Hybrid evolutionary approach for designing neural networks for classification

Z.-H. Tan

An approach for the automatic design of artificial neural networks is presented where a hybrid evolutionary algorithm (HEA) is applied to the structural and parametric learning of networks. The HEA combines genetic algorithms and evolutionary programming on the basis of a real-valued multi-matrix representation. Experimental results show that the proposed approach has a good generalisation and a low computational cost.

Introduction: The problem of finding a suitable architecture and the corresponding weights of the network is of central importance in the area of designing artificial neural networks (ANNs) [1]. In general, ANNs are designed by means of trial and error and fixed during the learning process and the parameters are trained by gradient-based algorithms liable to converge to a local minimum. To enable an automatic design of ANNs and perform a global search, evolutionary ANNs (EANNs) have been widely explored in recent years [2]. EANNs are often characterised by two classes of evolutionary computations, namely genetic algorithms (GAs) and evolutionary programming (EP). At present, there is a trend to prefer EP-based EANNs to GAs as GAs are theoretically not as well suited for evolving ANNs [1–4].

The distinguishing feature of GAs is the crossover operator, which often operates on strings called chromosomes or genotypes in the recombination space. The advantages of crossover come from the ability of identifying good sub-strings called building blocks and recombining these from parents into offspring. Each offspring genotype is then evaluated by mapping it into a solution to the task called a phenotype in the evaluation space. An interpretation function is required to map between the two distinct spaces [4]. This mapping may be many-to-one, termed the competing conventions problem [5]. For example, ANNs with hidden neurons defined in different orders may have very dissimilar genotypes even though they are functionally identical. This tends to prevent successful recombination of building blocks. This is the major argument against applying crossover to evolving ANNs [1, 4]. EP, however, applies a natural representation for the problem, allowing a direct manipulation of ANNs so that the problems associated with the dual space are avoided [1].

However, considerable effort and progress has been made in GA-based EANNs although without delving into the aforementioned theoretical considerations. Instead of abandoning crossover or avoiding the theoretical considerations, this Letter introduces a linear combination crossover operating on a real-valued multi-matrix encoding so that the problems previously discussed are overcome. Subsequently, a hybrid evolutionary algorithm (HEA) is proposed to combine the crossover operator in GAs with the mutation operator in EP. This leads to a method of designing ANNs called GAEPNet where connection weights and architecture of ANNs evolve simultaneously.

Real-valued multi-matrix encoding scheme: In the frequently used Miller-matrix encoding scheme [6], a network with N neurons is represented as a Boolean matrix of dimensions $N \times (N+1)$ in which element c_{ij} of the first N columns is zero if there is no link from neurons i to j and one if there is a link, and element $c_{i(N+1)}$ of the last column is zero if there is no bias in neuron i and one if there is a bias. The concatenation of the elements forms the chromosome representing the network architecture. The obvious deficiency of this scheme is the resulting huge genotype with massive redundant genes. In addition, this binary code is suitable for pure architecture evolution only.

Therefore, a real-valued multi-matrix encoding scheme is presented to encode each feedforward ANN with one hidden layer as a genotype encompassing four matrices: in-weight, hidden-bias, out-weight and out-bias, which describe weights from input neurons to hidden neurons, hidden neuron biases, weights from hidden neurons to output neurons and output neuron biases, all in real numbers, respectively. If a neuron is nonexistent, the corresponding element is set as an imaginary number and its weights as zero. This scheme encodes both architecture and weights and eliminates the need for an interpretation function indicating that the dual space problem is avoided. It is also a flexible scheme as the only restriction on the architecture is to set a maximum number of hidden neurons.

Hybrid evolutionary algorithm: Standard GAs are in general slow in fine-tuning a good solution once a promising region has been identified whereas EP, relying on Gaussian mutation, is often better at local search. In addition, EP emphasises the behavioural link between parents and offspring as opposed to the genetic link stressed in GAs. To benefit from both EP and GAs, an HEA is introduced where the proportion of crossover and mutation changes adaptively. In the beginning, crossover is the dominant variation operation whereas they are equal in the end.

A conventional crossover operator generates offspring genes via a combination of the genes in the parents. In this Letter, an arithmetic recombination is introduced. Assuming that $X=\{x_1, \ldots, x_n\}$ and $Y=\{y_1, \ldots, y_n\}$ are two real-valued chromosomes for crossover, the recombination is implemented as follows:

$$z_i = h_i \times x_i + (1 - h_i) \times y_i \tag{1}$$

where h_i is a random variable with a uniform distribution between 0 and 1, and x_i and y_i correspond to biases or weights.

There are two types of mutation: parametric and structural. Structural mutations include the deletion and addition of hidden neurons whereas parametric mutations consist of Gaussian mutation of weights and biases.

Both crossover and mutation operators are followed by a partial training using back-propagation (BP) with a fixed epoch aiming at increasing the behavioural link between the parents and their offspring but avoiding overtraining.

To achieve a good generalisation, known data are divided into two sets namely training set and validation set, both of which are utilised for the evaluation of ANNs. The fitness function is defined as follows:

$$F = \alpha E(\text{training set}) + (1 - \alpha)E(\text{validation set}) \tag{2a}$$

$$E(x) = \beta/MSE(x) + (1 - \beta)/CER(x) \tag{2b}$$

where α and β are scale factors and normally smaller than 0.5, and $MSE(x)$ and $CER(x)$ are the mean squared error and the classification error rate of a single ANN on data set x, respectively.

