sciendo

JAISCR, 2023, Vol. 13, No. 2, pp. 107– 126
10.2478/jaiscr-2023-0010

EMERGING MODULARITY DURING THE EVOLUTION OF NEURAL NETWORKS

Tomasz Praczyk

Computer Department, Polish Naval Academy,
ul. Smidowicza 69, 81-127 Gdynia, Poland

**E-mail: t.praczyk@amw.gdynia.pl*

Submitted: 9th February 2022; Accepted: 19th October 2022

Abstract

Modularity is a feature of most small, medium and large–scale living organisms that has evolved over many years of evolution. A lot of artificial systems are also modular, however, in this case, the modularity is the most frequently a consequence of a handmade design process. Modular systems that emerge automatically, as a result of a learning process, are very rare. What is more, we do not know mechanisms which result in modularity. The main goal of the paper is to continue the work of other researchers on the origins of modularity, which is a form of optimal organization of matter, and the mechanisms that led to the spontaneous formation of modular living forms in the process of evolution in response to limited resources and environmental variability. The paper focuses on artificial neural networks and proposes a number of mechanisms operating at the genetic level, both those borrowed from the natural world and those designed by hand, the use of which may lead to network modularity and hopefully to an increase in their effectiveness. In addition, the influence of external factors on the shape of the networks, such as the variability of tasks and the conditions in which these tasks are performed, is also analyzed. The analysis is performed using the Hill Climb Assembler Encoding constructive neuro-evolutionary algorithm. The algorithm was extended with various module-oriented mechanisms and tested under various conditions. The aim of the tests was to investigate how individual mechanisms involved in the evolutionary process and factors external to this process affect modularity and efficiency of neural networks.

Keywords: neuro–evolution, modular neural networks, emergent modularity, hybrid algorithms

1 Introduction

The nature has long proven that in the case of complex systems, only modular architectures have a chance of survival and development. From basic electromechanical devices like irons, vacuum cleaners, or razors to complex body cells, modularity works in a huge number of situations. Modularity itself is a widespread technique for organizing and simplifying complex systems. The concept of modularity is used primarily to reduce complexity by breaking a system into varying degrees of interdependence and independence across and to ”hide the complexity of each part behind an abstraction and interface” [3]. In technology, the term of modularity applies to systems that can be decomposed into a number of components that may be mixed and matched in a variety of configurations [3]. In the case of artificial neural networks (ANN), to which this paper is devoted, it means de-

composition of the entire network (modular neural network–MANN) into a number of sub–networks loosely connected with other sub–networks.

However, modularity in ANNs is not a goal in itself. Unlike monolithic networks, MANNs have features that make them an attractive tool for solving many problems. Usually, they are simpler in construction than their monolithic counterparts which is due to adjusting MANNs to a modular problem and the fact that the entire problem is typically more difficult to solve in one piece than after dividing it into sub–problems each of which is solved by a separate module [37].

The consequence of the simplicity in construction is the ease of analysis. The monolithic ANNs are often treated as black boxes – it is usually very difficult or even impossible to determine the role of individual neurons or connections in solving the whole task. In turn, the decomposition of MANNs into larger building blocks than neurons elevates the analysis of the networks to a higher level, instead of trying to understand the role which individual neurons play in the network we can do the same with respect to modules [37].

The training of MANNs is also usually easier compared to monolithic ANNs. Firstly, it is a consequence of the simpler construction of the modules themselves. Each of them is responsible for a single piece of a problem, so they do not have to be so complex as networks dealing with the entire problem. Secondly, if the division of responsibility between the modules and general modular architecture is known in advance, instead of training the modules together to cooperate and to solve the entire problem, they can be trained separately from each other, even with the use of different algorithms dedicated to monolithic networks. After the training of all the modules, they are combined together and tuned to a problem [37].

Undeniable advantages of MANNs compared to their monolithic counterparts and ubiquitous modularity in our daily lives contributed to to the large interest in this type of networks, both among researchers and practitioners. On the one hand, it is currently difficult to imagine image or natural language processing without Deep Neural Networks (DNN) which are an example of MANNs consisting of many sequentially organized modules. On the other hand, there is also a great interest among researchers in the phenomenon of modularity itself. How modularity arises in the natural world and what mechanisms lead to it – these are the questions many scientists are trying to answer.

The modularity of neural networks, which is not the result of manual, intuitive design based on the experience of a designer, and sometimes even simple guesswork, but of evolutionary processes and gradual growth of the networks from a single neuron to more complex structures is also addressed in the current paper. Its main contribution is the analysis of the influence of various factors on evolution of the networks, their modularity and effectiveness. In addition to connection cost, network size, environmental changes, and multiple selection pressures whose impact on network modularity and effectiveness has already been examined by many researchers, the paper also takes into consideration such factors as outgoing connection cost, connection length cost, inter–neuron distance cost, and the ability to gradually grow the networks.

