

# A novel optimized GA–Elman neural network algorithm

Weikuan Jia<sup>1,3</sup> · Dean Zhao<sup>2</sup> · Yuanjie Zheng<sup>1,4</sup> · Sujuan Hou<sup>1</sup>

Received: 26 September 2016 / Accepted: 15 June 2017 / Published online: 8 July 2017  
© The Natural Computing Applications Forum 2017

**Abstract** The Elman neural network has good dynamic properties and strong global stability, being most widely used to deal with nonlinear, dynamic, and complex data. However, as an optimization of the backpropagation (BP) neural network, the Elman model inevitably inherits some of its inherent deficiencies, influencing the recognition precision and operating efficiency. Many improvements have been proposed to resolve these problems, but it has proved difficult to balance the many relevant features such as storage space, algorithm efficiency, recognition precision, etc. Also, it is difficult to obtain a permanent solution from a temporary solution simultaneously. To address this, a genetic algorithm (GA) can be introduced into the Elman algorithm to optimize the connection weights and thresholds, which can prevent the neural network from becoming trapped in local minima and improve the training speed and success rate. The structure of the hidden layer can also be optimized using the GA, which can solve the difficult problem of determining the number of neurons. Most previous studies on such evolutionary Elman algorithms optimized the connection weights or network structure

individually, which represents a slight deficiency. We propose herein a novel optimized GA–Elman neural network algorithm where the connection weights are real-encoded, while the neurons of the hidden layer also adopt real-coding but with the addition of binary control genes. In this new algorithm, the connection weights and the number of hidden neurons are optimized using hybrid encoding and evolution simultaneously, greatly improving the performance of the resulting novel GA–Elman algorithm. The results of three experiments show that this new GA–Elman model is superior to the traditional model in terms of all calculated indexes.

**Keywords** Elman neural network · Genetic algorithm · Simultaneous evolution · GA–Elman optimized algorithm

## 1 Introduction

The recurrent Elman neural network, proposed by J.L. Elman in 1990, is a typical feedback-type network model, being an optimization of the backpropagation (BP) neural network. Elman proposed the addition of a context layer as a delay operator to the hidden layer of a BP neural network, achieving a memory function and enabling the model to adapt to a time-varying system, as well as offering strong global stability. The Elman model has been widely applied in many application fields for more than 20 years [1–3], greatly assisting with the solution of complex problems. However, the gradually expanding application scope of the method has revealed the deficiencies of this model. Indeed, the Elman network seems imperfect in terms of the properties of the sample and the structure of the neural network. The Elman model is considered to represent an optimization of the BP structure, enjoying its advantages but inevitably also inheriting some of its inherent deficiencies

✉ Weikuan Jia  
wkjia@sdu.edu.cn

<sup>1</sup> School of Information Science and Engineering, Shandong Normal University, Jinan 250358, China

<sup>2</sup> School of Electrical and Information Engineering, Jiangsu University, Zhenjiang 212013, China

<sup>3</sup> School of Agricultural Engineering, Shandong University of Technology, Zibo 255000, China

<sup>4</sup> Key Lab of Intelligent Computing & Information Security in Universities of Shandong, Institute of Life Sciences, Shandong Provincial Key Laboratory for Distributed Computer Software Novel Technology, and Key Lab of Intelligent Information Processing, at Shandong Normal University, Jinan 250014, China

[4], e.g., becoming easily trapped in local minima (which can cause overfitting of the neural network, and even training failure), a fixing learning rate (which limits the convergence speed and requires lots of training time), difficult determination of the number of hidden neurons (requiring lots of time by trial and error), etc., which can influence the operating efficiency and recognition precision of the Elman model.

All of the above problems have attracted attention from scholars, and various improved algorithms have been proposed, including use of a fuzzy strategy to adjust the learning rate [5], use of a normalized risk-aversion error criterion to avoid local minima [6], optimization of the hidden neurons using a genetic algorithm [7, 8], etc. These improvements can improve the performance to some degree, but it has proved difficult to balance the many relevant features, such as storage space, algorithm efficiency, and recognition precision, and some of these improved algorithms have a “stopgap” flavor, as they do not achieve ideal results and still require further improvement. The genetic algorithm (GA) [9, 10] has been introduced and used to optimize the Elman algorithm, being an evolutionary computation method that does not require problem-specific knowledge and being superior in terms of robustness and solution of nonlinear, parallel, and complex problems.

In this approach, the genetic algorithm is used to optimize the connection weights or structure of the Elman neural network [11, 12]. Such optimization of the connection weights and thresholds can improve the learning efficiency and avoid trapping in local minima. The number of hidden neurons can also be optimized, addressing the issue that the structure of the hidden layer is difficult to determine and reducing the time required to construct the network. In recent years, genetic algorithms have been used to optimize Elman neural networks in many fields, attracting great attention from many scholars with substantial progress being made [13–15]. Rohitash proposed an evolutionary Elman network model for prediction of chaotic time sequences [16]. Nate used an evolving Elman neural network to study decision-making [17]. Ding proposed an evolutionary Elman classification algorithm with improved network performance through use of optimized weights [18]. However, most past studies of such evolutionary Elman neural network algorithms have optimized the connection weights or network structure individually, achieving good results in terms of improved network performance but being slightly deficient overall.

