1/1	
Bound	BP [-50,50], PSO [0,1]		[0,1]		Initial [-2,2]	
NP	200		200		30	
Max iteration	200+1500		500		2000	
Hidden nodes	3	7	3	7	3	7
Avg MSE	4.8984e-04	1.4333e-04	9.6113e-03	6.7104e-03	**9.8220e-06**	**9.9500e-06**
Med MSE	n/a	n/a	9.9619e-03	5.4945e-03	**9.8745e-06**	**9.9757e-06**
STD	n/a	n/a	2.7267e-03	5.4070e-03	**1.6755e-07**	**4.2176e-08**
Best MSE	3.2781e-04	4.2074e-05	5.5411e-.3	9.5455e-04	**9.4330e-06**	**9.8665e-06**
Worst MSE	**7.266e-04**	**2.7363e-04**	n/a	n/a	9.9574e-06	9.9957e-06

Table 8. The comparison of PSO based method and our DEAW method on f_5

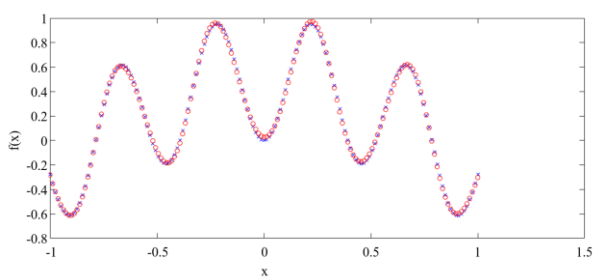
	PSO [15]	DEAW		
Input/Output	1/1	1/1		
Bound	Fixed [-1,1]	Initial [-2,2]		
NP	30	30		
Hidden nodes	8	8	8	10
iteration	2000	2000	30000	30000
RMSE	0.29±0.02	0.23±0.03	0.04±0.01	0.01±0.0001

Table 9. The comparison of RBFN and WNN methods and DEAW method on f_6

	ANN [29]		DEAW
	RBFN	WNN	
Input/Output	1/1	1/1	1/1
Hidden nodes	n/a	n/a	7
Bound	Fixed [-1,1]	Fixed [-1,1]	Initial [-2,2]
NP	-	-	30
iteration	n/a	n/a	30000
N_e	9.11658	0.207205	0.19260



(a)



(b)

Fig. 8. Illustrated graphs of different RMSEs. (a) RMES reaches 2e-1 while (b) reaches 1e-2

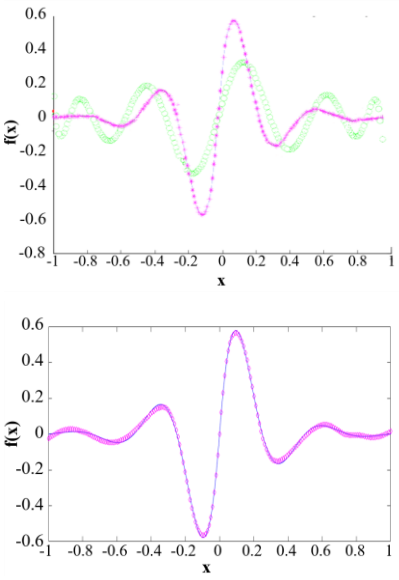


Fig. 9. Illustrated graphs of RBFN and WNN [29] (upper) and DEAW algorithm (lower). The result of the RBFN method is represented in green while the results of WNN and DEAW methods are in magenta

B. Comparison on 2D-function approximation problems

For more comparison tests, we applied the DEAW algorithm to approximate the 2D functions which are used in [5]. The functions are represented as follows.

$$f_7(x_1, x_2) = \frac{3.2(1.25+\cos(5.4x_2))}{6+6(3x_1-1)^2} \quad (16)$$

$$f_8(x_1, x_2) = \left(x_1^2 - x_2^2\right)\sin(5x_1) \quad (17)$$

where the data is uniformly generated on $x \in [-1,1]$, 100 data for training. For the ANN structure, we used a structure 2-14-1 and 2-9-1 with activation scale 1 for this experiment of f_7 and f_8, respectively. The iteration is set to 30000.

