Neurocomputing 87 (2012) 33–40

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

On the issue of separability for problem decomposition in cooperative neuro-evolution

Rohitash Chandra*, Marcus Frean, Mengjie Zhang

School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

A R T I C L E I N F O

Article history:
Received 13 May 2011
Received in revised form
29 November 2011
Accepted 4 February 2012
Communicated by V. Palade
Available online 24 February 2012

Keywords:
Cooperative coevolution
Neuro-evolution
Feedforward neural networks
Evolutionary algorithms
Genetic algorithms and separability

A B S T R A C T

Cooperative coevolution divides an optimisation problem into subcomponents and employs evolutionary algorithms for evolving them. Problem decomposition has been a major issue in using cooperative coevolution for neuro-evolution. Efficient problem decomposition methods group interacting variables into the same subcomponents. It is important to find out which problem decomposition methods efficiently group subcomponents and the behaviour of neural network during training in terms of the interaction among the synapses. In this paper, the interdependencies among the synapses are analysed and a problem decomposition method is introduced for feedforward neural networks on pattern classification problems. We show that the neural network training problem is partially separable and that the level of interdependencies changes during the learning process. The results confirm that the proposed problem decomposition method has improved performance compared to its counterparts.

1. Introduction

Cooperative co-evolution [17] divides a problem into subcomponents that are implemented as sub-populations. The sub-populations are genetically isolated and cooperation takes place during fitness evaluation. In the use for cooperative coevolution for training neural networks, problem decomposition determines how the neural network is broken down into subcomponents [6,12,11].

In the original cooperative co-evolutionary framework, the problem is decomposed by having a separate subcomponent for each variable [17]. Cooperative coevolution was targeted for solving partially-separable and non-separable problems through co-adaptation; these types of problems are more difficult than fully separable ones. In non-separable problems, there are interdependencies between the variables as opposed to separable ones [18]. In this paper, we refer to the interdependencies by the *degree of non-separability*. A high degree of non-separability will indicate that more interaction is present among the variables.

In this paper, problem decomposition is also referred as *encoding scheme* as the neural network problem has to be decomposed and encoded into subcomponents. The major problem decomposition methodologies include those based on the *neuron level* and *synapse level*. The neuron level encoding scheme employs each neuron in the hidden layer as the main reference point for the respective subcomponents [6,5,10,12]. In synapse level problem decomposition, each weight or synapse in the network forms a subcomponent. An example is the cooperatively coevolved synapse neuro-evolution (CoSyNE) algorithm that has been used for training feedforward and recurrent networks on pole balancing problems [11].

Cooperative coevolution has been used for training neural networks [16,8,10,19]; however, the attention has not been on the issue of separability and interacting variables. It is important to understand the nature of the neural network training problem in order to develop efficient problem decomposition methods. The problem decomposition method must take into account the architecture of the neural network in order to group interacting variables into subcomponents that can allow better performance in cooperative coevolution. Cooperative coevolution naturally appeals to separable problems where no interacting variables are present. In partially separable problems, the advantage of cooperative coevolution can only be taken if the problem is decomposed in such a way that there is least interaction amongst the subcomponents. In other words, the partially-separable problem has to be broken down into small groups where there is interaction of variables within the groups, but less interaction among the different groups.

In this paper, the interdependencies among the synapses are analysed and a problem decomposition method is introduced for feedforward neural networks for pattern classification problems. The proposed encoding scheme is called neuron-based sub-population (NSP). The NSP is based on the analysis of the interdependencies of the network during training. The goal of NSP is to achieve reduced optimisation time and better guarantee for convergence in terms of the success rate. We use benchmark problems from the UCI machine

* Corresponding author. Tel.: +64 4 463 5233 x7088.
E-mail address: c.rohitash@gmail.com (R. Chandra).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2012.02.005

learning repository [1] and provide a comparison with the problem decomposition methods from literature.

This paper is a substantial extension of the results presented in Chandra et al. [3]. The NSP encoding has been used for training recurrent neural network on grammatical inference problems in our previous study [2] where it showed better performance than synapse level encoding. This paper also investigates if similar behaviour is observed for training feedforward networks on pattern classification problems.

The main contributions of this paper are the analysis of the neural network in terms of separability and interacting variables and the introduction of a new problem decomposition method. The synapse level problem decomposition methods from literature are used for training feedforward networks in pattern recognition problems for the first time.

The rest of the paper is organised as follows. In Section 2, a background of cooperative coevolution and neuron-evolution is given. Section 3 presents the analysis on a feedforward neural network in terms of separability. Section 4 gives details of the proposed problem decomposition method and Section 5 presents the experimentations with results and analysis. Section 6 concludes the paper with discussion on future research.

2. Background: cooperative coevolution and neuro-evolution

Cooperative coevolution (CC) is an evolutionary computation framework inspired from nature which divides a larger problem into sub-problems and solves them collectively in order to solve the larger problem [17]. The sub-problems or subcomponents are implemented as sub-populations in the framework. Cooperative coevolution has mainly been used for tackling large-scale optimisation problems [14,23] and neuro-evolution of feedforward [16,5] and recurrent networks [10,11].

The original cooperative coevolution framework can be summarised as follows.

1. *Problem decomposition*: Decompose a high dimensional problem into subcomponents that can be solved by conventional evolutionary algorithms. The subcomponents can vary in sizes and are implemented as sub-populations.
2. *Subcomponent optimisation*: Evolve each subcomponent separately by an evolutionary algorithm where evolutionary operators such as crossover and mutation are restricted to a subcomponent and do not affect other subcomponents.
3. *Fitness evaluation*: Fitness of individuals in subcomponents are evaluated cooperatively with the rest of the subcomponents.

Problem decomposition is considered to be an important step in cooperative coevolution. The problem decomposition method solely relies on the nature of the optimisation problem. The original cooperative coevolution framework (CCEA) decomposed the problem in such a way that a single sub-population was used for each variable in function optimisation problems [17]. The sub-populations in the cooperative coevolution framework are evolved separately and the cooperation only takes place for fitness evaluation for the respective individuals in each sub-population.