The paper also proposes four hand-designed module–oriented mechanisms encoded in the network genotype and applied in the process of network growth. The first is aimed at building larger neural architectures from small fully formed modules with dense inner connectivity and one output neuron, the second groups neurons in densely packed clusters, the third replicates neuron concentrations in different network locations, and the fourth mechanism introduces a new type of neuron whose task is to split the neural network into separate clusters.

In order to examine the influence of the above mentioned mechanisms and factors on modularity and effectiveness of the networks, experiments were carried out. The networks of different size were applied to solve two binary classification problems, one simpler and one harder. The evolved networks were examined in terms of their modularity and efficiency. To measure modularity, the Louvain [5] method for community detection was applied [1].

In the experiments, the networks evolved according a generative neuro–evolutionary algorithm

[1] C++ implementation of Louvain was taken from https://github.com/riyadparvez/louvain-method

called Hill Climb Assembler Encoding (HCAE). It represents an incremental approach to evolution of neural networks. In HCAE, neural networks are designed in many iterations. Each iteration is a separate evolutionary run that is responsible for a small piece of a resultant network. The task of each run is to update neurons/connections that have already been established by previous runs or to add new neurons/connections. All changes made to the networks are performed by evolutionary shaped programs similar to simple assembler programs. To evolve the programs cooperative co–evolutionary algorithm is applied.

HCAE is a simplified version of Hill Climb Modular Assembler Encoding (HCMAE) [38], i.e. an algorithm designed to evolve modular neural networks. However, HCMAE assumes that the modular architecture of the networks is predetermined, it does not evolve. HCAE is basically a single–module HCMAE equipped with additional mechanisms that are designed to achieve modularity through evolution, not manually.

The rest of the paper is organized as follows: section two presents related work, section three describes HCAE, section four details module–oriented mechanisms, section five reports the experiments, and the final section concludes the paper.

2 Related work

Increasing complexity of problems solved with the help of neural networks contributed to the development of different algorithms dedicated especially to MANNs and extensions of traditional algorithms, like HCAE, to the concept of modularity.

With the simplest case of modularity in neural networks we deal when a problem can be manually decomposed into sub–problems. In that case, the architecture of a MANN reflects the structure of the problem: each sub–problem extracted from the original problem is linked to a separate module. Parallel processing in the modules requires an integration mechanism responsible for combining outputs of each module. Since the task of each module is known beforehand, they can be separately trained by means of a selected learning algorithm, e.g. BackPropagation. Examples of such an approach can be found among other things in [4, 7, 28, 40, 45, 46].

The methods for automatic design of MANNs are necessary when the decomposition of a problem into sub–problems is unknown in advance. In [10, 22, 35], the methods are presented which arbitrarily determine a general modular architecture of the network and then adjust individual modules to it. They use evolutionary techniques or self–organization to assign tasks to each module, to design the modules, and to determine cooperation between the modules. The number of modules and inter–module connectivity are imposed from above.

Cooperative Co–Evolutionary Neural Networks [37] is an ensemble–based approach in which modules collaborating within the same MANN are not connected and they negotiate to determine the output of the network. A module with the greatest negotiation strength is allowed to set one of the outputs of the entire MANN. To fix all the outputs, the modules negotiate many times. The architecture of individual modules is defined through evolutionary process, whereas the number of modules is predetermined.

In contrast to the methods mentioned above, the work [20] proposes the algorithm which automatically adjusts the number of modules to a problem and it applies a self–organization to this purpose. The scheme of inter–module connections still remains the decision of a network designer, whereas the architecture of individual modules is defined in the evolutionary way.

The existence of a set of predefined modules is assumed in [13]. To design a MANN, some modules are selected from the set, and then somehow combined together. Selecting modules to the networks, determining their optimal number and adjusting the topology of inter–module connections to a problem are performed in the evolutionary way.

Selective application of modules in response to a given input signal is also presented in [23]. A modular network has a set of modules at its disposal and once the network is fed with an input signal the modules that best match the input are selected, formed in a layer, and then supplied with the network input. In order to calculate the output signal of the layer, an integration mechanism is applied, e.g. outputs of all the selected modules are summed together. In general, the considered approach can be applied with respect to any network that can be decomposed into many layers somehow connected

with each other. In this case, each layer can be linked to other set of modules.

In [30, 32], a Genetic Programming (GP) approach is presented. In this case, the evolution can optimally shape modular architecture of a network as well as the inner architecture of the modules themselves.

Modular variant of EPNet [48] that is also based on GP and is called M–EPNet is presented in [25]. The authors compare both algorithms on two distinct classification tasks. The experiments revealed similar performance of purely modular and purely monolithic networks in generalization stage and a better performance of modular networks in the learning stage.

In Augmenting Modular Neural Networks [36], which is a generative Neuro–Evolutionary (NE) method, MANNs expand gradually. At the beginning of the evolutionary process, all networks consist of only input/output modules (or a single module). After some time, if the architecture of all evolved networks is insufficient to effectively perform a task, all of them are augmented by one hidden module. In the following generations, further hidden modules are also added and this procedure is continued until some stopping criterion is satisfied. In addition to the number of modules, the method decides about the inner architecture of each module as well as about inter–module connections.

