Modularity Adaptation in Cooperative Coevolution of Feedforward Neural Networks

Rohitash Chandra, Marcus Frean and Mengjie Zhang

***Abstract*— In this paper, an adaptive modularity cooperative coevolutionary framework is presented for training feedforward neural networks. The modularity adaptation framework is composed of different neural network encoding schemes which transform from one level to another based on the network error. The proposed framework is compared with canonical cooperative coevolutionary methods. The results show that the proposal outperforms its counterparts in terms of training time, success rate and scalability.**

I. Introduction

Cooperative coevolution (CC) divides a large problem into smaller subcomponents where each subcomponent uses a separate evolutionary algorithm (EA). The subcomponents evolve in a round-robin fashion and are genetically isolated. The only cooperation that takes place is during fitness evaluation. The original CC framework employed a separate subcomponent for each variable [1] and was only effective for problems that are separable [2]. In separable problems, there is no interdependency between the decision variables whereas in non-separable problems, interdependencies exist. In most problems, groups of interacting and non-interacting variables exist which determine the *degree of non-separability*. The degree of non-separability is used to refer to the nature of the problem in terms of interacting variables in the rest of the paper. The performance of evolutionary algorithms deteriorate when the problem becomes significantly large. Just as in other evolutionary algorithms, CC also faces the problems of scalability which has been tackled for non-separable problems [3], [4].

The CC framework has been used for training feedforward [5], [6] and recurrent neural networks[7], [8]; however, the attention has not been on the issue of separability and interacting variables. In recent years, two major subcomponent design methodologies have been proposed. These include subcomponent design on the *neuron level* and *synapse level*. The neuron level subcomponent design employs each neuron in the hidden layer as the main reference point for the respective subcomponent [9], [10], [11], [12]. In the synapse level encoding, each weight in the network forms a subcomponent of the CC framework. An example of the synapse level encoding has been the cooperatively coevolved synapse neuro-evolution (CoSyNE) algorithm that was used for training feedforward and recurrent networks on pole balancing problems [13]. In this encoding scheme, each subcomponent represents a single interconnection that is a single weight in the network. CoSyNE showed better performance than most evolutionary algorithms for pole balancing problems [13]. However, its performance was poor in pattern recognition when compared to neuron-based sub-population (NSP) [12].

The number of subcomponents used in the respective encoding schemes plays a significant role during evolution. Each encoding scheme groups interacting variables and makes an assumption on the degree of non-separability regardless of the problem. CoSyNE for instance, views the neural network training as a separable problem and has been only effective for training feedforward and recurrent networks on pole balancing problems. CoSyNE performed poorly in training feedforward networks for pattern recognition problems [12]. Therefore, it is reasonable to adapt the encoding scheme (modularity) as different levels of search in terms of non-separability are needed at different stages of evolution.

This paper presents an adaptive modularity cooperative coevolutionary framework (AMCC-FNN) for training feedforward neural networks for pattern classification. Instead of using a fixed level of modularity during the entire evolution, the AMCC-FNN adapts the number of modules at different stages of the evolutionary process. The performance of the AMCC-FNN is compared with an evolutionary algorithm and canonical cooperative coevolutionary approaches such as CoSyNE and NSP. The goal is to observe the neural network training performance in terms of the optimisation time, success rate and scalability. The scalability reflects on the ability of the training algorithm to perform when the problem size increases significantly.

The rest of the paper is organised as follows. Section II of the paper overviews cooperative coevolution. Section III provides details of the proposed adaptive modularity cooperative coevolutionary framework for feedforward networks. Section IV presents the experiments and results and section V concludes the paper with a discussion on future research.

II. Cooperative Coevolution for Neuro-Evolution

In the evolutionary process of nature, different species compete in order to survive with the given resources. The individuals of a particular group of species mate amongst themselves in order to produce stronger individuals. However, mating with different species is not feasible. The cooperative

Rohitash Chandra is with School of Engineering and Computer Science at Victoria University of Wellington, Wellington, New Zealand. This work was partially done when the first author was with the Department of Computing Science and Information Systems at Fiji National University, Suva, Fiji. (email: rohitash.chandra@ecs.vuw.ac.nz, rohitash_c@yahoo.com)

Marcus Frean and Mengjie Zhang are with School of Engineering and Computer Science at Victoria University of Wellington, Wellington, New Zealand. (email: {marcus.frean, mengjie.zhang}@ecs.vuw.ac.nz).

978-1-4244-9636-5/11/$26.00 ©2011 IEEE

coevolution framework is nature inspired where species are represented as subcomponents which are implemented as subpopulations. A major advantage of the CC framework in neuro-evolution is that it provides a mechanism for grouping network weights into different subpopulations.