2.1. Neuro-evolution with cooperative coevolution

The major advantage of cooperative coevolution for neuro-evolution is that it allows the neural network to be broken down into sub-networks where information can be preserved during evolution. This information is often lost in neuro-evolution with conventional evolutionary algorithms due to reproduction operators such as *crossover*. The sub-populations provide more diversity than a conventional evolutionary algorithm [16].

In feedforward networks, *cooperative co-evolutionary neural networks* (COVNET) have been proposed for pattern recognition problems [6]. Multi-objective cooperative coevolution (MOBNET) employs multi-objective methods for assigning fitness for subcomponents in evolving feedforward networks [5]. COVNET and MOBNET have shown promising results in terms of accuracy in classification problems and have been able to learn the problem in simpler or smaller network structures. Cooperative coevolution has also been used in the construction of Bayesian networks for data mining [22], designing neural networks ensembles [7] and in a cooperative constructive method [9] used in designing neural networks for pattern classification.

3. Analysis on the issue of separability

There has not been much study on how to encode interacting variables into separable subcomponents for the case of neuro-evolution. Most of the study has been focused on large scale function optimisation problems [14,21,20,23], which is not directly applicable to neuro-evolution. It is important to analyse how the synapses interact with each other.

In this section, we examine the interaction between the synapses in a feedforward neural network. Cooperative coevolution naturally appeals to problems which can be broken down and those that do not have any interdependencies among the variables. These types of problems are often expressed as *separable problems*, as given in Definition 1.

Definition 1. A function of n variables is separable if it can be written as a sum of n functions with just one variable as given in Eq. (1). The parameters of a separable function are called independent variables [15].

$$\arg \min_{(x_1,x_2,\ldots,x_n)} f(x_1,x_2,\ldots,x_n) = \left(\arg \min_{x_1} f(x_1,\ldots), \ldots \arg \min_{x_n} f(\cdots x_n) \right) \tag{1}$$

In real world application problems, there are interdependencies among variables which make the problem non-separable [13]. In order to break the neural network training problem into sub-networks, it is important to find how the synapses interact among themselves.

In this section, a measure known as the *degree of non-separability* is established, by monitoring the gradient of the rest of the variables when a single variable is perturbed. In a fully separable problem, there will be no change in the sign of the gradient of the rest of the variables when a single variable is perturbed.

The degree of non-separability is used to measure the interactions among variables in the case of neural networks. The neural network in Fig. 1 contains two sigmoid neurons in the hidden layer (h_1 and h_2) and a single sigmoid neuron y in the output layer. The set of synapses W, is given by $W=[a,b,c,d,e,g]$. i and j are the bias of h_1 and h_2, respectively. x_1 and x_2 are inputs and E is the error given for a single instance. t is the target output and y is the actual output of the network. There is no bias associated with the output neuron y. Eq. (2) gives further details of the network shown in Fig. 1 and Eq. (3) gives the derivatives of the error (E) with respect to its weights (W).

$$\begin{aligned}
\phi_1 &= x_1 a + x_2 c + i \\
\phi_2 &= x_1 b + x_2 d + j \\
h_1 &= F(\phi_1) \\
h_2 &= F(\phi_2) \\
y &= F(F(\phi_1)e + F(\phi_2)g) \\
E &= \tfrac{1}{2}(t-y)^2
\end{aligned} \tag{2}$$

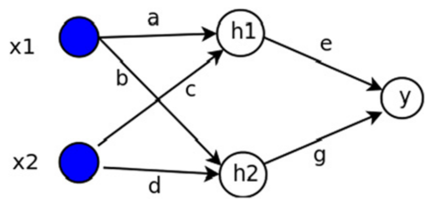

Fig. 1. The simple feedforward neural network used for analysis.

where $F(x) = 1/(1+e^{-x})$.

$$\frac{\partial E}{\partial a} = ex_1(t-y)F'(\phi_1)F'[F(eF(\phi_1)+gF(\phi_2))]$$
$$\frac{\partial E}{\partial b} = gx_1(t-y)F'(\phi_2)F'[F(eF(\phi_1)+gF(\phi_2))]$$
$$\frac{\partial E}{\partial c} = ex_2(t-y)F'(\phi_1)F'[F(eF(\phi_1)+gF(\phi_2))]$$
$$\frac{\partial E}{\partial d} = gx_2(t-y)F'(\phi_2)F'[F(eF(\phi_1)+g-kF(\phi_2))]$$
$$\frac{\partial E}{\partial e} = F(\phi_1)(t-y)F'[F(eF(\phi_1)+gF(\phi_2))]$$
$$\frac{\partial E}{\partial g} = F(\phi_2)F'(t-y)[F(eF(\phi_1)+gF(\phi_2))] \tag{3}$$

Algorithm 1. The degree of non-separability in feedforward neural networks.

```
1. Create vector $W$ of size $n$ where $n$ is total number of weights
   and biases
2. Create vector of gradients $P$ and $Q$ of size $n$
3. Create a matrix $C$ of size $n \times n$.
for each Monte-Carlo trial do
    Initialise the vector $W$ with random values in a range
    for each $i$ until $n$ is reached do
        i. Compute all the previous gradients ($P = \frac{\partial E}{\partial W}$)
        ii. Perturb $W[i]$, flip the sign ($W[i] = -1 * W[i]$)
        iii. Compute all the current gradients ($Q = \frac{\partial E}{\partial W}$)
        iv. Compare the previous and current gradients
        for each $j$ until $n$ is reached do
            Compare $P[j]$ and $Q[j]$ and increment $C[i][j]$ if the sign of
            the two gradients change
        end for
    end for
end for
```

Algorithm 1 shows how the degree of non-separability is computed for a feedforward neural network. A total number of m Monte-Carlo trials are done where vector W is initialised in the range of $[-5, 5]$. In each trial, a single synapse in the neural network is perturbed by multiplying the variable by -1 and the rest of the synapses are frozen. The gradients $\partial E/\partial w$ for the frozen weights are computed. For instance, when synapse a is perturbed, the behaviour on the rest of the synapses which include weights and biases $W = [i,j,b,c,d,e,g]$ are observed. The change in the sign of the gradient for a particular frozen weight indicates that there is interaction. Therefore, whenever there is a sign change in the gradient, the count of the number of interactions of the particular synapse is incremented. This is the measure for the degree of non-separability.