In these experiments, the DEAW approximates the 2D-function f_7, f_8 to compare with MMWNN-GA method in [5]. As seen in table 10, the error rate of our proposed method reaches 4.41E-03 which is higher than the MMWNN-GA method. However, an illustration of the approximated values using the DEAW algorithm in figure 10 shows that the DEAW can suitably represent the function of f_7.

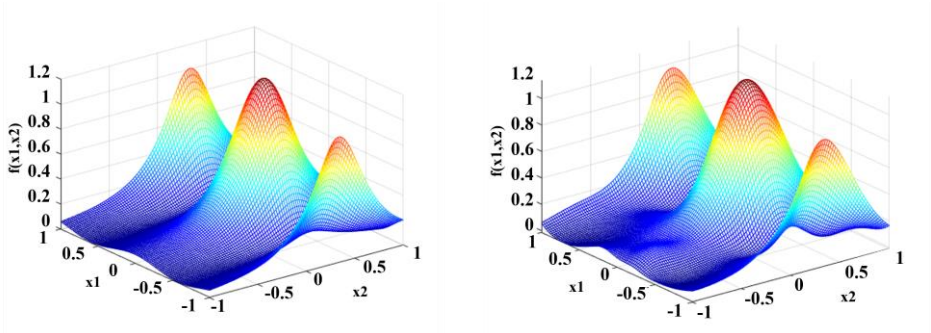
Table 10. Comparison results of 2D-function f_7 using MMWNN-GA [5] and the DEAW method

	MMWNN-GA	**DEAW**
No. of Hidden nodes	14	14
Initial range	-	[-2,2]
Bound capability	fixed	extendable
MSE	7.89E-04	4.41E-03

Next, the DEAW algorithm is demonstrated with the 2D-function f_8. The results in table 11 show that our proposed simple method approximates accurately as well as MMWNN-GA method. The approximation function of the DEAW algorithm displays in figure 11.

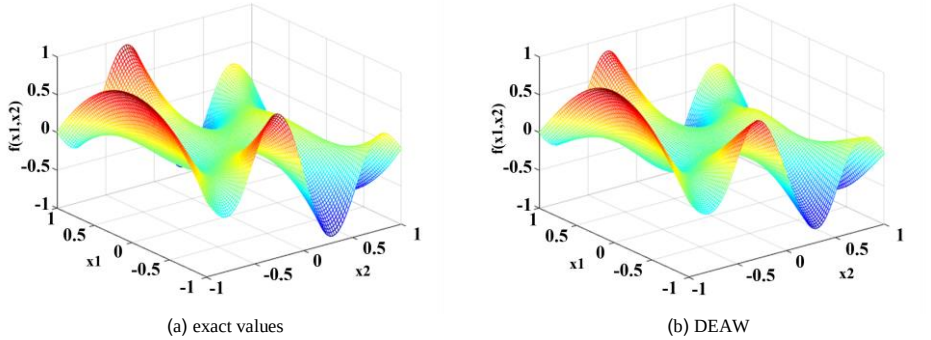
Table 11. Comparison results of 2D-function f_8 using MMWNN-GA [28] and the DEAW method

	MMWNN-GA	**DEAW**
No. of Hidden nodes	9	9
Initial range	-	[-2,2]
Bound capability	fixed	extendable
MSE	4.86E-03	4.09E-03



(a) exact values (b) DEAW

Fig. 10. Graphs of 2D-function f_7 (a) exact values, (b) approximated values obtained by the DEAW algorithm



(a) exact values (b) DEAW

Fig. 11. Graphs of 2D-function f_8 (a) exact values, (b) approximated values obtained by the DEAW algorithm

4. Conclusions

This study applies an alternative approach for training the feed-forward neural networks relying on the differential evolution (DE) algorithm for global search abilities and to overcome the limitation of local search methods. Our DEAW algorithm improves the performance of the DE training by using small initial weight bounds and adaptive adjustment strategies in the mutation process. The proposed method gives more efficient results than using the fixed limit bound. Moreover, increasing the scale of the activation function improves the efficiency of unsuitable fixed limit bound. The DEAW can achieve the solution at high accuracy with simple configurations for solving function approximation problems. In addition, testing on 1D-functions, the method performs better than the compared BP and PSO-based training methods in term of accuracy and convergence. For the 2D-function problems, the proposed method gives comparative results to MMWNN-GA. Future work study could investigate applying the proposed weight adjustment strategies to other learning tasks such as classification and system identification problems.