In this study, a novel strategy based on combined optimization of the connection weights and structure of the Elman neural network is proposed, i.e., based on hybrid encoding and evolution simultaneously. The connection weights are real-coded with direct expressions to overcome the disadvantages of binary code, whereas the number of

neurons in the hidden layer also adopts real-coding but with the addition of binary control genes. Genetic algorithms and neural networks are important bionic intelligent computing methods. Here, the GA is applied to optimize the connection weights and structure of the Elman neural network simultaneously, leading to a novel optimized GA–Elman algorithm. To verify its performance, three experiments were carried out: experiment I using a real agricultural pest forecasting dataset, to test the operating efficiency of the algorithm; experiment II using a waveform dataset, to test its overall performance; experiment III using an iris dataset, to test its generalizability. The datasets for experiments II and III are from the University of California Irvine (UCI) standard set. Comparison of the results of each experiment confirms the superiority of the proposed optimized GA–Elman algorithm.

The rest of this paper is organized as follows: In Sect. 2, a brief introduction to the Elman neural network and the genetic algorithm is given; Sect. 3 first describes the concept of the proposed GA–Elman algorithm, then the construction of the individual optimization algorithms; Sect. 4 describes the construction of the simultaneous optimization algorithm, and details the steps of the novel proposed algorithm; Sect. 5 presents the three experiments used to test the effectiveness of each algorithm; Sect. 6 discusses the results and presents the conclusions of this work.

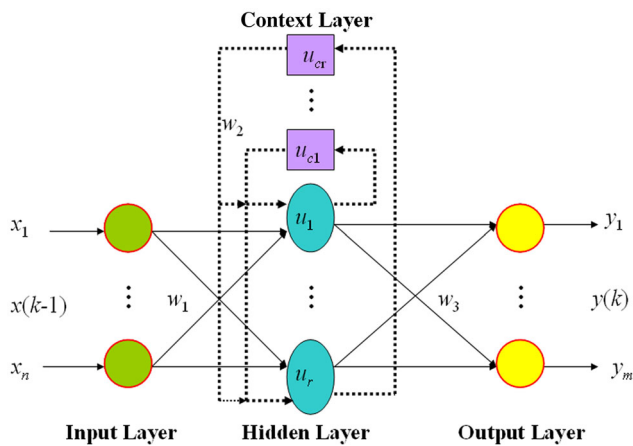
## 2 Elman model and genetic algorithm

### 2.1 Elman algorithm

In the recurrent Elman neural network, one type of feedback neural network, the context layer based on the hidden layer of the BP model is added; it can be regarded as a delay operator and introduces a memory function. It enables the network to adapt to dynamic, time-varying characteristics and ensures global stability. As shown in Fig. 1, the structure of the Elman network model can be described as follows:

The topological structure of the Elman network model usually contains four layers: An input layer, whose neurons are generally linear, passes the signal to the hidden layer, where the signal is translated or dilated via an activation function. Next is the context layer, which can remember the previous moment values of the hidden layer’s output; it can be regarded as a one-step delay operator and thus provides a feedback function. Finally, the output layer outputs the results.

The Elman network is based on the structure of the BP network model, but the delay and storage function of the context layer join the output of the hidden layer automatically to its input. How this joining occurs is sensitive to the historical data of the neural network itself, so this



**Fig. 1** Topological structure of Elman network model

internal feedback mechanism can improve the ability of the network to deal with dynamic information. Based on such mapping of the dynamics to a stored internal state, the system has the ability to adapt to time-varying characteristics.

Supposing that the system has  $n$  inputs and  $m$  outputs, the hidden layer has  $r$  neurons, the same number as in the context layer. The connection weight from the input to hidden layer is denoted as  $w_1$ , that from the context to hidden layer is denoted as  $w_2$ , and that from the hidden to output layer is denoted as  $w_3$ . The input of the network system is  $x(k - 1)$ , the output of the hidden layer is expressed as  $u(k)$ ,  $u_c(k)$  represents the output of the context layer, and  $y(k)$  represents the output of the network system.

Therefore,

$$u(k) = f(w_2 u_c(k) + w_1(x(k - 1))), \tag{1}$$

where

$$u_c(k) = u(k - 1), \tag{2}$$

$f$  represents the transfer function of the hidden layer, commonly an S-type function,

$$f(x) = (1 + e^{-x})^{-1}, \tag{3}$$

and  $g$  represents the transfer function of the output layer, often a linear function,

$$y(k) = g(w_3 u(k)). \tag{4}$$

In the Elman neural network, the weights of the BP algorithm are revised so that the network errors are as follows:

$$E = \sum_{k=1}^m (t_k - y_k)^2, \tag{5}$$

where  $t_k$  is the target output vector.

## 2.2 Genetic algorithm

The GA is a search algorithm, with highly parallel, randomized, and self-adaptive character, developed by reference to natural selection and the mechanism of evolution. It uses group search techniques based on a population that represents a set of problem solutions. A series of genetic manipulations, such as selection, crossover, mutation, etc., are applied to the current population, thus generating a next-generation population and gradually evolving the population to a state that includes the approximate optimal solution. The genetic algorithm starts from a population which may include the problem’s potential solution. The population is made up of a certain number of individuals expressed using gene encoding, where each individual is a chromosome representing a single entity.

The process of the genetic algorithm is depicted in Fig. 2, where Gen is the genetic operator,  $P_c$  is the selection probability,  $P_e$  is the crossover probability, and  $P_m$  is the mutation rate.