A total of 10,000 experiments with different values of weights and biases were carried out and the results averaged and shown as heat-maps in Figs. 2 and 3. The results in Figs. 2 and 3 show independent Monte-Carlo trials of four different combinations of input $x1$ and $x2$ ([0, 0], [0, 1], [1, 0] and [1, 1]). They show how the rest of the

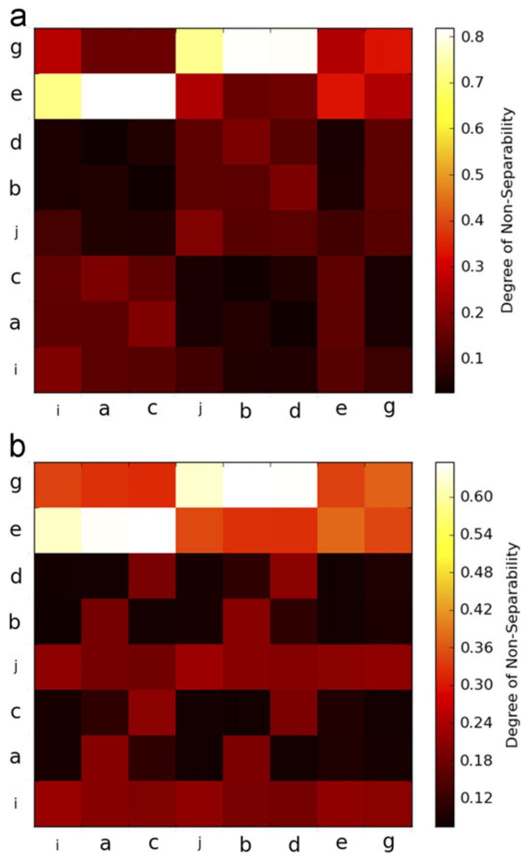

Fig. 2. The level of interaction between the synapses at the beginning of the evolutionary process. (a) Stage 1: RMSE is 0.381. (b) Stage 2: RMSE is 0.095.

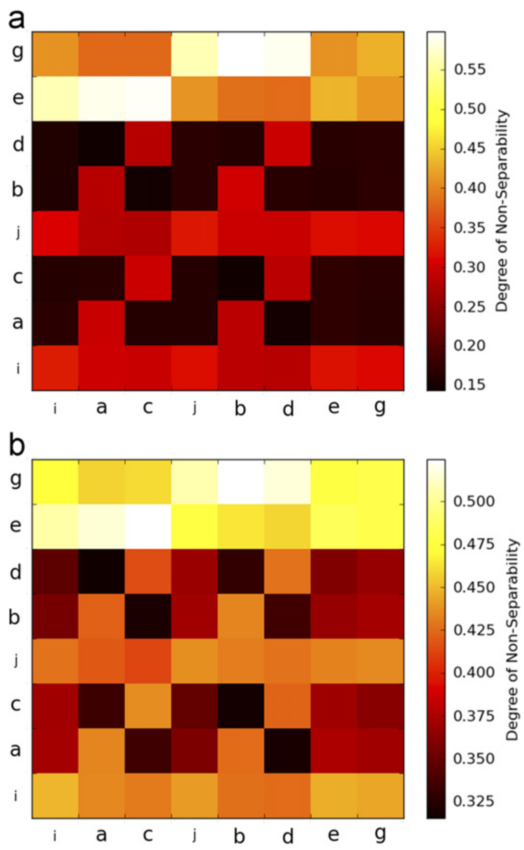

Fig. 3. The level of interaction between the synapses towards the end of the evolutionary process. (a) Stage 3: RMSE is 0.047. (b) Stage 4: RMSE is 0.009.

synapses in the neural network interact with each other when a synapse is perturbed independently in the Monte-Carlo trials.

The target of each input instance is heuristically generated to depict different stages in learning. The root-mean-squared-error (RMSE) is shown to indicate the learning process of the four different stages of evolution. Fig. 2 shows that at the beginning of the learning process in Stage 1, there is little interaction between the weights *e* and *g*. The weights in the input-hidden layer interact with the weights in the hidden-output layer. In Stage 2, the interaction between weights *e* and *g* becomes higher. Fig. 3 shows that the interaction between *e* and *g* remains high during the later stages of learning. In Stage 3, the interaction in $W = a,b,c,d$ grows and more interaction is seen between weights *e* and *g*. Stage 4 shows the end of the learning process and there is more interaction with *e* and *g* and the rest of the weights.

In most of the learning process, the interaction between *e* and *g* grows and hence it is reasonable to group these weights together into a separate subcomponent.

Note that interactions were asymmetrical in general. We can see that *a* interacts with *e*, but not the other way around i.e. *e* does not interact with *a*; therefore, they are not required to be grouped. Due to this asymmetry, it is reasonable to separate the groups of the hidden-output layer weights from the input-hidden layer weights.

We observe that there are two options for grouping the weights. *Option* 1 groups the weights as follows: [*a*,*c*,*i*] are grouped together, [*b*,*d*,*j*] are grouped together, and finally [*e*,*g*] are grouped together.

Another possibility is *Option* 2 where the weights are grouped as follows: [*e*,*g*] are be grouped together as in Option 1. In the input-hidden layer weights, [*a*,*b*,*i*] are grouped together and [*c*,*d*,*j*] are grouped together. This option seems to be more suitable according to the interaction in the input-hidden layer weights during the later stages of learning (Stages 2–4).

