Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A regeneratable dynamic differential evolution algorithm for neural networks with integer weights[#]

Jian BAO[†1], Yu CHEN[2], Jin-shou YU[1]

([1]*School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China*)

([2]*Institute of Software and Intelligent Technology, Hangzhou Dianzi University, Hangzhou 310018, China*)

[†]E-mail: baojian@hdu.edu.cn

Received May 5, 2010; Revision accepted July 23, 2010; Crosschecked Oct. 15, 2010

Abstract: Neural networks with integer weights are more suited for embedded systems and hardware implementations than those with real weights. However, many learning algorithms, which have been proposed for training neural networks with float weights, are inefficient and difficult to train for neural networks with integer weights. In this paper, a novel regeneratable dynamic differential evolution algorithm (RDDE) is presented. This algorithm is efficient for training networks with integer weights. In comparison with the conventional differential evolution algorithm (DE), RDDE has introduced three new strategies: (1) A regeneratable strategy is introduced to ensure further evolution, when all the individuals are the same after several iterations such that they cannot evolve further. In other words, there is an escape from the local minima. (2) A dynamic strategy is designed to speed up convergence and simplify the algorithm by updating its population dynamically. (3) A local greedy strategy is introduced to improve local searching ability when the population approaches the global optimal solution. In comparison with other gradient based algorithms, RDDE does not need the gradient information, which has been the main obstacle for training networks with integer weights. The experiment results show that RDDE can train integer-weight networks more efficiently.

Key words: Differential evolution, Integer weights, Neural networks, Greedy, Embedded systems, Function approximation
doi:10.1631/jzus.C1000137 **Document code:** A **CLC number:** TP183

1 Introduction

Neural networks (NNs) have proved powerful in speech recognition, handwritten character recognition, signal processing, and other applications in recent years (Fukushima and Wake, 1991; Robert *et al.*, 2002; Nejadgholi and Seyyedsalehi, 2007). Most of these applications in automotives, handheld devices, automatic control, home appliances, etc. have used embedded systems (ESs) which mostly use advanced RISC machines (ARMs), digital signal processing (DSP), field-programmable gate array (FPGA), and other systems. However, mapping of resultant NNs onto fast and compact ESs is a difficult task because conventional NNs that use real weights and a non-linear activation function are expensive when storing weights and implementing calculation in ESs. NNs with integer weights are less expensive and easier to implement in ESs, and the storage of integer weights is much easier as well. Another obstacle is the learning algorithm, because conventional algorithms are usually awkward in training networks with integer weights.

Since the back-propagation (BP) algorithm was successfully applied to NNs with real weights, many modified and new algorithms have also been proposed (Rumelhart *et al.*, 1986; Rumelhart and McClelland, 1986; Holmstrom and Koistinen, 1992). Many of them have a fast convergence rate, such as BP with momentum, and the Levenberg-Marquardt (LM) algorithm (Hagan and Menhaj, 1994; Phansalkar and Sastry, 1994). The BP algorithm is the most

[#] Presented at the 21st China Process Control Conference, August 6–8, 2010, Hangzhou, China popular training algorithm for NNs, but it requires a huge number of iterations to obtain the desired results and is easily trapped in local minima. The LM algorithm has a fast convergence speed and requires fewer iterations, but it requires large computation, and is not well suited for large size networks. However, these conventional algorithms use gradient information to converge to local optima over a continuous space. When these algorithms train the networks with integer weights, it is particularly difficult to converge over a discrete space, because only when weight increments are larger than or equal to 1 at one step, can the weights be updated. Otherwise, the increment will be quantized to zero by the quantization operation and the training will be trapped in one of the spurious local minima caused by weight quantization.

There are some studies focusing on algorithms for training networks with integer weights or limited precision weights. These studies can be divided into the following three categories:

1. Algorithms modifying the conventional training algorithms that are based on gradient information, such as BP and its variants (Woodland, 1989; Marchesi *et al*., 1990; Khan and Hines, 1994; Alibeik *et al*., 1995; Anguita and Gomes, 1996; Babri *et al*., 1998; Kamio *et al*., 2000; Behan *et al*., 2008).

One of these algorithms is the quantized gradient descent rule (QGDR) that has been proposed using a modified version of the BP algorithm (Alibeik *et al*., 1995). In this method, initially, the network is trained with floating-point weights using the BP algorithm. Then the weights are quantized and the network is trained using QGDR. Then, each weight (w_i) is changed in the opposite direction of $\dfrac{\mathrm{d}E}{\mathrm{d}w_i}$ (E is the error of neural network output) for an amount that equals the quantization of one step. After that, weights quantization operation and QGDR are repeated until the desired weight precision is reached.

2. Algorithms using different strategies that are not based on gradient information, such as the optimum descent point learning method (ODPLM) (Yan *et al*., 2008).

ODPLM searches the point with the smallest error function in a discrete space. Each iteration searches the optimum neighbor and then replaces the current point. This cycle repeats until the network converges.

3. Algorithms using evolutionary algorithms such as genetic algorithms, differential evolution and so on (Plagianakos and Vrahatis, 1999; 2000; 2002; Bao *et al*., 2009).

One of these algorithms is the differential evolution algorithm with threshold activation functions (DETAF), which has been proposed for training small size networks with integer weights in order to solve simple classification problems such as the XOR problem (Plagianakos and Vrahatis, 2000).