The basic steps of the genetic algorithm are as follows:

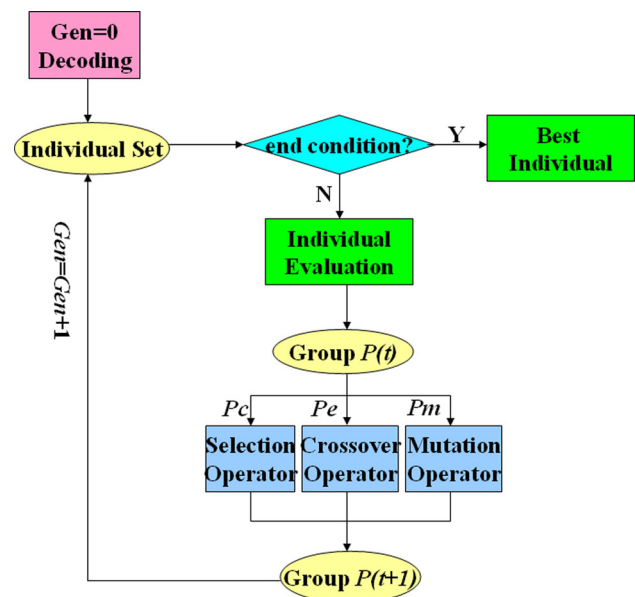
*Step 1* Create an initial population as a random string

*Step 2* Calculate the fitness of each individual

*Step 3* Using genetic probabilities, use the duplication, exchange, and mutation operations to generate a new population

*Step 4* Repeat steps 2 and 3, until given termination conditions are satisfied, and select the best individuals from the genetic algorithm population

This is an iterative process. In each iteration, a set of candidate solutions are retained, then sorted according to their merit. Some solutions are then selected based on



**Fig. 2** Flowchart of genetic algorithm

certain indicators. The genetic operators are used to carry out operations, then a new generation of candidate solutions is generated. This process is iterated until given convergence indexes are met.

### 3 GA–Elman optimized algorithm

Genetic algorithms can be applied to search nondifferentiable, multimode, and large problem spaces, not requiring information on the gradient of the error function, which is a unique advantage if such information is difficult to obtain. When the GA is used to optimize the connection weights of the neural network, one should insert a penalty term into the error function, which does not depend on whether it is differentiable. This can reduce the complexity of the neural network and improve its generalizability, having great potential for optimization of the connection weights of the Elman neural network. In terms of optimizing the network structure, the GA can explore different optimized topologies, addressing the problem of the difficult determination of the number of hidden neurons.

#### 3.1 Construction of the GA–Elman algorithm

The Elman neural network algorithm optimized using the GA is denoted as the GA–Elman algorithm. The main problem in constructing such a genetically optimized algorithm is to determine the encoding method and design the genetic operators. Different optimization problems require different encoding schemes and genetic operators based on understanding of the considered problem, all of them being key to successful application of the genetic algorithm. The main steps of the GA–Elman algorithm are chromosome encoding, fitness function definition, and genetic operator construction. The corresponding flowchart is shown in Fig. 3.

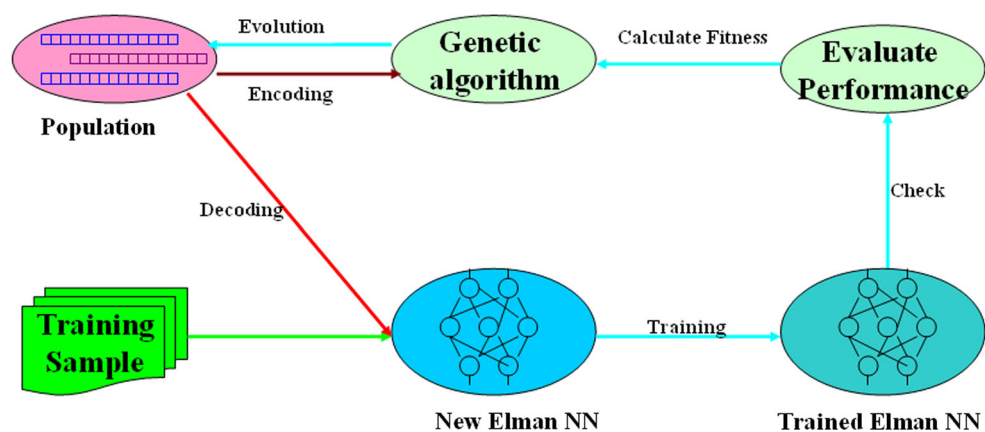
The significance of this optimized GA–Elman algorithm is that, because of its optimized connection weights and thresholds, the network can offer improved training convergence, reduced runtime, and improved operating efficiency, while use of the optimized number of hidden neurons determines the optimal structure of the network model and reduces the time required for trials to construct the network architecture. For the Elman neural network, the hidden layer is determined, and the context is also determined. Such use of the GA to optimize the Elman neural network can be viewed as resulting in a self-adaptive system with automatically adjusted connection weights and structure, without the requirement for human intervention. Finally, the new algorithm also realizes organic integration.

In this study, the connection weights and structure (number of hidden neurons) are first optimized separately using two kinds of individual optimization algorithm; then, on the basis of these individual optimization algorithms, the novel optimized GA–Elman algorithm is established, in which the connection weights and structure are optimized simultaneously. The optimization approach of this novel GA–Elman algorithm therefore adopts hybrid encoding and evolution simultaneously.