Both of the options will have more subcomponents than CME and can be generalised to a neural network with more than one hidden layer.

However, Option 1 will be used in this paper. With Option 1, the number of subcomponents depends mainly on the number of hidden neurons while in Option 2, it depends on the number of input units.

Fig. 4 shows the degree of non-separability on a six dimension input problem in the early stage of evolution. There are six inputs (plus a bias) on a neural network with five hidden units and one output. Indices 0–35 show the interaction among the input-hidden later weights. The results in this heat-map indicates that the degree of non-separability for a larger network architecture is similar to a small network structure as shown in Fig. 2(a).

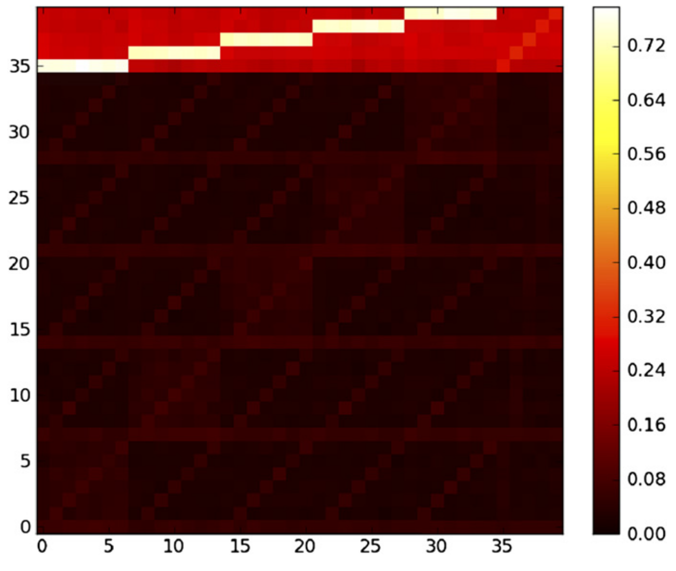

Fig. 4. The level of interaction between the synapses at the beginning of the evolutionary process on a 6 dimension problem. There are six inputs (plus a bias) on a neural network with five hidden units and one output.

4. A new encoding scheme: neuron based sub-population

4.1. Preliminaries

COVNET [6] and MOBNET [5] have been used for training feedforward networks. These cooperative coevolutionary methods have used neural level encoding scheme where the neurons in the hidden layer act as the main reference point for a subcomponent. ESP [10,12] employs a similar encoding scheme; however, recurrent connections are also present as it has been used mainly for training recurrent networks. Henceforth, in this paper, the encoding scheme used in COVNET, MOBNET and ESP is referred to as "CME" due to their similarities. This is done by taking the first letters from each abbreviation. Fig. 5 shows the schematic of the interconnected input and output links to a hidden neuron. It is assumed that the network has one hidden layer only. The number of hidden neurons is equal to the number of subcomponents. In this mapping, all sub-populations have the same size for the entire framework.

4.1.1. Development of neuron-based sub-population

A major limitation of the encoding scheme given in CME is that it cannot be extended to a neural network with more than one hidden layer. This is because the hidden neuron of a single layer acts as a major reference point and contains outgoing neurons in the subcomponents. We need a problem decomposition method that can be easily extended to a network with more hidden layers and at the same time it should efficiently group the interacting variables in a separate subcomponent.

Synapse level encoding in neural networks provides the most diversity and ensures global search. CME encoding has lower diversity than synapse level and gives more emphasis on interacting variables. Note that the diversity is dependent on the number of subcomponents. The problem in cooperative neuro-evolution is to seek a balance between diversity and interacting variables. A problem of CME encoding is that it cannot be generalised to a neural network with more than more hidden layer. If the CME encoding is broken down, it can be generalised to more than one hidden layer and also achieve more diversity. The new encoding scheme should also group interacting variables which is not a feature of synapse level encoding. The results in Section 3 have shown that it is reasonable to break the CME encoding. It is important to group the weights that are connected to the output neuron.

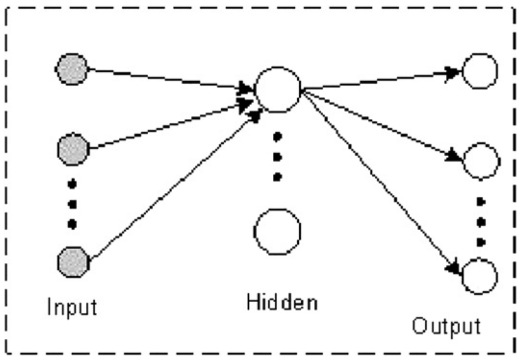

Fig. 5. The CME encoding scheme summarised from [12,6,5] is used for comparison in the experiments [3].

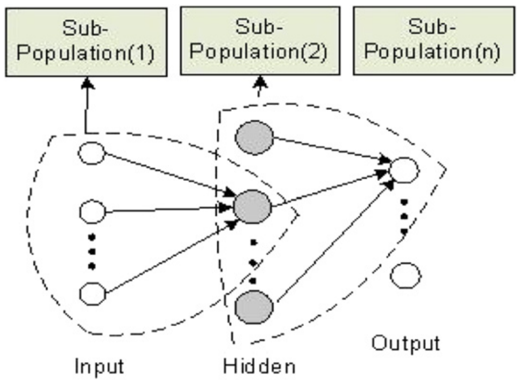

Fig. 6. The NSP encoding scheme for feedforward networks [3]. Each neuron in the hidden and output layer acts as a reference point to its subcomponents given as sub-populations. The same encoding is used in the rest of the neurons in the hidden and output layer. Note that only one hidden layer is used. NSP can also be used for more than one hidden layer.

