Evolutionary Neural Networks with Mixed-Integer Hybrid Differential Evolution

Yung-Chin Lin
Department of Electrical Engineering, National Yunlin University of Science & Technology, Yunlin County, Taiwan
Email: yclin@wu.edu.tw

Yung-Chien Lin
Department of Electrical Engineering, WuFeng University, Chiayi County, Taiwan
Email: chien-lin@wu.edu.tw

Wen-Cheng Chang
Department of Electrical Engineering, WuFeng University, Chiayi County, Taiwan
Email: wcchang@wu.edu.tw

Kuo-Lan Su
Department of Electrical Engineering, National Yunlin University of Science & Technology, Yunlin County, Taiwan
Email: sukl@yuntech.edu.tw

Abstract—A novel application to the optimization of neural networks is presented in this paper. Here, the weight and architecture optimization of neural networks can be formulated as a mixed-integer optimization problem. And then a mixed-integer evolutionary algorithm (Mixed-Integer Hybrid Differential Evolution, MIHDE) is used to optimize the neural network. Finally, the optimized neural network is applied to the prediction of chaotic time series. The satisfactory results are achieved, and demonstrate that the neural network optimized by MIHDE can effectively predict the chaotic time series.

Index Terms—neural networks, mixed-integer optimization, evolutionary algorithm

I. INTRODUCTION

Neural networks are computing systems made up of a number of interconnected processing nodes. Their applications are motivated by the capabilities of learning, generalization, and fault tolerance. Therefore, neural networks are well suited for a wide variety of engineering applications [1-5]. However, the performance of neural networks highly depends on the architecture of the networks and their weight parameters. Apart from training time, we know that a neural network with excessive number of nodes may cause overfitting, i.e., it simply memorizes the training data, but may produce worse generalization for unknown validation data. On the contrary, the neural network with too few nodes will not produce an efficient mapping result for the training data. Therefore, in addition to weight optimization, the architecture optimization is also crucial in a neural network design. In general, a neural network should be designed as small as possible, especially for the applications that require on-line adaptation.

In the last decade, an active research field has developed dealing with the neural network’s design, which aims at simultaneously optimize the neural network’s architecture and its weight parameters. Early, the neural network’s design was achieved by the handcrafted work. It highly depends on the expert experience and a trial-and-error process. There is no automatic method to design the neural networks. Later, some methods based on constructive (growing) and destructive (pruning) algorithms were developed toward the automatic design of network architectures. These methods automatically add or remove nodes, layers and connections during the training, but such approaches only explore a small subspace of all possible network architectures, and the result depends on the initial network architectures. Therefore, as stated by Angeline et al. [6], "such structural hill climbing methods are susceptible to becoming trapped at structural local optima."

In recent years, evolutionary algorithms (EAs) [7, 8] have been applied to the optimization of neural networks [6, 9-14]. EAs are powerful search algorithms based on the mechanism of natural selection. Unlike conventional search algorithms, they simultaneously consider many points in the search space so as to increase the chance of global convergence. Many considerable efforts at obtaining optimal neural networks based on EAs were reported in the literature and comprehensively reviewed by Yao [10]. However, due to over-emphasizing biological evolution behaviors of neural networks, these EA-based methods are frequently needed to tailor...
exclusive genetic operators, or to modify or remove original genetic operators to avoid the production of illegal or irregular network topologies.

We know that the weight optimization problem of neural networks belongs to the class of real-valued optimization problems. Similar to the weight optimization problem, the architecture optimization problem of neural networks can be formulated as an integer optimization problem since the architecture parameters can be expressed as integer-valued variables. Consequently, the simultaneous architecture and weight optimization of neural networks can be formulated as a mixed-integer optimization problem. In this paper, an evolutionary algorithm, called mixed-integer hybrid differential evolution (MIHDE) [15], is used to perform the optimization process. Finally, the optimized neural network is applied to the prediction of chaotic time series. The satisfactory results are achieved, and demonstrate that the neural network optimized by MIHDE can effectively predict the chaotic time series.

II. EVOLUTIONARY OPTIMIZATION OF NEURAL NETWORKS

In this paper, a mixed-integer encoding scheme is used to represent the optimization parameters of neural networks, including the number of hidden layers, the number of nodes in each hidden layer, the types of node transfer functions and the connection weights. Except for the connection weights, the other optimization parameters are defined as architecture parameters. The connection weights are encoded as real-valued variables, and the other architecture parameters are encoded as integer-valued variables.