#### 3.2 Individual optimization of connection weights using genetic algorithm

In the traditional Elman network model, learning of connection weights is commonly based on the gradient descent method, which can easily fall into local minima, thus leading to training failure or overfitting. Using the genetic algorithm to optimize the connection weights is an adaptive global training method; namely, the genetic algorithm replaces the traditional learning algorithm to overcome this deficiency.

**Fig. 3** Flowchart of GA–Elman algorithm



### 3.2.1 Basic steps

In this section, GA is only used to optimize the connection weights of the Elman neural network, using the following basic steps:

*Step 1* Randomly generate an initial population  $P(t)$ , the upper limit of individual is  $Ps$ , in which each individual corresponds to a weight distribution, using a real-coded scheme to encode each weight and threshold, and construct strips of encoded chain;

*Step 2* Use appropriate decoding methods to decode the encoding of step 1, where each encoded chain corresponds to an Elman neural network with specific connection weights and thresholds;

*Step 3* Use the training data to train the population of Elman neural networks obtained in step 2, and calculate the error function to determine the fitness of each individual (the bigger the error, the lower the fitness);

*Step 4* Select some individuals with the highest fitness values to retain in the next generation directly;

*Step 5* Use crossover operations, according to the roulette method, to select two individuals from  $P(t)$  to cross, obtaining two new individuals to add to the next generation  $P'(t)$ ,  $Crossover(P(t)) = P'(t)$ , according to the crossover probability  $P_c$ ;

*Step 6* Carry out the mutation operation,  $Mutate(P'(t)) = P''(t)$ , according to the mutation probability  $P_m$ , and change the value at a random location (insert a random number);

*Step 7* Calculate the fitness of each individual in the population  $P''(t)$  according to the error function;

*Step 8* Set  $t = t + 1$ . If  $t > Ps$ , go onto step 9; otherwise return to step 4;

*Step 9* Set  $G = G + 1$ . If  $G = G_{max}$ , the algorithm terminates; otherwise return to step 3.

### 3.2.2 Set parameters

The main aspects of the optimization algorithm are as follows:

**3.2.2.1 Encoding scheme** In the GA–Elman design process, the greatest challenge is how to design a proper encoding scheme for effective presentation of the network connection weights. We assume herein an Elman neural network with  $n$  inputs,  $m$  outputs, and  $r$  hidden neurons. The connection weights are encoded by real-coding, with each value represented by a real number; this has the advantage of perceptual intuition and overcomes the disadvantages of binary encoding. This encoding scheme, for the conditions of the given network structure, has encoding length of

$$L = n \times r + r \times m + r + m = r \times (n + m + 1) + m. \quad (6)$$

**3.2.2.2 Definition of the fitness function** The GA is used to optimize the connection weights of the Elman network model; once its structure has been determined, generally speaking, the greater the error, the lower its fitness. In this paper, the training error is adopted to determine the chromosome fitness, i.e., corresponding to each neural network. The fitness function is defined as

$$F = C - e, \quad (7)$$

where  $C$  is a constant and  $e$  is the network training error. Generally, for each individual, the greater the error, the lower its fitness.

**3.2.2.3 Genetic operators** Three different kinds of genetic operator, viz. selection, crossover, and mutation, were set as follows:

When applying the selection operator based on the “roulette” selection method, each individual enters the next generation with probability equal to its fitness divided by the fitness of the entire population. Therefore, the higher the fitness value, the greater the probability that an individual will go into the next generation. This maintains the diversity of the population while applying the condition of “survival of the fittest.”

The crossover operator applies the single-point crossover method, according to which part of the parental genes are exchanged or restructured according to a given probability.

For the mutation operator, real-encoded Gaussian mutation is adopted.

The results of the genetic algorithm and Elman algorithm are highly sensitive to the parameters used during the training process. The result depends on the initial status of the network model. The Elman algorithm is very effective for local search, and the genetic algorithm for global search. Thus, the connection weights evolve as follows:

(1) The genetic algorithm is used to optimize the distribution of initial connection weights, identifying an appropriate search space within the entire solution space.

(2) In this small solution space, the Elman algorithm is used to search for a relatively optimal solution, then iterated.

Generally, the operating efficiency of such a mixed training method will be better than when using either the genetic algorithm or Elman training method alone.

## 3.3 Individual optimization of the structure using genetic algorithm

A simple neural network structure for a given problem has few connections and neurons, limiting its processing ability and making it unsuitable for processing of complex non-linear data. However, inclusion of too many connections

and neurons will result in noise during training, limiting the generalizability of the network. In previous engineering applications, a trial-and-error method was generally applied to design the neural network structure, relying too heavily on subjective experience and requiring a lot of time. Many other methods can be applied, including the add-structure method, cut-structure method, etc., but they still cannot improve significantly beyond the “blind climbing” approach.

The topological structure of the Elman neural network includes connection routes and node transition functions. Use of a good topology can solve a problem successfully without redundant nodes and connections. The structure of the neural network is designed for each search problem, and the ability for generalization and recognition precision are evaluated based on given criteria to identify the best Elman neural network structure from the architecture space. The structure of the evolutionary Elman neural network mainly depends on the structural encoding and operator design, as the encoding scheme directly affects the operator design. In this case, the structure of the Elman neural network mainly depends on the number of neurons in the hidden layer.

