# A memetic imperialist competitive algorithm with chaotic maps for multi-layer neural network training

# A Memetic Imperialist Competitive Algorithm with Chaotic Maps for Multi-Layer Neural Network Training


## Seyed Jalaleddin Mousavirad

Department of Computer and Electrical Engineering,
University of Kashan,
Kashan, Iran
E-mail: jalalmoosavirad@gmail.com, mousavirad@grad.kashanu.ac.ir

## Azam Asilian Bidgoli

Department of Computer and Electrical Engineering,
University of Kashan
Kashan, Iran
E-mail: asilian@grad.kashanu.ac.ir

## Hossein Ebrahimpour Komleh

Department of Computer and Electrical Engineering,
University of Kashan
Kashan, Iran
E-mail: ebrahimpour@kashanu.ac.ir

## Gerald Schaefer

Department of Computer Science,
Loughborough University
Loughborough, U.K.
E-mail: gerald.schaefer@ieee.org



**Abstract:** The performance of artificial neural networks (ANNs) is largely dependent on the success of the training process. Gradient descent-based methods are the most widely used training algorithms but have drawbacks such as ending up in local minima. One approach to overcome this is to use population-based algorithms such as the imperialist competitive algorithm (ICA) which is inspired by the imperialist competition between countries. In this paper, we present a new memetic approach for neural network training to improve the efficacy of ANNs. Our proposed approach – Memetic Imperialist Competitive Algorithm with Chaotic Maps (MICA-CM) – is based on a memetic ICA and chaotic maps, which are responsible for exploration of the search space, while back-propagation is used for an effective local search on the best solution obtained by ICA. Experiment results confirm our proposed algorithm to be highly competitive compared to other recently reported methods.



**Keywords:** Neural network training; imperialist competitive algorithm; memetic computing; chaotic map; back-propagation.


**Biographical notes:** Seyed Jalaleddin Mousavirad received his Ph.D. degree in Computer Engineering from the University of Kashan, Iran in 2017. He also worked as a visiting student at Xian Jiaotong Liverpool University, Suzhou, China under the supervision of Prof. Yuhui Shi. His research interests include evolutionary computation, multi-objective optimisation, and evolutionary computation applications in scientific fields such as image processing and pattern recognition.

Azam Asilian Bidgoli is currently pursuing a Ph.D. degree in Computer Engineering at Kashan University, Iran. She is also working as a visiting scholar at the University of Ontario Institute of Technology, Canada. Her research interests include evolutionary computation, multi-objective optimisation, and pattern recognition.

Hossein Ebrahimpour-Komleh is currently an Assistant Professor at the Department of Electrical and Computer Engineering at the University of Kashan, Iran. His main areas of research include computer vision, image processing, and pattern recognition.

Gerald Schaefer is currently with the Department of Computer Science, Loughborough University, U.K. His research interests include image analysis, medical imaging, pattern recognition and computational intelligence and he has about 500 publications in these areas.

## 1 Introduction

Artificial neural networks (ANNs) have been extensively employed for tasks such as pattern classification, feature selection, and prediction (Lam et al., 2014; Mousavirad et al., 2011). ANNs are computational models inspired by the human brain and one of the most important pattern recognition techniques applied for both supervised and unsupervised learning. One of the most commonly used ANN architectures is the multi-layer perceptron (MLP), a feed-forward neural network that consists of simple neurons called perceptrons.

ANN training is performed by learning from examples. During training, the MLP weight connections between nodes are adjusted to minimise the error between the actual output of the MLP and the desired output Ebrahimpour-Komleh and Mousavirad (2013). Gradient descent-based algorithms, which perform stochastic gradient descent on the weights, such as the back-propagation (BP) algorithm are among the most frequently used techniques for MLP training. These algorithms are applied in many different applications, however, they suffer from backdraws such as overfitting, sensitivity to weight initialisation, and a tendency to converge to local minima of the error function, thus not being able to find a global minimum in multi-modal error functions Yao (1999). One approach to overcome these limitations is to use population-based algorithms such as genetic algorithms (GAs), particle swarm optimisation (PSO), or ant colony optimisation (ACO). In recent years, these algorithms have been applied as training methods and have been shown to be able to address the disadvantages of gradient-based algorithms Mandischer (2002).

According to the "no free lunch" theorem Wolpert and Macready (1997), no population-based algorithm is best suited to solve all optimisation problems. This motivates many researchers to consider the performance of various population-based algorithms to train MLPs. Evolutionary algorithms (EAs) are a category of population-based algorithms that are inspired by natural evolution, and have been widely applied for neural network training. Sexton and Dorsey (2000) compared BP with a GA for neural network training and found GAs to give better performance. Mandischer (2002) compared evolution strategies (ES) and BP and showed that BP outperforms ES for most of the chosen learning problems. Cantú-Paz and Kamath (2005) presented a comprehensive empirical evaluation of eight combinations of EAs and ANNs on 15 classification problems to identify methods that consistently produce accurate classifiers that generalise well.