Unfortunately, QGDR does not perform well in convergence since it is based on gradient information with continuous weights. ODPLM has a good convergence. However, ODPLM requires a very long computational time. DETAF using conventional differential evolution (DE) to solve simple classification problems is easily trapped in local minima and is not well suited for solving complex problems. Thus, our main goal is to propose a novel DE algorithm called the regeneratable dynamic differential evolution algorithm (RDDE) to train relatively large size networks with integer weights more efficiently, in order to make these networks more powerful and efficient.

2 Learning capability of integer-weight neural networks

Formally, a network consists of Q layers, where the first layer denotes the inputs, the final one is the outputs, and the intermediate layers are the hidden layers (Fig. 1). It is assumed that the qth layer has n_q neurons. The neurons operate according to the following equation:

$$y_i^q = f\left(\sum_{j=1}^{n_{q-1}} w_{ij}^q y_j^{q-1} + \theta_i^q\right), \tag{1}$$

where w_{ij}^q is the connection weight from the jth neuron at the (q−1)th layer to the ith neuron at the qth layer, y_j^{q-1} is the output of the jth neuron belonging to the (q−1)th layer, θ_i^q denotes the bias of the ith neuron at the qth layer, and $f(x)$ is the activation function. Each neuron implements a hyperplane in the n_{q-1}-dimensional space of its inputs, which can be presented as follows:

$$\sum_{j=1}^{n_{q-1}} w_{ij}^q y_j^{q-1} + \theta_i^q = 0. \tag{2}$$

We can then study how the integer weight range affects the learning capability of integer-weight neural networks (IWNNs) (Draghici, 2002). Considering a 2D space, hyperplanes can be drawn as a mesh according to Eq. (2). When the weights lie in a very restricted range, the mesh becomes very thin with large interstices in-between hyperplanes, so different patterns will be easily put in the same interstice, such that the patterns from different classes are more difficult to separate by the hyperplanes. In other words, the learning capability of IWNNs is relatively weak when the weight range is smaller. Oppositely, the mesh becomes denser as the weight range becomes larger. Thus, if the weight range is large enough, the IWNNs can be used to approximate the response of their counterparts with continuous weights (Khan and Wilson, 1996).

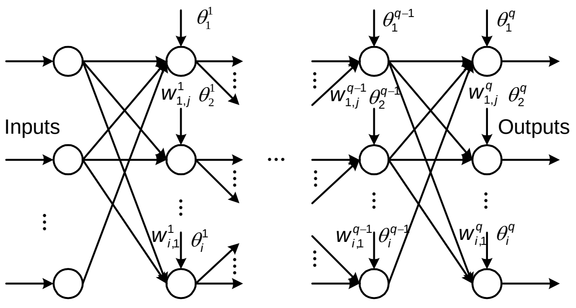


Fig. 1 Architecture of the feedforward neural network

3 Differential evolution

DE has been introduced recently by Storn and Price (1997). It is an efficient heuristic algorithm for global optimization over a continuous space (Fig. 2).

NP is the number of individuals $\boldsymbol{X}_{i,G}=[x_{1i,G}, x_{2i,G}, ..., x_{Di,G}]$ (i=1, 2, …, NP) in population, and it does not change during the process; F denotes the mutation factor, $F\in[0, 2]$; CR is the crossover factor, CR$\in[0, 1]$; D is the dimension of each individual; G denotes the Gth generation, and $G_{\max}$ is the maximum generation; $\boldsymbol{X}_{\text{opt}}$ is the optimum individual, and $f(\boldsymbol{x})$ is the fitness function. The main operations are described below.

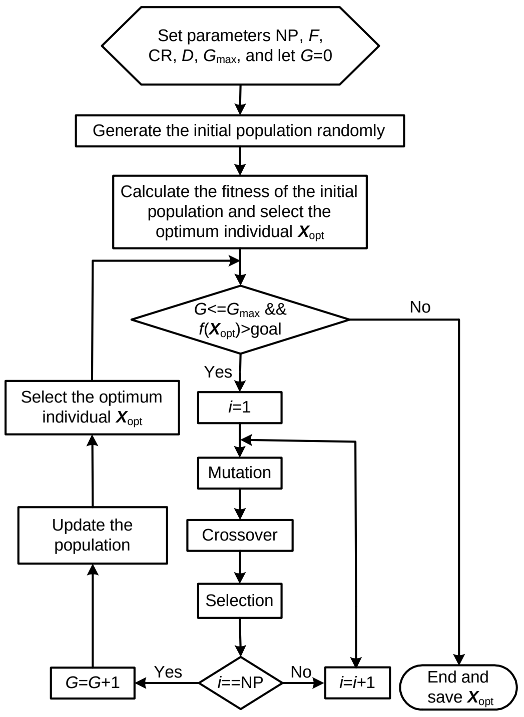

Set parameters NP, F, CR, D, Gmax, and let G=0
Generate the initial population randomly
Calculate the fitness of the initial population and select the optimum individual Xopt
G<=Gmax && f(Xopt)>goal
No
Yes
i=1
Mutation
Crossover
Selection
i==NP
Yes
No
i=i+1
G=G+1
Update the population
Select the optimum individual Xopt
End and save Xopt

Fig. 2 Flowchart of the differential evolution algorithm

3.1 Mutation

For each target individual $\boldsymbol{X}_{i,G}$ (i=1, 2, …, NP), a mutant individual can be generated according to

$$\boldsymbol{V}_{i,G+1} = \boldsymbol{X}_{r_1,G} + F\left(\boldsymbol{X}_{r_2,G} - \boldsymbol{X}_{r_3,G}\right), \tag{3}$$

where $i\neq r_1\neq r_2\neq r_3$, and $r_1, r_2, r_3\in\{1, 2, ..., \text{NP}\}$.