A single subcomponent in CME encodes the incoming and outgoing connections in reference to a hidden neuron. The proposed neuron based sub-population (NSP) breaks down this encodings scheme into a lower level. Each subcomponent in the NSP consists of incoming connections associated with neurons in the hidden and output layers. NSP implements one sub-population for each neuron which groups interacting variables (synapses) that are connected to the neuron. Therefore, each sub-population for a layer is composed of the following individuals:

1. Hidden layer sub-populations: weight-links from each neuron in the *hidden* layer are connected to all *input* neurons and the bias of *hidden* layer.
2. Output layer sub-populations: weight-links from each neuron in the *output* layer are connected to all *hidden* neurons and the bias of *output* layer.

Fig. 6 shows a detailed diagram of the NSP encoding. The general framework for NSP in training feedforward networks is summarised in Algorithm 2. Each neuron in the hidden and output layer acts as a reference point to its subcomponents given as sub-populations.

Algorithm 2. The NSP for training feedforward networks.

```
Step 1. Decompose the problem into $k$ subcomponents. $k$ is the
        number of hidden neurons plus the number of output
        neurons.
Step 2. Encode each subcomponent in a sub-population in the
        following order:
   1. Hidden layer sub-populations
   2. Output layer sub-populations
Step 3. Initialise and cooperatively evaluate each sub-
population
for each Cycle until termination do
   for each Subpopulation do
      for $n$ Generations do
         (i) Select and create new offspring
         (ii) Cooperatively evaluate the new offspring
         (iii) Update the sub-population
      end for
   end for
end for
```

In Algorithm 2, the network is decomposed into *k* subcomponents where *k* is equal to the number of hidden neurons, plus the number of output neurons. Each sub-population contains all the weight links from the previous layer connecting to a particular neuron. A *cycle* is completed when all the sub-populations are evolved in a round-robin fashion for a fixed number of generations. The algorithm halts if the termination condition is satisfied. The termination condition is when the network correctly classifies a given percentage of the training data or when the maximum training time is reached.

5. Simulation: problem decomposition in feedforward networks

This section provides an empirical study on the performance of the proposed problem decomposition method. The simulation reports the performance of the different problem decomposition methods given different *depth of search* specified by the number of generations.

The neural network optimisation time in terms of the number of function evaluations and the success rate are considered to be the main performance measures for the method presented in this study. The success rate determines how well a particular algorithm can guarantee a solution within a specified time. A run is considered successful if a desired solution is found before the maximum time is reached. The desired solution for neural network training is specified by a predefined minimum network error or minimum classification performance depending on the type of the problem.

This section presents an experimental study of the NSP and compares it with CoSyNE, CME and evolutionary algorithms (EA). The G3-PCX [4] evolutionary algorithm is used in all the respective CC frameworks and the EA. The G3-PCX algorithm employs the population size of 100, two parents and two offspring for the EA and CC frameworks. The same set-up has been in the sub-populations of all the respective CC frameworks. This set-up has been used in Deb et al. [4] and has shown good results for general optimisation problems. The sub-populations are seeded with random real numbers in the range of $[-5, 5]$ in all experiments.

5.1. Real-world problems and neural network configuration

The *Iris*, *Wine*, *Breast-Cancer* and *Heart-Disease* datasets used in the experiments have been taken from machine learning repository [1]. Table 1 shows the neural network configuration for all the experiments. The data is split into a training set and a test set where 70% of the data was used for training and the remaining 30% for testing. The maximum training time given by the number of function evaluations in all the problems is fixed as 100,000. The table also shows the minimum training performance required for each problem. Note that in the Heart-Disease classification problem only, the neural network is trained until it reaches at-least 88% classification performance on the training data. This value was determined in trial runs and it has been observed that reaching a better classification performance is difficult for this problem. In all other problems, the minimum classification performance required on the training data is 95%. The network topology configuration for each problem is also given in Table 1.

Table 1
Dataset information and neural network configuration.

Domain	No. cases	Input	Out.	Hid.	Min. train (%)
Iris	150	4	3	4	95
Wine	178	13	3	4	95
Breast-Cancer	699	9	2	5	95
Heart-Disease	303	13	2	7	88

5.2. Depth of search in the sub-populations

Each sub-population is evolved for a fixed number of generations in NSP as shown in Algorithm 2. The study proceeds after determining the optimal number of generations needed for the sub-population in each method. Note that all the sub-populations are meant to evolve for the same number of generations in a round-robin fashion which must be fixed beforehand.

The results in Figs. 7–10 report the performance of the respective methods in terms of the number of function evaluations and the success rate out of 30 independent runs. Note that the results in this section do not include the number of function evaluations from the initialisation stage for each method. The depth of search given by the number of generations used for all sub-populations is also shown. Note that the results from the unsuccessful runs are also included in the mean and the 95% confidence interval shown as error bars in the histograms. Note that the least number of function evaluations with high success rate is desired.

The results for the Iris classification problem presented in Fig. 7 show that NSP outperforms the other methods in terms of the function evaluations and success rate. CoSyNE takes the most training time and gives the worst/lowest success rate. CME performs slightly better than CoSyNE but tends to be weaker than NSP. NSP is the best choice for this problem. The depth of search in NSP does not make a major difference in its performance.

In the results for the Wine classification problem shown in Fig. 8, NSP gives the best performance. The depth of search in NSP does not play a major role. The depth of search impacts the performance of CME and mostly, CoSyNE. CoSyNE shows good performance for the depth of one generation only. The best results are given by NSP in terms of least optimisation time and the best success rate.

The results from the Heart-Disease classification problem shown in Fig. 9 confirms that NSP outperforms CME and CoSyNE. The CoSyNE method delivered a solution only with the depth of one generation only, performing poorly in comparison to CME and NSP. The depth of search in NSP does not show a major difference. Similar performance is shown for the Breast-Cancer classification problem as shown in Fig. 10, where NSP outperforms other methods. The depth of search for NSP and CME does not play a significant role in their performance which implies that NSP and CME have been able to group the interacting variables in the different sub-populations.