Another class of population-based algorithms that is applied as training algorithms is swarm intelligence (SI). These originate from the collective behaviour of swarming species such as ant and birds. Carvalho and Ludermir (2006) analysed the use of PSO and two variants with a local search operator for neural network training on three medical benchmark classification problems. Yaghini et al. (2013) presented a method for ANN training based on PSO and BP with a momentum term, and employed opposition-based learning and random perturbation for population diversity during each iteration. Some other popular population-based algorithms which are used for neural network training are ant colony optimisation (Socha and Blum, 2007), cuckoo optimisation algorithm (Ebrahimpour-Komleh and Mousavirad, 2013), social spider optimisation (Mirjalili et al., 2015), and grey wolf optimiser (Mirjalili, 2015).

The imperialist competitive algorithm (ICA) (Atashpaz-Gargari and Lucas, 2007) is a population-based algorithm which is inspired by socio-political evolution. It has advantages over traditional optimisation algorithms, not needing to calculate the gradient and reducing the probability of getting stuck in local minima, and has been shown to yield good performance both in terms of convergence rate and global search success for various applications (Mahmoudi et al., 2013; Bashiri, 2014). Similar to other population-based algorithms, it starts with a random population, which are termed countries. Countries are divided into two groups: imperialists for the best candidate solutions, and colonies. The main mechanisms that steer the search for a better solution are imperialist competition and assimilation. Weaker empires get destroyed in the imperialist competition process, and, as a result, the probability of reaching the optimal solution is increased.

In recent years, theories and applications of non-linear dynamical systems, especially of chaos, have received increased attention in various disciplines. Chaos is a kind of characteristic of non-linear systems with highly unstable motion. An essential feature of chaotic systems is the sensitivity to initial conditions such as the starting value of data. Population-based optimisation algorithms similar to chaos can help order to grow from disturbance. Due to these common properties between chaos and optimisation algorithms, the combination of these concepts lead to improved optimisation algorithms (Talatahari et al., 2012). In chaos-based optimisation algorithms, random numbers can be generated by chaotic variables instead of random variables. Chaos-based optimisation algorithms can more likely escape from a local optimum compared to stochastic optimisation algorithms (Tavazoei and Haeri, 2007), although this has not been mathematically proved yet (He et al., 2009).

Memetic algorithms are an interesting approach to solve optimisation problems. Memetic algorithms combine a population-based metaheuristic algorithm with a problem-specific local search operator. Therefore, memetic algorithms have the ability to use the advantages of population-based metaheuristic algorithms (generality, robustness, and high efficacy



Copyright © 201X Inderscience Enterprises Ltd.

of global search) and local search algorithms (rapid convergence toward local optima) simultaneously, leading to improved performance.

In this paper, we first introduce an improved ICA based on chaos theory, and then propose a memetic method for neural network training based on the improved ICA. The proposed method – Memetic Imperialist Competitive Algorithm with Chaotic Maps (MICA-CM) – combines the global search strategy of the chaotic imperialist competitive algorithm with the local search ability of gradient descent back-propagation (GDBP). This memetic algorithm uses the chaotic ICA algorithm to perform a global search and then uses back-propagation for a local search around the global optimum. Moreover, an imperialist movement operator is proposed to improve the exploitation of ICA.

The remainder of the paper is organised as follows: Section 2 introduces the multi-layer perceptron, while Section 3 explains the original imperialist competitive algorithm. Section 4 then presents our novel MICA-CM algorithm. Experimental results are given in Section 5, while Section 6 concludes the paper.

## 2 Multi-Layer Perceptron

The multi-layer perceptron (MLP) is one of the most widely used neural networks. An MLP has one input layer and one output layer of neurons, with hidden layers between the input layer and the output layer to extract useful features from the data that is fed to the network. Fig. 1(a) shows a typical MLP. Input values are transmitted from the nodes of the input layer to the nodes of the output layer by unidirectional links. Each link sends a signal from one node to another, affected by a weight which indicates the link strength between the two nodes. Moreover, an activation function is employed to control the output of a neuron. One of the most commonly used activation function, and the one employed in this paper, is the sigmoid function, defined as

$$\delta(net) = \frac{1}{1 + e^{-net}}. \quad (1)$$

Since the weights and biases have a significant impact on the output, finding suitable values for them is required to define a desirable relationship between input and output. This is achieved during the training phase of an MLP based on learning from examples.

## 3 Imperialist Competitive Algorithm

The imperialist competitive algorithm (ICA) (Atashpaz-Gargari and Lucas, 2007) is a global search metaheuristic. This population-based strategy has been shown to work well for various optimisation problems (Mahmoudi et al., 2013; Bashiri, 2014). The algorithm is based on imperialist competition among

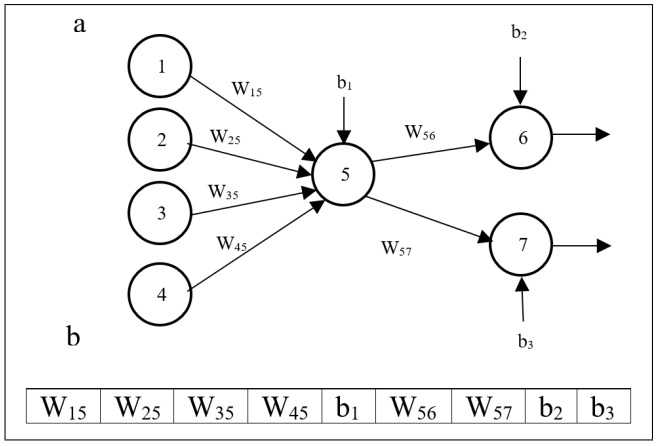


**Figure 1**: (a) An ANN with 4-1-2 structure (b) The corresponding countries.

countries. ICA starts with an initial population of countries. These are divided into two groups, colonies and imperialists, that together form empires. Then, all colonies are distributed among the imperialists according to each imperialist's power, with more powerful empires having more colonies while weaker ones have fewer. After the formation of empires, colonies start moving towards their dependent imperialist country based on an assimilation policy, based on the distance between the colonies and the imperialist, a (constant) assimilation factor, and a random number between 0 and 1, by