The subpopulations in the cooperative coevolution framework evolve separately and the cooperation only takes place for fitness evaluation for the respective individuals in each subpopulation. The size of a subcomponent and the way it is encoded is dependent on the problem. The way the algorithm cooperatively evaluates each subcomponent has been a major study in the CC framework. A method for estimating fitness has been proposed by Potter and Jong [5]. This method obtains the fitness of each individual in a subpopulation by combining it with the best individuals for the rest of the subpopulations. This method of fitness assignment has been used to train cascade networks on the two-spirals problem and has shown to learn the task with smaller networks when compared to the cascade correlation learning architecture [5].

A. Cooperative evaluation of subcomponents

A major concern in the general framework is the cooperative evaluation of each subcomponent in every subpopulation. There are two main phases of evolution in the cooperative coevolution framework. The first is the *initialisation phase* and the second is the *evolution phase*.

In order to cooperatively evaluate the ith individual of the kth subcomponent (chosen individual) in the initialisation phase, arbitrary individuals from the rest of the subpopulations are selected and combined with the chosen individual and cooperatively evaluated. This is done in the initialisation stage since the individuals from the rest of the subpopulations do not have fitness values assigned. Once the fitness has been assigned to all the individuals of a particular sub-population, then the best individual can be chosen. A similar approach is shown in [5].

In the evolution phase, cooperative evaluation is done by combining or concatenating the individual from the selected subpopulation with the best individuals from the rest of the subpopulations. After the initialisation phase, the best individuals from the rest of subpopulations can easily be found through ranking according to fitness. The concatenated individual is encoded into the neural network and the fitness is calculated through the training error. The goal of the evolutionary process is to increase the fitness which tends to decrease the network error. In this way, the fitness of each subcomponent in the network is evaluated until the *cycle* is completed. A cycle is when all the subpopulations are evolved for n generations in a round-robin fashion.

B. Problem Decomposition

Problem decomposition has been a major study in using the cooperative coevolution framework for neuro-evolution which determines the way a subcomponent is designed. We also refer to problem decomposition as encoding scheme in this paper. The major problem decomposition methods include those on the *neuron level* and *synapse level*. The neuron level subcomponent design uses each neuron in the hidden layer as the main reference point for the respective subcomponent. Each subcomponent consists of the incoming and outgoing weight connections. Examples include the cooperative coevolution model for evolving artificial neural networks (COVNET) [9] and multi-objective cooperative networks (MOBNET) [10] that encode subcomponents by mapping all input and output connections from the respective hidden neuron. They have been used for training feedforward network architectures. This problem decomposition method is similar to that of *enforced subpopulations* (ESP) [7], [11]. The NSP [12] decomposes the network to the neuron level and unlike ESP, the subcomponents do not include the outgoing weight links associated with a neuron as shown in Figure 1.

In this paper, the NSP for feedforward networks in [12] is used. We assume that the network has one hidden layer and each subpopulation in the respective layer is composed of the following individuals.

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

Each neuron in the hidden and output layer acts as a reference point to its subpopulations. Note that the NSP can be easily extended for multiple hidden layers.

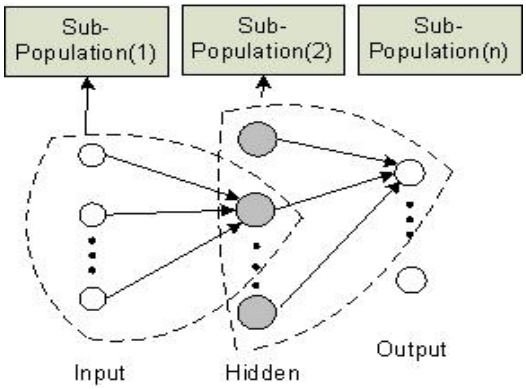


Fig. 1. The NSP encoding scheme. The same encoding scheme is used in the rest of the neurons in the hidden and output layer [12]. Note that the NSP can be easily extended for multiple hidden layers.

III. Adaptation of Modules in Cooperative Coevolution of Feedforward Networks

This section introduces the framework for the adaptation of modules in cooperative coevolution (AMCC) for training feedforward neural networks (FNN). The AMCC framework changes its level of modularity during evolution. It employs the level of modularity which provide a greater level of flexibility (allowing evolution in separable search space) during the initial stage and decreases the level of modularity during the later stages of evolution.

