Proc. Congress on Evolutionary Computation CEC, Edinburgh, Vol 1, pp 2691-2698

Co-evolutionary Modular Neural Networks for Automatic Problem Decomposition

Vineet R. Khare, Xin Yao
School of Computer Science
The University of Birmingham
Birmingham B15 2TT, UK
{ v.r.khare, x.yao }@cs.bham.ac.uk

Bernhard Sendhoff, Yaochu Jin, Heiko Wersing
Honda Research Institute Europe GmbH
Carl-Legien-Straße 30
D-63073 Offenbach/Main, Germany
{ bs, yaochu.jin, heiko.wersing }@honda-ri.de

Abstract- Decomposing a complex computational problem into sub-problems, which are computationally simpler to solve individually and which can be combined to produce a solution to the full problem, can efficiently lead to compact and general solutions. Modular neural networks represent one of the ways in which this *divide-and-conquer* **strategy can be implemented. Here we present a co-evolutionary model which is used to design and optimize modular neural networks with task-specific modules. The model consists of two populations. The first population consists of a pool of modules and the second population synthesizes complete systems by drawing elements from the pool of modules. Modules represent a part of the solution, which co-operates with others in the module population to form a complete solution. With the help of two artificial supervised learning tasks created by mixing two sub-tasks we demonstrate that if a particular task decomposition is better in terms of performance on the overall task, it can be evolved using this co-evolutionary model.**

1 Introduction

Decomposing a complex computational problem into sub-problems, which are computationally simpler to solve individually and which can be combined to produce a solution to the full problem, can efficiently lead to compact and general solutions. Ideally for a good decomposition, these sub-problems will be much easier than the corresponding monolithic problem. In most cases such a decomposition relies on human experts and domain analysis. A system that can produce modules, which solve a subset of a big problem, can save us from manually crafting them. Ideally, both - the number of modules and the role that each one plays in the solution, should emerge automatically within the system. This work is an attempt towards developing a system that would automatically discover natural decompositions of complex problems, while simultaneously solving the sub-problems.

Many problems can only be decomposed into interdependent subcomponents hence changes in the role of one subcomponent effects the others. So we want the solutions to co-adapt and collectively solve the problem. Co-evolution is well suited for modelling the interdependencies among the subcomponents of the system and has been used in the literature to implement the *divide-and-conquer* strategy for tackling complex computational problems. These co-evolutionary methods can be subdivided further into two categories – single and two-level co-evolutionary methods. In single level co-evolutionary methods [14, 15, 16] the subcomponents/modules are evolved in separate genetically isolated sub-populations and fitness evaluations for these individuals are carried out by combining representative individuals from these subpopulations and then passing back the fitness of the system, thus created, to the representative individual. In two-level co-evolutionary methods [3, 7, 12] modules and complete systems are co-evolved in two separate populations.

Here we use one such two-level co-evolutionary method to design and optimize modular neural networks which have sub-task specific modules. A module represents a part of the solution, which co-operates with others in the module population to form a complete solution. Fitness of individuals in the module population is determined by their contribution towards various systems in system population. Evolutionary pressure to increase the overall fitness of the two populations provides the needed stimulus for the emergence of the sub-task specific modules.

The rest of the paper is organised as follows. In Sec. 2 a brief background on automatic problem decomposition and its relationship with modularity in neural networks is presented. Sec. 3 describes the artificial tasks, made up of other sub-tasks, that we use to evaluate our co-evolutionary model for its ability to evolve sub-task specific modules. Also, in this section we test our hypothesis that a pure-modular structure is indeed beneficial in solving the complete task. In Sec. 4 we describe the co-evolutionary model in two stages. In *stage 1* to test the effectiveness of our model, we proceed with a heavily constrained system, which makes the decomposition into modules much simpler. In *stage 2*, we remove those constraints and test our system on a much more generic decomposition problem. We then list the experiments and results in Sec. 5 followed by a discussion on these results in Sec. 6. Finally we conclude in Sec. 7.

2 Automatic Problem Decomposition & Modular Neural Networks

In the context of learning, we can think of *sequential* and *parallel* problem decompositions. Sequential decomposition, where we try to divide the overall learning task into steps (examples include feature selection methods [13, 17] and learning from hints [1]). Parallel decomposition involves dealing with sub-tasks simultaneously but separately

e.g. the classic what-and-where vision task (examples include feature decomposition methods [4, 8], mixture of experts [5] and class relations based decomposition methods [10]). In the literature, for parallel decomposition, we usually find the following instances:

