Biasing the Evolution of Modular Neural Networks

V. Landassuri-Moreno
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
Email: V.Landassuri-Moreno@cs.bham.ac.uk

John A. Bullinaria
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
Email: J.A.Bullinaria@cs.bham.ac.uk

***Abstract*—Modularity is known to have benefits for neural systems and their evolution, and this paper aims to improve the evolutionary neural network algorithm EPNet to take advantage of those benefits. Neural networks exist with varying degrees of modularity ranging from pure modular networks characterized by disjoint partitions of hidden nodes with no communication between modules, to pure homogeneous networks with significant connections throughout. In between are apparently homogeneous networks that can be seen to have some degree of modularity if the hidden nodes are reorganized appropriately. In this paper, a modularity measure is presented and extended that can be applied to any neuron at any level in the network to provide a fine analysis of node partitioning. It also allows the rearrangement of nodes to create modules in homogeneous networks, and that is used to improve the EPNet algorithm to evolve modular neural networks. Experimental results on a simple classification task confirm that the new modular EPNet algorithm does indeed lead to more modular networks than the classical EPNet algorithm, without compromising the performance on the given task.**

***Index Terms*—EANNs, Modular Neural Networks, Modularity, Classification.**

I. Introduction

Modular systems are usually seen as a collection of independent components that work together for specific purposes, e.g. with each component specialized to perform a particular subtask that may be used multiple times. This occurs naturally in neural networks where it is possible to have disjoint partitions of the neurons (i.e. modules). If there is no communication between modules (i.e. independent partitions), one has a pure Modular Neural Network (MNN), whilst if the nodes are highly interconnected (i.e. with no independent partitions), one has a homogeneous or non-modular network.

There is considerable evidence that MNNs can provide numerous advantages over non-modular networks, e.g., leading to faster learning for some combinations of tasks [1], or facilitating the evolution of module re-use [2]. Moreover, evolutionary algorithms have proven to be extremely useful in the search for appropriate architectures for various tasks [3], [4]. However, a practical problem for real world applications is that the advantages of modularity (particularly module re-use) will only become apparent (and hence result in a tendency for modularity to emerge through evolution) if sufficiently large patterns of successful module re-use occur in the evolutionary simulations. In practice, most simulations are far too small-scale for such modularity and its advantages for module re-use to occur. The aim of this paper is to explore how to help modularity and its advantages emerge, without the need for such computationally infeasible evolutionary simulations. This will be done in the context of the evolutionary neural network algorithm EPNet [3], which is based on Fogel's Evolutionary Programming (EP) approach.

First one needs an approach for identifying modules in a network. Some researchers have focussed on the emergence of modules within neural networks [1], [5], [6], while others have considered more general graph based representations [7]–[10]. To measure the emergence of modules in evolutionary neural network systems, one needs to develop some kind of modularity measure that indicates the degree of node partitioning during evolution [1], [5], [6], [10]. The highest degree of modularity indicates pure MNNs, the lowest values indicate pure homogeneous architectures, and intermediate values correspond to various types of partial partitioning.

Clearly it is the interaction of the modules that results in their successful behavior. That interaction usually takes the form of each *final module* gathering and processing the outputs of the other modules, but there may also be cross-connections between contributing modules. One needs to analyze the communication between modules and the neural substructures that may emerge during evolution. This paper does that by attempting to reorganize the neural network hidden nodes into modules using the dependency of each node to each module provided by an appropriately chosen modularity measure (see Sec. II). That is shown to work for any type of neural network, from pure homogeneous to pure modular. The modularity measure presented by [5] has proven to be suitable to analyze modules during evolution for the much studied what–where task [11], so that will be used here. However, it was developed for networks that contain more than one output unit, so a required contribution of this paper is to develop a modification that also works for tasks with a single output node.

The module creation process, based on node rearrangement using the improved modularity measure, allows the discovery of hidden units that contribute to more than one module and cross-connections between nodes from different modules, i.e. *shared nodes* and *shared connections* respectively. Those *shared* components then lead to the second contribution of this work, which is the improvement of the standard EPNet algorithm [3] into a modular algorithm (M-EPNet) by introducing two new mutation operators corresponding to shared node and shared connection deletion.

978-1-4244-7833-0/11/$26.00 ©2011 IEEE

Both algorithms (EPNet and M-EPNet) were tested on an artificial data set that involves performing two distinct classification tasks on the same set of real valued input data. Previous studies on that task [1] have shown that, although it can be performed by either pure modular or pure homogeneous networks, it is learned more easily by modular neural networks, and that tends to drive the evolution of modular neural architectures for it.

That earlier study [1] used Multi-Layer Perceptron (MLP) architectures with a generational algorithm to evolve networks aimed at learning the tasks as quickly as possible. Here, the EPNet algorithms is a 'steady state' approach based on the generalized MLP (GMLP) [12, pp. 272-273] which allows a larger range of connectivity patterns among nodes and therefore adds complexity to the search for pure MNNs. A crucial difference is that generational algorithms test several combination per generation, while 'steady state' algorithms only test a few. That could render it difficult for the EPNet algorithm to evolve pure MNNs, so an important further aim of this work is to explore the extent to which modularity evolves in both the EPNet and M-EPNet algorithms. That will lead to a more promising approach for investigating module re-use within this kind of algorithm, although the results from that must remain a topic for future papers.