### 3.3.1 Basic steps

The basic steps of the GA used to optimize the structure of the Elman neural network are as follows:

*Step 1* Randomly generate  $N$  structures, each encoded by real-coding, and add a binary control gene;

*Step 2* Use different initial weights to train the  $N$  individuals;

*Step 3* According to the training results and evolution strategy, determine the fitness of each individual;

*Step 4* Select some individuals with the highest fitness values and directly retain them for the next generation;

*Step 5* Apply the crossover and mutation operators on the current population, to generate the next population;

*Step 6* Repeat steps 2–5 until an individual of the current population meets the requirements.

The main difficulty with this algorithm is the encoding of the structure. In practical applications, the number of neurons in the input and output layers is given. Therefore, determining the number of neurons in the hidden layer has become the focus of study. Herein, it is supposed that the Elman network model has one hidden layer, and that the number of input and output neurons is known.

### 3.3.2 Set parameters

The main issues involved in this optimization algorithm are as follows:

**3.3.2.1 Encoding scheme** Based on the complexity of the problem, the original binary coding is obviously not suitable. Herein, real-coding is adopted, and a binary control gene is added to the hidden neurons, generated using a random function. When the value of this control gene is 0, the corresponding neuron of the hidden layer has no effect on the output layer; however, when the value of the control gene is 1, the neuron does have an effect. This encoding approach must determine a maximum number  $r$  of hidden neurons. Supposing that the Elman network model has one hidden layer,  $n$  inputs, and  $m$  outputs, its encoding length is

$$L = r + r \times m + n \times r + r + m = r \times (n + m + 2) + m. \quad (8)$$

In the Elman network model, the context layer corresponds to the hidden layer, so its encoding scheme is the same as for the hidden layer.

**3.3.2.2 Genetic operators** When applying the mutation operator, the real-coded value undergoes “Gaussian” mutation, where the binary coding is subject to bit-flipping mutation.

The remaining parameters are set as in Sect. 3.2.

## 4 Novel GA–Elman algorithm

On the basis of the considerations described above, we attempted to optimize the connection weights and structure simultaneously to construct a novel GA–Elman algorithm. The connection weights of the Elman neural network were evolved on the premise that its structure had been determined. During the optimization algorithm, the design of the structure is critical. A neural network with a good structure can effectively solve the given problem. Herein, an automatic design optimization approach is proposed, based on the idea that, when the structure is completely optimized, the connection weights will also be optimized; namely, the connection weights and structure are optimized simultaneously. This represents the core of the new algorithm.

The proposed evolutionary optimization training process is essentially a hybrid simultaneous training process, whose effect is superior to either the genetic algorithm or Elman training alone. The decision regarding when to switch between the two algorithms is just like designing the topological structure of a network model, depending on the given dataset and parameter values.

### 4.1 Basic steps of optimized algorithm

In this section, the GA is used to optimize the structure and connection weights simultaneously, using the following basic steps:

*Step 1* Encode the structure and connection weights as described above to construct different encoded chains;

*Step 2* Decode the encoded chains from step 1 to obtain  $N$  different neural networks;

*Step 3* Predefine the learning parameters based on the given training samples;

*Step 4* Determine whether there is a neural network that meets the precision requirements. If so, go to step 10; otherwise, go to step 5;

*Step 5* Calculate the fitness of each individual according to the error function and training results;

*Step 6* Sort the individuals according to their fitness value, and select some individuals with the highest fitness values to retain for the next generation directly;

*Step 7* Apply the crossover and mutation operators described above to the current population to add to the next generation;

*Step 8* Determine whether the size of the population has reached the upper limit. If yes, go to step 9; otherwise go to step 5;

*Step 9* Determine whether the number of iterations of the genetic algorithm has reached the preset limit. If yes, go to step 10; otherwise, go to step 4;

*Step 10* Identify the optimal initial Elman neural network and use the preset parameters to continue training;

*Step 11* If the precision meets the requirement in terms of the error function, or the number of iterations of the Elman algorithm has reached the preset limit, the algorithm terminates.

The significance of the proposed algorithm is the optimization of the connection weights and thresholds to improve the convergence performance, such as training speed, and thus decrease the training time and thereby improve the operating efficiency of the neural network. This enables the design of an optimal structure with the optimal number of neurons in the hidden layer, thereby improving its problem-solving ability.

## 4.2 Set parameters

The proposed algorithm is based on the assumption of a single hidden layer, with given number of input and output neurons. Therefore, the evolution of the neural network structure reduces to optimization of the number of neurons.

The encoding problem and genetic operators were addressed in Sects. 3.2 and 3.3.

## 5 Experiments

### 5.1 Experimental design

To verify the performance of the proposed algorithm, three experiments were carried out and their results compared.

Experiment I tested the efficiency of the optimized network model using pest forecasting data from actual agricultural production [19]. Experiment II tested the overall performance of each algorithm using data from the waveform generator of the UCI standard dataset [20]. Experiment III tested the generalizability of the optimized algorithm using data from the iris dataset of the UCI standard dataset [21].

#### 5.1.1 Algorithm notations

1. The traditional Elman neural network is denoted as “Elman”;
2. The algorithm using the GA only to optimize the connection weights is denoted as “GA–Elman W”;
3. The algorithm using the GA only to optimize the structure is denoted as “GA–Elman S”;
4. The algorithm using the GA to optimize the connection weights and structure simultaneously is denoted as “GA–Elman W + S”.