Without loss of generality, we will restrict our discussions within a multilayer feedforward neural network. A typical multilayer feedforward neural network shown in Figure 1 is assumed to have \( n_0 \) input nodes, \( L \) hidden layers with \( n_i \) hidden nodes in the \( i \)th hidden layer and \( n_{L+1} \) output nodes.

![Figure 1. A multilayer feedforward neural network.](image)

In this paper, four types of transfer functions are used in the optimal design of neural networks. These transfer functions include identity function, unipolar sigmoid function, bipolar sigmoid function, and radial basis function. Therefore, the \( j \)th node in the \( i \)th layer, \( h_{ij} \), can be described by the following equations.

1) identity function:

\[
h_{ij} = s_{ij} = \begin{cases} 1, & i = 1, 2, \ldots, L + 1; j = 1, 2, \ldots, n_i \\
\end{cases}
\]

where

\[
s_{ij} = \begin{cases} \frac{1}{1 + \exp(-s_{ij})}, & i = 1, 2, \ldots, L + 1; j = 1, 2, \ldots, n_i \\
\end{cases}
\]

2) unipolar sigmoid function:

\[
h_{ij} = \frac{1 - \exp(-s_{ij})}{1 + \exp(-s_{ij})} = \begin{cases} 0, & i = 1, 2, \ldots, L + 1; j = 1, 2, \ldots, n_i \\
\end{cases}
\]

where \( \exp(\cdot) \) is an exponential function and the value of \( h_{ij} \) is in the range \((0, 1)\).

3) bipolar sigmoid function:

\[
h_{ij} = \frac{1 - \exp(-s_{ij})}{1 + \exp(-s_{ij})} = \begin{cases} 1, & i = 1, 2, \ldots, L + 1; j = 1, 2, \ldots, n_i \\
\end{cases}
\]

where the value of \( h_{ij} \) is in the range \((-1, 1)\).

4) radial basis function:

\[
h_{ij} = \exp\left(\sum_{k=0}^{L} \frac{(h_{i-k} - w_{ij,k})^2}{2h_{ij}^2}\right)
\]

\[
W_{ij} = \begin{cases} h_{i-k} - w_{ij,k}, & i = 1, 2, \ldots, L + 1; j = 1, 2, \ldots, n_i \\
\end{cases}
\]

where \( W_{ij} = [w_{ij,1}, w_{ij,2}, \ldots, w_{ij,n_i}] \)

\[
H_{ij} = [h_{i-1,k}, h_{i-2,k}, \ldots, h_{i,n_j}]^T
\]

The value of radial basis function is in the range \((0, 1)\).

To implement the evolutionary optimization process of neural networks, a mixed-integer encoding scheme is developed to represent the genotype chromosome of neural networks. For example, consider a 3-input and 3-output feedforward neural network. Assume that there are at most 2 hidden layers in the neural network and at most 3 hidden nodes in each hidden layer. A genotype chromosome is designed to describe the architecture of the neural network as illustrated in Figure 2. The genotype chromosome consists of three types of genes: layer gene, node genes and weight genes.

The layer gene, in the form of an integer-valued code, denotes the number of active (or existent) hidden layers. In Figure 2, the value of the layer gene is one. It means
that there is only one hidden layer in the neural network. Furthermore, the neural network is assumed to have at most 2 hidden layers, so the first hidden layer is set be deactivated (nonexistent) and the second hidden layer is set to be activated (existent). On the other hand, the output layer is necessarily active at any time.

Since the neural network is assumed to have at most 2 hidden layers, three sets of node genes are created to correspond to two hidden layers and one output layer, respectively. Similarly, each set of node genes is composed of three integer-valued codes because the neural network is assumed to have at most 3 hidden nodes in each hidden layer. The significance of node gene is defined as follows: “0” denoting an inactive (nonexistent) node, “1” denoting an identity function, “2” denoting a unipolar sigmoid function, “3” denoting a bipolar sigmoid function, and “4” denoting a radial basis function.

In Figure 2, the second set of node genes is (3, 0, 2). It means that in the second hidden layer the first and third nodes are activated (existent) and the second node is inactivated (nonexistent), and the active transfer functions of the first and third nodes are bipolar sigmoid function and unipolar sigmoid function. In addition, the third set of node genes is (1, 4, 1). It means that in the output layer two identity functions and one radial basis function are chosen to donate the neural network outputs.

After achieving the mixed-integer encoding, a corresponding phenotype neural network can be obtained by pruning the inactive layers, nodes and redundant connections, as shown in Figure 3.