Acknowledgement

The Development and Promotion of Science and Technology Talents Project and Department of Mathematics, Faculty of Science, Khon Kaen University, Fiscal Year 2022.

References

[1] Baioletti M., Di Bari G., Milani A., Poggioni V.: Differential Evolution for Neural Networks Optimization. Mathematics 8(1), 2020, 69 [http://doi.org/10.3390/math8010069].
[2] Bartlett P. L.: For Valid Generalization, the Size of the Weights is More Important than the Size of the Network. Proceedings of the 9th International Conference on Neural Information Processing Systems, 1996, 134–140.
[3] Bartlett P. L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Transactions on Information Theory 44, 1998, 525–536 [http://doi.org/10.1109/18.661502].
[4] Chen L.: A global optimization algorithm for neural network training. Proceedings of International Conference on Neural Networks 1993, 443–446 [http://doi.org/10.1109/IJCNN.1993.713950].
[5] Chihaoui M., Bellil W., Amar C. B.: Multi Mother Wavelet Neural Network based on Genetic Algorithm for 1D and 2D Functions Approximation. Proceedings of the International Conference on Fuzzy Computation and International Conference on Neural Computation 2010, 429–434 [http://doi.org/10.5220/0003083704290434].
[6] Cong H., Nguyen N., Huy V. N., Bùi T.: The Influence of Initial Weights on Neural Network Training. Journal of Science and Technology 95, 2013, 18–25.
[7] Das S., Suganthan P. N.: Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation 15, 2011, 4–31 [http://doi.org/10.1109/TEVC.2010.2059031].
[8] Das S., Mullick S. S., Suganthan P. N.: Recent advances in differential evolution – An updated survey. Swarm and Evolutionary Computation 17, 2016, 1–30 [http://doi.org/10.1016/j.swevo.2016.01.004].
[9] Dhar V. K., Tickoo A. K., Koul R., Dubey B. P.: Comparative performance of some popular artificial neural network algorithms on benchmark and function approximation problems. Pramana 74, 2010, 307–324 [http://doi.org/10.1007/s12043-010-0029-4].
[10] Gao Y., Liu J.: A modified differential evolution algorithm and its application in the training of BP neural network. IEEE/ASME International Conference on Advanced Intelligent Mechatronics 2008, 1373–1377.
[11] Garro B. A., Sossa H., Vázquez R. A.: Evolving Neural Networks: A Comparison between Differential Evolution and Particle Swarm Optimization. Advances in Swarm Intelligence 2011, 447–454 [http://doi.org/10.1007/978-3-642-21515-5_53].
[12] Hahm N., Hong B. I.: An approximation by neural networkswith a fixed weight. Computers and Mathematics with Applications 47, 2004, 1897–1903 [http://doi.org/10.1016/j.camwa.2003.06.008].
[13] Ismailov V. E.: Approximation by neural networks with weights varying on a finite set of directions. Journal of Mathematical Analysis and Applications 389, 2012, 72–83 [http://doi.org/10.1016/j.jmaa.2011.11.037].
[14] Jesus R. J., Antunes M. L., da Costa R. A., Dorogovtsev S. N., Mendes J. F., Aguiar R. L.: Effect of the initial configuration of weights on the training and function of artificial neural networks. Mathematics 9, 2021, 1–16 [http://doi.org/10.3390/math9182246].
[15] Mendes R., Cortez P., Rocha M., Neves J.: Particle swarms for feedforward neural network training. Proceedings of the International Joint Conference on Neural Networks – IJCNN'02 2002, 1895–1899 [http://doi.org/10.1109/IJCNN.2002.1007808].
[16] Mezura M. E., Velázquez R. J., Coello C.: A comparative study of differential evolution variants for global optimization. GECCO 2006 – Genetic and Evolutionary Computation Conference 1, 2006, 485–492 [http://doi.org/10.1145/1143997.1144086].