The efforts to modularize NEAT [41] and its successor HyperNEAT [19], which are the state–of–the–art generative NE methods which do not have special mechanisms for evolving modular ANNs, are presented in [8, 21,31]. To this end, the authors apply a connection cost technique which prefers networks with fewer number of connections, multi–objective PNSGA algorithm [9] or directly add a modularity component to the fitness function. The experiments reported in the above mentioned works showed that both algorithms are able to evolve MANNs with regularities.

In [14], the task of the authors was to design a modular controller for a robot arm capable of working with different objects. To this end, they also applied the connection cost technique and pNSGA, which is a probabilistic version of the popular multiobjective optimization algorithm NSGA-II [12]. Their experiments revealed that evolutionary pressure to form sparser, more modular networks, can be beneficial when modeling multiple objects.

A grammatical approach to the evolution of neural networks is presented in [39]. The algorithm proposed by the authors, called Modular Grammatical Evolution (MGE), is meant for evolving modular neural networks with a layered architecture. It was validated on different classification benchmarks with different sizes, feature counts, and output class counts. The tests reported in the mentioned work showed superiority of MGE in relation to selected NE algorithms as well as its ability to evolve simple neural classifiers.

All the papers mentioned above present algorithms dedicated to small or medium–size MANNs that contain at most a few modules and a few or at most a dozen of neurons in each module. Meanwhile, Deep ANNs (DNN) which are typically composed of many modules and neurons need a different approach from those presented above.

A typical approach to design DNNs is to manually determine the number of modules, inter–module connectivity, size of modules and their inner connectivity, and then to apply gradient decent algorithms or evolutionary techniques to find optimal parameters of the network. An example of such approach is presented among other things in [**?**].

Other approaches like MENNDL [49], CMA–ES [27], CoDeepNEAT [26, 29], and the ones proposed in [1, 2, 42, 47, 50], use gradient descent to learn DNNs and apply the NE to evolve topology, hyper–parameters of the networks (e.g. the number of convolutional layers, skip connections, the number of kernels in each convolutional layer, kernel size in each convolutional layer), and learning parameters (e.g. learning rate, batch size).

A similar approach is presented in [17]. In this case, the NE is applied to evolve topology of a differentiable variant of Compositional Pattern Producing Network [19] which is then trained with the use of the gradient descent. The authors show that their networks can rediscover (approximate) convolutional network architectures.

A separate branch of research on modularity is inquiring its origins and the influence on natural and artificial systems. The role of modularity in the ability to adopt to environmental changes is considered in [6]. In order to explore this role,

the authors carried out two sets of artificial life experiments in which they measured modularity of evolved neural networks and programs in response to changes of tasks they had to perform. The experiments showed strong correlation between modularity of the evolved agents and their performance when the tasks of the agents became increasingly complex.

The work [44] examines the impact of network size on its modularity and performance. The simulations performed on three types of networks of different size (non–modular, non–modular sparsely connected, and modular; small and large) revealed that modularity becomes effective as the network size increases.

Analysis of the interdependencies between network size, modularity and efficiency is continued in [43]. The authors show that computational efficiency is insufficient for modularity to arise. What is more, they also demonstrate that even in modular problems, modular networks are not always more effective than their monolithic counterparts. Their experiments revealed that the size of the networks is a key factor in this case, meaning that small modular networks can be less computationally efficient than their non–modular rivals, however, larger modular networks should be rather more efficient than the non–modular ones. To obtain modularity in the networks, the authors simply replicated initially trained existing neural substructures.

The influence of weight and node–based pruning is investigated in [18]. The experiments carried out by the authors show that weight–based pruning performed at the end of the training process makes networks more modular than random networks and networks with the same sparsity and distribution of weights. Moreover, the experiments also revealed that training combined with node–based pruning (dropout) contributes to modularity.

In [15], the authors focus on a problem called catastrophic forgetting which applies to losing previously acquired skills. They convince that modularity evolved in neural networks combined with neuromodulation, which is a type of reinforcement learning, alleviates catastrophic forgetting effect. To obtain modular networks they use connection cost technique.

Modularity in gene regulatory networks which are modeled as recurrent neural networks is considered in [16]. The authors show that modularity in their networks arises when two conditions are met. First, networks that have been previously trained to reproduce one activity pattern are trained to attain previously learned pattern, and additionally, a new pattern. Each pattern corresponds to a specific activity of each gene (or neuron) in the network. Second, both patterns must have a common part, and a different part, meaning that some genes are active in the same way in both patterns, whereas the remaining genes differ in their activity. The authors explain that modularity in the networks arises in order to reduce the impact of one group of genes on the other group.

3 Hill Climb Assembler Encoding

HCAE is based on three key components, i.e. Network Definition Matrix (NDM) which represents a neural network, Assembler Encoding Program (AEP) which operates on NDM, and Evolutionary Algorithm whose task is to produce optimal AEPs, NDMs, and in consequence, the networks. All the three components are described below.