In order to reduce the number of architecture parameters and simultaneously increase the search efficiency, a fixed output node transfer function is used in this paper. The identity function is selected since the output can be easily obtained by summing up the weighted preceding nodes. Finally, the weight genes, in the form of real-valued nodes, are used to denote the weight and bias parameters of each corresponding node. Therefore, the mixed-integer encoding scheme, which combines the layer gene, the node genes and the weight genes, can be used to guide the evolutionary optimization of the neural networks.

To evaluate the neural network, the objective function may be defined according to the performance requirements. The error function on the training data is chosen as the objective function to assess the performance. For simplicity, the mean squared error function (MSE) is used and stated by the following equation:

$$E = \frac{1}{2mn} \sum_{m}^{n} \sum_{j}^{L} (d_j(k) - o_j(k))^2$$  \hspace{1cm} (10)

where $o_j$ is an output of the neural network, $d_j$ is the corresponding desired output, and $m$ is the number of training data patterns.

The mixed-integer hybrid differential evolution (MIHDE) is evolutionary algorithm. It was successfully applied to many complex mixed-integer optimization problems [15-19]. In this paper, the MIHDE algorithm is used to conduct the evolutionary training process.

### III. MIXED-INTEGER HYBRID DIFFERENTIAL EVOLUTION

First, let us consider a mixed-integer optimization problem as follows:

$$\min f(x, y)$$  \hspace{1cm} (11)

$$x^L \leq x \leq x^U$$

$$y^L \leq y \leq y^U$$

where $x$ represents an $n_x$-dimensional vector of real-valued variables, $y$ is an $n_y$-dimensional vector of integer-valued variables, and $(x^L, y^L)$ and $(x^U, y^U)$ are the lower and upper bounds of the corresponding decision vectors.

The procedure of MIHDE is shown in Table 1. And the main operations of MIHDE are stated as follows.

<table>
<thead>
<tr>
<th>Table 1. Procedure of the MIHDE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed-Integer Hybrid Differential Evolution</td>
</tr>
<tr>
<td>1. Representation and Initialization.</td>
</tr>
<tr>
<td>2. Mutation.</td>
</tr>
<tr>
<td>3. Crossover.</td>
</tr>
<tr>
<td>4. Evaluation and Selection.</td>
</tr>
<tr>
<td>5. Migration if necessary.</td>
</tr>
<tr>
<td>6. Repeat step 2 to 5 until stop criterion is satisfied.</td>
</tr>
</tbody>
</table>

1) Representation and Initialization

The MIHDE uses $N_p$ decision vectors $\{x^i, y^i\}, i = 1, ..., N_p$ to denote a population of
individuals in the G-th generation. The decision vector (chromosome), \((x, y)_i\), is represented as \((x_{1i}, \ldots, x_{pi}, y_{1i}, \ldots, y_{qi})\). The decision variables (genes), \(x_j\) and \(y_j\), are directly coded as real-valued and integer-valued numbers. The initialization process generates \(N_p\) decision vectors \((x, y)_i\), randomly, and should try to cover the entire search space uniformly as in the form:

\[ (x^k, y^k)_i = (x^k, y^k) + \rho_i \left( (x^U - x^L) - (x^L - y^L) \right) \]

where \( \rho_i = \text{Diag}(\rho_{i1}, \rho_{i2}, \ldots, \rho_{in}) \) is a diagonal matrix, the diagonal elements \((\rho_{i1}, \rho_{i2}, \ldots, \rho_{in})\) are random numbers in the range \([0,1]\), the other elements are zero, and the rounding operator \(\left\lfloor a, b \right\rfloor = \rho_i (x^U - x^L), b = \rho_i (y^U - y^L)\) in (12) is defined as \((a, \text{INT}\{b\})\) in which the operator \(\text{INT}\{b\}\) is expressed as the nearest integer-valued vector to the real-valued vector \(b\).

2) Mutation

The i-th mutant individual \((u^G, v^G)_i\) is obtained by the difference for two random individuals as expressed in the form

\[ (u^G, v^G)_i = (x^G, y^G)_i + \rho_n \left( (x^G, y^G)_i - (x^L, y^L)_i\right) \]

where random indices \(k, l \in 1, \ldots, N_p\) are mutually different. The operator \(\text{INT}\{b\} = \rho_n (y^G - y^L)_i\) in (13) is to find the nearest integer vector to the real vector \(b\). The mutation factor \(\rho_n\) is a random real number between zero and one. This factor is used to control the search step among the direction of the differential variation \((x^G, y^G)_i = (x^G, y^G)_i\).