[17] Migdady H.: Boundness of a Neural Network Weights Using the Notion of a Limit of a Sequence. International Journal of Data Mining and Knowledge Management Process 4, 2014, 1–8 [http://doi.org/10.5121/ijdkp.2014.4301].
[18] Mirjalili S. A., Hashim S. Z. M., Sardroudi H. M. Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. Applied Mathematics and Computation 218, 2012, 11125–11137 [http://doi.org/10.1016/j.amc.2012.04.069].
[19] Morse G., Stanley K. O.: Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks. Proceedings of the Genetic and Evolutionary Computation Conference 2016, 477–484 [http://doi.org/10.1145/2908812.2908916].
[20] Mohamad F. A., Nor A. M. I., Wei H. L., Koon M. A.: Differential evolution: A recent review based on state-of-the-art works. Alexandria Engineering Journal 61(5), 2022, 3831–3872 [http://doi.org/10.1016/j.aej.2021.09.013].
[21] Piotrowski A. P.: Differential Evolution algorithms applied to Neural Network training suffer from stagnation. Applied Soft Computing 21, 2014, 382–406 [http://doi.org/10.1016/j.asoc.2014.03.039].
[22] Prechelt L.: A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. Neural Networks 9, 1996, 457–462 [http://doi.org/10.1016/0893-6080(95)00123-9].
[23] Prieto A., Prieto B., Ortigosa E. M., Ros E.: Neural networks: An overview of early research, current frameworks and new challenges. Neurocomputing 214, 2016, 242–268 [http://doi.org/10.1016/j.neucom.2016.06.014].
[24] Rumelhart D. E., Hinton G. E., Williams R. J.: Learning representations by back-propagating errors. Nature 323, 1986, 533–536 [http://doi.org/10.1038/323533a0].
[25] Si T., Hazra S., Jana N. D.: Artificial Neural Network Training Using Differential Evolutionary Algorithm for Classification. Advances in Intelligent and Soft Computing 232, 2012, 769–778 [http://doi.org/10.1007/978-3-642-27443-5-88].
[26] Storn R., Price K.: Differential Evolution A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 11, 1997, 341–359 [http://doi.org/10.1023/A:1008202821328].
[27] Subudhi B., Jena D.: An improved differential evolution trained neural network scheme for nonlinear system identification. International Journal of Automation and Computing 6, 2009, 137–144 [http://doi.org/10.1007/s11633-009-0137-0].
[28] Yang S., Ting T. O., Man K. L., Guan S. U.: Investigation of Neural Networks for Function Approximation. Procedia Computer Science 17, 2013, 586–594 [http://doi.org/10.1016/j.procs.2013.05.076].
[29] Zainuddin Z., Pauline O.: Function Approximation Using Artificial Neural Networks. International Journal of Systems Applications, Engineering and Development 1, 2007, 173–178 [http://doi.org/10.5555/1466915.1466916].
[30] Zhang J. R., Lok T. M., Lyu M. R.: A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. Applied Mathematics and Computation 185, 2007, 1026–1037 [http://doi.org/10.1016/j.amc.2006.07.025].
[31] UCI machine learning benchmark repository. the UC Irvine Machine Learning Repository, 2019 [http://archive.ics.uci.edu/ml/datasets.php].

M.Sc. Saithip Limtrakul
e-mail: saithiplim@kkumail.com

Ph.D. candidate in Applied Mathematics at Department of Mathematics, Faculty of Science, Khon Kaen University, Thailand. The author's research area focuses on artificial intelligence, neural network, optimization, image processing, simulation and visualization.

http://orcid.org/0000-0002-7207-6640

Ph.D. Jeerayut Wetweerapong
e-mail: wjeera@kku.ac.th

Assistant Professor in Department of Mathematics at the Faculty of Science, Khon Kaen University, Thailand. The author's research interests include optimization and computational intelligence.

http://orcid.org/0000-0001-5053-3989