3.2 Crossover

To increase the diversity of the population, crossover is introduced. The trial individual is $\boldsymbol{U}_{i,G}=[u_{1i,G}, u_{2i,G}, ..., u_{Di,G}]$ (i=1, 2, …, NP). In the crossover operation, the jth gene of the ith individual of the next generation is generated from the perturbed individuals $\boldsymbol{V}_{i,G+1}$ and $\boldsymbol{X}_{i,G}$. The gene is presented as follows:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if rand}(j) \leq \text{CR or } j = \text{rni}(i), \\ x_{ji,G}, & \text{else}, \end{cases} \tag{4}$$
$$j = 1, 2, \ldots, D,$$

where random number rand(j)$\in[0, 1]$ and random integer number rni(i)$\in\{1, 2, …, D\}$.

3.3 Selection

The trial individual $\boldsymbol{U}_{i,G+1}$ competes only with its parent $\boldsymbol{X}_{i,G}$ to decide whether it should be a member of the next generation. The parent is replaced by its trial individual if the trial individual's fitness is equal to or smaller than its parent's fitness. Otherwise, the parent is retained. Its operation can be expressed as follows:

$$\boldsymbol{X}_{i,G+1}=\begin{cases}\boldsymbol{U}_{i,G+1}, & \text{if } f(\boldsymbol{U}_{i,G+1})\le f(\boldsymbol{X}_{i,G}),\\ \boldsymbol{X}_{i,G}, & \text{else.}\end{cases} \quad (5)$$

There are several variants of DE that can be denoted as DE/*x*/*y*/*z*. The most popular variants are DE/rand/1/bin and DE/best/2/bin. The basic DE algorithm described above is the first one. The second one has different mutation and crossover operations:

$$\boldsymbol{V}_{i,G+1}=\boldsymbol{X}_{\text{best},G}+F(\boldsymbol{X}_{r_1,G}+\boldsymbol{X}_{r_2,G}-\boldsymbol{X}_{r_3,G}-\boldsymbol{X}_{r_4,G}), \quad (6)$$

$$u_{ji,G+1}=\begin{cases}v_{ji,G+1}, & \text{if rand}(j)\le \text{CR},\\ x_{ji,G}, & \text{else,}\end{cases} \quad j=1, 2, \ldots, D, \quad (7)$$

where $\boldsymbol{X}_{\text{best},G}$ is the optimum individual which has the smallest fitness. In addition, best$\neq r_1\neq r_2\neq r_3\neq r_4$, and r_1, r_2, r_3, $r_4\in\{1, 2, ..., \text{NP}\}$.

4 Regeneratable dynamic differential evolution for networks with integer weights

There have been studies examining training NNs with DE over a continuous and discrete space (Plagianakos and Vrahatis, 1999; 2000; 2002; Ilonen *et al*., 2003; Slowik and Bialko, 2008; Basturk and Gunay, 2009).

In this work, however, our goal is to train IWNNs that will be used in ESs. The DE methods proposed for training continuous-weight networks are not suitable for integer-weight networks. On the other hand, the DE methods, which have been proposed for integer-weight networks by other researchers, are useful for solving simple classification problems. Furthermore, our experiments show that these methods do not perform well in convergence or generalization on function approximation and difficult classification problems.

In order to train NNs with integer weights efficiently and to obtain more powerful integer-weight networks, RDDE is presented in this work based on DE/best/2/bin. Its three newly introduced strategies are presented below.

4.1 Regeneratable strategy

In the optimization process over a constrained integer space, all individuals will easily evolve to an identical state after some generations, and then the population cannot evolve further. Because the mutation operation does not make a contribution to the population evolution process, this will probably not reach the global optimization or the desired optimization, and then the subsequent operations are all useless.

To avoid this problem and lead the population to further evolutions, a regeneratable strategy is introduced to eliminate the whole population, while reserving only one individual. In other words, this is the best individual to form the next new population. Then one regenerates NP−1 individuals randomly in the problem space, and after this operation, these individuals and the reserved one form a new population to evolve further. This will preserve the previous results, and the new random population will help the preserved one escape from the local minima.