$$Position_{i+1} = Position_i + \gamma \times \delta \oplus d, \quad (2)$$

where $Position_i$ is the colony's position in the $i$-th iteration, $\gamma$ is the assimilation coefficient, $\delta$ is a random vector normally distributed in $[0; 1]$, and $d$ is the distance between colony and its imperialist.

Another ICA operator is revolution, which signifies a sudden change in a part of country characteristics, and improves the exploration of ICA.

The total power of an empire depends on both the power of the imperialist country and of its colonies,

$$T.C._n = Cost(imperialist_n) + \xi mean\{Cost(Colonies\ of\ empires)\}, \quad (3)$$

where $T.C._n$ is the total cost of the $n$-th empire and $\xi$ is a positive number. A small value for $\xi$ means the total power of the empire is specified mainly by the imperialist, while increasing the term will enhance the impact of the colonies.

In imperialist competition, all empires try to take possession of colonies of other empires in order to increase their own power while decreasing the power of others. This imperialist competition is simulated by picking some of the weakest colonies of the weakest empires and empires competing for these with more powerful empires having a higher chance of acquiring the colonies. Eventually, all empires except for the most powerful one will exterminate, and all colonies will have the same position.

## 4 The Proposed Algorithm

ICA is a global search algorithm with a strong ability to determine global solutions, but its ability to search around global solutions is weak. On the other hand, local operators are well suited to determine local solutions but perform poorly in identifying the global solution. In this paper, we develop a memetic algorithm named MICA-CM – for Memetic Imperialist Competitive Algorithm with Chaotic Maps – by combining ICA and local operators to exploit the advantages of both strategies. In addition, in our proposed method, a chaotic map is used to increase the diversity of the proposed algorithm. In the following, we first introduce the components of the proposed algorithm and then explain its general structure.

### *4.1 Encoding strategy*

We us a real-valued vector encoding strategy where the weights and biases constitute a vector in order to form a country. That is, a country represents the weights and biases of an MLP structure as shown in Fig. 1(b) for an example of one hidden layer, four input nodes, and two output nodes.

### *4.2 Cost function*

For classification problems, the classification error to evaluate the cost is defined as

$$\text{Classification error} = 100 \times \sum_{p=1}^{P} \xi(\sum_{n=1}^{P} \overrightarrow{p}/P) \tag{4}$$

with

$$\xi(\overrightarrow{p}) = \begin{cases} 1 & \text{if } \overrightarrow{o_p} \neq \overrightarrow{d_p} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

For each $P$-dimensional input vector $x_P$, the corresponding desired $k$-dimensional output vector is $d_p = (d_{p,1}, d_{p,2}, ..., d_{p,k})$. The goal of ANN training is to modify the weights in the network so that the network output vector $o_p = (o_{p,1}, o_{p,2}, ..., o_{p,k})$ is as close to the desired output vector $d_p$. As an approximation problem, the mean squared error (MSE) is employed and is defined as

$$MSE = \frac{1}{P} \sum_{i=1}^{P} (d_i - o_i)^2. \tag{6}$$

### *4.3 Sequences generated from chaotic maps*

Generating random sequences with a sensible uniformity is critical for population-based algorithms. Chaos is a deterministic and random-like process in dynamical systems that includes infinite bounded unstable motions. Mathematically, chaos may be explored as a source of randomness. A chaotic map (CM) is a discrete-time map $x_{k+1} = f(x_k)$ that exhibits chaotic behaviour and the resulting chaotic sequence $\{x_k, k = 1, 2, ..., \}$ can be considered as an alternative form of randomness.

Using the randomness and regularity properties of chaos, the optimal solution moves in a chaotic way among the local minima and eventually converges to the global optimal solution with a high probability (Tavazoei and Haeri, 2007). There are various chaotic maps of which we consider the following in this paper: circle map, logistic map, tent map, sinusoidal map, and ICMIC map.

In our approach, we adopt sequences generated from chaotic systems instead of random sequences for the ICA parameters. Our novel chaotic imperialist competitive algorithm applies chaotic sequences in two ways: generating the initial empires and during the assimilation process.

#### *4.3.1 Generating initial empires*

Initial countries are produced by iterating the selected chaotic maps as shown in Algorithm 1, where $pop_{i,j}$ is the $j$-th dimension of the $i$-th country in the population.

**input** : $N$: dimensionality of problem; $Varmin$, $Varmax$: lower and upper bound
**output**: pop
**for** $i \leftarrow 1$ **to** *population size* **do**
  Randomly initialise the first chaotic variables;
  **for** $j \leftarrow 1$ **to** $N$ **do**
    Generate chaotic variable $cm_{i,j}$ according to the selected map;
    $pop_{i,j} =$
    $Varmin + cm_{i,j} \times (Varmax - Varmin)$;
  **end**
**end**

**Algorithm 1:** Pseudo code of algorithm to generate the initial countries.

#### *4.3.2 Assimilation process*

Parameter $\delta$ of Eq. (2) is modified by

$$Position_{i+1} = Position_i + \gamma \times cm \oplus d, \tag{7}$$

where $cm$ is a vector generated based on the selected chaotic map.