- One instance one sub-task [5, 10]

$$f^t(\vec{X},\vec{Y}) = f_1(\vec{X}) \quad OR \quad f_2(\vec{Y}) \tag{1}$$

- Sub-tasks on separate outputs [4, 6]

$$\vec{f}(\vec{X},\vec{Y}) = \{f_1(\vec{X}), f_2(\vec{Y})\} \tag{2}$$

- Combination of sub-tasks at one output [8]

$$f(\vec{X},\vec{Y}) = g(f_1(\vec{X}), f_2(\vec{Y})), \tag{3}$$

where $f^t(\cdot)$, $\vec{f}(\cdot)$ and $f(\cdot)$ are instances of tasks made up of sub-tasks $f_1(\cdot)$ and $f_2(\cdot)$. $\vec{X}$ and $\vec{Y}$ are subsets of the attributes of the problem, which may or may not be overlapping. In the first two instances the decomposition is known a priori and we have separate feedback available to the learning system for separate sub-tasks. This can be used to embed a priori knowledge into the system and possibly train the modules independent of each other. For this work, we are interested in the third instance where we have much less knowledge about the problem (we do not know the function g).

To achieve this kind of problem decomposition we use modular neural networks, where modules solve various subcomponents of the problem. For instance for the problem decomposition in eq. 3, we require a modular neural network with three modules, two for problems $f_1(\vec{X})$ and $f_2(\vec{Y})$ and one for the combination function g.

Modularity in neural networks can prove useful for many reasons. Modular neural networks can incorporate a priori knowledge, generalize well, avoid temporal and spatial cross-talks [5]. All of these are useful in problem decomposition but with an emphasis on *automatic* problem decomposition, for this work, we are mainly left with the performance of the network on the complete problem as a measure of its usefulness. We use a separate test set error for this purpose and explore the possibility of using this measure to evolve modular neural networks consisting of sub-task specific modules.

3 Test Problems & Candidate Solutions

To test the feasibility of using co-evolution as a means of evolving sub-task specific modules for a given task, we construct an artificial time series prediction task (eq. 3) by combining two sub-tasks using a linear and a nonlinear combination function. Mackey-Glass (MG) [11] and Lorenz(LO) [9] time series prediction problems are used as the two sub-tasks. The Mackey-Glass time series is generated by the following differential equation using the fourth order Runge-Kutta method with initial condition $x(0) = 0.9$ and time step of $\Delta_1 = 1$.

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{10}(t-\tau)}, \tag{4}$$

where $\alpha = 0.2$, $\beta = -0.1$, $\tau = 30$. The z- component of the 3-dimensional Lorenz time series is used to create the mixture problem and is generated by solving the following differential equation system, again, using the fourth order Runge-Kutta method with a time step of $\Delta_2 = 0.02$.

$$\begin{aligned} \dot{x}(t) &= -\sigma(x(t) - y(t)) \\ \dot{y}(t) &= -x(t)\cdot z(t) + r\cdot x(t) - y(t) \\ \dot{z}(t) &= -x(t)\cdot y(t) - \beta\cdot z(t), \end{aligned} \tag{5}$$

where $\sigma = 16$, $r = 45.92$, $\beta = 4$. Now these two time series (MG and LO) are mixed according to Table 1 to create the mixture problems. Both sub-tasks involve prediction of the time series at time step t based on three previous time steps. For instance for Mackey-Glass task, MG(t) is to be predicted using MG(t-3Δ_1), MG(t-2Δ_1) and MG(t-Δ_1). The complete task is to predict g(MG(t), LO(t)). For all problems thus created the only feedback, the network (modular or not) gets, is its performance on the combined task (g). These two time series are relatively independent with a correlation coefficient of 0.032 for 1500 points and hence create mixture problems which favour a decomposition into independent modules. A linear combination problem is constructed by using g as averaging ($g(MG(t), LO(t)) = 0.5(MG(t) + LO(t))$) and a nonlinear combination problem is constructed using g as a product ($g(MG(t), LO(t)) = MG(t)\cdot LO(t)$) function. Let us call them *Averaging* and *Product* problems, respectively. For the MG-LO problem, in Table 1, decomposition of features into Mackey-Glass and Lorenz is an example of parallel decomposition and calculation of MG(t) and LO(t) as an intermediate step while predicting g is an example of sequential decomposition, as discussed in Sec. 2. Given this setup (Fig. 1) we want our co-evolutionary model to come up with a system with three modules. Two modules, M_{MG} and M_{LO}, that take inputs only from one source (Fig. 2(b)) or, in other words, predict the output for a single time series and a third that predicts their combination function (M_g).