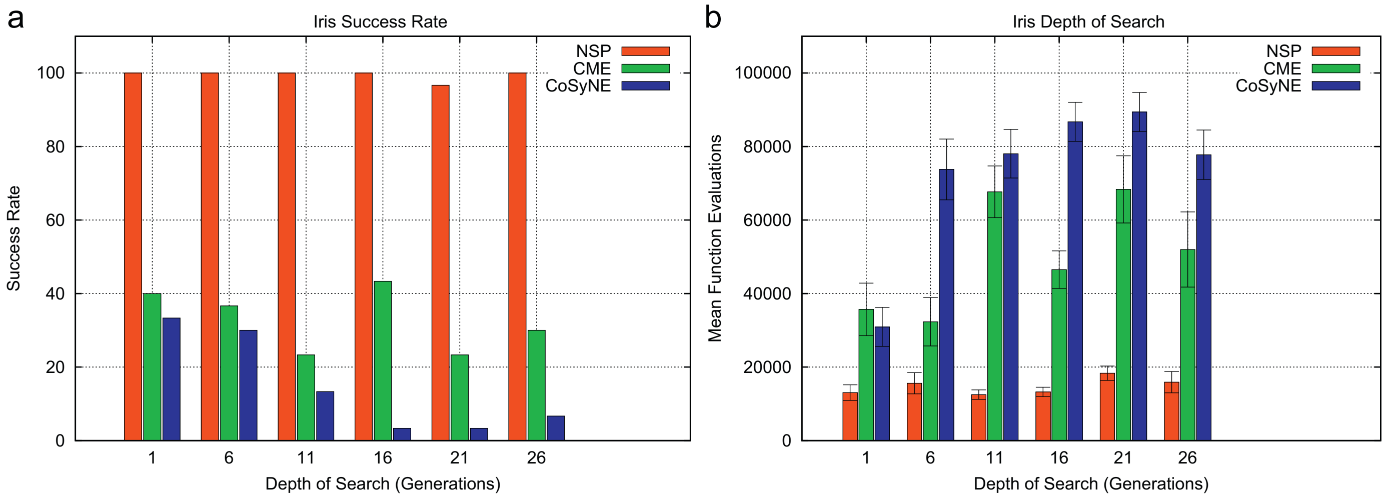

Fig. 7. The evaluation of the depth of search in the different problem decomposition methods for the *Iris* classification problem. The success rate is shown in (a) while the optimisation time in terms of the number of function evaluations is shown in (b).

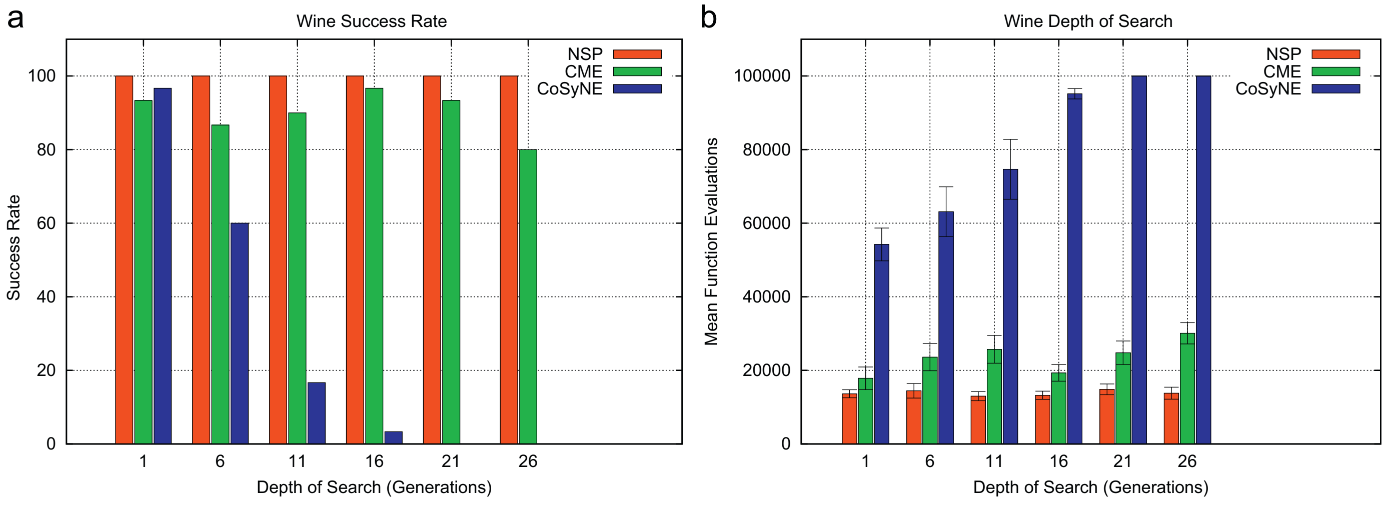

Fig. 8. The evaluation of the depth of search in the different problem decomposition methods for the *Wine* classification problem. The success rate is shown in (a) and the optimisation time in terms of number of function evaluations is shown in (b).