In Section II the $module$ and $modularity$ concepts are explained, along with the chosen modularity measure and the extension of it that works for networks with a single output unit. This leads, in Section III, to a presentation of the improved EPNet algorithm that enhances modularity, followed by the experimental set-up and results in Sections IV and V. The paper ends with some conclusions in Section VI.

II. Modules and Modularity

Modularity has been much studied in recent years, and a range of definitions for modules [1], [5], [13], [14] and modularity [5], [6], [10] have emerged. For example, $modularity$ can be defined as a property of neural networks where the connectivity patterns are organized in such a way that modules consist of disjointed subsets of hidden units [1], [5]. Some definitions consider modules to be non-interacting subsystems [1], [5], while others allow them to interact [13]. In this study, communication between modules is allowed and the shared neural components among partitions are analyzed.

A suitable modularity measure applied to neural architectures can indicate the degree of modularity, i.e. how close the network is to being pure modular. The modularity measure for networks with more than one output unit is presented next, with the method used to discover shared nodes and connections. Then the extension to the single output unit case is outlined.

This study is based on the method presented in [5] to measure the degree of modularity of the networks during evolution, and further details can be found there. There were two main reasons for choosing that measure: A) it provides a way to calculate the dependency of nodes against modules, and therefore to rearrange the nodes into modules and allow the identification of shared nodes and connections, i.e. neural components shared by two or more modules; B) it is less computationally expensive than other modularity measures [6], [7]. For example, [7] is based on the modularity measure of [8], [9] that uses Simulated Annealing to find the modules inside the network. Nevertheless, the choice of measure is not crucial. Any modularity measure could be used for the same purpose (identifying shared neural components in different modules) after the required modifications.

The chosen modularity measure is based on the assumption that the neural networks have to deal with completely separable problems (non-interacting subsystems) to determine the dependency of nodes against each module. Here, it is assumed that it is known a priori how to partition the given data set into modular tasks, i.e. how many modules are appropriate and which input or output units belong to each module. The algorithm was implemented in this way (as it was developed) to keep the approach as simple as possible. For the purpose of this work, this modularity definition is sufficient to explore the interaction between modules during evolution. Further improvements may automatically determine the data partition during evolution.

The algorithm is based on the idea of data partition driven by the network's input and output nodes. The data set used in this study involves two distinct classification tasks, so there are two modules defined by the output partition. Since there are two possible partition types (either from inputs or outputs), there are at least two ways to calculate the modularity, i.e. in a "top-down" fashion (from outputs to inputs units) if the output partition is known, or in a "bottom-up" fashion (from inputs to outputs nodes) in the other case. The following explanation is focused on the top-down version, but its counterpart differs little (refer to [5] for the 'bottom-up' version).

Since a separable problem is assumed, the set of output nodes $(y_1, ..., y_n)$ can be partitioned into m disjoint subsets $(S_1, ..., S_m)$. Nodes that are connected directly or indirectly to one subset of outputs S_j are called $pure$ nodes as they only contribute to one module (output partition). The modularity measure M is defined as the average degree of pureness of the hidden and input nodes given a m-tuple $(d_i(1), ..., d_i(m))$ for each node i. This tuple indicates the degree of dependency of each node i on the m different partition or module (S_j). The m-tuple for outputs is the first to be assigned:

$$d_i(j) = \begin{cases} 1 & y_i \in S_j \\ 0 & y_i \notin S_j \end{cases} \quad (1)$$

After the $d_i(j)$ are calculated for the output nodes, the m-tuple is calculated for the hidden and input nodes recursively:

$$d'_i(j) = \sum_k |w_{ik}| d_k(j) \quad (2)$$

$$d_i(j) = \frac{d'_i(j)}{\sum_{j'=1}^{m} d'_i(j')} \quad (3)$$

where w_{ik} is the connection weight from node k to node i, and $d'_i(j)$ is a partial processing value concerning w_{ik} with

the tuple of node k at every position j. The pureness of each node i is calculated by the variance expression:

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}\left(d_i(j) - \frac{1}{m}\right)^2 \quad (4)$$

where higher σ_i^2 indicates higher pureness of node i. The maximum value of pureness possible is $\frac{m-1}{m^2}$ which corresponds to pure nodes. Finally, the modularity measure is given by the average variance of all N hidden and input nodes:

$$M^{(weight)} = \frac{m^2}{m-1}\frac{1}{N}\sum_i \sigma_i^2 \quad (5)$$

where $M^{(weight)}$ falls in the interval [0,1], with 1 indicating a completely separable network, and 0 a completely homogeneous network. If the weights were not included in equation 2, it would lead instead to a measure $M^{(arch.)}$ of the modularity only in term of the structure of the network. Therefore, it can be seen that calculating $M^{(weight)}$ produces a finer measure of the dependency of each node against all output partitions, since the weights are involved in the calculation.