3.1 Network Definition Matrix

To represent a neural network, HCAE uses a matrix called Network Definition Matrix (NDM). The matrix includes all the parameters of the network, including the weights of inter–neuron connections, bias, etc (see Figure 1). The matrix which contains non–zero elements above and below the diagonal encodes a recurrent neural network (RANN), whereas the matrix with the only content above the diagonal represents a feed–forward network (FFANN).

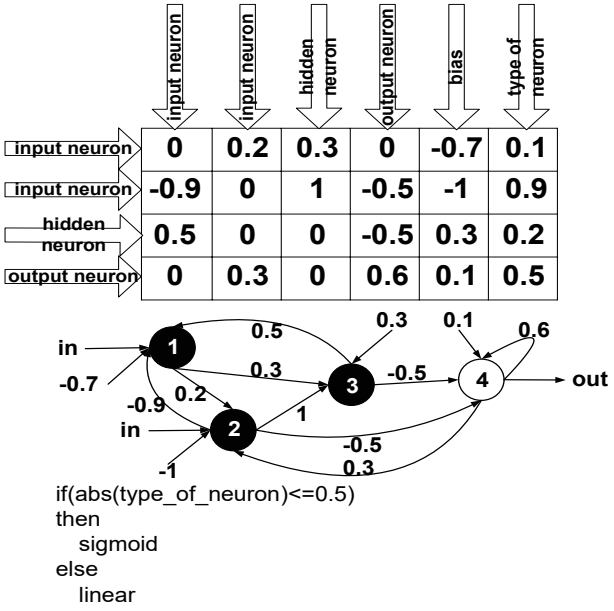


Figure 1. The way of encoding ANN in the form of Network Definition Matrix (NDM) [38]

3.2 Assembler Encoding Program

In HCAE, filling up the matrix, and, in consequence, constructing an ANN is the task of Assembler Encoding Program (AEP) which like an ordinary assembler program consists of a list of operations and a sequence of data. Each operation implements a fixed algorithm and its role is to modify a piece of NDM. The operations are run one after another and their working areas can overlap which means that modifications made by one operation can be overwritten by other operations which are placed further in the program.

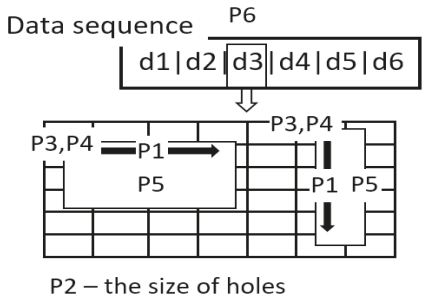


Figure 2. Illustration of changes made on NDM by two different HCAE–operations, p1 – direction of changes, along rows or columns, p2 – size of holes (zeros) between subsequent updates, p3, p4 – starting item in NDM, p5 – size of modified area, p6 – pointer to starting data item

Algorithm 1. Pseudo–code of HCAE–operation

Input: operation parameters (p), data sequence (d), NDM
Output: NDM
(1) *filled* ← 0;
(2) *where* ← p[6];
(3) *holes* ← 0;
(4) **if** p[1] **mod** 2 = 0
(5) **foreach** $k \in < 0..NDM.numberOfColumns)$
(6) **foreach** $j \in < 0..NDM.numberOfRows)$
(7) NDM[*j*,*k*] ← fill(*k*,*j*,param,data,*filled*,*where*,*holes*);
(8) **end for**
(9) **end for**
(10) **else**
(11) **foreach** $k \in < 0..NDM.numberOfRows)$
(12) **foreach** $j \in < 0..NDM.numberOfColumns)$
(13) NDM[*k*,*j*] ← fill(*j*,*k*,p,d,*filled*,*where*,*holes*);
(14) **end for**
(15) **end for**
(16) **end if**
(17) **Return** NDM.

The way each operation works depends on the one hand on its algorithm and on the other hand on its parameters. Each operation can be fed with its "private" parameters, linked exclusively to it, or with a list of shared parameters concentrated in the data sequence. Parametrization allows operations with the same algorithm to work in a different manner, for example, to work in different fragments of NDM.

The pseudo–code of HCAE operation is given in Algorithm 1 and Algorithm 2. It directly fills NDM with values from the data sequence of AEP: where NDM is updated, and which and how many data items are used, are determined by operation parameters. The first parameter indicates the direction according to which NDM is modified, that is, whether it is changed along columns or rows (lines (4) and (10) in Algorithm 1). The second parameter determines the size of holes between NDM updates, that is, the number of zeros that separate consecutive updates (line (3) in Algorithm 2). The next two parameters point out the location in NDM where the operation starts to work, i.e. they indicate starting row and column (line (1) in Algorithm 2). The fifth parameter determines the size of altered NDM area, in other words, it indicates how many NDM items are updated (line (1) in Algorithm 2). And the last sixth parameter points out the location in the se-

Algorithm 2. Pseudo–code of fill() [38]

```
Input: number of column (c), number of row (r), operation parameters (p),
data sequence (d), number of updated items (f),
starting position in data (w), number of holes (h)
Output: new value for NDM item
(1) if f < p[5] and c ≥ p[4] and r ≥ p[3]
(2)     f++;
(3)     if h = p[2]
(4)         h ← 0;
(5)         w++;
(6)         Return d[w mod d.length];
(7)     else
(8)         h++;
(9)         Return 0.
(10)    end if
(11) end if
```

quence of data from where the operation starts to take data items and put them into the NDM (line (2) in Algorithm 1).