The selection mechanism is rank based. Given M sorted individuals numbered as $0, 1, \ldots, M-1$, with the 0th being the fittest, the $(M-i)$th individual is selected with a probability of

$$p(M - i) = \frac{\sqrt[4]{i}}{\sum_{j=1}^{M} \sqrt[4]{j}} \tag{3}$$

GAEPNet: On the basis of the encoding scheme and the HEA presented previously, GAEPNet evolves ANNs as follows:

Step 1: Randomly generate an initial population of networks with the number of hidden neurons, neuron biases and connection weights assigned at random. Subsequently, each network is partially trained.

Step 2: Compute the fitness of each network and according to fitness values, rank the networks from the best to the worst. The first half in the ranked population is directly copied to next generation while the remaining half offspring come from the following evolutionary process.

Step 3: Select parents and conduct crossover according to the current crossover rate. Each offspring network is partially trained.

Step 4: Select parents and conduct mutations according to the current mutation rate. The mutation operators are always applied in the order of bias and weights Gaussian mutation, hidden neuron deletion and hidden neuron addition. Structural mutations take place only when parametric mutations fail to increase the fitness of an ANN. All operators are immediately followed by a partial training.

Step 5: If neither the performance of the best ANN meets the requirement nor the maximal evolutionary epoch is reached, go to Step 2.

Experiments and discussion: To evaluate the performance, GAEPNet is applied to the classification of breast cancer problem based on the well-known Wisconsin breast cancer dataset [7]. The dataset consists of 699 patterns of which 458 are benign samples and 241 malignant samples. The dataset was partitioned into three sets as follows: the first 349 samples for training, the following 175 samples for validation and the last 175 samples for test [3].

The ANNs were multilayer perceptrons with one hidden layer. The activation function of hidden neurons was sigmoid whereas it was linear for output neurons. Parameters were chosen aiming at high efficiency as the cost of computational time is in general high when performing evolutionary search. The partial training adopted resilient BP with a learning rate of 0.2 and a training epoch of 15. The maximum number of hidden neurons was set to 6, the population size to 20 and the generation of evolution was set to 30. The proportion between mutation and crossover is 1:59 in the beginning and linearly increases up to 1:1 in the end. The scale factors α and β in the fitness function were both set to 0.25.

The experimental results of GAEPNet over 30 runs are summarised in Table 1. As comparison, results of the EPNet system [3] and the FNNCA [8] are also included. The EPNet is a well-established evolutionary system with an emphasis on the generalisation ability. It is observed that GAEPNet achieves the lowest error rate for the test set and a substantially smaller variation indicating that GAEPNet has the advantages of consistency, stability and accuracy.

Table 1: Percentage error rates across classifiers for breast cancer task

Classifiers		Training set	Validation set	Test set
GAEPNet (30 runs)	Mean	3.734	1.600	0.724
	Max.	5.158	2.286	2.286
	Min.	2.865	0.571	0.000
EPNet (30 runs)	Mean	3.773	0.590	1.376
	Max.	4.585	1.143	4.000
	Min.	1.719	0.000	0.000
FNNCA (50 runs)	Mean	—	—	1.95

GAEPNet performs well not only in classification accuracy but also in computational time. The total time spent by GAEPNet could be estimated by adding the initial training time and the evolving time, resulting in maximum 9300 epochs as opposed to 109 000 epochs for EPNet for a single run.

Conclusion: An evolutionary approach to the design of ANNs has been presented for classification. Experimental results verify that the approach is capable of generating ANNs with high stability and generalisation. Further improvement is the significant reduction in computational time.

3 May 2004
Electronics Letters online no: 20045250
doi: 10.1049/el:20045250

Z.-H. Tan (*Department of Communication Technology, Aalborg University, Aalborg 9220, Denmark*)

E-mail: zt@kom.aau.dk

References

1 Garcia-Pedrajas, N., Hervas-Martinez, C., and Munoz-Perez, J.: 'COVNET: a cooperative coevolutionary model for evolving artificial neural networks', *IEEE Trans. Neural Netw.*, 2003, **14**, (3), pp. 575–596
2 Abraham, A.: 'Meta learning evolutionary artificial neural networks', *Neurocomputing*, 2004, **56**, pp. 1–38
3 Yao, X., and Liu, Y.: 'A new evolutionary system for evolving artificial neural networks', *IEEE Trans. Neural Netw.*, 1997, **8**, (3), pp. 694–713
4 Angeline, P.J., Saunders, G.M., and Pollack, J.B.: 'An evolutionary algorithm that constructs recurrent neural networks', *IEEE Trans. Neural Netw.*, 1994, **5**, (1), pp. 54–65
5 Schaffer, J.D., Whitley, D., and Eshelman, L.J.: 'Combinations of genetic algorithms and neural networks: a survey of the state of the art'. Int. Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, MD, USA, 1992, pp. 1–37
6 Miller, G.F., Todd, P.M., and Hegde, S.U.: 'Designing neural networks using genetic algorithm'. Third Int. Conf. Genetic Algorithms, 1989 pp. 379–384
7 Blake, C.L., and Merz, C.J.: 'UCI repository of machine learning databases', Univ. California, Irvine, CA, 1998
8 Setiono, R., and Hui, L.C.K.: 'Use of a quasi-Newton method in a feedforward neural network construction algorithm', *IEEE Trans. Neural Netw.*, 1995, **6**, (1), pp. 273–277