At the beginning of the evolution, lower modularity values are expected because of strong interaction among modules. Then as the generations advance, the interactions are expected to decrease, producing an increase in the modularity if the given task finds modular architectures beneficial, or if the evolutionary algorithm is biased towards producing modularity.

Thus far, it has been assumed that the output partition is known, which implies the existence of two or more outputs. Thus the above $M^{(weight)}$ and $M^{(arch.)}$ can only be applied to networks with more than one output. Modifications are required for applications with a single output (see Sec. II-C).

A. Shared nodes

Shared nodes and shared connections are neural components that contribute to more than one module or output partition.

Figure 1 shows a neural network with two output nodes in its normal representation (Fig. 1a) and with the nodes rearranged into modules (Fig. 1b). Input nodes are represented at the bottom, hidden nodes in the middle, and output nodes at the top. Using equations 3 and 4 one can determine the extent to which a given node is only connected with a single output partition. In this case, each node will be bound (associated) with the module for which it presents the biggest dependency, i.e. the hidden nodes are sorted into modules.

Thus, from equation 3 and 4, nodes 1, 5, 6, 7 and 9 in Fig. 1a are pure nodes in the top-down fashion as they are only connected directly or indirectly to one output partition (module). The rest of the input and hidden nodes are shared to a certain degree. Note that in this case, only $M^{(arch.)}$ has been used to determine the purity of nodes, but $M^{(weight)}$ might give a finer grained measure.

B. Shared Connections

If certain weights are assigned to the connections in the network presented in Fig. 1a, equation 3 can be used to sort

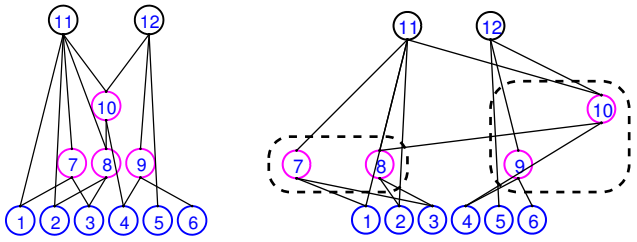

Fig. 1. Typical neural network with two output nodes, shown as normal (Fig. 1a) and modular (Fig. 1b) representations

the nodes into modules as presented in Fig. 1b. The input and output nodes are not included in the graphical representation of the modules here (i.e. nodes grouped by dashed lines), but they may be included if desired.

After the nodes have been organized into modules and shared nodes detected, it is straightforward to discover the cross-connections between modules, as illustrated in Fig. 1b where connections $c_{8,10}$ and $c_{10,11}$ are shared by two modules. Note that if the weights take different values, such shared nodes may be assigned to different modules, e.g. node 8 could be moved into the module formed with node 12.

This allows the classification of two types of connection: a) *strong connections* are the outbound connections from a given node to other units in the same module, which resulted in the node belonging to that module; b) *weak connections* are the cross-connections between units from different modules (i.e. shared connections between modules).

Note that equation 3, leading to $M^{(weight)}$, allows a unique rearrangement of nodes into modules, and shared nodes and connections can be found in the same modularity calculation. However, that is not possible using equation 3 without the weights, leading to $M^{(arch.)}$, because it only measures the architecture modularity. For example, there is no way to know if node 10 contributes more to the output partition formed by node 11 or the partition composed by node 12 without considering both weights $w_{10,11}$ and $w_{10,12}$ in Fig. 1a.

C. Improved Modularity Measure

It is not straightforward to apply the above modularity measure to networks with only a single output unit. In that case, the network presented in Fig. 2a would have $M^{(arch.)} = M^{(weight)} = 0$ in the "top down" approach, as there is only one output node and thus only one partition. However, it is possible to treat them (networks with one output) as being comprised of a number of sub-networks (or an ensemble of sub-networks [15]–[18]) and base the partitioning on the combined set of outputs of the constituent networks.

To adapt the above modularity measure to networks with a single output unit, one needs to look for the internal connectivity patterns inside the network to discover similar structures as in modular architectures without taking into account the output node. Thus the improved modularity measure will not consider the single output unit in the calculation of $M^{(arch.)}$

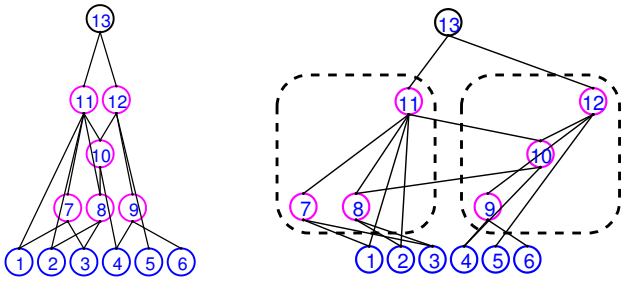

Fig. 2. Neural network with one output unit, normal (Fig. 2a) and modular (Fig. 2b) representation

or $M^{(weight)}$, but this node and its inbound connections will still be used elsewhere.