4.2 Dynamic strategy

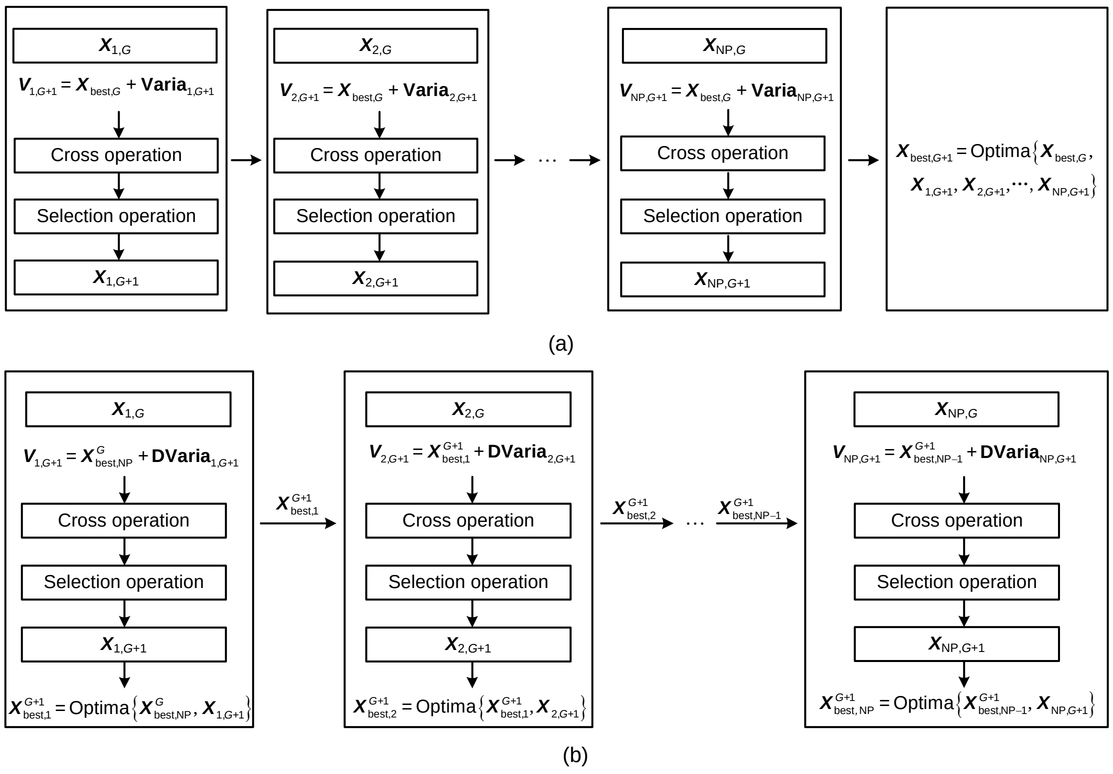
In order to speed up the convergence, a dynamic strategy is introduced (Qing, 2006). The current individual $\boldsymbol{X}_{i,G}$ is replaced by its trial individual if the trial one's fitness is smaller than the current one's. Then the trial one $\boldsymbol{X}_{i,G+1}$ is used immediately in the subsequent evolution operations by Eq. (9). Moreover, the current optimum individual competes with the trial one that has replaced its parent. The current optimum individual is replaced by this trial one if the trial one's fitness is also smaller than the current optimum one's fitness. The updated optimum individual is also used in the subsequent evolution operations immediately according to Eq. (10).

Fig. 3 shows the diagrams of DE/best/2/bin and the dynamic strategy of RDDE in the *G*th generation. **Varia**$_{i,G+1}$ and **DVaria**$_{i,G+1}$ in Fig. 3 can be generated according to

$$\textbf{Varia}_{i,G+1}=F\left(\boldsymbol{X}_{r_1,G}+\boldsymbol{X}_{r_2,G}-\boldsymbol{X}_{r_3,G}-\boldsymbol{X}_{r_4,G}\right), \quad (8)$$

$$\textbf{DVaria}_{i,G+1}=F(\boldsymbol{X}_{r_1,Ga}+\boldsymbol{X}_{r_2,Gb}-\boldsymbol{X}_{r_3,Gc}-\boldsymbol{X}_{r_4,Gd}), \quad (9)$$

$$\boldsymbol{V}_{i,G+1}=\boldsymbol{X}^{G+1}_{\text{best},i-1}+\textbf{DVaria}_{i,G+1}, \; 1<i\le \text{NP}, \quad (10)$$



$X_{1,G}$
$V_{1,G+1} = X_{\text{best},G} + \text{Varia}_{1,G+1}$
Cross operation
Selection operation
$X_{1,G+1}$
$X_{2,G}$
$V_{2,G+1} = X_{\text{best},G} + \text{Varia}_{2,G+1}$
$X_{2,G+1}$
$X_{\text{NP},G}$
$V_{\text{NP},G+1} = X_{\text{best},G} + \text{Varia}_{\text{NP},G+1}$
$X_{\text{NP},G+1}$
$X_{\text{best},G+1} = \text{Optima}\{X_{\text{best},G}, X_{1,G+1}, X_{2,G+1}, \cdots, X_{\text{NP},G+1}\}$
(a)
$V_{1,G+1} = X^{G}_{\text{best,NP}} + \text{DVaria}_{1,G+1}$
$X^{G+1}_{\text{best},1} = \text{Optima}\{X^{G}_{\text{best,NP}}, X_{1,G+1}\}$
$X^{G+1}_{\text{best},1}$
$V_{2,G+1} = X^{G+1}_{\text{best},1} + \text{DVaria}_{2,G+1}$
$X^{G+1}_{\text{best},2} = \text{Optima}\{X^{G+1}_{\text{best},1}, X_{2,G+1}\}$
$X^{G+1}_{\text{best},2}$
$X^{G+1}_{\text{best,NP}-1}$
$V_{\text{NP},G+1} = X^{G+1}_{\text{best,NP}-1} + \text{DVaria}_{\text{NP},G+1}$
$X^{G+1}_{\text{best, NP}} = \text{Optima}\{X^{G+1}_{\text{best,NP}-1}, X_{\text{NP},G+1}\}$
(b)