3) Crossover

In crossover operation, each gene of the i-th individual is reproduced from the mutant vector \((u^G, v^G)_i\) and the current individual \((x^G, y^G)_i\) as follows:

\[ u^G_{i}^{G+1} = \begin{cases} x^G_{i}, & \text{if a random number} > \rho_c \\ u^G_{i}, & \text{otherwise} \end{cases} \]

\[ v^G_{i}^{G+1} = \begin{cases} y^G_{i}, & \text{if a random number} > \rho_c \\ v^G_{i}, & \text{otherwise} \end{cases} \]

where the crossover factor \(\rho_c \in [0,1]\) is a constant and the value can be specified by the user.

4) Evaluation and Selection

The operation includes two evaluation phases. The first phase is performed to produce the new population in the next generation as (16). The second phase is used to obtain the best individual as (17).

\[
(x^G_{i}^{G+1}, y^G_{i}^{G+1}) = \arg\min \{ f((x^G_{i}^{G+1}, y^G_{i}^{G+1})), f((u^G_{i}^{G+1}, v^G_{i}^{G+1})) \}, i = 1, \ldots, N_p
\]

\[
(x^G_{i}^{G+1}, y^G_{i}^{G+1}) = \arg\min \{ f((x^G_{i}^{G+1}, y^G_{i}^{G+1})), i = 1, \ldots, N_p \}
\]

where \((x^G_{i}^{G+1}, y^G_{i}^{G+1})\) is the best individual with the smallest objective function value.

5) Migration

In order to increase the exploration of the search space, a migration operation is introduced to generate a diversified population. Based on the best individual \((x^G_{i}^{G+1}, y^G_{i}^{G+1})_h = (x^G_{i}^{G+1}, x^G_{i}^{G+1}, \ldots, x^G_{i}^{G+1}, y^G_{i}^{G+1}, y^G_{i}^{G+1}, \ldots, y^G_{i}^{G+1})\), the j-th gene of the i-th individual can be diversified by the following equations:

\[
X^{jG}_{i, \text{ij}} = \begin{cases} x^G_{ij} + \rho_j (x^G_j - x^G_{ij}), & \text{if a random number} < \frac{x^G_{ij} - x^G_j}{x^G_j - x^G_{ij}} \\ x^G_{ij} + \rho_j (x^G_j - x^G_{ij}), & \text{otherwise} \end{cases}
\]

where \(\rho_j\) and \(\rho_2\) are the random numbers in the range \([0,1]\).

The migration operation in MHDE is performed only if a measure for the population diversity is not satisfied, that is when most of individuals have clustered together, the migration has to be actuated to make some improvements. In this study, we propose a measure called the population diversity degree in order to define whether the migration operation should be performed. In order to define the measure, we first introduce the following gene diversity index for each real-valued gene \(x^G_{i, j}\) and for each integer-valued gene \(y^G_{i, j}\),

\[
dx^G_{i, j} = \begin{cases} 0, & \text{if } \frac{x^G_{ij} - x^G_{ij}}{x^G_{ij} - x^G_{ij}} < \varepsilon_2; j = 1, \ldots, n_c, i = 1, \ldots, N_p, i \neq b \\ 1, & \text{otherwise} \end{cases}
\]

\[
dy^G_{i, j} = \begin{cases} 0, & \text{if } y^G_{ij} = y^G_{ij}; j = 1, \ldots, n_c, i = 1, \ldots, N_p, i \neq b \\ 1, & \text{otherwise} \end{cases}
\]

where \(dx^G_{i, j}\) and \(dy^G_{i, j}\) are the gene diversity indices and \(\varepsilon_2 \in [0,1]\) is a tolerance for real-valued gene provided by the user. According to (20) and (21), we assign the j-th gene diversity index for the i-th individual to zero if this gene clusters to the best gene. We now define the population diversity degree \(\eta\) as a ratio of total diversified genes in the population. From (20) and (21) we have the population diversity degree as
\[ \eta = \left[ \sum_{i=1}^{N_p} \sum_{j=1}^{N} w_{ij} d^i_p + \sum_{j=1}^{N} d_j \right] / \left[ (n_c + n_i)(N_p - 1) \right] \] (22)

From equation (20), (21) and (22), the value of \( \eta \) is in the range [0,1]. Consequently, we can set a tolerance for population diversity, \( \varepsilon_1 \in [0,1] \), to assess whether the migration should be actuated. If \( \eta \) is smaller than \( \varepsilon_1 \), then MIHDE performs the migration to generate a new population to escape a local solution. Contrary, if \( \eta \) is not less than \( \varepsilon_1 \), then MIHDE suspends the migration operation to keep a constant search direction to a target solution.