3.3 Evolutionary Algorithm

HCAE is a hill–climber whose each step is made by Cooperative Co–Evolutionary GA (CCEGA) [33, 34] (see Algorithm 3). A starting point of the algorithm is a blank network represented by a blank NDM (line (1)). Then, the network as well as NDM are improved in subsequent evolutionary runs of CCEGA (line (5)). Each next run works on the best network/NDM found so far by all earlier runs (each AEP works on its own copy of NDM), is interrupted after a specified number of iterations without progress (*MAX_ITER_NO_PROG*), and delegates outside, to HCAE main loop, the best network/NDM that evolved within the run (temp-NDM). If this network/NDM is better than those generated by earlier CCEGA runs, a next HCAE step is made – each subsequent network/NDM has to be better than its predecessor (line (7)).

AEPs evolve according to CCEGA. The algorithm determines: the number of the operations, the parameters of the operations, the length of the data sequence, and its content. Each evolved component of AEP evolves in a separate population, that is, an AEP with n operations and the sequence of data evolves in $n+1$ populations (see Figure 3).

To construct a complete AEP, NDM, and finally, a network, the operations and the data are combined according to the procedure applied in CCEGA. An individual (for example, an operation) from the evaluated population is linked to the best leader individuals from the remaining populations that evolved in all previous CCEGA iterations. Each population maintains the leader individuals which are applied as building blocks of all AEPs constructed during the evolutionary process. In order to evaluate newborn individuals, they are combined with the leader individuals from the remaining populations [38].

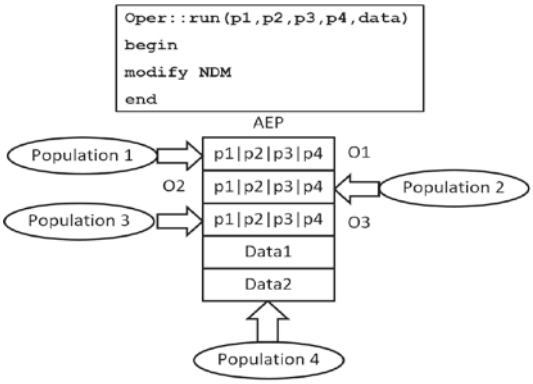


Figure 3. Evolution of AEPs according to CCEGA. In the figure, the case is presented with AEP homogeneous in terms of operations: all operations implement the same algorithm (in the upper part of the figure). In consequence, the operations are encoded as a set of parameters: p1–p4. In addition to the list of operations, AEP also includes the sequence of data. Each single component of AEP evolves in a separate population.

HCAE uses constant–length programs of a small size. They include at most two operations and the sequence of data, the number of operations does not change over time. Such construction of AEPs affects the structure of CCEGA. AEPs evolve in two or at most three populations, the number of

Algorithm 3. Evolution in HCAE

Input: CCEGA parameters, for example crossover probability
Output: Neural network
(1) NDM ← 0;
(2) *numberOfIter* ← 0;
(3) *fitness* ← evaluation of NDM;
(4) **while** *numberOfIter* < *maxEval* **and** *fitness* < *acceptedFitness*
(5) tempNDM ← CCEGA.run(NDM,*MAX_ITER_NO_PROG*);
(6) **if** tempNDM.*fitness* > *fitness*
(7) NDM ← tempNDM;
(8) *fitness* ← tempNDM.*fitness*;
(9) **end if**
(10) *numberOfIter* ← *numberOfIter* + 1;
(11) **end while**
(12) **Return** Neural network decoded from NDM.

populations is invariable. One population includes sequences of data, i.e. chromosomes–data, whereas the remaining populations contain encoded operations, i.e. chromosomes–operations. The operations are encoded as integer strings, whereas the data as real–valued vectors. Both chromosomes–operations and chromosomes–data are of constant length [38].

In HCAE, evolution in all the populations takes place according to a simple Canonical Genetic Algorithm with a tournament selection. The chromosomes undergo two classical genetic operators, i.e. one–point crossover and mutation. The crossover is performed with a constant probability P_c, whereas the mutation is adjusted to the current state of the evolutionary process. Its probability (P_m^d – probability of mutation in data sequences, P_m^o – probability of mutation in operations) grows once there is no progress for a time and it decreases once progress is noticed [38].