The modularity of the AMCC-FNN framework can transform from synapse level encoding (CoSyNE) to neuron level encoding (NSP) and finally to network level encoding (EA). The general AMCC frameworks employs all the three levels of encoding for the adaptation of modularity. The respective levels of encoding are further described as follows:

1) **Synapse level encoding**: Decomposes the network into its lowest level to form a single module [13]. The number of connections in the network determines the number of modules.
2) **Neuron level encoding**: Decomposes the network into the neuron level. The number of neurons in the hidden and output layers determines the number of modules [12].
3) **Network level encoding**: The standard neuro-evolutionary encoding scheme where only one population represents the entire network. There is no decomposition present in this level of encoding.

The assumption in using the above levels of modularity at the respective stages in evolution is that the nature of the neural network training in terms of non-separability changes during evolution. We assume that in training neural networks, lower degree of non-separability is needed during the initial stages when compared to the final stages of evolution. The experimental section will show whether this assumption is correct.

A. Overview

The AMCC-FNN framework in Algorithm 1 follows three main stages of evolution. At the beginning, all the subpopulations of the synapse level, neuron level and network level encoding are seeded with random real values in a range. In Stage 1, the subpopulations at synapse level encoding are only cooperatively evaluated. Neuron level and Network level encoding are left to be cooperatively evaluated at later stages.

Stage 1 employs synapse level encoding where the subpopulations are evolved until the maximum number of function evaluations (MaxGlobal) is reached. The optimal solution is returned if the minimum error is reached (MinError). However, if MaxGlobal and MinError are not satisfied, the change of modularity is done and the framework proceeds to neuron level encoding in Stage 2. The best individuals are transferred to the subpopulations of the neuron level at Stage 2. The transfer is done by encoding the best individuals at Synapse level and encoding it in the way a subcomponent is build in the neuron level encoding as shown in Figure 1. All the subpopulations are cooperatively evaluated. During neuron level evolution, if the MinError is not reached within the MaxGlobal, then the change of modularity is done in order to proceed to the network level encoding at Stage 3. The best individuals at neuron level from each subpopulation are concatenated and added to the population at the network level. Note that the neuron level subpopulations and the network level population initially contain randomly initialised genetic material. The best individual from the previous stage is added to them and then the evolution proceeds.

Alg. 1 Three-Stage AMCC-FNN

Initialise Synapse, Neuron and Network level

Stage 1: Synapse level encoding
Cooperatively evaluate synapse level only

while *FuncEval* $<$ *MaxGlobal* **do**
  foreach *each Subpopulation at synapse level* **do**
    Apply Evolutionary Operators
    Cooperative Evaluation
  end
  if *(Error* $\leq$ *MinError)* **then**
    break
    (Exit)
  end
  if *(ChangeMod)* **then**
    break
    (Change Modularity)
  end
end
Stage 2: Neuron level encoding
i. Carry best individuals from synapse level into neuron level
ii. Cooperatively evaluate neuron level

while *FuncEval* $<$ *MaxGlobal* **do**
  foreach *each Subpopulation at neuron level* **do**
    Apply Evolutionary Operators
    Cooperative Evaluation
  end
  if *(Error* $\leq$ *MinError)* **then**
    break
    (Exit)
  end
  if *(ChangeMod)* **then**
    break
    (Change Modularity)
  end
end
Stage 3: Network level encoding
i. Carry best individuals into network level
ii. Evaluate Network level

while *FuncEval* $<$ *MaxGlobal* **do**
  Network level Evolutionary Operators
  if *(Error* leq *MinError)* **then**
    break
    (Exit)
  end
  if *(FuncEval* $=$ *MinGlobal)* **then**
    Go back to neuron level encoding
  end
end

Note that in Stage 3, if the minimum number of function evaluations (MinGlobal) is reached and the problem has not been solved by reaching the MinError, then the framework assumes that there is convergence in local minimum. Therefore, the modularity is changed to the neuron level in order to employ different levels of evolution. In this case, the best individual from the Network level is also copied to the neuron level along with its fitness. This is also shown in the general AMCC framework in Figure 2, which gives a visualisation of how the framework transforms the evolutionary process with different levels of encoding.