### *4.4 Imperialist movement*

In the original ICA, imperialists are fixed and never move which prevents reaching a better solution. Therefore, an improved ICA is proposed to enhance the exploitation of ICA. In the improved ICA, for each imperialist, a few random motions are considered, and the cost of these hypothetical positions computed. If the cost of one of these new positions is lower than the cost of the current position, the imperialist is moved to

$$Position_{imp}(new) = Position_{imp}(old) + \alpha, \tag{8}$$

where $\alpha$ is a vector of small random numbers, otherwise the imperialist remains in the current position.

### 4.5 Gradient descent back-propagation

Gradient descent back-propagation (GDBP) is a classic algorithm to train an ANN where the network weights are modified in the direction that corresponds to the negative gradient of the error function. In this paper, GDBP is used as a problem-specific search operator in the memetic ICA and is applied to search around the best solution.

### 4.6 Overall structure of proposed algorithm

As mentioned, our algorithm combines two local search operators together with a chaotic ICA. The first one generates some random motions around each imperialist, while the second uses GDBP to search around the global optimum. Moreover, chaotic maps increase the exploration of the algorithm. Similar to other algorithms, MICA-CM generates some countries with chaotic maps. Then, the countries are divided into two groups: imperialists and colonies. In the next step, colonies are distributed among the imperialists. Colonies are updated using chaotic assimilation and the revolution operator. Afterwards, imperialist competition is applied. GDBP is employed to search around the best solution, and the whole process is terminated when some stopping conditions are satisfied.

In the following, the various steps of the proposed algorithm are explained in more detail:

Step 1: Initialise the parameters of ICA, including the number of countries, $NumCountries$, the number of imperialists, $NumImp$, and the number of colonies, $NumCol$ ($NumCountries = NumImp + NumCol$).

Denote the maximum iteration with $max_{Dec}$ and the current iteration with $Dec$. Initialise the starting iteration $Dec = 1$.

Step 2: Generate the initial countries by iterating through the selected chaotic map following Algorithm 1. The structure of each country is described as in Fig. 1.

Step 3: Calculate the cost value of each country.

Step 4: Select $NumImp$ of the most powerful countries to form the empires. The remaining $NumCol$ countries of the initial population are colonies. To create the initial empires, colonies are distributed among imperialists based on their power. For this purpose, the normalised cost of an imperialist, defined as

$$C_k = c_k - \max(c_i), \tag{9}$$

is used. Here, $c_k$ is the cost of the $k$-th imperialist and $C_k$ is its normalised cost.

The normalised power of each imperialist is calculated as

$$Power_k = \left| \frac{C_k}{\sum_{i=1}^{NumImp} C_i} \right|, \tag{10}$$

and gives the number of initial colonies that should be possessed by the imperialists as

$$N.C_k = round(Power_k \times NumCol), \tag{11}$$

where $N.C._k$ is the initial number of colonies of the $k$-th empire. Colonies are chosen randomly and given to the $k$-th imperialist, and these colonies together with the $k$-th imperialist form the $k$-th empire. Bigger empires comprise more colonies while weaker ones contain fewer.

Step 5: Move colonies towards their imperialist position chaotically according to Eq. (7).

Step 6: Use the imperialist movement operator, as described in Section 4.4, to search locally around each imperialist.

Step 7: Apply the revolution operator to each colony. Revolution is a random change in the position of a country. A revolution occurs according to a user-defined revolution probability. In each iteration, for each colony, a random number in the range $[0; 1]$ is generated. If the generated value is lower than the revolution probability, a revolution occurs.

Step 8: Calculate the cost of all colonies and imperialists in each empire.

During the previous steps, a colony may reach a position with lower cost than its relevant imperialist. In this case, the colony and the imperialist exchange their positions.

Step 9: Calculate the total power of the empires according to Eq. (3).

Step 10: Perform imperialist competition. All empires try to take possession of colonies of other empires according to their power (total cost). The normalised total cost of empire $i$ is calculated by

$$P = [P_{p1}, P_{p2}, ..., P_{pNumImp}] \tag{12}$$

and

$$R = [r_1, r_2, ..., r_{NumImp}], \tag{13}$$

to yield

$$D = P - R = [D_1, D_2, ..., D_{NumImp}]. \tag{14}$$

The empire chosen to possess the colony is the one with the biggest value of $D$. The weakest colony of the weakest empire will be given to the winner empire.

Step 11: An empire collapses and is removed when it loses all its colonies.

Step 12: Apply GDBP to search around the best solution obtained in the previous steps.

Step 13: Increase $Dec$ to $Dec + 1$.

Step 14: If $Dec > max_{Dec}$ the algorithm has terminated, otherwise go to the Step 5.

## 5 Experimental Results

We used several benchmarks datasets, include three function approximations and five real classification problems, to evaluate the performance of our proposed MICA-CM algorithm for MLP training. Upper and lower bounds of all weights were limited from -1 to +1, and all weights were chaotically initialised as described in Algorithm 1. We evaluated five chaos maps: circle map, logistic map, tent map, sinusoidal map, and ICMIC map.