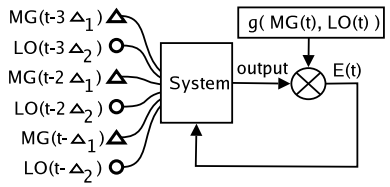

Figure 1: Problem Construction

Now let us consider four different networks in Fig. 2. These two-level RBF networks represent four different decompositions for the combined problem. In order to validate our assumption that the pure-modular structure (Fig. 2(b)), being the intuitive decomposition for the problem, is the optimal one we compare the learning curves of all these structures for the combined problem. In addition, we also compare them to the *pre-trained pure-modular structure*. As the

Problem	Inputs						Prediction Task
Mackey-Glass	MG(t-3Δ_1)		MG(t-2Δ_1)		MG(t-Δ_1)		MG(t)
Lorenz-z		LO(t-3Δ_2)		LO(t-2Δ_2)		LO(t-Δ_2)	LO(t)
MG-LO	MG(t-3Δ_1)	LO(t-3Δ_2)	MG(t-2Δ_1)	LO(t-2Δ_2)	MG(t-Δ_1)	LO(t-Δ_2)	g(MG(t), LO(t))

Table 1: Artificial time series mixture problems

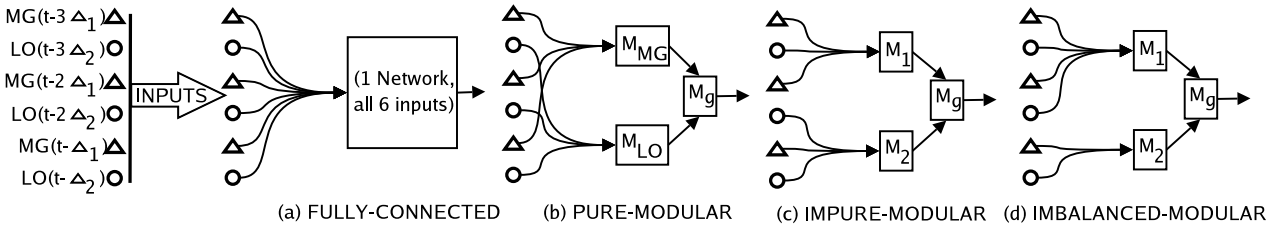

Figure 2: Four two-level RBF network structures representing four possible problem decompositions.

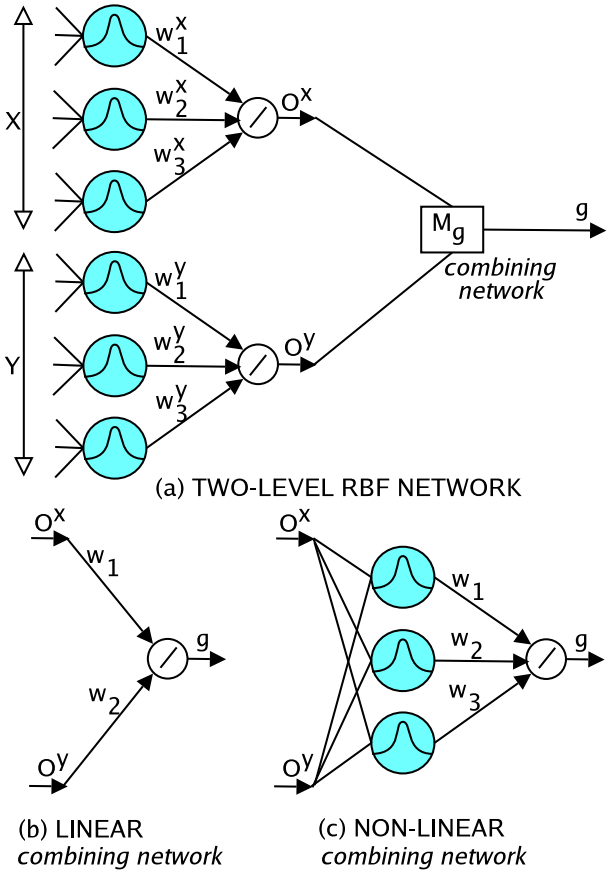

Figure 3: A two-level (modular) RBF Network.

name suggests, modules in pre-trained pure-modular structure are trained separately from each other on individual sub-tasks. Since this structure has separate feedback available from all the modules, which is not the case with any other structure and has pure modules, it can be used as a base case / ideal solution to the combination problem.

Fig. 2 shows four different networks (fully-connected network, pure-modular network, impure-modular network and imbalanced-modular network). For linear problems in *stage 1* of experiments (Sec. 5) the combination module M_g, performs averaging of the other two module outputs (Fig. 3(a), 3(b)). For generic non-linear problems in *stage 2* of experiments, the outputs of the modules need to be combined non-linearly. For this purpose we use a RBF network as the combination module (Fig. 3(a), 3(b)), let us call it *combining network*. Within 100 epochs of stochastic gradient descent (SGD) [2] learning, pure-modular structure emerges as the best structure for both types of problems. Corresponding learning curves are shown in Fig. 4 for the *Averaging* problem. Similar results are also obtained for the *Product* problem but are omitted here for space constraints. We also observe the sub-task specialization in the modules of the pure-modular structure. This specialization in a two-level RBF network is shown in Fig. 5 for the *Product* problem. Also, the performance of the pure-modular structure is comparable to that of pre-trained pure-modular structure.