Figure 2 illustrates the use of the improved modularity measure for networks with a single output node. Note that this network is the same network as in Fig. 1, except for the new output node 13 which relegates the former output nodes 11 and 12 to the position of hidden nodes. This simple relation will prove helpful in illustrating the improved modularity measure. Since the output node 13 will not now be considered in the modularity evaluation, it and its connections $c_{11,13}$ and $c_{12,13}$ will be referred to as *virtual nodes* and *virtual connections* for the purpose of this section, because they exist in the real network, and are used to identify modules, but do not take part in the modularity evaluation.

In such networks, all the inbound connections into the final output neuron come from the outputs of each module, i.e. one output per module is connected to the final output unit (virtual node 13 in Fig. 2a). After the final output neuron has been omitted, one can identify all the inbound connections to it and take the corresponding nodes to be potential new outputs that will lead to the structure of a multiple-output modular neural network. However, not all of these potential new outputs should be considered as actual new outputs of the network (i.e. module outputs of the full network). For example, suppose the nodes are ordered in an incremental way over a connectivity matrix C, and nodes i and j are connected to the output unit k (i.e. there exist connections $c_{i,k}$ and $c_{j,k}$). If there exists another connection $c_{i,j}$ where $i < j < k$, the connection $c_{i,k}$ should not be considered to be the output of a possible module because one of the outputs of node i is connected to another hidden node (node j, because $c_{i,j}$ exists). In a hierarchical representation, node i is then an internal unit of a possible module formed by node j because node j needs to process the output of node i to produce its output, regardless of whether node i is directly connected to node k.

For that reason, the following two restrictions need to be imposed to determine which hidden nodes will be considered as the new outputs of the network, i.e. nodes that have a virtual connection. If H and O are the sets of hidden and output nodes respectively, then:

1) Node i is considered an internal unit inside the module formed by node j if $\exists c_{i,j}, \forall i, j \in H$ where $i < j$.
2) The output of node j ($c_{j,k}, k \in O$) is considered the output of module k until $\exists c_{j,l}, \forall j, l \in H$ where $j < l$.

This means that a node i will be considered as the output of a new module k if and only if node i has a virtual connection and there is no other outbound connection to a subsequent node. In Fig. 2a, nodes 11 and 12 are the only nodes that have a virtual connection to node 13. If other hidden nodes had a connection to node 13, they could not be considered as a new module's output because they have connections to subsequent nodes that in turn are connected to the original output node. Finally, applying equations 1 to 5 to the new partitioning leads to the improved modularity measure.

Clearly, there will always be at least one hidden node in the original network that satisfies the requirements for becoming a new output node, i.e. the last hidden node in the network which was originally connected to the virtual node. If there is only one such new output node, that would again lead to $M^{(weight)} = M^{(arch.)} = 0$, as there were no modular structures identified inside the network. However, one can then repeat the above process to identify modules that feed into that output node. This definition of modularity for "top-down" networks with a single output unit fits with the conventional definition of ensembles and MNNs. Each module has one output node, and all the module outputs are fed into the final node of the complete network, even if there exist other nodes that have a direct connection to the final output node.

An important aspect of this process for identifying modules is that it is not restricted to looking for modules that feed into the final output of a network, but it can equally well be applied to any node (for any sub-network) in the network to produce a fine-grained measure of modularity at a local level. For example, if a detailed analysis of a single node j is required, it is possible to apply the improved modularity measure to the sub-network feeding into it. That will allow the discovery of smaller partitions of nodes in the sub-structure of the network. This is a similar process to that of Newman [8] where the nodes are divided into communities to create a "dendrogram" (a tree representing the connectivity clusters among nodes). However, that approach needs to search for the best partition of nodes that increases a modularity measure Q. In the process defined above, there is no need for an extra optimization stage to determine the modularity, and that makes it much more efficient for using repeatedly in an extended evolutionary process.

If the nodes of the network in Fig. 2a are rearranged according to the above process (using the same connection weights as for Fig. 1b up to node 12), the modular representation shown in Fig. 2b is obtained. Comparing Figs. 1b and 2b shows that the process has, as required, formed the same modules with the same nodes. In the examples presented here, a graphical representation of the shared modules has not been shown, i.e. modules formed by shared nodes. If that had been done, the networks presented in Figs. 1b and 2b would have an extra module containing the nodes 8 and 10.

The process presented here allows the analysis of neural networks to create modules using a modularity measure,

identifies modules using a shared module (discovering shared nodes), and enables the analysis of cross-connection between modules over networks with one or multiple output units. Although such an automated process for identifying modules and measuring modularity can clearly be used as a very general tool for analyzing neural network structures, in this paper it will simply be used within an evolutionary neural network algorithm to facilitate the evolution of neural architectures with increased modularity.

III. EPNet and Modular EPNet

The EPNet algorithm [3] is a steady state algorithm that uses Lamarckian inheritance to evolve populations of neural networks for particular tasks. It is based on Fogel's Evolutionary Programming approach, which means that mutations are applied to individuals in the population, but crossover is not used as an operator. Every time an individual is mutated, it is partially trained, its fitness determined, and the probability of replacing another individual in the population computed. The mutation operators are applied in a hierarchical fashion with the aim of finding the smallest architectures that do not compromise performance. Thus, deletion mutations (of nodes and connections) are applied before addition mutations.