The chromosomes–data and chromosomes–operations are mutated differently, and it is performed according to Eq. 1 and 2 [38].

$$d_{new} = \begin{cases} d + \text{randU}(-a,a) \text{ if randU}(0,1) \leq P_m^d \\ d \text{ otherwise} \end{cases} \tag{1}$$

$$o_{new} = \begin{cases} o + \text{randI}(-b,b) \text{ if randU}(0,1) \leq P_m^o \text{ and} \\ \quad \text{randU}(0,1) \geq P_m^{o,zero} \\ 0 \text{ if randU}(0,1) \leq P_m^o \text{ and} \\ \quad \text{randU}(0,1) \leq P_m^{o,zero} \\ o \text{ otherwise} \end{cases} \tag{2}$$

where

d – is a gene in the chromosome–data
o – is a gene in the chromosome–operation
randU$(-a,a)$ – is a uniformly distributed random real value from the range $< -a,a >$
randI$(-b,b)$ – is a uniformly distributed random integer value from the range $< -b,b >$
$P_m^{o,zero}$ – is the probability of the mutated gene to be zero.

4 Module–oriented mechanisms and factors affecting modularity

The main goal of the paper is to discover how to construct neural networks with modular architecture that does not result from manual network design, but from the slow spontaneous process of the evolutionary formation from a single neuron to more complex structures. Moreover, the goal is not a modular architecture in itself, but an architecture that is optimally formed by its perfect fit to the problem to be solved.

The paper proposes a number of mechanisms which could contribute to the spontaneous formation of modular neural networks through the process of evolution. Some mechanisms have a natural basis, while the others are hand-designed network construction procedures which according to the knowledge of the authors have no counterpart in the natural world.

In addition, the paper also analyzes the influence of fluctuations in the environment and the size of the networks on their modularity.

4.1 Minimal and compact architectures

The first mechanism that derives from the natural world and which, apart from efficiency, affects the shape of the evolved neural networks, both natural and artificial, are the laws of physics and the desire to minimize the energy used to form and then maintain a single nerve cell and the entire nervous system. This mechanism is implemented in the form of evolutionary bias towards properly organized and compact neural architectures characterized by the presence of concentrations of neurons. This bias takes the form of a pressure to evolve lightweight architectures with a small number of neurons and to create architectures with an appropriate organization, that is, architectures with short connections, architectures with connections evenly distributed among all neurons, or architectures with densely connected clusters of neurons that are geographically close to each other. A detailed description of the individual mechanisms is given below:

1. Reward for a small number of connections (RSNC): This is an equivalent of the connection cost (CC) mechanism applied, among other things, in [8, 14, 21], and it simply results in a pressure to reduce connectivity in neural networks. RSNC neglects the organization of neurons in the network. They can be linked in any way, the only important thing is the pressure towards light architectures. In the experiments reported in the following section RSNC was calculated as follows:

$$RSNC = \frac{W^{RSNC}}{(1+N_c)} \tag{1}$$

where W^{RSNC} is a weight of reward (the greater the weight, the greater the pressure to evolve networks with a small number of connections) whereas N_c is the number of all connections in the network.

2. Reward for a small number of outgoing connections (RSNOC): This is a variant of RSNC which instead of being oriented solely towards reducing the overall network connectivity, is oriented towards reducing neuron outgoing connections, that is, it is the first mechanism focused not only on the minimization of architecture but also on appropriate internal network organization. In RSNC, all that matters is a small number of connections which can be distributed over neurons in any way. This means that there is no strong pressure to form balanced clusters of neurons loosely connected with other clusters. Some neurons can be isolated from the rest of the network, whereas others can be linked to many network regions gluing them together. A small number of the latter neurons may be positive for modularity, however, a large number can make it very difficult to form neuron clusters. To avoid such problems, RSNOC encourages to evenly distribute connections over the whole set of neurons, the networks which include neurons with many outgoing connections are penalized (or they are rewarded for a small number of such neurons). Formally, RSNOC is defined as follows:

$$RSNOC = \begin{cases} W^{RSNOC} & \text{if } N_n(N_{oc}^{max}) \leq N_n^{max} \\ \frac{W^{RSNOC}}{(1+N_n(N_{oc}^{max}))} & \text{otherwise} \end{cases} \tag{2}$$

where W^{RSNOC} is a weight of reward, N_{oc}^{max} is the maximum acceptable number of neuron outgoing connections, $N_n(N_{oc}^{max})$ is the number of neurons for which $N_{oc} \geq N_{oc}^{max}$, and N_n^{max} is the maximum number of neurons with at least N_{oc}^{max} outgoing connections which does not reduce the reward.

3. Reward for dense concentrations of neighboring neurons (RDCNN): This is a variant of RSC which aims to concentrate neurons in dense clusters with short neuron connections. To do so, it encourages to form tight clusters of connections in NDMs. It assumes that if there is a connection $i \rightarrow j$ between two neighboring neurons ($\mathrm{NDM}(i,j) \neq 0$), then there should be also connections: $(i+k) \rightarrow (j+l)\ \forall_{k,l \in -1..1}$. To calculate the reward, RDCNN counts isolated items in NDM, that is, the non–zero items surrounded with at least N_0^{min} zeros – the more the items, the smaller the reward. Formal definition of RDCNN is as follows:

$$RDCNN = \frac{W^{RDCNN}}{(1+N_i)} \tag{3}$$

where W^{RDCNN} is a weight of reward, and N_i is the number of the isolated items in NDM.