These comparisons indicate that the pure-modular structure represents a good decomposition of the task, especially because it fares well against the pre-trained pure-modular structure, but only a few other structures are tested and it can not be claimed that it is the optimal one. We can expect our co-evolutionary model to find the built-in decomposition of the problem only if the corresponding structure is better than other possible structures, so we expect it to find either this pure structure or other structures (at least) equivalent in performance to this structure.

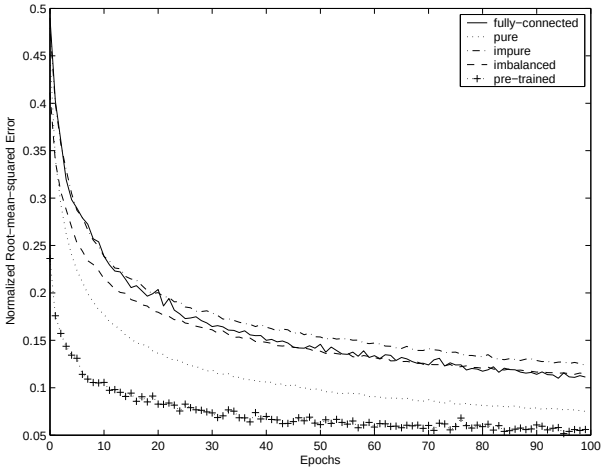

Figure 4: Learning curves for two-level RBF network structures shown in Fig. 2. SGD ($\eta = 0.01$, $\mu = 0.00$) learning for *Averaging* problem. Average of 30 runs. Fully-connected network has 79 parameters, others have 72.

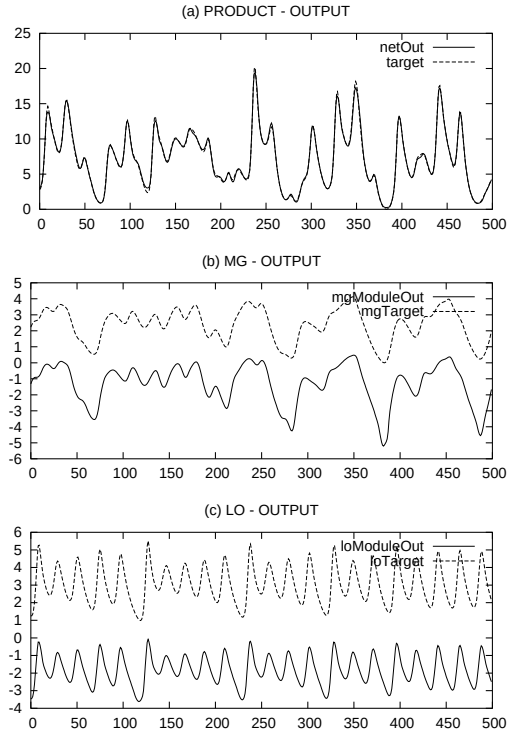

Figure 5: Specialization in a modular two-level RBF Network. (a) Target output for *Product* problem and output of two-level RBF network on 500 test data points. (b) Target output for MG sub-task and output of MG module on 500 test data points. (c) Target output for LO sub-task and output of LO module on 500 test data points.

4 Co-evolving Problem Decomposition

We use a two level co-evolutionary architecture (Fig. 6 and 7), the lower level (level 1) has a population of modules and the higher level (level 2) has a population of complete systems made up of modules from module population. Within this general co-evolutionary framework, both the structure and parameterization of the modules as well as the parameterization and structure of the systems can be evolved. Evolution at level 1 searches for good building blocks for the system and at level 2 it searches for good combinations of modules in level 1. This kind of co-evolutionary architecture has been used before [3, 7, 12] where neurons and artificial neural networks were used as modules and systems, respectively. We use a similar approach, in *stage 1*, for the linear combination problem and further extend it up a level, in *stage 2*, to use networks and their combinations (two-level RBF networks) as the lower and the higher levels, respectively, for the non linear combination problem.