The whole approach is based on a Generalized Multi-Layered Perceptron (GMLP) which allows feed-forward connections between any pairs of nodes, unlike a standard MLP which only allows connections between adjacent layers of a layered network. A GMPL may therefore have as many layers as hidden nodes, e.g. in the situation that each non-output node i is connected to node $i+1$. For a more complete description of the algorithm, see [3], [16]–[18].

A. Connectivity Matrix

To represent the GMLP connectivity, the EPNet algorithm uses a network connectivity matrix C as shown in Fig. 3. As feed-forward networks are used here, the connectivity matrix for this approach consists of five sub-matrices (IH, IO, HH, HO and OO) in the upper-right part of it, where the rows represent source nodes and columns represent destination nodes. Thus, the GMLP allows connections from inputs to hidden nodes (IH), inputs to outputs (IO) and so on until connections from output to output nodes (OO sub-matrix) are reached. This can be used to highlight the differences between MLPs and GMLPs for general NNs and MNNs.

If GMLPs are compared with MLPs, it can be seen that MLPs use IH, HH and HO when there is more than one hidden layer. For the simplest case of one hidden layer, MLPs only require the submatrices IH and HO. In general, IH and HH are more restricted in MLPs as not all possible connections are allowed in those sub-matrices.

If an MLP or a pure MNN is analyzed using the connectivity matrix, a clear pattern of non overlapping sections will be evident in each sub-matrix (not shown in Fig. 3). Since the GMLP is used here as the basis for evolving MNNs with the EPNet algorithm, it will require considerable effort during evolution to reach such non-overlapping patterns. However,

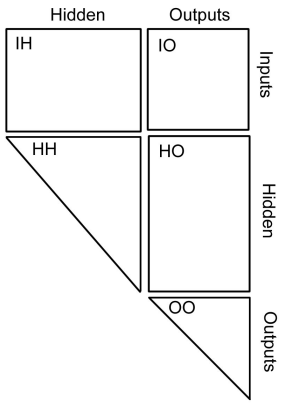

Fig. 3. Sub-matrices in the network connectivity matrix

that is the inevitable cost of allowing the evolution of more general, and potentially more powerful, architectures. This is why a biasing modification of the standard EPNet algorithm is required.

B. Modular EPNet Algorithm (M-EPNet)

The aim to implement a modular version of the EPNet algorithm arises because the normal algorithm is found to be too slow to delete the appropriate neural components that increase the average modularity in the population. Thus, the required modifications of the EPNet algorithm are the addition of new operators that delete shared nodes and shared connections and thus allow the increase of the modularity during evolution.

The idea of shared node deletion has previously been implemented in a more direct manner [1]. In that case, there is a simple set of parameters which specify how many hidden nodes in a single hidden layer MLP connect to each sub-set of the outputs. If there are only two output units, deleting the shared nodes then simply corresponds to reducing a single parameter value. In this way, the evolution of modules is controlled, and the degree of modularity can be monitored, directly with that single parameter, and no additional measure of modularity is required. Clearly, the more complex connectivity patterns of the GMLP networks of interest here require a more sophisticated approach. Moreover, simply counting the number of connections does not take into account how many of the associated connection weights have been reduced to zero (or near zero) by the learning algorithm. This is where the above modularity measures are required.

The new M-EPNet algorithm is presented in Fig. 4, with the overall procedure on the left, and the mutation procedure on the right. The first modification to the standard EPNet algorithm is in the training stage, where the usual hybrid learning involving both Back-Propagation (BP) and Simulated Annealing (SA) has been simplified to only use the BP algorithm. This keeps the procedure as simple as possible, and enables a more direct comparison with earlier studies (e.g., [1], [5]). The second modification, of direct relevance to modularity, is in the introduction of two new mutation operators (shared node and shared connection deletions) as

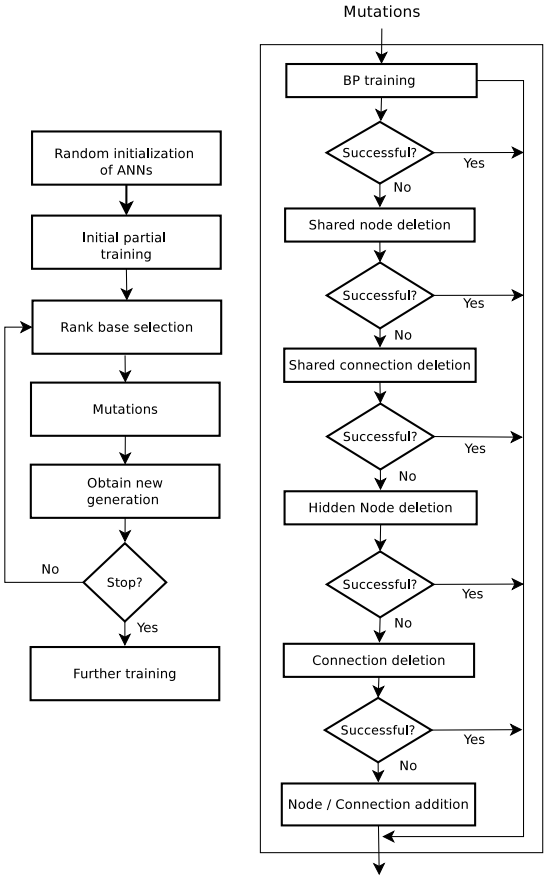

Fig. 4. The Modular EPNet algorithm

the first architectural mutations. If these are not successful (i.e. they do not produce a better individual than the least fit one in the existing population), then the general node and connection deletion mutations are followed as in the original algorithm. At the end are the standard addition mutations.