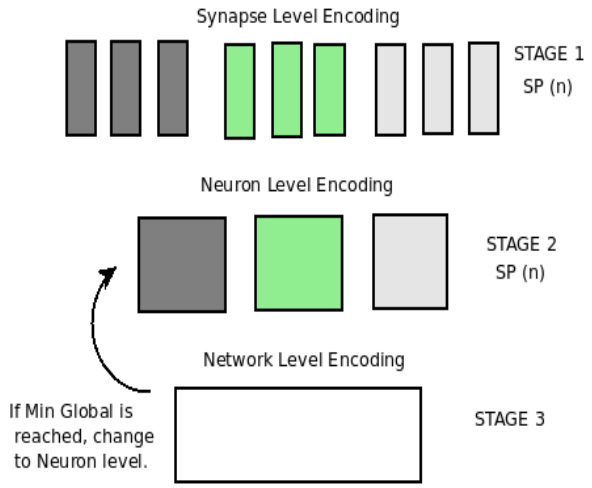


Fig. 2. The 3 Stage AMCC framework. The figure shows how the framework transforms the evolutionary process with different levels of encoding. The subpopulations *SP (n)* at the synapse level and the neuron level are shown. The *n* represents the number of subpopulations. Note that in Stage 3, if the problem is not solved before the global minimum time in terms of function evaluations is reached, then the modularity is changed from the network level to the neuron level.

B. Transfer of Valuable Information

The transition from one level of modularity to another ensures that the information gained using the existing modularity is transferred to the next level of encoding. In the AMCC framework, the best individuals in each population at each level are transferred to the next level. The transfer of information requires the best individuals from the respective subpopulations in Synapse and neuron level to be concatenated.

Note that the number of subpopulations in each level are different. The synapse level encoding has more subpopulations than the neuron level, however, information from the best individuals synapse level must be added to neuron level.

Note that neuron level encoding contains several subpopulations while network level encoding contains one population only. The best individuals from all the subpopulations are concatenated and copied into the single population of the Network level. Consequently, the best solution obtained from the neuron level encoding is transferred and retained in the network level encoding using elitism. It is important to retain the positions of all the synapses during the transfer. Information will be lost if the synapses are placed in the wrong position.

In changing from Synapse to neuron level, the best fitness value of the best individuals in different subpopulations at the synapse level is used as the fitness of the subcomponent at the neuron level. The same is done for neuron-network level fitness transfer of best individuals.

C. A Heuristic to Change Modularity

The general idea in changing modularity is to check whether the neural network is making good improvement in terms of the overall network *Error* over time. If no or low improvement is made, then a change in modularity is done.

α and β are two small constants that act as a threshold and determine how much difference is acceptable, and if these thresholds are passed, then change in modularity is enforced. α and β can be chosen empirically with trial experiments. α will usually be bigger than β as there are greater differences in change of error during the earlier stages of evolution. During the final stage of evolution, the framework will transform from Neuron to Network level of modularity. During this stage, the change in error is lower than the initial stages.

ChangeMod is dependent on the change in error (ChangeError) which can be calculated as follows.

$$ChangeError = Error[current - n] - Error[current]$$

where $Error$ is the *mean-squared-error* of the neural network. $current$ is the current cycle or generation and n is the number in previous time steps the error has to be captured and deducted from the current error. The heuristic of changing the modularity from the synapse level to neuron level is given as follows.

If($ChangeError < \alpha$)
Set $ChangeMod$ to TRUE

This heuristic will be used in the AMCC framework shown in Algorithm 1. Once ChangeMod is TRUE, the modularity is changed from the synapse level to the neuron level. Similar approach is used for the transfer from the neuron level to network level where β is used as the threshold.

The optimal value for the change of modularity can be determined by doing experimental runs with iterative values for Error change in the synapse-neuron level and the neuron-network level. This is shown in the experimental section and compared with the heuristic method. Note that this framework is not confined to three stages (synapse-neuron-network) of modularity adaptation only. More stages can be incorporated according to the nature of the problem; however, the performance can deteriorate if more levels are used than required. In Stage 3 of Algorithm 1, the system will return to the neuron level if convergence is not achieved within a specified time which is close to Maxtime. This means that if local convergence is achieved at the network level, the AMCC framework will revert to the neuron level.

IV. Results and Analysis

This section presents an experimental study of the proposed AMCC framework and compares it with the neuron level (NL)[12] and the synapse level (SL) [13] encodings for training feedforward networks on pattern recognition problems. Comparison with an evolutionary algorithm (EA) is also done.

The G3-PCX evolutionary algorithm [14] is used in SL, NL, EA [14] and AMCC framework for comparison. The population size of 100, 2 offspring and 2 parents are used. This set-up has been used in [14] and [12]. The individuals in the respective populations are seeded with random real numbers in the range of [-5, 5] in all experiments.

A. Real-World Problems and Neural Network Configuration

The datasets chosen from the UCI machine learning repository [15] are *Iris, Wine* and *Zoo* classification.

Table I shows the neural network configuration and dataset details used for all the experiments. 70 percent of the data is used for training. The maximum training time given by the number of function evaluations in all the problems was fixed as 15000 for Iris and Wine classification and 30000 for Zoo classification. The table also shows the minimum training performance required for each problem. In all the problems, the network is trained until the minimum classification accuracy of 95 % is reached on the training data. The network topology configuration for each problem is given in Table I.