One important aspect of an MLP is its structure. In this paper, we do not focus on finding the optimal structure, and used the suggestions in (Wdaa, 2008; Mirjalili et al., 2015) to set the number of neurons in the hidden layer to $2 \times N + 1$ ($N$ is the number of MLP inputs) and to 15 in the classification and function approximation problems, respectively.

### 5.1 Function approximation problems

We used three function approximation datasets, given in Table 1, to evaluate the proposed algorithm. We trained MLPs of structure 1-15-1 to approximate these functions. In all experiments, the maximal iteration number was set to 500, the number of countries to 40, the number of imperialists to 5, and revolution rate to 0.1. Due to the random nature of population-based algorithms, MICA-CM was run 50 times, and we report statistical results in form of average MSE and standard deviation on the test dataset.

For comparison, we evaluated also the original ICA algorithm, a standard GDBP approach and various other optimisation algorithms, namely (Mirjalili et al., 2014; Mirjalili, 2015) genetic algorithm (GA), particle swarm optimisation (PSO), ant colony optimisation (ACO), evolutionary strategy (ES), population-based incremental learning (PBIL), biogeography-based optimisation (BBO), and grey wolf optimiser (GWO).

#### 5.1.1 Sigmoid function

The sigmoid function, $y = 1/1(1 + e^{-x})$, is the simplest function used in our experiments. There are 61 training and 121 test samples in the dataset. Table 2 indicates that the logistic map gives the best results not only in terms of average MSE but also in terms of standard deviation. Table 3 clearly shows that our MICA-CM outperforms all other evaluated approaches.

**Table 1** Function approximation datasets.

| function | training samples | test samples |
|---|---|---|
| sigmoid | 61: $x \in$ [-3:0.1:3] | 121: $x \in$ [-3:0.05:3] |
| cosine | 31: $x \in$ [1.25:0.05:2.75] | 38: $x \in$ [1.25:0.04:2.75] |
| since | 126: $x \in$ [-2:0.05:2] | 252: $x \in$ [-2:0.05:2] |

**Table 2** Results for chaotic maps on *sigmoid* function.

| chaotic map | avg. MSE | std.dev. MSE |
|---|---|---|
| circle map | 3.19E-06 | 1.99E-06 |
| logistic map | **7.93E-13** | **8.02E-13** |
| tent map | 4.99E-11 | 9.22E-11 |
| sinusoidal map | 1.60E-11 | 2.00E-11 |
| ICMIC map | 1.19E-05 | 8.72E-06 |

**Table 3** Comparison of algorithms on *sigmoid* function.

| algorithm | avg. MSE | std.dev. MSE |
|---|---|---|
| GDBP | 2.75E-01 | 2.32E-03 |
| ICA | 5.53E-04 | 1.55E-04 |
| GA | 1.09E-03 | 9.16E-04 |
| PSO | 2.30E-02 | 9.43E-03 |
| ACO | 2.35E-02 | 1.00E-02 |
| ES | 7.56E-02 | 1.64E-02 |
| PBIL | 4.05E-03 | **2.74E-17** |
| BBO | 1.33E-05 | 3.57E-04 |
| GWO | 2.03E-04 | 2.26E-04 |
| MICA-CM | **7.93E-13** | 8.02E-13 |

#### 5.1.2 Cosine function

This dataset, based on $y = cos(x\pi/2)$, is more challenging than the *sigmoid* function. Data is from the interval $[-1.25; 2.75]$ with increments of 0.05 for training data and 0.04 for test data, giving 31 training samples and 38 test samples. From Table 4 it can be seen that the tent map yields the lowest MSE although the logistic and

**Table 4** Results for chaotic maps on *cosine* function.

| chaotic map | avg. MSE | std.dev. MSE |
|---|---|---|
| circle map | 4.37E-04 | 4.14E-04 |
| logistic map | 3.09E-10 | **8.92E-11** |
| tent map | **1.77E-10** | 1.95E-10 |
| sinusoidal map | 1.02E-09 | 3.26E-10 |
| ICMIC map | 4.92E-05 | 5.50E-05 |

**Table 5** Comparison of algorithms on *cosine* function.

| algorithm | avg. MSE | std.dev. MSE |
|---|---|---|
| GDBP | 1.75E-01 | 6.56E-02 |
| ICA | 4.23E-02 | 4.83E-03 |
| GA | 1.09E-02 | 6.32E-03 |
| PSO | 5.90E-02 | 2.10E-02 |
| ACO | 5.09E-02 | 1.08E-02 |
| ES | 8.66E-02 | 2.22E-02 |
| PBIL | 9.43E-02 | 1.85E-02 |
| BBO | 1.37E-02 | **1.83E-18** |
| GWO | 3.11E-03 | 2.16E-03 |
| MICA-CM | **1.77E-10** | 1.95E-10 |

**Table 6** Results for chaotic maps on *sine* function.

| chaotic map | avg. MSE | std.dev. MSE |
|---|---|---|
| circle map | 4.53E-01 | 2.25E-03 |
| logistic map | **7.33E-09** | 1.94E-09 |
| tent map | 1.02E-08 | **1.29E-09** |
| sinusoidal map | 1.06E-08 | 3.48E-09 |
| ICMIC map | 4.51E-01 | 7.78E-04 |