If the task at hand does not require a modular network, then the operators designed to increase the modularity (shared node and connection deletion) may not be used. However, if the task can be performed better or equally well by a modular network, then it may be expected that the shared node and connection deletion mutations will be used more than the other mutations in the algorithm.

If a shared node is deleted, it is likely to decrease the number of shared connections in the network. Similarly, a shared connection deletion can cause a shared node to become a pure node. Also, every time the weights are updated (by the BP training stage), there is the possibility that a node will switch to another module, i.e. while the learning procedure does not affect $M^{(arch.)}$, it can change $M^{(weight)}$.

IV. Experimental Set-up

The data set used to test the algorithms was generated randomly with two inputs in the range [0,1] and two distinct classification tasks (with one output each) as shown in Fig. 5, i.e. the networks have to learn the classification boundaries in

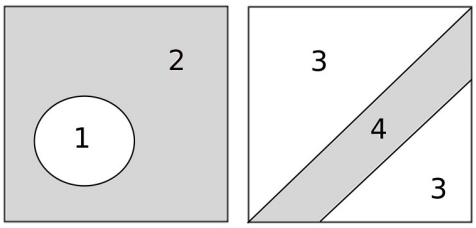

Fig. 5. Data sets formed by random points in a two dimensional input space and two outputs correponding to distinct classification tasks

the two dimensional input space for each output task. This data set has been studied previously and shown to result in modular MLP architectures evolving if the fitness is measured in terms of the time required to learn optimal generalization [1]. Here, 1000 such patterns were used for training, 100 patterns were used for validation during evolution (to provide a measure of fitness), and 100 patterns were used to test the networks after the evolution is finished.

The algorithms were tested using 5000 generations of evolution, with 20 individuals in the population, and 10 independent trials were run to provide reliable statistics. The population of networks was initialized before the evolution starts with a simplified connection scheme set using the same probability ϕ for each connection in the connectivity matrix C. For a recurrent network with n nodes, the expected number of connections would be $n^2\phi - n$, assuming there are no loops from any node to itself. In the feedforward network case studied here it is $N\phi$, where N is the maximum total number of connections allowed (the upper-right part of the connectivity matrix as shown in Fig. 3). The connectivity level ϕ set in the initialization was maintained during evolution (as done in [1], [6]), and different values of it were tested to determine how it affected the performance of the algorithms. The numbers of hidden nodes were initialized randomly between 50 and 100 nodes. The EPNet algorithm uses a partial training at each generation, for which 50 epochs of training were used, with BP learning rate fixed at 0.15, and no early stopping.

The data set shown in Fig. 5 may be learned well by either pure modular or pure homogeneous networks, and that can potentially make it problematic to use the classification generalization error as the fitness. For that reason, two different fitness measures were used to test the EPNet algorithms. The first one was the standard error percentage [19] as previously used in the EPNet algorithm [3], and defined by equation 6:

$$E_p = \frac{100}{Tn(Z_{max} - Z_{min})^2} \sum_{t=1}^{T} \sum_{i=1}^{n} (Y_i(t) - Z_i(t))^2 \qquad (6)$$

where T is the number of patterns, n the number of output nodes, and $Y_i(t)$ and $Z_i(t)$ are the actual and desired outputs of node i for pattern t. The second fitness function is a modified version of the error percentage in which the network is penalized according to the modularity measure $M^{(arch.)}$:

$$E_b = \alpha E_p + (1 - \alpha) \frac{1}{M^{(arch.)}} \qquad (7)$$

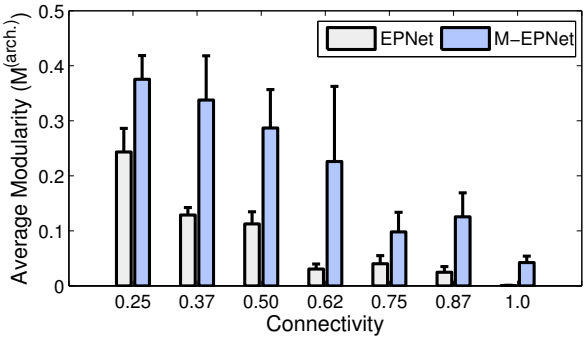

Fig. 6. Evolved modularity values for both algorithms with different connectivity values using the standard error E_p as fitness function