In addition to the three real world problems, the 4-Bit parity problem was used where an even parity is determined by the even number of 1s in the input. For the 4-bit-parity problem, the network is training until the mean-squared error goes below 0.001. The maximum number of function evaluations is 20000.

TABLE I

Dataset information and neural network configuration

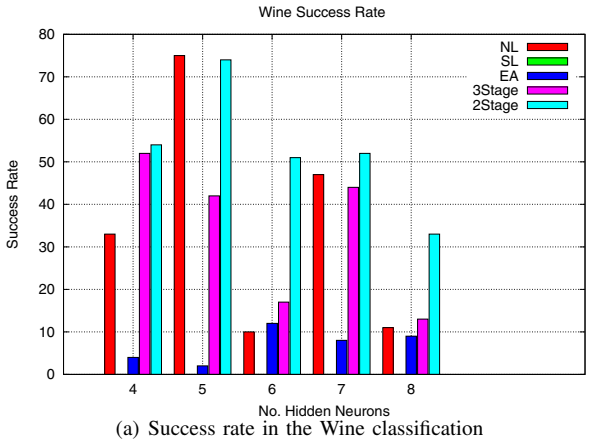
Problem	No. Cases	Class	Input	Out.	Min. Train (%)
4-Bit	16	2	4	1	100
Wine	178	3	13	3	95
Iris	150	3	4	3	95
Zoo	102	7	16	7	95

B. Experiments: The Scalability of AMCC

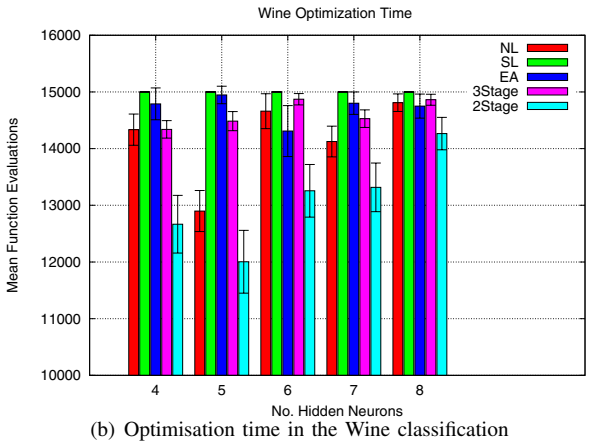
This section evaluates the performance of the AMC-FNN framework and compares it with neuron, synapse level encoding and evolutionary algorithms (EA). The EA employs the G3-PCX. A 2-Stage AMCC-FNN framework is also used here where only the neuron and network level are used; i.e. the neuron level is employed initially and later modularity is adapted to the network level. The 3-Stage AMCC-FNN framework employs the synapse, neuron and network level encodings as shown in Figure 2 and Algorithm 1. In both cases (2-Stage and 3-Stage AMCC), the framework reverts to the neuron level encoding from network level if desired convergence is not achieved within 70 percent of the maximum time. This is done in order to ensure that local convergence is not achieved using the network level encoding.

For the change of modularity in the 3-Stage framework, the α of 1E-2 is used to change from synapse to neuron level. The β of 1E-5 is used to change from neuron to network level. In the 2-Stage framework, 1E-5 is used to change from neuron to network level. These values have been determined during trial experiments.

The goal of the experimental study is to observe the performance of the respective algorithms in relation to a particular topology; i.e. fixed number of hidden neurons. Note that the number of hidden neurons directly influences the difficulty of the learning problem. It is more difficult to learn the problem if enough neurons are not present in the hidden layer. It is also difficult for evolutionary algorithms to optimise a problem when their chromosome size increases according to the number of hidden neurons.