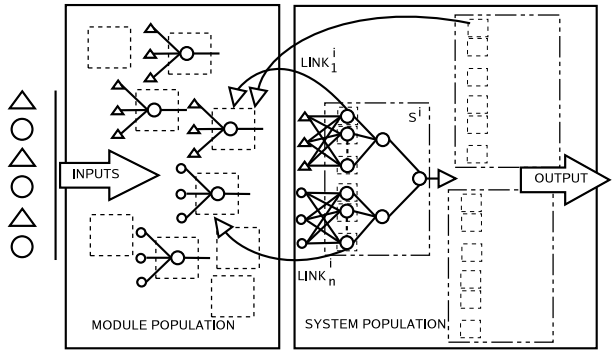

Figure 6: Co-evolutionary Model for the linear combination problem. Each individual in module population is a Gaussian neuron and each individual in system population contains two RBF (sub) networks made out of these neurons in module population.

- *Stage 1*: In case of the linear combination (*Averaging*) problem, each system in system population represents (Fig. 6) a combination of two RBF networks which have inputs complimentary to each other, e.g., if one network takes 3 inputs from one of the time-series then the other takes the remaining 3 inputs, thus representing the complete solution to the problem. The output of the combination is the average output of the individual networks. Modules are Gaussian neurons differing from each other in terms of parameters (centres and widths) and also in terms of the inputs that they get out of all 6 inputs from the combined problem, hence each module can have between 1 and 6 inputs. Here we evolve structure and parameters of the network together. Each system is trained partially (refer to Table. 2) in each generation.
- *Stage 2*: For non-linear combination (*Product*) problems, modules are RBF networks (Fig. 7) differing from each other in terms of the inputs that they get out of all 6 inputs from the combined problem, hence

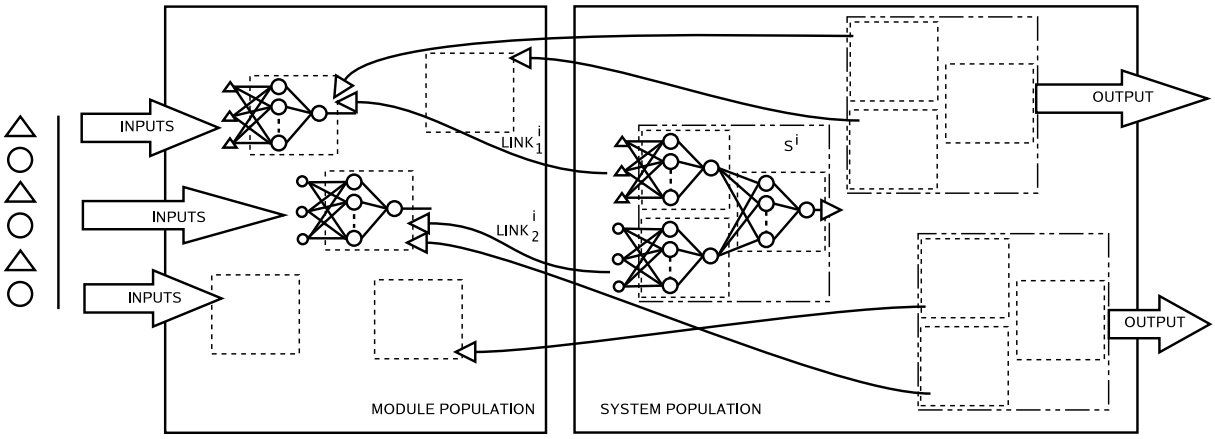

Figure 7: Co-evolutionary Model for the non-linear combination problem. Each individual in module population is a RBF network and each individual in system population contains a RBF network and pointers to two modules in module population.

each module can have between 1 and 6 inputs. Each system contains pointers to two modules in module population and the output of the system is obtained with the help of *combining network*, with 2 inputs and 1 output. Here we do not evolve parameter values for these networks and others in module population because we allow modules to change structure and after adding or deleting an input to a module the previous parameter values might not remain any good. Hence each system needs to be trained in every generation.

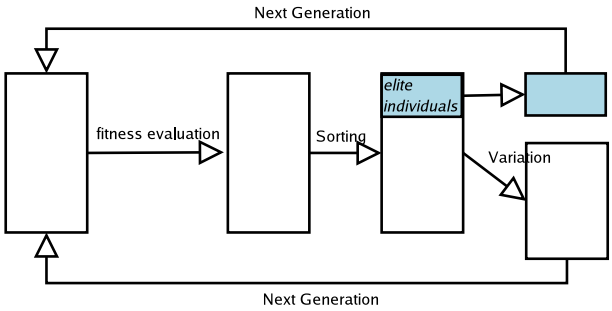

Figure 8: Both populations undergo these stages in a generation. In addition all individuals in system population undergo partial training before fitness evaluation.