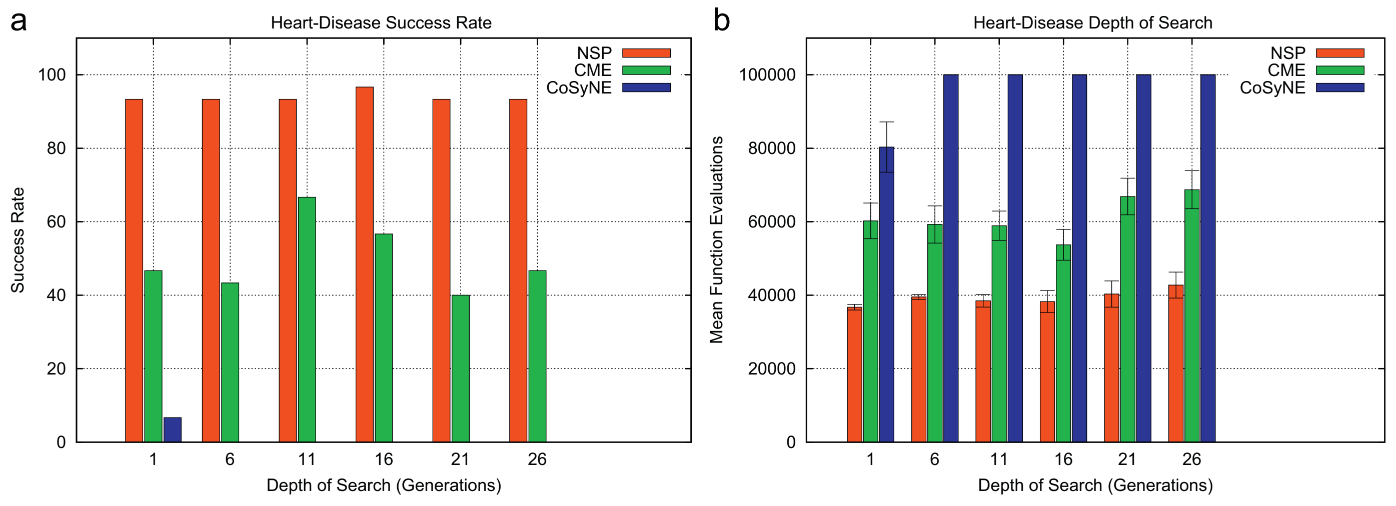

Fig. 9. The evaluation of the depth of search in the different problem decomposition methods for the *Heart-Disease* classification problem. The success rate is shown in (a) and the optimisation time in terms of the number of function evaluations is shown in (b).

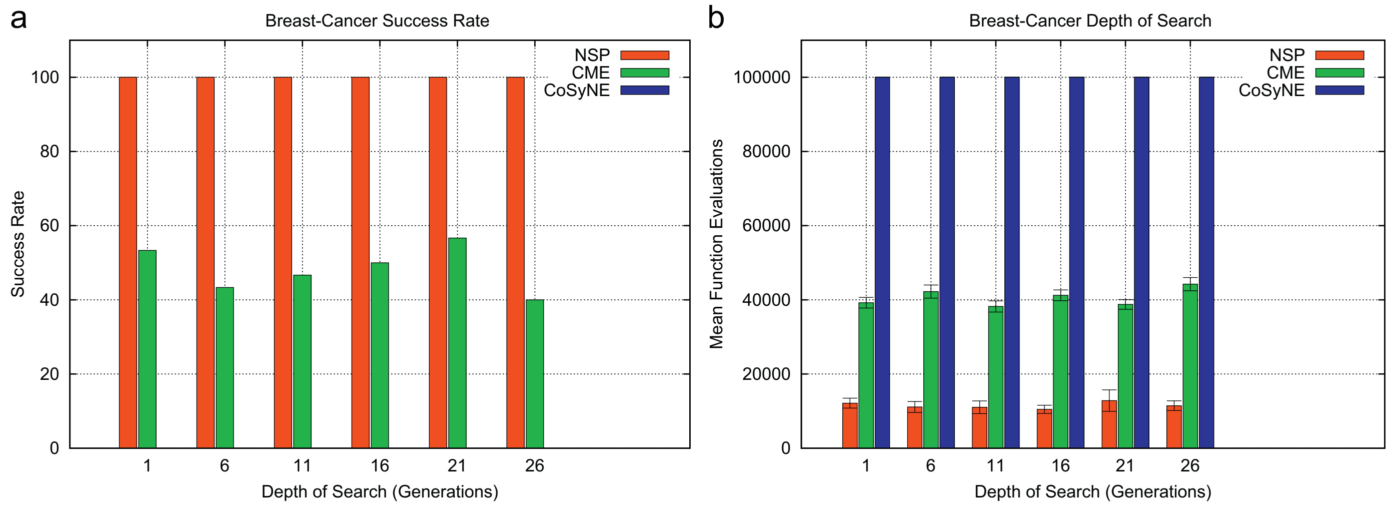

Fig. 10. The evaluation of the depth of search in the different problem decomposition methods for the *Breast-Cancer* classification problem. The success rate is shown in (a) and the optimisation time in terms of the number of function evaluations is shown in (b).

NSP performs well regardless of the depth of search used, while CoSyNE shows good performance for the depth of one generation only. In general, CoSyNE is unreliable in converging to a solution. The results have shown that a lower depth of search used for a subcomponent is important for CoSyNE. The depth of search for NSP does not show a significant effect. This indicates that NSP encoding scheme can efficiently group interacting variables in separate subcomponents. The efficient grouping of interacting variables allows less interaction amongst different subcomponents. A higher depth of search is possible only if there is less interactions amongst subcomponents. CoSyNE fails with higher depth of search. This indicates that the neural network training problem is partially separable and interaction amongst subcomponents is present in problem decomposition using CoSyNE. Given an increased depth of search, CME shows better performance when compared with CoSyNE while NSP shows the best performance. This is due to the difference in the way the subcomponents are encoded using the different methods. Note that the CoSyNE uses a higher number of subcomponents and it does not group interacting variables as the number of variables in a subcomponent is restricted to one.

The comparison of NSP and CME in terms of the function evaluations shows that NSP is more stable in its performance when the depth of search in increased. This is applicable to all the problems except the Breast-Cancer problem.

5.3. Discussion

The results reveal that good performance has been achieved when each sub-population is evolved for one generation in a round-robin fashion for the entire cycle of CoSyNE and NSP. In neural networks, interacting variables exist which determine the degree of non-separability. The degree of non-separability is dependent on the particular neural network architecture and application problem. The deep greedy search for a large number of generations for each sub-population has not shown good performance and gives an indication that neural networks on the given problems are partially separable.