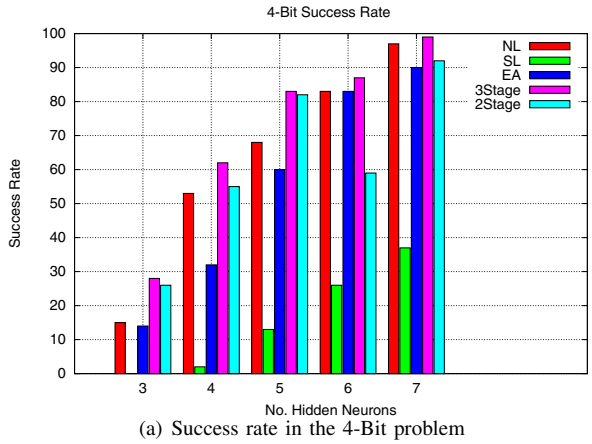


(a) Success rate in the Wine classification

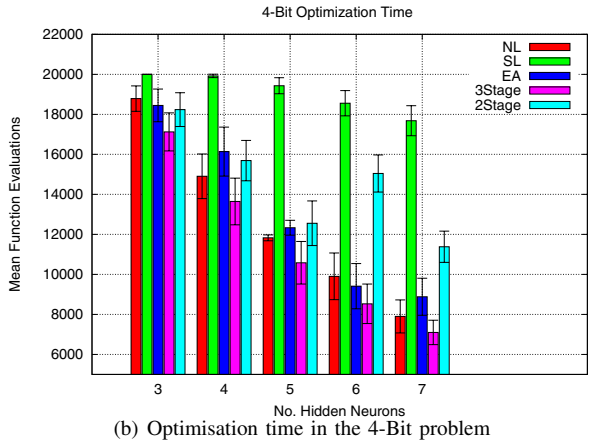


(b) Optimisation time in the Wine classification

Fig. 3. The performance the 2 Stage and 3 Stage AMCC-FNN framework on different number of hidden neurons for the Wine classification problem. The performance of NL, SL and EA is also given. The success rate is shown in (a) while the optimisation time in terms of the number of function evaluations is shown in (b).

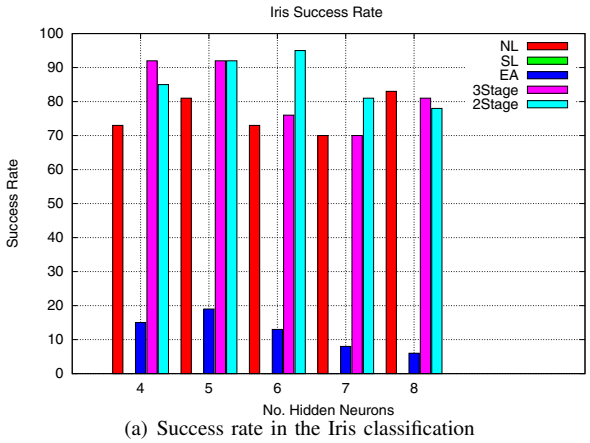


(a) Success rate in the 4-Bit problem

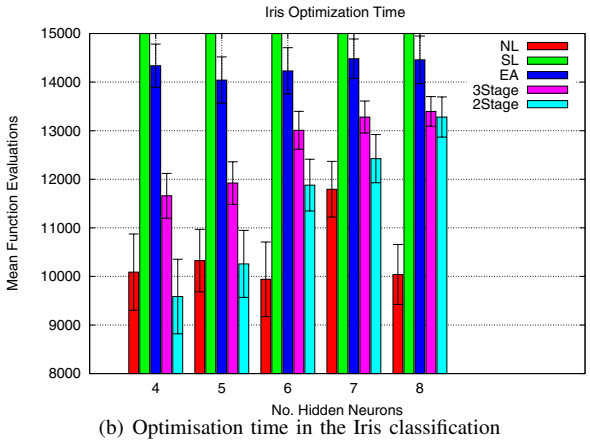


(b) Optimisation time in the 4-Bit problem

Fig. 4. The performance of 2 Stage and 3 Stage AMCC-FNN on different number of hidden neurons for the 4-Bit problem. The performance of NL, SL and EA is also given. The success rate is shown in (a) while the optimisation time in terms of the number of function evaluations is shown in (b).



(a) Success rate in the Iris classification



(b) Optimisation time in the Iris classification

Fig. 5. The performance of the 2 Stage and 3 Stage AMCC-FNN framework on different number of hidden neurons for the Iris classification problem. The performance of NL, SL and EA is also given. The success rate is shown in (a) while the optimisation time in terms of the number of function evaluations is shown in (b).

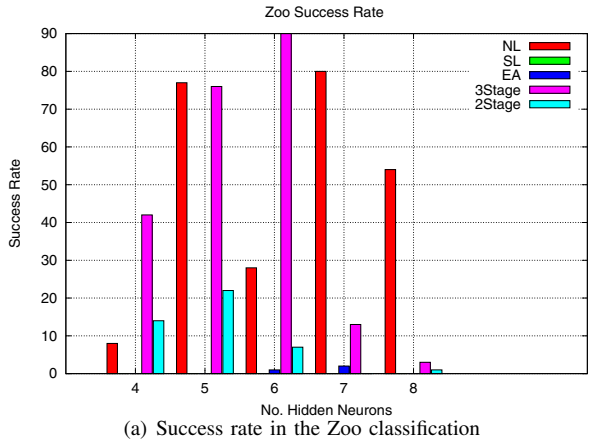
The results are given in Figures 3 to 6. The confidence interval is shown as error bars in the histograms showing the optimisation time. In the wine classification problem shown in Figure 3, NL, SL and EA do not scale well for all the different number of hidden neurons in (a) and (b). The 2-Stage framework shows the best performance given by the high success rates in (a) and least optimisation time in (b). It shows better performance when compared to the 3-Stage framework. SL encoding failed to converge in all the cases.