Fig. 3 Diagram of DE/best/2/bin (a) vs. the dynamic strategy of the regeneratable dynamic differential evolution algorithm (RDDE) (b) in one generation

where $r_1 \neq r_2 \neq r_3 \neq r_4$, and $r_1, r_2, r_3, r_4 \in \{1, 2, ..., \text{NP}\}$, and $X_{r_1,Ga}$, $X_{r_2,Gb}$, $X_{r_3,Gc}$, and $X_{r_4,Gd}$ are generated according to

$$X_{r_j,Gx} = \begin{cases} X_{r_s,G}, \ i \le r_s \le \text{NP}, \\ X_{r_d,G+1}, \ 1 \le r_d < i, \end{cases} \tag{11}$$
$$x \in \{a, b, c, d,\}, \ j \in \{1,2,3,4\},$$

and Optima() denotes selecting the best individual, which has the smallest fitness, from a candidate set.

4.3 Local greedy strategy

When the population approaches the global optimization or the desired optimization, there still needs to be many generations to reach the optimization because the local search ability of DE is relatively weak. To solve this problem, a local greedy strategy, which makes the locally optimal choice at each stage, with the hope of finding the desired optimization, is introduced when the population approaches the desired optimization. In this work, the local greedy strategy searches some neighbors, but not all, because the computation is large when the parameter dimension is large.

Training NNs with integer weights is equivalent to minimizing the error function $E(x)$, which is defined as follows:

$$\begin{cases} E(\boldsymbol{W}) = \dfrac{1}{2}\sum_{i=1}^{p} E_i(\boldsymbol{W}) = \dfrac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{N}(d_{ij} - y_{ij})^2, \\ \boldsymbol{W} = [w_1, w_2, \ \ldots, w_D], \end{cases} \tag{12}$$

where $\boldsymbol{W}$ is the integer weight vector, p is the number of input patterns, N is the number of output neurons, d_{ij} is the desired value of the jth output neuron at the ith pattern, y_{ij} is the actual response of the jth output neuron at the ith pattern, and D is the total number of weights and biases.

In order to apply RDDE to training NNs with integer weights, the fitness function is defined as

$$f(\boldsymbol{W}) = E(\boldsymbol{W}), \tag{13}$$

and the mutation is modified as follows:

$$\boldsymbol{V}_1 = \boldsymbol{W}_{r_1,Ga} - \boldsymbol{W}_{r_2,Gc}, \tag{14}$$

$$\boldsymbol{V}_2 = \boldsymbol{W}_{r_3,Gb} - \boldsymbol{W}_{r_4,Gd}, \tag{15}$$

$$\boldsymbol{V}_{i,G+1} = \text{round}\left(\boldsymbol{W}_{\text{best},i-1}^{G+1} + F\left(\boldsymbol{V}_1 - \boldsymbol{V}_2\right)\right), \tag{16}$$

where round(x) rounds x to the nearest integer, and $\boldsymbol{W}_{\text{best},i-1}^{G+1}$ is the optimum integer weight vector.

The crossover factor is computed as follows:

$$\text{CR} = \text{CR}_{\max} - \left(\text{CR}_{\max} - \text{CR}_{\min}\right)G / G_{\max}, \tag{17}$$

where $\text{CR}_{\max}$ is the largest crossover factor and $\text{CR}_{\min}$ is the smallest one. Storn and Price (1997) studied the choice of CR. CR=0.1 is a good choice but slow and CR=0.9 is appropriate to speed up convergence, with CR=0.5 being recommended. Our experiments show that the algorithm works well with $\text{CR}_{\max}$=0.9 and $\text{CR}_{\min}$=0.5. According to Eq. (17), the crossover factor is large at the beginning of training. This will increase the diversity of the population and speed up the convergence. With the training process carried on, the factor is smaller, which will bring a detailed search in the weights space.

In the end, the activation function is quantized as a look-up table since RDDE does not depend on the gradient information (Fig. 4). The procedure of RDDE for training NNs with integer weights consists of the following 13 steps.

Step 1: Set parameters NP, F, CR, D, $G_{\max}$, and G.

Step 2: Generate the initial population with integers randomly.

Step 3: Calculate the fitness of the initial population and select the optimum individual $\boldsymbol{W}_{\text{best}}$.

Step 4: If $G \le G_{\max}$ and $f(\boldsymbol{W}_{\text{best}}) > E_{\text{allowed}}$, set i=1; else, end the procedure and save $\boldsymbol{W}_{\text{best}}$.

Step 5: Select $\boldsymbol{W}_{r_1}$, $\boldsymbol{W}_{r_2}$, $\boldsymbol{W}_{r_3}$, and $\boldsymbol{W}_{r_4}$ randomly, $r_1 \ne r_2 \ne r_3 \ne r_4$, and $r_1, r_2, r_3, r_4 \in \{1, 2, ..., \text{NP}\}$.

Step 6: Calculate the mutant individual $\boldsymbol{V}_{i,G+1}$ by Eq. (16) and the crossover factor CR by Eq. (17), and then set j=1.

Step 7: If rand(j)<CR, set $u_{ji,G+1}=v_{ji,G+1}$; else, $u_{ji,G+1}=w_{ji,G}$.

Step 8: If $j \ne D$, set $j=j+1$ and jump to Step 7.