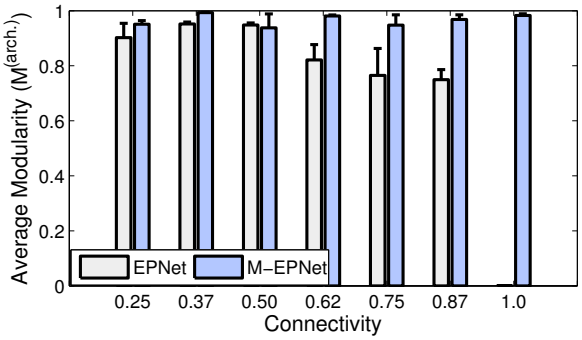

Fig. 7. Evolved modularity values for both algorithms with different connectivity values using the modified fitness function E_b

where α is a parameter controlling the relative importance of the E_p and $M^{(arch.)}$ components. After some preliminary experiments it was determined that α should be set to between 0.8 and 0.9, since lower values tend to result in no learning at all, because the modularity has a bigger contribution to the modified fitness than E_p, which causes an increment in the modularity in the early generations regardless of whether the networks improve their performance on the task.

V. Experimental Results

Figure 6 presents the resulting evolved modularity values $M^{(arch.)}$ for the EPNet and M-EPNet algorithms, using different connectivity values, with the standard percentage error (eq. 6) as the fitness function. As hoped, the new modular algorithm (M-EPNet) results in larger modularity values than the classical algorithm (EPNet), which confirms the effectiveness of the new mutation operators introduced for deleting shared neural components between modules. Both algorithms tend to evolve more modular networks when there are a smaller number of connections in the networks.

It was discovered that the relatively low modularity values for both algorithms were mainly due to two factors: a) there are many more connections in GMLPs in comparison with MLPs, and b) the data set may be classified correctly by different networks with different degrees of modularity, thus there is little pressure to select more modular individuals because any individual in the population performs similarly on the classification task and thus has similar fitness, i.e. the task can be solved through evolution without increasing the modularity more than that presented in Fig. 6.

That confirms the need for the modified fitness function (eq. 7) to bias the evolution of modularity. The result of repeating the above experiments using that fitness function are presented in Fig. 7. In this case, both the EPNet and M-EPNet algorithms produce, on average, individuals with larger modularity values after 5000 generation of evolution than when the standard E_p fitness function was used (Fig. 6). Interestingly, the EPNet algorithm continues to result in architectures with lower modularity values than M-EPNet in the majority of the cases. That again demonstrates the advantage of the shared node and shared connection deletion mutations not present in the conventional EPNet algorithm.

Moreover, it can also be seen in Fig. 7, for all connectivity values, that the M-EPNet algorithm has almost pure modular networks at the end of evolution. Another interesting finding occurred in the experiment with connectivity level of 1.0 for EPNet. As seen in Fig. 7, the algorithm could not then delete shared architectural components due to the high connectivity, which results in the evolution of pure homogeneous networks. That is similar to the results from the EPNet algorithm for the same connectivity value with standard fitness in Fig. 6.

In Fig. 8 is presented a comparison of the average modularity $M^{(arch.)}$ throughout the evolution for both algorithms and both fitness functions for connectivity 0.25. It is clear that the modularity increases more rapidly when the M-EPNet algorithm is used with either fitness measure, than with the EPNet algorithm. When the standard error E_p is used as the fitness, the modularity produced by M-EPNet is larger throughout evolution than with the EPNet algorithm. However, both algorithms are reaching similar levels of modularity at the end of the evolution when the modified fitness E_b is used.

In both experiments, using E_p and E_b, it was found that the M-EPNet algorithm resulted in architectures at the end of the evolution with significantly fewer connections than the EPNet algorithm, for almost all connectivity cases investigated. That implies that the specific mutations designed with particular knowledge of the network work more effectively here than random mutations. Fig. 9 shows the evolution of the average numbers of shared connection for the M-EPNet algorithm

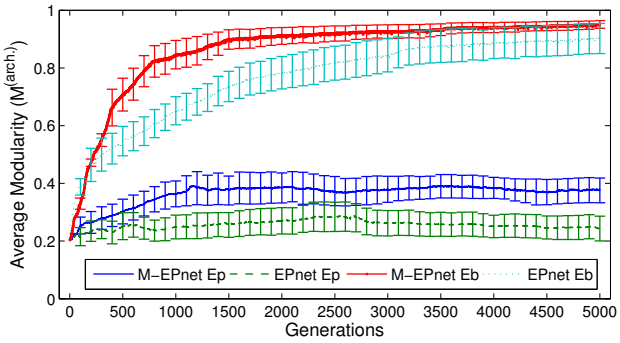

Fig. 8. Comparison of the evolving modularity values for EPNet and M-EPNet for both fitness measures (E_p and E_b) and connectivity set at 0.25

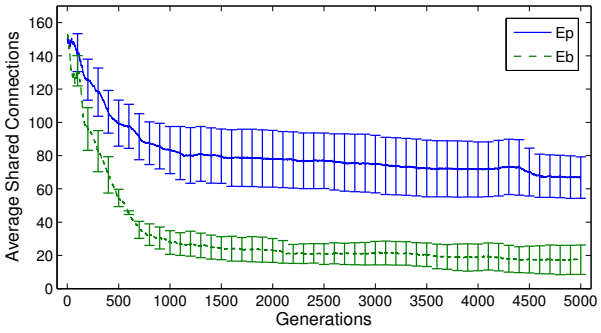

Fig. 9. Comparison of the evolving numbers of shared connections for M-EPNet with both fitness measures (E_p and E_b) and connectivity set at 0.25

using both fitness measures with a connectivity of 0.25. It is seen that the biased E_b reduces the shared connections between modules faster than the normal fitness measure E_p. The numbers of shared nodes follow a similar pattern.