In the results of the 4-Bit parity problem shown in Figure 4, the 3-Stage framework gives the best performance followed by EA in terms of scalability shown by the high success rates in (a) and the least optimisation time in (b). NL shows better performance than the 2-Stage framework in terms of the optimisation time in (b). The SL encoding scheme fails to converge for 3 hidden neurons and shows the worst performance.

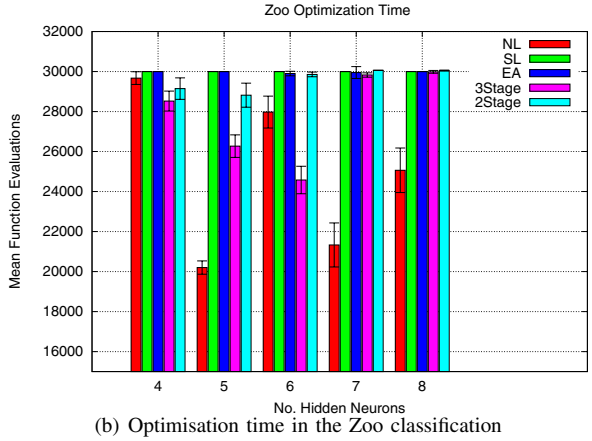
In the results of the Iris classification problem shown in Figure 5, the 2-Stage framework gives the best results in terms of the least optimisation time for 4 and 5 hidden neurons as shown in (b). NL takes the least optimisation time for 6 to 8 neurons in (b). The 2-Stage and 3-Stage gives the highest success rates in (a). The SL encoding scheme fails to converge within the maximum time in all the cases. EA also shows poor performance.

In the results of the Zoo classification problem shown in Figure 6, the 2-Stage framework, EA and SL shows poor performance. The NL and 3-Stage framework shows better performance, however they do not scale well for different number of hidden neurons. The best performance is achieved by the 3-Stage framework for 4 and 6 hidden neurons in (a) and (b), while NL gives the best performance for 5, 7 and 8 hidden neurons in (a) and (b).

In general, SL gives poor performance in all four problems. EA gives poor performance in all problems. In most cases, it had difficulty to converge within the maximum number of function evaluations. In Wine classification, the 2 Stage framework showed better performance, while in the 4-Bit parity problem, the 3-Stage framework showed the best performance. In the Iris classification, the 2 Stage



(a) Success rate in the Zoo classification



(b) Optimisation time in the Zoo classification

Fig. 6. The performance the 2 Stage and 3 Stage AMCC-FNN framework for the Zoo classification problem. The performance of NL, SL and EA is also given. The success rate is shown in (a) and the optimisation time in terms of the number of function evaluations is shown in (b).

framework showed better performance for lower number of hidden neurons where NL encoding scheme showed better for greater number of hidden neurons. In Zoo classification, better performance is achieved by the 3-Stage framework and NL.

Although the goal of this paper is not the generalisation performance, Table II gives a comparison of NL and 3-Stage AMCC framework for Iris and Wine classification performance. The Iris classification performance shows minor improvement for 5 or more hidden neurons while the Wine classification problem achieves similar generalisation using both methods. The subscript for the corresponding values in Table II gives the 95 % confidence interval. Note that 30 % of the data has been used for testing the generalisation performance.

C. Discussion

The results in general suggest that multi-stage (2 or 3 Stage) achieve the best performance in most cases.

The difference in the performance of the different methods indicates that the nature of the neural network training changes during evolution, which is why adaptation is necessary. The nature of the change during training is dependent on the problem type, learning difficulty and adaptation in terms of the degree of non-separability.

The nature of the problem also changes when different numbers of hidden neurons are used to represent the problem. The problem becomes too difficult to be solved when there is not sufficient neurons present in the hidden layer or when there are more than enough neurons present and the problem size increases. This can be seen in the performance of the Iris classification in Figure 5 where the 2-Stage framework shows best performance for 4 and 5 hidden neurons and deteriorates in performance for 6 to 8 neurons. The success of the 2 Stage evolution in this case indicates that when a lower number of neurons are present, a greater level of non-separability is needed and hence the change in modularity is needed. However, for 6 to 8 neurons, the success of NL where no modularity is adapted indicates that the level of non-separability is not increased during later stages in evolution. Here it seems that the level of non-separability related to the number of hidden neurons.