4. Reward for short connections (RSC): This mechanism is in line with two previous mechanisms because it also aims to reduce the number

of connections in neural networks. In this case, however, the reduction applies to long connections. The objective of such strategy is to form clusters of neighboring neurons. There are only few long connections ($N_c^{max}(D_c^{max})$) which are allowed without impact on the reward. These connections are intended to be links between remote concentrations of neurons. Formal definition of RSC is as follows:

$$RSC = \begin{cases} W^{RSC} & \text{if } N_c(D_c^{max}) \leq N_c^{max} \\ \frac{W^{RSC}}{(1+N_c(D_c^{max}))} & \text{otherwise} \end{cases} \quad (4)$$

where W^{RSC} is a weight of reward, D_c^{max} is the maximum acceptable length of neuron connections, $N_c(D_c^{max})$ is the number of connections for which $D_c \geq D_c^{max}$, and N_c^{max} is the maximum number of connections longer or equal to D_c^{max} which does not reduce the reward. Since HCAE represents each network in the form of NDM in which each column and row indicates a neuron and its location in 2D space, the length of a connection between i–th and j–th neuron can be simply calculated by $D_c(i, j) = |i - j|$.

4.2 Hand-designed module-oriented mechanisms

In addition to the pressure to create architectures with densely packed neural matter, the impact of which can be seen from generation to generation, the paper also proposes hand-designed mechanisms that have no natural counterpart, and which also aim to build compact neural architectures. These mechanisms are: a new type of neuron, a mechanism of self-regulation of the network size, and a network construction mechanism based on ready-made building-blocks consisting of groups of neurons connected to each other. All the mechanisms are detailed below:

1. Limited number of connections (LNC): This is a combination of RSNOC and RSC which means that LNC also has two goals, namely to minimize the "weight" of the overall architecture and to appropriately organize neurons in the network. In LNC, all neurons are allowed to have at most N_{oc}^{max} outgoing connections. If a neuron has more connections, the longest connections are removed from NDM (see Figure 4).

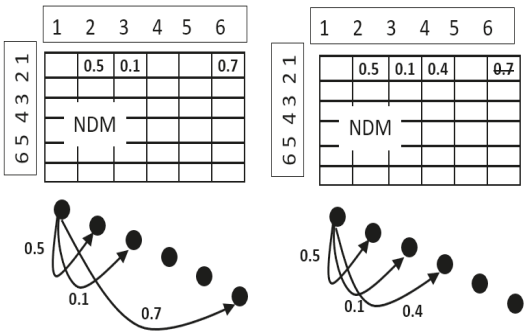


Figure 4. Example application of LNC: each neuron has permission to have maximally $N_{oc}^{max} = 3$ outgoing connections, adding the connection "0.4" from neuron no. 1 to neuron no. 4 removed the longest connection of neuron no. 1, that is, the connection to neuron no. 6

2. Ready modules (RM): This mechanism forms a ready low-level module with N_m densely connected neurons and with only one output neuron – see Figure 5. The neuron can be connected with only one hidden neuron from outside the module or with only one output neuron of the network. RM is implemented as a separate HCAE operation. AEPs applying RM include only operations that implement RM, they do not contain operations specified in Algorithm 1. Since each operation forms a single module in a selected region of the network, the modules generated by different operations can overlap which makes it difficult to create high-level modules.

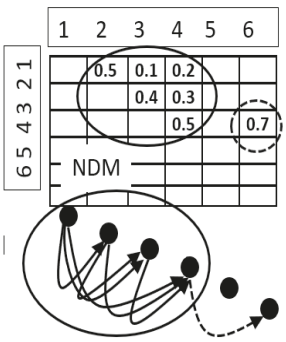


Figure 5. Example RM module consisting of four neurons (no. 1 – no. 4), neuron no. 4 is an output neuron which connects the module with neuron no. 6

3. Multiplication neuron (MN): MN multiplies inputs instead of summing them. It is enough that only one input to the neuron is equal to zero

then its output is also equal to zero. This way it is possible to turn on/off some neurons or even sub–networks. The networks can include any number of MNs which are used along with traditional sigmoid neurons. The number of MNs and their location in the network depends on HCAE decisions.

4.3 Repeated use of the same connection patterns

Copying connectivity schemes in different regions of the network (CCS) by repeated use of the same information stored in a genotype is a mechanism that, in some form, also occurs in the natural world. CCS is implemented as a double execution of operation specified in Algorithm 3, each run is performed in a different region of the network. This way, it increases network regularity, at least at a low–level, but in the case of problems with high–level regularities, it may help with building repeated modules with the same or similar function. Example application of CCS on simple network consisting of six neurons is depicted in Figure 6.