The final important question to consider concerns how the increased degrees of modularity affect the neural networks' performances on the given task. Measuring performance in terms of the percentage of correct classifications shows that all four approaches (EPNet-E_p, EPNet-E_b, M-EPNet-E_p, M-EPNet-E_b) learn the task well, with no significant differences between their performance levels.

VI. Conclusions

This paper has explored how the evolutionary neural network algorithm EPNet can be biased to produce more modular networks that will facilitate module re-use across a range of tasks, without compromising performance on those tasks.

It has been shown how a modularity measure can be used to identify specific neural components for additional deletion mutations, thus increasing the degree of modularity arising through evolution. The same modularity measure was also improved to work for networks with a single output unit, and to allow a finer-grained analysis of the internal partitions of nodes inside a network. Although this approach has assumed prior knowledge about the task data partitions to measure the modularity, there is scope for dynamically adjusting the output partition into modules.

The information provided by the modularity measure allowed the implementation of a new Modular EPNet algorithm (M-EPNet) that biases the evolution of modularity compared to the classical EPNet algorithm, and also enabled a modification to the standard generalization error fitness that further biases the evolution of modularity.

The general conclusion is that the rearrangement of nodes into modules using the modularity measure enables the discovery of shared neural components through modules, and that information can be used to evolve almost pure modular neural networks with the EPNet and M-EPNet algorithms using a fitness function penalized by lower modularity values.

Both the biasing deletion mutations and biasing fitness function are not special to the EPNet algorithm - they can easily be applied to bias towards modularity in other evolutionary neural network approaches. Moreover, they are not restricted in any way to the test task chosen for this paper, but can be applied to any classification or regression task.

Clearly the next stages of this work are to run tests using a wider range of tasks, to demonstrate explicitly how the evolved modules facilitate neural re-use for new tasks, and to explore what other advantages the more modular architectures have. That will be reported in a future paper.

Acknowledgment

This research was supported by the University of Birmingham UK through the BlueBEAR cluster, and by CONACYT through a scholarship granted to the first author.

References

[1] J. A. Bullinaria, "Understanding the emergence of modularity in neural systems." *Cognitive Science*, vol. 31, no. 4, pp. 673–695, 2007.

[2] J. Reisinger, K. O.Stanley, and R. Miikkulainen, "Evolving reusable neural modules," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*. New York, UK: Springer-Verlag, 2004, pp. 68–81.

[3] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 1997.

[4] K. O. Stanley and R. Miikkulainen, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, no. 1, pp. 63–100, 2004.

[5] M. Hüsken, C. Igel, and M. Toussaint, "Task-dependent evolution of modularity in neural networks," *Connection Science*, vol. 14, 2002.

[6] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 39, pp. 13 773 – 13 778, Sep. 2005.

[7] R. Guimera and L. A. Nunes Amaral, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, pp. 895 – 900, 2005.

[8] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Phys. Rev. E*, vol. 69, no. 6, p. 066133, Jun. 2004.

[9] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb 2004.

[10] R. V. Solé and S. Valverde, "Spontaneous emergence of modularity in cellular networks," *Journal of The Royal Society Interface*, vol. 5, no. 18, pp. 129 – 133, January 2008.

[11] J. G. Rueckl, K. R. Cave, and S. M. Kosslyn, "Why are what and where processed by separate cortical visual systems? a computational investigation," *Journal of Cognitive Neuroscience*, vol. 1, no. 2, pp. 171–186, 1989.

[12] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. New York, NY, USA: Wiley-Interscience, 1994.

[13] G. P. W. Gerhard Schlosser, *Modularity in Development and Evolution*. Chicago U. Press, 2004.

[14] J. A. Fodor, *The Modularity of Mind: an Essay on Faculty Psychology*. Cambridge, MA: The MIT Press, April 1983.

[15] J. A. Bullinaria, "Using evolution to improve neural network learning: pitfalls and solutions," *Neural Computing and Applications*, vol. 16, no. 3, pp. 209–226, 2007.

[16] V. Landassuri-Moreno and J. A. Bullinaria, "Neural network ensembles for time series forecasting," in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2009, pp. 1235–1242.

[17] X. Yao and M. Islam, "Evolving artificial neural network ensembles," *Computational Intelligence Magazine, IEEE*, vol. 3, no. 1, pp. 31–42, Feb. 2008.

[18] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 28, no. 3, pp. 417–425, Jun. 1998.

[19] L. Prechelt, "PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms," Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Tech. Rep. 21/94, Sep. 1994.