TABLE II
GENERALISATION PERFORMANCE OF 3-STAGE AMCC AND NL

Problem	Hidden	3-Stage AMCC		NL	
Iris	4	95.35	0.46	95.58	0.47
	5	96.46	0.31	94.74	0.73
	6	96.94	0.25	95.93	0.47
	7	97.03	0.26	95.37	0.61
	8	97.31	0.14	95.24	0.54
Wine	4	93.36	0.76	93.93	1.44
	5	91.56	0.95	92.50	0.92
	6	90.44	0.88	90.90	1.13
	7	94.66	0.87	93.39	0.81
	8	92.34	1.23	91.75	1.96

V. CONCLUSIONS AND FUTURE WORK

This paper introduced a novel cooperative co-evolution framework for adapting modularity during evolution for training feedforward neural networks.

Our experiments indicate that in training feedforward neural networks, the nature of the problem changes as the problem is being learnt, in terms of the degree of non-separability between interacting variables. The success of the proposed adaptive modularity framework indicates that during the later stages of evolution, the strength of the connection between the interacting variables (weights) increases and the problem becomes more non-separable. This requires larger sub-populations, and finally, evolution with a single population only. This is the main reason that canonical cooperative coevolution problem decomposition methods like neuron level and synapse level encoding failed to perform as they have a fixed problem decomposition method throughout the entire evolution. The AMCC framework has the feature of adaptation for different degrees of non-separability which delivered improved performance when compared to the neu-

ron and the synapse level encoding as the size of the problem increased.

The level of non-separability has only been considered for the neural network weights through modularity adaptation, however, there exist dependencies and interacting variables within the training data which have major effect on the network. This was observed for the four different problems. For instance, in the 4-bit parity problem, the 3-Stage framework has been effective and for the Wine classification, the 2-Stage framework has been effective. This shows that the training data is a major factor in determining the change of modularity in evolution. Hence, future work can be directed towards building neural network architectures and models based on the nature of the training data; i.e. dependencies among the attributes in the training data.

Neural network training requires a global search in the initial stage and a local search in the final stage for further refining the solution. The two main issues of concern are 1) global versus local search and 2) degree of non-separability. In most problems, a global search is needed in the beginning and refinements using local search are required during the final stages. The second issue is the balance in the degree of the non-separability. Future research questions can be directed to find out how the degree of non-separability affects global and local search.

REFERENCES

[1] M. A. Potter and 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*. London, UK: Springer-Verlag, 1994, pp. 249–257.

[2] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 2001.

[3] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, vol. 178, no. 15, pp. 2985–2999, 2008.

[4] M. Omidvar, X. Li, and X. Yao, "Cooperative co-evolution with delta grouping for large scale non-separable function optimization," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010, pp. 1 –8.

[5] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.

[6] N. García-Pedrajas and D. Ortiz-Boyer, "A cooperative constructive method for neural networks for pattern recognition," *Pattern Recogn.*, vol. 40, no. 1, pp. 80–98, 2007.

[7] F. Gomez and R. Mikkulainen, "Incremental evolution of complex general behavior," *Adapt. Behav.*, vol. 5, no. 3-4, pp. 317–342, 1997.

[8] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by evolino," *Neural Comput.*, vol. 19, no. 3, pp. 757–779, 2007.

[9] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "COVNET: a cooperative coevolutionary model for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 575–596, 2003.

[10] ——, "Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks)," *Neural Netw.*, vol. 15, no. 10, pp. 1259–1278, 2002.

[11] F. J. Gomez, "Robust non-linear control through neuroevolution," PhD Thesis, Department of Computer Science, The University of Texas at Austin, Technical Report AI-TR-03-303, 2003.

[12] R. Chandra, M. Frean, and M. Zhang, "An encoding scheme for cooperative coevolutionary feedforward neural networks," in *AI 2010: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, J. Li, Ed. Springer Berlin / Heidelberg, 2011, vol. 6464, pp. 253–262.

[13] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *J. Mach. Learn. Res.*, vol. 9, pp. 937–965, 2008.

[14] K. Deb, A. Anand, and D. Joshi, "A computationally efficient evolutionary algorithm for real-parameter optimization," *Evol. Comput.*, vol. 10, no. 4, pp. 371–395, 2002.

[15] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://archive.ics.uci.edu/ml/datasets.html

ACKNOWLEDGEMENT

The first author wishes to express his gratitude towards Mohammad Omidvar and Edwin Leong for their comments.