In both both stages, if a pure modular structure is advantageous over others then we expect that some of the modules in module population will specialize in MG prediction task and some in LO prediction task or in other words starting from a completely random configuration we will converge to a state where some modules will take their inputs only from Mackey-Glass time-series and some from Lorenz time-series. In each generation both populations go through the stages illustrated in Fig. 8. Systems are trained using SGD learning, with random parameter initializations, in each generation and, depending on their performance on validation data set, are assigned with a fitness value. Fitness of system S_i is evaluated as,

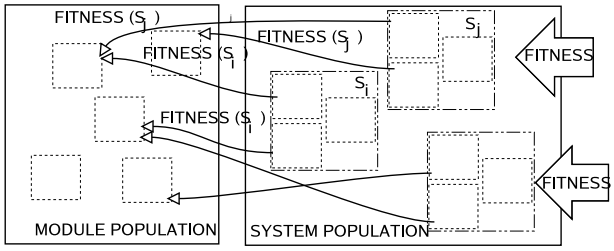

Figure 9: Individuals in system population get their fitness based on performance on validation set, while modules derive it from the systems they participate in.

$$FITNESS(S_i) = 1/(nrmse + c), \tag{6}$$

where $nrmse$ is the normalised root-mean-squared error on validation data, obtained as the ratio of root-mean-squared error and the standard deviation of target values, and c is a small constant to prevent the singularity. Modules derive their fitness from the systems (Fig 9) to which they contribute in the system population. They are to be judged on the basis of their contribution towards the complete problem. Listed below are the two credit assignment strategies that we use to evaluate module fitness. After the fitness evaluation both populations are sorted and top 25 percent of individuals in both populations are then labelled as *elite individuals*. The variation operators used in the two stages are described later.

- *Credit assignment using a few good systems*: Each module gets the summed fitness of the best 25 percent systems in which this module participates.
- *Credit assignment using module frequency*: Each module is assigned with a fitness value equal to the number of appearances it made, in any system in system population, during the last 10 generations.

In *stage 1* only mutation is used in the two populations. In module population, parameters associated with a neuron,

not in *elite individuals*, are mutated by adding normally distributed noises with 0 mean and 1 standard deviation. Mutation rate of p_m was used for each real parameter. In system population, the input positions are flipped between the two complimentary networks again using a mutation rate of p_m per input position.

In *stage 2*, in addition to mutation, crossover is also used as a variation operator in system population as we do not need to enforce complementarity among the two modules within a system. Each of the *elite individuals* in system population is crossed with another *elite individual* with a certain probability p_c to produce two offspring which replace the worst two individuals in the population. One-point crossover is used, which swaps the modules in the two systems. Systems other than *elite individuals* are then mutated, with a certain probability p_m, by replacing one of its modules with another randomly chosen module from module population. Like *stage 1* only mutation is used for module population, where each module, not in *elite individuals*, is mutated again with probability p_m. For this purpose, either one of the randomly selected inputs is deleted or a new input, randomly chosen from the ones which are not present in the module, is added to the module.

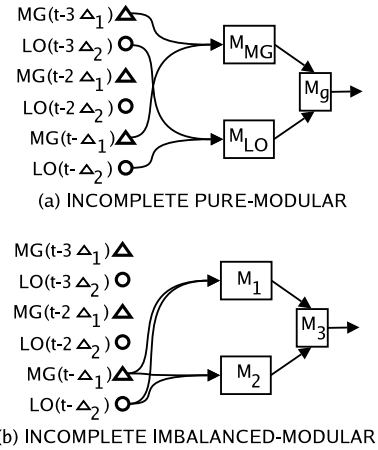

Figure 10: Two solutions for the *Product* problem from the co-evolutionary model.

5 Experiments & Results

Below, we describe the two stages (Sec. 4) of experimentation with the co-evolutionary model for *Averaging* and *Product* problems. Parameter values for these experiments are listed in Table 2. Both credit assignment strategies (Sec. 4) produce very similar results. Results listed here are with the credit assignment using a few good networks.

- *Stage 1 (Averaging problem)*: In *stage 1* of experimentation we assume that the combination function is linear and evaluate the system output as the average of its networks, which are fixed to two modules per system. A complementarity condition between the modules of a system is also assumed which says that the modules can not share inputs among themselves within a system. So only parallel decomposition (Sec. 2), in MG and LO modules, is required of the co-evolutionary model, which it achieves. With these assumptions the co-evolutionary model converges to the pure-modular structure in all of the 30 runs conducted. This was observed with both credit assignment strategies, but credit assignment using module frequency resulted in faster convergences. A population is assumed to be converged when all *elite individuals* (Sec. 4) in the system population have the same structure. Once this structure is found it is trained on the *Averaging* problem for 100 epochs of SGD learning and the results on a test set are given in Table 3. These results are comparable to our base case / ideal solution (pre-trained pure-modular structure), results for which are also listed in the table.
- *Stage 2 (Product problem)*: Here we move further towards our aim of having an *automatic* decomposition and remove assumptions about the combination function and complementarity of inputs, so the only domain knowledge we use is that there are two modules in the system. But since the two problems (MG and LO) are relatively independent (Sec. 3) we still expect complementarity to be beneficial and the co-evolutionary model to discover it. Now the problem is much harder as it requires both parallel (MG and LO modules) and sequential (first evaluation MG and LO modules and then the product module) decompositions. Here, the search space being much larger, the co-evolutionary model does not always converge to the pure-modular structure. Out of 10 runs, with credit assignment using module frequency, we only