The performance of CoSyNE is weak when compared to NSP for training feedforward neural networks in pattern recognition problems. However, CoSyNE showed impressive results for pole balancing problems in Gomez et al. [11]. This is due to the nature of the problem as the pole balancing problem is a control problem, where neuro-evolution is most appropriate as gradient information is not available.

The generalisation performance of NSP has been similar to the other methods. This indicates that the NSP has achieved the same solution quality in lower optimisation time and better success rate which reflects on robustness.

6. Conclusions

In neural networks, interacting variables exist which determine the degree of non-separability. In this paper, the degree of non-separability has been analysed and a new problem decomposition method has been presented. The analysis has shown that the level of interactions among the synapses changes during the learning process.

The proposed problem decomposition method has been shown to efficiently decompose the problem by grouping interacting variables into the separate subcomponents. This has been verified as the depth of search does not give a significant difference in the performance of the proposed method. This indicates that there is little interaction amongst the subcomponents during evolution and the problem decomposition method has been effective.

In future research, it would be interesting to examine the interactions in a recurrent neural network during the learning process. Moreover, it would be also interesting to apply the proposed problem decomposition methods in real world problems.

References

[1] A. Asuncion, D. Newman, UCI Machine Learning Repository, 2007, URL ⟨http://archive.ics.uci.edu/ml/datasets.html⟩.
[2] Rohitash Chandra, Marcus Frean, Mengjie Zhang, Christian W. Omlin, Encoding subcomponents in cooperative co-evolutionary recurrent neural networks, Neurocomputing, 74 (17) (2011) 3223–3234.
[3] R. Chandra, M. Frean, M. Zhang, An encoding scheme for cooperative coevolutionary feedforward neural networks, in: J. Li (Ed.), AI 2010: Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 6464, Springer, Berlin/Heidelberg, 2010, pp. 253–262.
[4] K. Deb, A. Anand, D. Joshi, A computationally efficient evolutionary algorithm for real-parameter optimization, Evol. Comput. 10 (4) (2002) 371–395.
[5] N. Garcia-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks), Neural Netw. 15 (10) (2002) 1259–1278.
[6] N. Garcia-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, COVNET: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Trans. Neural Netw. 14 (3) (2003) 575–596.
[7] N. Garcia-Pedrajas, C. Hervas-Martinez, D. Ortiz-Boyer, Cooperative coevolution of artificial neural network ensembles for pattern classification, IEEE Trans. Evol. Comput. 9 (3) (2005) 271–302.
[8] N. García-Pedrajas, D. Ortiz-Boyer, A cooperative constructive method for neural networks for pattern recognition, Pattern Recogn. 40 (1) (2007) 80–98.
[9] N. García-Pedrajas, D. Ortiz-Boyer, A cooperative constructive method for neural networks for pattern recognition, Pattern Recogn. 40 (1) (2007) 80–98.
[10] F. Gomez, R. Mikkulainen, Incremental evolution of complex general behavior, Adapt. Behav. 5 (3-4) (1997) 317–342.
[11] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 9 (2008) 937–965.
[12] F.J. Gomez, Robust Non-linear Control Through Neuroevolution, Technical Report AI-TR-03-303, PhD Thesis, Department of Computer Science, The University of Texas at Austin, 2003.
[13] K. Tang, L. P.N.S.Z.Y. Xiaodong Li, T. Weise, Benchmark Functions for the CEC'2010 Special Session and Competition on Large Scale Global Optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009, URL ⟨http://nical.ustc.edu.cn/cec10ss.php⟩.
[14] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of the 2001 Congress on Evolutionary Computation, 2001.
[15] D. Ortiz-boyer, C. Hervs-martnez, N. Garca-pedrajas, Cixl2—a crossover operator for evolutionary algorithms based on population features, J. Artif. Intell. Res. 24 (2005) 2005.
[16] M.A. Potter, K.A. DeJong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[17] M.A. Potter, K.A.D. Jong, A cooperative coevolutionary approach to function optimization, in: PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, 1994, pp. 249–257.
[18] R. Salomon, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms, Biosystems 39 (3) (1996) 263–278.
[19] J. Schmidhuber, D. Wierstra, M. Gagliolo, F. Gomez, Training recurrent networks by evolino, Neural Comput. 19 (3) (2007) 757–779.
[20] Y.-j. Shi, H.-f. Teng, Z.-q. Li, Cooperative co-evolutionary differential evolution for function optimization, in: L. Wang, K. Chen, Y. Ong (Eds.), Advances in Natural Computation. Lecture Notes in Computer Science, vol. 3611, Springer, Berlin/Heidelberg, 2005, pp. 1080–1088.
[21] F. van den Bergh, A. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 225–239.
[22] M.L. Wong, S.Y. Lee, K.S. Leung, Data mining of Bayesian networks using cooperative coevolution, Decis. Support Syst. 38 (3) (2004) 451–472.
[23] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Inf. Sci. 178 (15) (2008) 2985–2999.

Rohitash Chandra is a research assistant in computer science at the School of Engineering and Computer Science, Victoria University of Wellington. He has recently completed his PhD in computer science at the same institution. He holds a MSc in computer science from the University of Fiji and BSc from the University of the South Pacific. His research interests in general encircle the methodologies and applications of artificial intelligence. More specifically, he is interested in neural and evolutionary computation methods such as feedforward and recurrent networks, genetic algorithms, cooperative coevolution and neuro-evolution with applications in pattern classification, time series prediction, control, robot kinematics and environmental informatics. He is currently working on problems in Computational Biology. Apart the editor of the Blue Fog Journal. He is a poet and his third poetry collection is titled "Being at Home" which is due to be launched in 2012. He is also the founder of the Software Foundation of Fiji.

Marcus Frean is a senior lecturer in computing science at the School of Engineering and Computer Science, Victoria University of Wellington. His research interests are in artificial intelligence and machine learning.

Mengjie Zhang is a professor of computer science at the School of Engineering and Computer Science, Victoria University of Wellington. His research interests are in genetic programming, swarm intelligence, data mining and machine learning.