#### 5.1.2 Data preprocessing

Preprocessing to normalize or standardize the original data can eliminate incomparability due to different data index distributions or numerical differences in the source, which can ensure the quality of the data used. Standardized preprocessing was applied here to obtain data with distribution  $N(0, 1)$ , using the formula

$$x'_{ij} = (x_{ij} - \bar{x}_j) / \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}. \quad (9)$$

#### 5.1.3 Parameter setting

The evolutionary algorithms used real coding. The chromosome's length indicates the total number of network models. The parameters were set as follows:

Population size  $S = 30$ ; Crossover probability  $P_c = 0.9$ ; Mutation rate  $P_m = 0.01$ ;

Genetic algorithm iterations: 500 generations; Target error: mean square error (MSE) of  $1 \times 10^{-5}$ ; Elman iterations: 2000;

Elman learning algorithm: traingdx (gradient-descent learning algorithm for adaptive learning rate);

Use elitism strategy (four fittest individuals passed to next generation directly);

Apply linear scaling (first the original fitness is linearly scaled, then the selection operation is applied);

The environment used to run the experiments was equipped with a Duo 2.60-GHz E7300 CPU with 1.99 GB of RAM.

**Table 1** Results of experiment I

Test no.	Elman		GA–Elman W		GA–Elman S		GA–Elman W + S	
	Hidden neurons	Convergence steps	Hidden neurons	Convergence steps	Hidden neurons	Convergence steps	Hidden neurons	Convergence steps
1	2	Failed	2	Failed	5	1138	7	351
2	3	Failed	3	Failed	7	893	6	530
3	4	Failed	4	572	5	862	6	377
4	5	Failed	5	Failed	6	726	5	237
5	6	1762	6	330	8	Failed	3	1915
6	7	Failed	7	355	2	Failed	2	Failed
7	8	Failed	8	252	5	798	8	269
8	9	Failed	9	237	5	949	5	347

### 5.1.4 Evaluation criteria

For experiments II and III, the algorithm was evaluated based on the following five aspects: successful training rate, runtime, sum of squared errors (SSE), convergence steps, and recognition accuracy. The successful training rate is the fraction of successful trainings among a certain number of attempts (50 in this case). The number of convergence steps is the number of steps required for convergence, averaged over 50 training runs. The runtime is the average runtime of each algorithm. The SSE is the difference of the sum of squared error between the recognized and actual value, which can be used to measure the closeness of the forecast and actual value. The recognition accuracy is the number ratio of correctly recognized samples to all testing samples. For given recognition accuracy, smaller sum of squared errors corresponds to higher recognition precision.

## 5.2 Experiment I

The primary purpose of this experiment is to test the operating efficiency of each algorithm and thereby reveal the effect of optimization. The dataset was used to train the four algorithms described above, eight times each.

The data for this experiment came from actual agriculture production, representing agricultural pest forecasting data for wheat midge occurrence based on meteorological factors. These data extend from 1941 to 2000, representing a 60-year sample, with 14 meteorological factors as variables. Therefore, in this experiment, the Elman neural network has 14 inputs, representing the 14 features, and 1 output, representing the wheat midge occurrence degree.

### 5.2.1 Experimental test

For the traditional Elman and GA–Elman W algorithms, the number of hidden neurons was set by trial and error, requiring 2 to 9 attempts with a total of 8. The GA–Elman S algorithm sets an upper limit (supposed to be 15). The GA–Elman W + S algorithm also sets an upper limit, and optimizing the structure optimizes the connection weights simultaneously.

Each algorithm was tested eight times. The results for the four algorithms are presented in Table 1.

### 5.2.2 Experimental results

According to the results in this table, the traditional Elman algorithm was successful in only one of eight experiments, suggesting that this dataset is not very suitable for direct processing using the traditional Elman algorithm. Its fatal weakness is being easily trapped in local minima. Use of the GA with 500 generations was sufficient to take the GA–Elman W algorithm out of local minima, increasing the number of training successes. If the number of hidden neurons is increased, the length of the genetic algorithm encoding also increases, and the complexity of the genetic algorithm rises, so a reasonable number of hidden neurons in this test is six or seven.

The GA–Elman S algorithm obtained the optimal number of neurons. In the subsequent learning of the connection weights, the number of iterations required was relatively larger, and the algorithm was still easily trapped in local minima. Meanwhile, the GA–Elman W + S algorithm, after the GA optimization, soon converged to the target error. These two optimized algorithms suggest that the optimal number is five, six or seven, similar to the number of classical trial-and-error attempts.



**Table 2** Comparison of performance of four algorithms

Network model	Training success rate (%)	Convergence steps	Runtime (s)	SSE	Recognition accuracy (%)
Elman	80.0	1163	8.3	17.6860	78.0
GA–Elman W	94.0	500 + 132	11.7	6.9317	93.0
GA–Elman S	84.0	500 + 379	13.6	11.2566	90.0
GA–Elman W + S	100	500 + 86	6.7	2.2034	99.0

**Table 3** Comparison of each classifier’s accuracy

Neural network model	Simulated and actual value					SSE
	1	0	1	2	0	
Elman	1.2773	0.2576	1.8169	2.2617	0.4368	1.0699
GA–Elman W	0.7233	0.2883	1.3007	2.3178	−0.2649	0.4213
GA–Elman S	0.8437	−0.2571	0.7864	1.8162	0.7677	0.7593
GA–Elman W + S	0.8187	0.1436	1.1027	1.9344	−0.2002	0.1084

According to the results of this experiment, the greatest advantage of the GA–Elman W + S optimized algorithm is that it is adaptive to determine the optimal model.