IV. TIME SERIES PREDICTION

A benchmark problem is used to test the performance of the MIHDE algorithm in neural network design. For implementation, the setting parameters used in MIHDE are listed as follows: population size \( N_p = 5 \), crossover factor \( \rho_c = 0.5 \), and two tolerances \( \varepsilon_1 = \varepsilon_2 = 0.1 \). In addition, the neural network is assumed to have at most 2 hidden layers and at most 3 hidden nodes in each hidden layer.

Example: Mackey-Glass Chaotic Time Series Prediction

The prediction of the Mackey-Glass chaotic time series [20] is a benchmark problem to test the ability of a neural network. The chaotic time series is generated by the Mackey-Glass differential delay equation:

\[ \frac{dx(t)}{dt} = \frac{0.02x(t-r)x(t-r-10)}{1 + x^{10}(t-r)} - 0.1x(t) \] (23)

The chaotic behavior of Mackey-Glass time series is determined by the delay parameter \( r \). Large \( r \) can produce a more chaotic dynamic behaviour that is much difficult to predict. In this paper, the value of \( r \) is set to 17 and the initial condition \( x(0) = 1.2 \). The neural network is used to predict \( x(t+6) \) based on \( x(t) \), \( x(t-6) \), \( x(t-12) \) and \( x(t-18) \), where \( t=118 \) to 1117. The first 500 input-output data pairs are used as the training data set for the neural network, while the remaining 500 input-output data pairs are used as the checking data set for testing the evolved neural network. The number of training generations is identical to 3000.

After 3000 training generations, an optimal neural network with one hidden layer is obtained. The hidden layer consists of three hidden nodes: unipolar sigmoid function node, radial basis function node and bipolar sigmoid function node. The training mean squared error of the best result is 0.0221. The training and testing results for the optimal neural network are shown in Figure 4. The prediction error is given in Figure 5. In Figure 4, it can be observed that the training result (\( t=118 \) to 617) is quite good since the prediction output is almost equal to the plant output. The testing result (\( t=618 \) to 1117) is slightly worse than the training result. However, the result can be acceptable since the prediction error is quite small as shown in Figure 5. Figure 6 displays the convergence progress of the training error. From Figure 6, it is important to note that an improved neural network should be obtained if increasing the number of training generations.

![Figure 4](image-url)  
Figure 4. Training and testing results of the neural network.

![Figure 5](image-url)  
Figure 5. Prediction error of the neural network.

![Figure 6](image-url)  
Figure 6. Convergence progress of training error.
V. CONCLUSIONS

In this paper, a mixed-integer encoding scheme is proposed to represent the neural network’s architecture and weight parameters including types of transfer functions, network topology and connection weights. And then a mixed-integer evolutionary algorithm, MIHDE, is used to optimize the weights and architectures of neural networks. Finally, the problem of time series prediction is used to test the capability of the optimized neural network. Computational results demonstrate that the optimized neural network can effectively predict the Mackey-Glass chaotic time series. The successful results show a significant progress in research on the optimization of neural networks.

REFERENCES


Yung-Chin Lin is a PhD student in Department of Electrical Engineering at Yunlin University of Science & Technology, Yunlin County, Taiwan. He received his MS degree in Automatic Control from Feng Chia University, Taichung City, Taiwan. His research interests include control systems, evolutionary algorithms, and fuzzy systems.

Yung-Chien Lin is an Assistant Professor in Department of Electrical Engineering at WuFeng University, Chia yi County, Taiwan. He received his PhD degree in Electrical Engineering from National Chung Cheng University, Chia yi County, Taiwan. His research interests include evolutionary algorithms, neural networks, and fuzzy systems.

Wen-Cheng Chang is an instructor in Department of Electrical Engineering at WuFeng University, Chia yi County, Taiwan. He received his MS degree in Electrical Engineering from Taiwan University of Science & Technology, Taipei City, Taiwan. His research interests focus on fuzzy control and neural networks.

Kuo-Lan Su is an Assistant Professor in Department of Electrical Engineering at Yunlin University of Science & Technology, Yunlin County, Taiwan. He received his PhD degree in Electrical Engineering from National Chung Cheng University, Chia yi County, Taiwan. His research interests include robotic systems and intelligent control.