Step 9: If the fitness $f(\boldsymbol{U}_{i,G+1}) < f(\boldsymbol{W}_{i,G})$, update the weights $\boldsymbol{W}_{i,G}=\boldsymbol{U}_{i,G+1}$; else, jump to Step 11.

Step 10: Compare the current weights' fitness $f(\boldsymbol{W}_{i,G})$ and the best weights' fitness $f(\boldsymbol{W}_{\text{best}})$. If $f(\boldsymbol{W}_{i,G}) < f(\boldsymbol{W}_{\text{best}})$, set $\boldsymbol{W}_{\text{best}}=\boldsymbol{W}_{i,G}$.

Step 11: If $i \ne$NP, set $i=i+1$ and jump to Step 5.

Step 12: If all the weight vectors are the same, select the regeneratable strategy; else, if the best fitness $f(\boldsymbol{W}_{\text{best}})$ is close to E_{allowed} judging by $E_{\text{allowed}} < f(\boldsymbol{W}_{\text{best}}) < E_{\text{allowed}}+e$, select the greedy strategy.

Step 13: $G=G+1$, jump to Step 4.

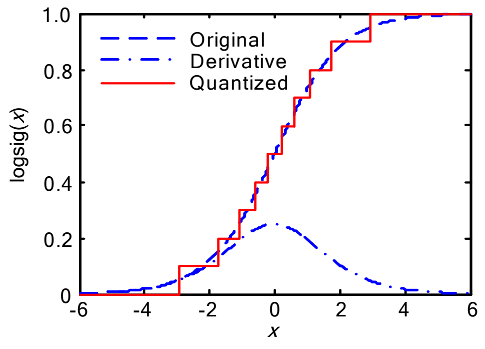


Fig. 4 Quantized log-sigmoid function

The parameter e, which denotes the radius close to the desired optimization E_{allowed}, has to be determined by the user. According to our experiments, an appropriate value for e is E_{allowed} in this paper.

5 Experiments

In this section, the performance of RDDE in integer-weight NNs training is evaluated using two test problems.

First, RDDE is compared with DE/rand/1/bin, DE/best/2/bin, QGDR, DETAF, and ODPLM, respectively, to approximate the function

$$g(x) = \mathrm{e}^{-x}\sin(2\pi x),\ 0 \le x \le 1 \tag{18}$$

that has been tested by Yan *et al.* (2008). The parameters of the network are all the same as those listed in Yan *et al.* (2008), such as 1-4-1 network structure which has 13 weights, the activation function for the hidden layer and the output layer are log-sigmoid (quantized as a look-up table individually, Fig. 4) and linear, the allowed value of error function E_{allowed}=0.01, the weights range is [−10, 10], and the training set is obtained by sampling the function at the points x=0, 0.1, …, 0.9, 1.0. Then the parameters of DE/rand/1/bin, DE/best/2/bin, and RDDE are: D=13, NP=26, $G_{\max}$=5000, F=0.4. The weight vectors are initialized with random integers from the interval [−1, 1] following a uniform probability distribution, and 20 simulations are conducted in this work.

The simulation results of the six algorithms are listed in Table 1. A weight vector obtained from training by RDDE is $\boldsymbol{W}$=[9, 0, 9, 10, −4, −1, 1, −9, −2, 1, 5, 1, −4] and the corresponding value of error function is E=0.0099.

As demonstrated in Table 1, RDDE shows greater capability in convergence than DE/rand/1/bin, DE/best/2/bin, QGDR, and DETAF. In addition, the time of RDDE is far less than that of ODPLM, and its convergence rate is satisfactory. Further investigation reveals that ODPLM requires an unreasonable amount of time as the network size becomes large, because 3^D-1 neighbor points need to be searched at each iteration, where D is the weight vector dimension. Thus, ODPLM is not suited for large-scale networks; this will limit the application of integer-weight NNs. On the contrary, this problem has little influence on RDDE.

The simulation results of RDDE for training IWNNs with different weight ranges and network structures are listed in Table 2. In this simulation, the function to be approximated is still $g(x)$ according to Eq. (18), and the other parameters are the same except the weight range, network structure, and weight vector dimension D.

Second, a classification problem has also been used for testing the functionality. In this experiment, we classify the circle and triangle points that are distributed in the 2D double helix space (Fig. 5). A 2-15-1 network has been used and $E_{\text{allowed}}\leq0.05$, D=61, NP=50, $G_{\max}$=2000, F=0.4, and the weight range is [−20, 20]. In addition, all the trainings by different algorithms have been stopped when the allowed running time is greater than 30 min.

The simulation results of the six algorithms are shown in Table 3. RDDE outperforms DE/rand/1/bin, DE/best/2/bin, QGDR, DETAF, and ODPLM greatly. ODPLM cannot converge in the allowed time because there are $3^{61}-1$ adjacent points to be searched at each iteration and the implementation calculation is too large to finish one iteration in the allowed time. Fortunately, RDDE is not significantly affected by the

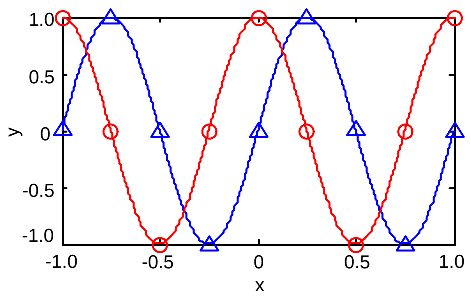


Fig. 5 Circle and triangle points distributed in the 2D double helix space

Table 1 Simulation results on function approximation by DE/rand/1, DE/best/2, QGDR, DETAF, ODPLM, and RDDE