**5.3 Experiment II**

The major aim of this experiment is to test the overall performance and recognition ability of each algorithm. The experiment was divided into two steps. The first step tested the overall effect, whereas the second step evaluated each algorithm based on the recognition precision for a single sample. The data adopted for this experiment were from the UCI standard dataset; we chose the waveform generator dataset with 5000 samples and 40 features to describe the data characteristic, namely data with 40 inputs and 1 output.

*5.3.1 Overall performance of algorithms*

In this experiment, the first 500 samples were selected as training samples, then 100 samples were selected as simulation samples. The four algorithms were tested, and the simulation results are presented in Table 2.

*5.3.2 Recognition precision of algorithms*

To further illustrate the recognition ability of each algorithm, five samples were selected and the test and actual values compared; the results are presented in Table 3.

*5.3.3 Experimental results*

Tables 2 and 3 show that, based on all six evaluation criteria, the GA–Elman W + S algorithm was superior to the other three algorithms. The traditional Elman algorithm was worst except in terms of runtime. Compared with the GA–Elman S, the GA–Elman W algorithm was slightly better.

**5.4 Experiment III**

The main aim of this experiment was to test the generalizability of each algorithm. The experimental data used were the iris dataset from the UCI standard dataset. This dataset has 150 samples with 4 features and 3 categories.

*5.4.1 Experimental method*

To test generalizability, the experiment data were divided randomly into training and testing sample in ratio of 4:1. The dataset was redivided randomly for each training attempt, and we recorded the number of iterations, recognition precision, runtime, etc. Each algorithm was run 20 times, and the average value of each index was calculated. The results for each algorithm are presented in Table 4. For the genetic algorithm training alone, the number of iterations was 3000.

*5.4.2 Experimental results*

Table 4 shows that, in the case of no trapping in local minima, the convergence speed of the Elman network

**Table 4** Comparison of results of algorithms

Network model	Convergence steps	Recognition accuracy (%)	Runtime (s)	Successful training
Elman	1757	88.83	9.3	14
Genetic algorithm	1869	90.17	16.7	17
GA–Elman W	500 + 865	94.33	12.5	18
GA–Elman S	500 + 1176	91.50	13.3	17
Reference [18] algorithm	500 + 672	96.17	11.2	20
GA–Elman W + S	500 + 429	98.67	8.7	20

model is faster. Compared with the genetic algorithm and other optimization algorithms, although not easily trapped in local minima, several optimization algorithms are better than the simple genetic algorithm in terms of both operating efficiency and recognition accuracy. Compared with the algorithm in Ref. [18], in terms of recognition accuracy and runtime, the GA–Elman W + S algorithm was better. All of these results suggest that the GA–Elman W + S algorithm exhibited greatly improved generalizability.

Note that it is meaningless to compare the number of iterations of the genetic algorithm with the Elman neural network, because they are not based on the same concept, so runtime is listed instead.

### 5.5 Experiment analysis

The results of experiment I show that, if the Elman neural network does not become trapped in local minima, its convergence speed is very fast, but it is easily trapped, resulting in training failure. The Elman neural network inherits this deficiency from the BP algorithm, for which trapping in local minima during training can lead to lots of wasted time. The GA was then used to optimize the connection weights in the GA–Elman W and GA–Elman W + S algorithms, enabling them to escape from trapping in local minima, improving both the operating efficiency and generalizability. Using the genetic algorithm to optimize the structure, the GA–Elman S and GA–Elman W + S algorithms can determine the number of neurons adaptively, reducing the time required compared with the trial-and-error method, and also improving their operating efficiency and generalizability.

The results of experiment II show that, by using the GA to optimize the connection weights of the Elman network model, its operating efficiency and recognition precision can be significantly improved. By optimizing the weights, the neural network is no longer easily trapped in local minima, and its training efficiency and success rate are improved. By optimizing the structure, the number of hidden neurons can be determined adaptively. All of these

effects improve the operating efficiency and generalizability, and the recognition precision is also improved sharply.

The results of experiment III further show that, in terms of operating efficiency and generalizability, the GA–Elman W + S algorithm is best. Experiments I and III show that the Elman neural network is not suitable for some datasets. Although optimization algorithms can improve its generalizability, it is not effective for all datasets.

## 6 Conclusions

Aiming to address various defects of the Elman neural network, the GA was introduced in this work and a novel optimization approach designed based on combined optimization of the hidden neurons and connection weights, with hybrid encoding and evolution simultaneously. The genetic algorithm's global search and Elman neural network's local search abilities resulted in more precise adjustment of the hidden neurons and connection weights, resulting in an optimal Elman neural network algorithm. The results of three experiments show that this simultaneously optimized Elman neural network algorithm performed best in terms of all calculated performance indexes. Note that, in experiments II and III, one cannot simply compare the convergence steps of the Elman versus other optimization algorithms, as their iterations are not based on a common concept. The cost of each genetic algorithm iteration is different from the Elman neural network, so runtime is listed instead.