Data	
Training data	500 points
Validation data	500 points
Testing data	500 points
Individuals	
Neurons per module	5
Neurons per *combining network* (*P*)	9
Lifetime learning	
Partial training per generation	20 epochs
Learning rate (η)	0.01
Momentum (μ)	0.0
Co-evolution	
Module population size (*A*)	480 neurons
Module population size (*P*)	80 RBF networks
System population size	40 two-level RBF networks
Number of generations (*A*)	100
Number of generations (*P*)	500
mutation probability (p_m)	0.2
crossover probability (p_m)	0.8

Table 2: Parameter values used for experimentation. Parameters marked with an *A* or a *P* are specific to either the *Averaging* or the *Product* problem, respectively.

converge to the pure-modular structure 5 times. In another 2 runs it converges to an incomplete pure modular solution (Fig. 10(a)), which after 100 epochs of SGD learning is equivalent to the pure-modular structure in terms of performance on the combined problem. This is an interesting result as it indicates that all three inputs are not needed to solve the problem. The remaining 3 runs converge to a suboptimal solution consisting of two imbalanced structures (Fig. 10(b)). Again, all these three structures obtained from all 10 runs are trained on the *Product* problem for 100 epochs of SGD learning and the results on a test set are given in Table 3 alongside the results for a pre-trained pure-modular structure.

6 Discussion

We can expect our co-evolutionary model to find built-in decomposition of the problem only if the corresponding modular neural network structure is better than other possible structures. The difference in performance between this structure and others provides the co-evolutionary model with the selection pressure towards this structure (problem decomposition). In *stage 1* of experiments, with the combination function known and with the complementarity of inputs condition enforced, we only need to discover the optimal feature decomposition for the *Averaging* problem, which we are able to do quite successfully because of two reasons. First one being the limited search space and second being the advantage of pure-modular structure over others as the combination function of the two modules is fixed (Averaging), which is suited for the pure-modular structure.

On the other hand our experiments in *stage 2* were not quite so successful, where we were only able to achieve the intuitive decomposition 7 out of 10 runs. This again can be attributed to the fact that the search space is much bigger ($2^{12} - 1 = 4095$ possible structures) as there are many different ways to decompose this problem, and there are other possible structures which are very close to the pure-modular structure in terms of performance on the complete problem. A couple of such structures are shown in Fig. 10(a) and 10(b). A few other experiments with a slightly different setup, and hence not reported in this work, indicate that the performance of the co-evolutionary model in *stage 2* depends critically on system population size. Larger system population size results in better convergence to the pure-modular structure.

In the experiments presented here, 2 out of 7 successful runs produced an incomplete-pure structure (Fig. 10(a)), which indicates that all three inputs are not needed to solve the problem. Interestingly, the remaining 3 runs converged to a structure (Fig. 10(b)) which has two imbalanced modules. Since we know that for a time series prediction problem the last time step is the most important one, this structure represents another good solution to the problem where modules M_1 and M_2 are only focusing on the two most relevant inputs and the combination (M_3) is producing an *ensemble effect* of two very similar modules. At this point we would like to emphasize the importance of the *combining network* being able to approximate the combination function (product in this case). If it is unable to approximate the function properly the decomposition in terms of MG and LO modules would not be favoured, which is quite expected because we are trying to evolve the structure for two modules while keeping the structure for the third fixed. Ideally this *combining network* should be allowed to co-adapt with other modules.

7 Conclusions

We have presented a two-level co-evolutionary model to design and optimize modular neural networks with sub-task specific modules. The first level population consists of a pool of modules and the second level synthesizes systems by drawing elements from this pool. Modules represent a part of the solution, which co-operates with others in the module population to form a complete solution. Fitness of individuals in module population is determined by their contribution towards various systems in system population. Evolutionary pressure to increase the overall fitness of the two populations provides the needed stimulus for the emergence of the sub-task specific modules. With the help of artificial tasks created by mixing two sub-tasks (MG and LO time series prediction) we demonstrated that if a particular task decomposition is better in terms of performance (mean-squared-error in this case) on the overall task it can be evolved using this co-evolutionary model. This is also evident from the emergence of some good decompositions which one would not think of while designing such a modular system manually.