**Table 7** Comparison of algorithms on *sine* function.

| algorithm | avg. MSE | std.dev. MSE |
|---|---|---|
| GDBP | 1.2 | 1.11E-01 |
| ICA | 4.72E-01 | 1.42E-03 |
| GA | 4.21E-01 | 6.12E-02 |
| PSO | 5.27E-01 | 7.29E-02 |
| ACO | 5.30E-01 | 5.33E-02 |
| ES | 7.07E-01 | 7.74E-02 |
| PBIL | 4.83E-01 | 7.94E-03 |
| BBO | 1.03E-01 | 0 |
| GWO | 2.62E-01 | 1.15E-01 |
| MICA-CM | **7.33E-09** | **1.94E-09** |

sinusoidal maps give close results. Table 5 confirms that while MICA-CM performs very well, all other algorithms give relatively poor results.

### *5.1.3 Sine function*

This dataset, based on $y = sin(2x)$, is the most difficult one because this function has four peaks. Table 6 shows that MICA-CM provides an excellent approximation of the real function. The logistic map gives the best results, closely followed by the tent and sinusoidal maps as shown in Table 6. Table 7 again impressively shows the superiority of our MICA-CM algorithm compared to all other methods.

## *5.2 Classification problems*

We evaluated our MICA-CM algorithm on five benchmark datasets, listed in Table 8, from the UCI machine learning repository (Asuncion and Newman, 2007). These datasets were deliberately selected with different features so as to challenge the algorithms. Of the selected datasets, Iris is the simplest and Pima the most challenging one.

As is standard for evaluation of classification algorithms, $k$-fold cross-validation was used. That is, the dataset is randomly divided into $k$ equally sized partitions, one partition used as test data and the remaining $k-1$ partitions as training data, with the process repeated $k$ times so that each partition is used once for testing. Average classification accuracy over the test data is used as performance measure.

**Table 8** Classification datasets.

| dataset | data | samples | features | classes |
|---|---|---|---|---|
| Iris | real | 150 | 4 | 3 |
| Wine | integer, real | 178 | 13 | 3 |
| Ionosphere | integer, real | 351 | 34 | 2 |
| WBC | integer | 699 | 9 | 2 |
| Pima | integer, real | 768 | 8 | 2 |

**Table 9** Results for chaotic maps on *Iris* dataset.

| chaotic map | accuracy | std.dev. |
|---|---|---|
| circle map | 99.33 | 2.11 |
| logistic map | 99.33 | 2.11 |
| tent map | **99.93** | **0.23** |
| sinusoidal map | 98.67 | 2.81 |
| ICMIC map | 98.67 | 2.81 |

**Table 10** Comparison of algorithms on *Iris* dataset.

| algorithm | accuracy |
|---|---|
| GDBP | 65.32 |
| ICA | 96.67 |
| GA | 89.33 |
| PSO | 37.33 |
| ACO | 32.66 |
| ES | 46.66 |
| PBIL | 86.66 |
| BBO | 90.00 |
| GWO | 91.33 |
| IOPSO-BPA | 81.99 |
| ISO-FLANN | 99.03 |
| BMOANN | 96.12 |
| SGHS | 98.00 |
| ICA-GA | 97.78 |
| R-ICA-GA | 93.33 |
| FNNSSO | 91.33 |
| MICA-CM | **99.93** |

In our experiments, the parameter settings of the ICA algorithm were as follows: the number of countries was set to 20, the number of imperialists to 2, the revolution rate to 0.1, and the maximum number of iterations to 100.

For comparison, we again compared our algorithm to ICA, GDBP, GA, PSO, ACO, ES, PBIL, BBO, GWO, as well as IOPSO-BPA (Yaghini et al., 2013), ISO-FLANN (Dehuri et al., 2012), BMOANN (Askarzadeh and Rezazadeh, 2013), SGHS (Kulluk et al., 2012), ICA-GA (Khorani et al., 2011), R-ICA-GA (Khorani et al., 2011), FNNSSO (Mirjalili et al., 2015), ACOR (Socha and Blum, 2007), ACOR-BP (Socha and Blum, 2007), and ACOR-LM Socha and Blum (2007). Note that reported results are not available for all algorithms on all datasets.

### *5.2.1 Iris dataset*

This dataset is one of the most widely used in the literature. It contains 3 classes of 50 samples each where each class refers a type of iris plant. One class is linearly separable from the other two, while the latter are not linearly separable from each other.

The structure of the employed ANN for this dataset is 4-9-3. Therefore, every country in MICA-CM has 75

**Table 11** Results for chaotic maps on *Wine* dataset.

| chaotic map | accuracy | std.dev. |
|---|---|---|
| circle map | **100** | **0** |
| logistic map | **100** | **0** |
| tent map | **100** | **0** |
| sinusoidal map | **100** | **0** |
| ICMIC map | 99.44 | 1.76 |

**Table 12** Comparison of algorithms on *Wine* dataset.

| algorithm | accuracy |
|---|---|
| GDBP | 84.32 |
| ICA | 86.96 |
| ISO-FLANN | 96.56 |
| SGHS | 96.63 |
| ICA-GA | 98.15 |
| R-ICA-GA | 98.15 |
| MICA-CM | **100** |

variables to be optimised. The results for the different chaotic maps are given in Table 9. From there we can see that all maps give excellent results with the tent map outperforming the others both in terms of accuracy and robustness (i.e., standard deviation). As Table 10 clearly indicates, MICA-CM provides significantly better performance compared to all other methods.