Although the optimization using the genetic algorithm will consume a certain amount of time during Elman neural network training, the convergence is accelerated and the training success rate and generalizability improved. This can reduce the time required for repeated training attempts based on the trial-and-error method, improving the operating efficiency and recognition precision. According to the characteristics of the considered problem, we plan to investigate use of other intelligent computation methods

[22, 23] to optimize the neural network, to obtain even more superior models. We also expect the proposed algorithm to be applicable in other fields, such as target recognition [24], medical imaging [25], sliding control [26, 27], etc. These topics will be the focus of our further study.

**Acknowledgements** This work is supported by the National Natural Science Foundation of China (nos. 61572300, 31571571, and 61379101), Natural Science Foundation of Shandong Province in China (no. ZR2014FM001) and Taishan Scholar Program of Shandong Province of China (no. TSHW201502038).

#### Compliance with ethical standards

**Conflict of interest** The authors declare that there are no conflicts of interest regarding publication of this paper.

## References

- Arriandiaga A, Portillo E, Sánchez JA et al (2016) A new approach for dynamic modelling of energy consumption in the grinding process using recurrent neural networks. *Neural Comput Appl* 27(6):1577–1592
- Raghu S, Sriraam N, Kumar GP (2017) Classification of epileptic seizures using wavelet packet log energy and norm entropies with recurrent Elman neural network classifier. *Cogn Neurodyn* 11(1):51–56
- Lu JJ, Chen H (2006) Researching development on BP neural networks. *Control Eng China* 13(5):449–451 (in Chinese)
- Zhang HX, Lu J (2010) Creating ensembles of classifiers via fuzzy clustering and deflection. *Fuzzy Sets Syst* 161(13):1790–1802
- Ltaief M, Bezine H, Alimi AM (2016) A spiking neural network model with fuzzy learning rate application for complex handwriting movements generation. *Adv Intell Syst Comput* 552:403–412
- Lo TH, Gui Y, Peng Y (2015) The normalized risk-averting error criterion for avoiding nonglobal local minima in training neural networks. *Neurocomputing* 149:3–12
- Kapanova KG, Dimov I, Sellier JM (2016) A genetic approach to automatic neural network architecture optimization. *Neural Comput Appl*. doi:10.1007/s00521-016-2510-6
- Jia WK, Zhao DA, Ding L (2016) An optimized RBF neural network algorithm based on partial least squares and genetic algorithm for classification of small sample. *Appl Soft Comput* 48:373–384
- Sivaraj R, Ravichandran DT (2011) A review of selection methods in genetic algorithm. *Int J Eng Sci Technol* 3(5):3792–3797
- Qi F, Liu XY, Ma YH (2012) Synthesis of neural tree models by improved breeder genetic programming. *Neural Comput Appl* 21(3):515–521
- Yao X (1999) Evolving artificial neural networks. *Proc IEEE* 87(9):1423–1447
- Ding SF, Li H, Su CY et al (2013) Evolutionary artificial neural networks: a review. *Artif Intell Rev* 39(3):251–260
- Kalinli A (2012) Simulated annealing algorithm-based Elman network for dynamic system identification. *Turk J Electr Eng Comput Sci* 20(4):569–582
- Sheikhan M, Arabi MA, Gharavian D (2015) Structure and weights optimisation of a modified Elman network emotion classifier using hybrid computational intelligence algorithms: a comparative study. *Connect Sci* 27(4):1–18
- Chen H, Zeng Z, Tang H (2015) Landslide deformation prediction based on recurrent neural network. *Neural Process Lett* 41(2):1–10
- Chandra R, Zhang MJ (2012) Cooperative coevolution of Elman recurrent neural networks for chaotic time series prediction. *Neurocomputing* 86(1):116–123
- Nate K, Risto M (2009) Evolving neural networks for strategic decision-making problems. *Neural Netw* 22(3):326–337
- Ding SF, Zhang YN, Chen JR et al (2013) Research on using genetic algorithms to optimize Elman neural networks. *Neural Comput Appl* 23(2):293–297
- Zhang YM (2003) The application of artificial neural network in the forecasting of wheat midge. Master's thesis, Northwest A&F University, Xi'an, China
- <http://www.ics.uci.edu/~mlearn/databases/Waveform> Database Generator (Version 2). Accessed 16 Nov 2014
- <http://archive.ics.uci.edu/ml/datasets/Iris>. Accessed 16 Nov 2014
- Zheng XW, Lu DJ, Wang XG et al (2015) A cooperative coevolutionary biogeography-based optimizer. *Appl Intell* 43(1):1–17
- Azali S, Sheikhan M (2016) Intelligent control of photovoltaic system using BPSO-GSA-optimized neural network and fuzzy-based PID for maximum power point tracking. *Appl Intell* 44(1):1–23
- Hou SJ, Chen L, Tao D et al (2017) Multi-layer multi-view topic model for classifying advertising video. *Pattern Recognit* 68:66–81
- Pareek NK, Patidar V (2016) Medical image protection using genetic algorithm operations. *Soft Comput* 20(2):763–772
- Song XM, Ju HP (2017) Linear optimal estimation for discrete-time measurement delay systems with multichannel multiplicative noise. *IEEE Trans Circuits Syst II Express Briefs* 64(2):156–160
- Ding SH, Li SH (2017) Second-order sliding mode controller design subject to mismatched term. *Automatica* 77:388–392