This model has its share of limitations as well. Firstly, the number of modules is fixed and is supplied to the model as a priori knowledge and secondly, within these modules we are not optimizing the structure of the module which combines the other modules to produce the final output. Ideally, and in line with the idea of co-adapting all the modules together, the structure for this module (*combining network*) should also be evolved along with other modules. Adaptation of the number of modules in a system can be achieved in two different ways. First one is a stage-wise approach, where new modules are added or deleted simultaneously in all the systems present in the population. This happens only when there is a stagnation in performance of systems in system population. A more global approach is where systems with different number of modules are present at the same time in the population. To overcome the second limitation the model can be extended to co-evolve the modules for combining other modules alongside those other modules in the module population.

Finally, this generic model can be applied to a variety of problems ranging from feature decomposition and feature selection in neural network ensembles to problems which require pre-processing. Feature decomposition can be viewed as an example of parallel decomposition and, feature selection and pre-processing can be viewed as an examples of first stages in a multi-stage sequential decomposition problem.

Problem	Runs	Evolved Structure	Evolved Structure Performance	Pre-trained Structure Performance
Averaging problem (*stage 1*)	30	pure-modular (Fig. 2(b))	0.0751 (3.4754e-04)	0.0560 (8.8997e-04)
Product problem (*stage 2*)	5	pure-modular (Fig. 2(b))	0.1175 (0.0491)	0.0933 (7.0329e-04)
	2	incomplete--pure-modular (Fig. 10(a))	0.1137 (0.0524)	
	3	imbalanced (Fig. 10(b))	0.2098 (0.0335)	

Table 3: Normalized Root Mean-squared errors achieved by various structures (obtained from the co-evolutionary model) after 100 epochs of SGD ($\eta = 0.01$, $\mu = 0.00$) learning. These structures are also compared with the base case / ideal solution.

Bibliography

[1] Yaser. S. Abu-Mostafa. A method for learning from hints. In S. J. Hanson, J. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 73–80, San Mateo, CA, 1993. Morgan Kaufmann.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1996.

[3] Nicolás. García, Cesar Hervás-Martínez, and J. Muñoz-Pérez. Multi-objective cooperative coevolution of artificial neural networks. *Neural Networks*, 15:1259–1278, 2002.

[4] Michael Hüsken, Christian Igel, and Marc Toussaint. Task-dependent evolution of modularity in neural networks. *Connection Science*, 14:219–229, 2002.

[5] Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Cognitive Science*, 15:219–250, 1991.

[6] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991.

[7] Vineet R. Khare, Xin Yao, and Bernhard Sendhoff. Credit Assignment among Neurons in Co-evolving Populations. In Xin Yao et al., editor, *8th International Conference on Parallel Problem Solving from Nature, PPSN VIII*, pages 882–891, Birmingham, UK, September 2004. Springer. Lecture Notes in Computer Science. Volume 3242.

[8] Yuansong Liao and John Moody. Constructing heterogeneous committees using input feature grouping: Application to economic forecasting. *Advances in Neural Information Processing Systems*, 12:921–927, 1999.

[9] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric Science*, 20:130–141, 1963.

[10] Bao-Liang Lu and Masami Ito. Task decomposition and module combination based on class relations: A modular neural network for pattern classification. *IEEE Transactions on Neural Networks*, 10:1244–1256, 1999.

[11] Michael. C. Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197:287–289, 1977.

[12] David E. Moriarty and Risto Miikkulainen. Forming Neural Networks Through Efficient and Adaptive Coevolution. *Evolutionary Computation*, 5(4):373–399, 1997.

[13] David Opitz. Feature selection for ensembles. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI)*, pages 379–384, 1999.

[14] Gary B. Parker and H. Joseph Blumenthal. Comparison of sampling sizes for the co-evolution of cooperative agents. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 536–543, Canberra, 8-12 December 2003. IEEE Press.

[15] Mitchell A. Potter and Kenneth A. De Jong. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[16] Chern Han Yong and Risto Miikkulainen. Cooperative Coevolution of Multi-Agent Systems. Technical Report AI01-287, Department of computer Sciences, The University of Texas at Austin, Austin, TX 78712 USA, 2001.

[17] Marco Zaffalon and Marcus Hutter. Robust feature selection by mutual information distributions. In *Proc. of the 18th Conf. on Uncertainty in Artificial Intelligence*, pages 577–584, San Francisco, 2002. Morgan Kaufmann.