### *5.2.2 Wine dataset*

This dataset is the result of chemical analysis of three different types of wines grown in the same region in Italy. The study determined the quantities of 13 constituents found in each of the three types of wines. The results for different chaotic maps are presented in Table 11 from where we can see that all maps, except ICMIC, achieve perfect classification. From Table 12 it is apparent that none of the other algorithms matches this perfect performance.

### *5.2.3 Ionosphere dataset*

This dataset was obtained from a phased array of 16 high-frequency antennas. The targets are free electrons in the ionosphere. "Good" radar returns are those showing evidence of some structures in the ionosphere, while "bad" returns are those that do not as their signals pass through the ionosphere.

ANNs of structure 34-69-2 were trained for this dataset. The results for different chaotic maps are given in Table 13 which shows that the logistic map leads to the highest accuracy and lowest standard deviation, while the circle and tent maps yield the same accuracy but with higher standard deviations. The comparison with other methods from the literature in Table 14 confirms our approach to outperform competing methods.

### *5.2.4 WBC dataset*

This dataset, collected by the University of Wisconsin, contains 699 samples, and 9 continuous features that classify a breast tumor as either benign or malignant.

**Table 13** Results for chaotic maps on *Ionosphere* dataset.

| chaotic map | accuracy | std.dev. |
|---|---|---|
| circle map | **98.57** | 3.63 |
| logistic map | **98.57** | 2.02 |
| tent map | **98.57** | 3.63 |
| sinusoidal map | 98.00 | 4.48 |
| ICMIC map | 96.29 | 7.01 |

**Table 14** Comparison of algorithms on *Ionosphere* dataset.

| algorithm | accuracy |
|---|---|
| GDBP | 75.62 |
| ICA | 84.05 |
| ISO-FLANN | 90.38 |
| SGHS | 93.746 |
| MICA-CM | **98.57** |

MLPs with structure 9-19-2 are trained. Table 15 lists the results for different chaotic maps and shows the circle map to perform best. Table 16 compares the results of our MICA-CM algorithm with other methods. As we can see, GWO gives the highest accuracy for this dataset. However, the result of MICA-CM is close to GWO.

**Table 15** Results for chaotic maps on *WBC* dataset.

| chaotic map | accuracy | std.dev. |
|---|---|---|
| circle map | **98.73** | **1.94** |
| logistic map | 95.00 | 3.37 |
| tent map | 94.57 | 2.41 |
| sinusoidal map | 95.57 | 2.07 |
| ICMIC map | 94.13 | 2.47 |

**Table 16** Comparison of algorithms on *WBC* dataset.

| algorithm | accuracy |
|---|---|
| GDBP | 85.61 |
| CA | 94.25 |
| GA | 98.00 |
| PSO | 11.00 |
| ACO | 40.00 |
| ES | 6.00 |
| PBIL | 7.00 |
| BBO | 95.00 |
| GWO | **99.00** |
| IOPSO-BPA | 98.86 |
| BMOANN | 98.84 |
| SGHS | 97.00 |
| ACOR | 95.98 |
| ACOR-BP | 96.55 |
| ACOR-LM | 95.98 |
| MICA-CM | 98.73 |

### 5.2.5 Pima dataset

This dataset investigates whether a patient shows signs of diabetes according to World Health Organisation criteria. It contains of 768 samples, 2 classes, and 8 features; 500 subjects have diabetes symptoms and the remaining 268 do not. This dataset poses a difficult problem because it is comparatively small and is heavily polluted by noise.

MLPs with structure 8-17-2 are trained and therefore the problem has a dimensionality of 188. The results for different chaotic maps are provided in Table 17. Here, the logistic map performs better than the other ones. Table 18 compares the accuracy of MICA-CM against other algorithms and confirms our approach to yield significantly better results compared to all other approaches.

**Table 17** Results for chaotic maps on *Pima* dataset.

| chaotic map | accuracy | std.dev. |
|---|---|---|
| circle map | 80.32 | 5.16 |
| logistic map | **81.25** | **4.76** |
| tent map | 78.76 | 5.26 |
| sinusoidal map | 80.60 | 7.36 |
| ICMIC map | 80.32 | 4.76 |

**Table 18** Comparison of algorithms on *Pima* dataset.

| algorithm | accuracy |
|---|---|
| GDBP | 65.32 |
| ICA | 77.63 |
| IOPSO-BPA | 77.08 |
| ISO-FLANN | 79.63 |
| BMOANN | 77.45 |
| ICA-GA | 76.19 |
| R-ICA-GA | 76.623 |
| ACOR | 75.52 |
| ACOR-BP | 76.04 |
| ACOR-LM | 77.08 |
| MICA-CM | **81.25** |

## 6 Conclusions

Learning is an important aspect in ANNs. Most ANN training algorithms are based on gradient descent methods and thus suffer from a likelihood of getting trapped in local optima. One approach to overcome this problem is to use population-based algorithms.