Method	Min	Max	Mean	Time (s)	Succ (%)
DE/rand/1	5000	5000	5000	228.2	0
DE/best/2	5000	5000	5000	311	0
QGDR	30 000	30 000	30 000	189	0
DETAF	5000	5000	5000	269	0
ODPLM*	6	14	11	420	100
RDDE	241	4655	1213	44	75

* From Yan *et al.* (2008). QGDR: quantized gradient descent rule; DETAF: differential evolution algorithm with threshold activation functions; ODPLM: optimum descent point learning method; RDDE: regeneratable dynamic differential evolution algorithm. Min: the minimum number of generations; Max: the maximum number of generations; Mean: the mean number of generations; Time: the mean time of training, Succ: the success rate in 20 simulations

Table 2 Simulation by RDDE with different weight ranges and network structures

Weight range	Network structure	Min	Mean	Time (s)	Succ (%)
[−5, 5]	1-4-1	5000	5000	178.4	0
[−5, 5]	1-8-1	2275	2483	95.5	30
[−5, 5]	1-15-1	508	962	37.5	60
[−10, 10]	1-4-1	241	1213	44.0	75
[−10, 10]	1-8-1	177	747	29.7	95
[−10, 10]	1-15-1	228	970	38.7	90
[−20, 20]	1-4-1	155	982	37.4	80
[−20, 20]	1-8-1	164	1133	42.1	90
[−20, 20]	1-15-1	132	705	26.6	90

RDDE: regeneratable dynamic differential evolution algorithm. Min: the minimum number of generations; Max: the max number of generations; Mean: the mean number of generations; Time: the mean time of training; Succ: the success rate in 20 simulations weight dimension. Thus, RDDE will be more suited for relatively large scale integer-weight networks.

Finally, the full trained NNs are used in different embedded platforms such as ARM 9 and DSP. Table 4 compares the computing efficiency of float-weight neural networks (FWNNs) and IWNNs working on different platforms. Table 5 compares the computing efficiency of FWNNs and IWNNs with different network structures, which has been proposed above, working on ARM 9 (0.075 GHz) embedded systems. The results show that the computing efficiency has been greatly improved when IWNNs are working on ARM 9 embedded systems (about 10–18 times).

Table 3 Simulation results on the classification problem by DE/rand/1, DE/best/2, QGDR, DETAF, ODPLM, and RDDE

Method	Min	Max	Mean	Time (s)	Succ (%)
DE/rand/1	2000	2000	2000	389	0
DE/best/2	2000	2000	2000	552	0
QGDR	30 000	30 000	30 000	272	0
DETAF	2000	2000	2000	469	0
ODPLM*				1800	0
RDDE	305	1337	794	132	85

* Does not finish one iteration in the allowed time. QGDR: quantized gradient descent rule; DETAF: differential evolution algorithm with threshold activation functions; ODPLM: optimum descent point learning method; RDDE: regeneratable dynamic differential evolution algorithm. Min: the minimum number of generations; Max: the maximum number of generations; Mean: the mean number of generations; Time: the mean time of training; Succ: the success rate in 20 simulations

Table 4 Comparison of computing efficiency of 1-15-1 neural networks (NNs) on different platforms for 1000 tests

Platform	Time (ms)		Rate
	FWNNs	IWNNs	
PC	79	32	2.5:1
DSP	257	85	3:1
ARM 9	7734	432	18:1

At 2.8 GHz for PC, 0.3 GHz for DSP, and 0.075 GHz for ARM 9. FWNNs: float-weight neural networks; IWNNs: integer-weight NNs

Table 5 Comparison of computing efficiency on ARM 9 (0.075 GHz) embedded systems for 1000 tests

Network structure	Time (ms)		Rate
	FWNNs	IWNNs	
1-4-1	3525	366	10:1
1-15-1	7734	432	18:1
2-15-1	8021	489	16:1

FWNNs: float-weight neural networks; IWNNs: integer-weight NNs

6 Conclusions and future research

In this paper, the regeneratable dynamic differential evolution algorithm (RDDE) has been proposed to train neural networks with integer weights. The results indicate that the new algorithm is effective for training integer-weight neural networks (IWNNs), and that IWNNs can effectively reduce the complexity of network computing. As a result, the mapping of resultant IWNNs onto the embedded systems is promising. However, further research is needed. For example, an advisable method should be developed to obtain the minimum integer weight range to ensure that the integer-weight networks have the capability of solving the function approximation problem and other given problems. Accurate theory analysis can be performed to examine both integer-weight networks and the strategy of parameter selection in RDDE.