In this paper, we have presented the Memetic Imperialist Competitive Algorithm with Chaotic Maps (MICA-CM), a novel algorithm for ANN training based on an improved imperialist competitive algorithm coupled with chaotic maps to maintain the diversity of the population. A memetic ICA is used to enhance the probability of reaching the global optimum, while gradient descent back-propagation is employed as a problem-specific local search operator.

Extensive experimental evaluation has conclusively demonstrated the superiority of the proposed algorithm compared to other recent approaches from the literature, confirming MICA-CM to be a good candidate for neural network training for classification and function approximation problems, while selection of a suitable chaotic map depends on the nature of dataset.

In the future, the algorithm can be extended to evolve weights and the structure simultaneously. Also, the chaotic ICA can be applied to train other types of neural networks such as Kohonen self-organising networks, RBF networks, or recurrent neural networks.

## References

A. Askarzadeh and A. Rezazadeh. Artificial neural network training using a new efficient optimization algorithm. *Applied Soft Computing*, 13(2):1206–1213, 2013.

A. Asuncion and D. Newman. UCI machine learning repository, 2007.

E. Atashpaz-Gargari and C. Lucas. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In *IEEE Congress on Evolutionary computation*, pages 4661–4667, 2007.

M. Bashiri. Optimal scheduling of distributed energy resources in a distribution system based on imperialist competitive algorithm considering reliability worth. *Neural Computing and Applications*, 25(3-4):967–974, 2014.

E. Cantú-Paz and C. Kamath. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(5):915–927, 2005.

M. Carvalho and T.B. Ludermir. An analysis of PSO hybrid algorithms for feed-forward neural networks training. In *9th Brazilian Symposium on Neural Networks*, pages 6–11, 2006.

S. Dehuri, R. Roy, and A. Ghosh S.B. Cho. An improved swarm optimized functional link artificial neural network (ISO-FLANN) for classification. *Journal of Systems and Software*, 85(6):1333–1345, 2012.

H. Ebrahimpour-Komleh and S.J. Mousavirad. Cuckoo optimization algorithm for feedforward neural network training. In *21st Iranian Conference on Electrical Engineering*, 2013.

Y. He, Y.Z. Zhou, X.Q. Xiang, H. Chenand, and H. Qin. Comparison of different chaotic maps in particle swarm optimization algorithm for long-term cascaded

hydroelectric system scheduling. *Chaos, Solitons & Fractals*, 42(5):3169–3176, 2009.

V. Khorani, N. Forouzideh, and A.M. Nasrabadi. Artificial neural network weights optimization using ICA, GA, ICA-GA and R-ICA-GA: Comparing performances. In *IEEE Workshop on Hybrid Intelligent Models And Applications*, pages 61–67, 2011.

S. Kulluk, L. Ozbakir, and A. Baykasoglu. Training neural networks with harmony search algorithms for classification problems. *Engineering Applications of Artificial Intelligence*, 25(1):11–19, 2012.

H.K. Lam, U. Econg, H. Liu, B. Xiao, H.A. Araujo, S.H. Ling, and K.Y. Chan. A study of neural-network-based classifiers for material classification. *Neurocomputing*, 144:367–377, 2014.

M. Mahmoudi, F. Taghiyareh, N. Forouzideh, and C. Lucas. Evolving artificial neural network structure using grammar encoding and colonial competitive algorithm. *Neural Computing and Applications*, 22(1): 1–16, 2013.

M. Mandischer. A comparison of evolution strategies and backpropagation for neural network training. *Neurocomputing*, 42(1-4):87–117, 2002.

S.A. Mirjalili. How effective is the grey wolf optimizer in training multi-layer perceptrons. *Applied Intelligence*, 43(1):150–161, 2015.

S.A. Mirjalili, S.M. Mirjalili, and A. Lewis. Let a biogeography-based optimizer train your multi-layer perceptron. *Information Sciences*, 269:188–209, 2014.

S.Z. Mirjalili, S. Saremi, and S.M. Mirjalili. Designing evolutionary feedforward neural networks using social spider optimization algorithm. *Neural Computing and Applications*, 26(8):1919–1928, 2015.

S.J. Mousavirad, F. Akhlaghian, and K. Mollazade. Classification of rice varieties using optimal color and texture features and BP neural networks. In *7th Iranian Conference on Machine Vision and Image Processing*, pages 1–5, 2011.

R.S. Sexton and R.E. Dorsey. Reliable classification using neural networks: a genetic algorithm and backpropagation comparison. *Decision Support Systems*, 30(1):11–22, 2000.

K. Socha and C. Blum. An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing and Applications*, 16(3):235–247, 2007.

S. Talatahari, B.F. Azar, R. Sheikholeslami, and A.H. Gandomi. Imperialist competitive algorithm combined with chaos for global optimization. *Communications in Nonlinear Science and Numerical Simulation*, 17 (3):1312–1319, 2012.

M. Tavazoei and M. Haeri. Comparison of different one-dimensional maps as chaotic search pattern in chaos optimization algorithms. *Applied Mathematics and Computation*, 187(2):1076–1085, 2007.

A.S.I. Wdaa. *Differential evolution for neural networks learning enhancement.* PhD thesis, Universiti Teknologi Malaysia, 2008.

D.H. Wolpert and G.W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

M. Yaghini, M. Khoshraftar, and M. Fallahi. A hybrid algorithm for artificial neural network training. *Engineering Applications of Artificial Intelligence*, 26 (1):293–301, 2013.

X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.