References

Alibeik, S.A., Nemati, F., Sharif-Bakhtiar, M., 1995. Analog Feedforward Neural Networks with Very Low Precision Weights. IEEE Int. Conf. on Neural Networks, p.90-94. [doi:10.1109/ICNN.1995.487908]

Anguita, D., Gomes, B.A., 1996. Mixing floating- and fixed-point formats for neural network learning on neuroprocessors. *Microprocess. & Microprogr.*, **41**(10): 757-769. [doi:10.1016/0165-6074(96)00012-9]

Babri, H.A., Chen, Y.Q., Yin, T., 1998. Improving backpropagation learning under limited precision. *Pattern Recogn. Lett.*, **19**(11):1007-1016. [doi:10.1016/S0167-8655(98)00081-6]

Bao, J., Zhou, B., Yan, Y., 2009. A Genetic-Algorithm-Based Weight Discretization Paradigm for Neural Networks. WRI World Conf. on Computer Science and Information Engineering, p.655-659. [doi:10.1109/CSIE.2009.601]

Basturk, A., Gunay, E., 2009. Efficient edge detection in digital images using a cellular neural network optimized by differential evolution algorithm. *Exp. Syst. Appl.*, **36**(2):2645-2650. [doi:10.1016/j.eswa.2008.01.082]

Behan, T., Liao, Z., Zhao, L., Yang, C.T., 2008. Accelerating Integer Neural Networks on Low Cost DSPs. Proc. Int. Conf. on Intelligent Systems, p.1270-1273.

Draghici, S., 2002. On the capabilities of neural networks using limited precision weights. *Neur. Networks*, **15**(3): 395-414. [doi:10.1016/S0893-6080(02)00032-1]

Fukushima, K., Wake, N., 1991. Handwritten alphanumeric character recognition by the neocognitron. *IEEE Trans. Neur. Networks*, **2**(3):355-365. [doi:10.1109/72.97912]

Hagan, M.T., Menhaj, M.B., 1994. Training feedforward networks with the Marquardt algorithm. *IEEE Trans. Neur. Networks*, **5**(6):989-993. [doi:10.1109/72.329697]

Holmstrom, L., Koistinen, P., 1992. Using additive noise in back-propagation training. *IEEE Trans. Neur. Networks*, **3**(1):24-38. [doi:10.1109/72.105415]

Ilonen, J., Kamarainen, J.K., Lampinen, J., 2003. Differential evolution training algorithm for feed-froward neural networks. *Neur. Process. Lett.*, **17**(1):93-105. [doi:10.1023/A:1022995128597]

Kamio, T., Tanaka, S., Morisue, M., 2000. Backpropagation Algorithm for Logic Oriented Neural Networks. Proc. IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks, **2**:123-128. [doi:10.1109/IJCNN.2000.857885]

Khan, A.H., Hines, E.L., 1994. Integer-weight nueral nets. *Electron. Lett.*, **30**(15):1237-1238. [doi:10.1049/el:19940817]

Khan, A.H., Wilson, R.G., 1996. Integer-Weight Approximation of Continuous-Weight Multilayer Feedforward Nets. IEEE Int. Conf. on Neural Networks, p.392-397. [doi:10.1109/ICNN.1996.548924]

Marchesi, M., Orlandi, G., Piazza, F., Pollonara, L., Uncini, A., 1990. Multi-layer Perceptrons with Discrete Weights. Int. Joint Conf. on Neural Networks, **2**:623-630. [doi:10.1109/IJCNN.1990.137772]

Nejadgholi, I., Seyyedsalehi, S.A., 2007. Nonlinear normalization of input patterns to speaker variability in speech recognition neural networks. *Neur. Comput. Appl.*, **18**(1):45-55. [doi:10.1007/S00521-007-0151-5]

Phansalkar, V.V., Sastry, P.S., 1994. Analysis of the back-propagation algorithm with momentum. *IEEE Trans. Neur. Networks*, **5**(3):505-506. [doi:10.1109/72.286925]

Plagianakos, V.P., Vrahatis, M.N., 1999. Neural Network Training with Constrained Integer Weights. Proc. Conf. on Evolutionary Computation, **3**:2007-2013. [doi:10.1109/CEC.1999.785521]

Plagianakos, V.P., Vrahatis, M.N., 2002. Parallel evolutionary training algorithms for “hardware-friendly” neural networks. *Nat. Comput.*, **1**(2-3):307-322. [doi:10.1023/A:1016545907026]

Qing, A., 2006. Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems. *IEEE Trans. Geosci. Remote Sens.*, **44**(1):116-125. [doi:10.1109/TGRS.2005.859347]

Robert, C., Gaudy, J.F., Limoge, A., 2002. Electroencephalogram processing using neural networks. *Clin. Neurophysiol.*, **113**(5):694-701. [doi:10.1016/S1388-2457(02)00033-0]

Rumelhart, D.E., McClelland, J.L., 1986. Paralle Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Fundations. MIT Press, Cambridge, MA, USA.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. *Nature*, **323**:533-536. [doi:10.1038/323533a0]

Slowik, A., Bialko, M., 2008. Training of Artificial Neural Networks Using Differential Evolution Algorithm. Conf. on Human System Interactions, p.60-65. [doi:10.1109/HSI.2008.4581409]

Storn, R., Price, K., 1997. Differential evolution—a simple and efficient heuristic for global optimization over continuous space. *J. Glob. Optim.*, **11**(4):341-359. [doi:10.1023/A:1008202821328]

Woodland, P.C., 1989. Weight Limiting, Weight Quantisation and Generalization in Multi-layer Perceptrons. First IEE Int. Conf. on Artificial Neural Networks, p.297-300.

Yan, Y., Zhang, H., Zhou, B., 2008. A New Learning Algorithm for Neural Networks with Integer Weights and Quantized Non-linear Activation Functions. International Federation for Information Processing, **276**:427-431. [doi:10.1007/978-0-387-09695-7_42]