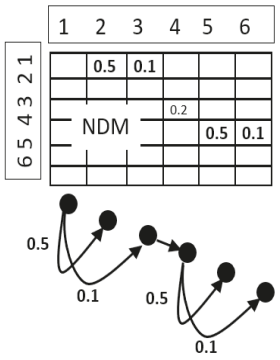


Figure 6. Example application of CCS

4.4 Environmental changes

In addition to internal factors working at the genotypic level or the level of processes responsible for forming the networks, there are also external factors which may affect their shape. Fluctuations in the environment in which the networks work are one of such factors. In the paper, the influence of the environment takes the form of different tasks performed by the networks (alternating goals) and different conditions under which these tasks are performed. In the experiments reported further, six different scenarios of environmental changes were examined. All they assume two tasks to perform by the networks, i.e. the task no. 1 (hard task) and the task no. 2 (easy task):

1. Gradual appearance of a new task (GA): Initially, only the task no. 1 is solved and if the fitness exceeds a threshold, the task no. 2 is gradually added. The more competent is the network in performing the harder task, the greater is the impact of the easier task on the final combined fitness. The effectiveness of each network is calculated, in this case, as follows:

$$E(ANN) = \frac{1}{2}\sum_{j}^{2} I_j E_j \tag{5}$$

$$I_j = \begin{cases} 1 \text{ if } j = 1 \\ S(E_1) \text{ otherwise} \end{cases} \tag{6}$$

where E_j is effectiveness of the network ANN in j–th task, $j = 1,2$, and $S(E_1) \in <0,1>$ is a function which makes E_2 dependent on E_1. If $E_1 < T$, where T is a threshold, than $S(E_1) = 0$, otherwise it linearly grows to 1 along with the increase of E_1.

2. Quick change in the environment (QC): As above, the task no. 2 is not considered in the cumulative effectiveness until fitness exceeds a threshold. Once the threshold is achieved, the network receives evaluation for performing both tasks. It corresponds to binary function $S(E_1)$.

3. Simple task switching (STS): The networks perform either task no. 1 or task no. 2. Switching between the tasks is periodical:

$$E(ANN,i) = \begin{cases} E_1 \text{ if } i \text{ is iteration assigned to task 1} \\ E_2 \text{ otherwise} \end{cases} \tag{7}$$

4. Task switching (TS): The networks perform either task no. 1 or tasks no. 1 and 2. Switching between the tasks is periodical:

$$E(ANN,i) = \begin{cases} E_1 \text{ if } i \text{ is iteration assigned to task 1} \\ \frac{E_1+E_2}{2} \text{ otherwise} \end{cases} \tag{8}$$

5. Gene activity patterns (GAP): This is an adaptation of a solution suggested in [16]. The authors show that modularity in their networks arises when two conditions are met. First, networks

that have been previously trained to reproduce one activity pattern are trained to attain previously learned pattern, and additionally, a new pattern. Each pattern corresponds to a specific activity of each gene (or neuron) in the network. Second, both patterns must have a common part, and a different part, meaning that some genes are active in the same way in both patterns, whereas the remaining genes differ in their activity. The authors explain that modularity in the networks arises in order to reduce the impact of one group of genes on the other group.

In order to implement the above idea, the networks, like in QC, are first trained on the task no. 1 (one activity pattern) and if fitness exceeds a threshold they are trained on tasks no. 1 and no. 2 (new activity pattern). Both tasks correspond to their own specific activity of all four output neurons. However, two output neurons are common for both tasks and two other neurons are specific for each task. In other words, two output neurons have the same activity for the task no. 1 and no. 2, and two other neurons differ in activity for the task no. 1 and for the task no. 2.

6. Gene activity patterns after a specified number of iterations (GAPI): This is a variant of GAP in which the change of the network task does not take place after exceeding a fitness threshold but after a specified number of HCAE iterations.

4.5 Size of networks

Another factor that may affect modularity of the networks is their size, which is understood in the paper as the maximum allowable number of neurons. In the natural world, limited resources determine the size of nature's creations and enforce the appropriate organization of living matter also through its modularization. The larger the structure, the greater the need for more efficient use of resources and optimal organization of matter. In order to examine how the size of the networks evolved by HCAE affects their modularity and efficiency, the tests were performed in which the networks could have a large number of neurons in relation to the complexity of the problem. Moreover, the maximum size of the networks was either infinite and increased gradually over time, or it was determined in advance and did not change during evolution.

5 Experiments

The goal of the experiments was twofold. Firstly, they were a next attempt to discover sources of modularity. To this end, the mechanisms specified in the previous section, both already examined mechanisms and the new ones, different variants of environmental changes, the size of the networks, and their ability to growth were examined in terms of their influence on the architecture and efficiency of the neural networks. Secondly, the research also had a more practical goal, which was to test whether it is possible to increase efficiency of HCAE through biasing the evolution towards modular architectures.

The experiments were carried out in three phases. First, HCAE with inclusion of module–oriented mechanisms was compared with the original variant of the algorithm which does not have any tools to support modularity. The purpose was to estimate how individual mechanisms affect network modularity and effectiveness, the original HCAE was the point of reference in this case. Next, a selected mechanism was examined in the conditions of different environment fluctuations. The influence of the size of the networks on their modularity and effectiveness was examined in the last third phase of the experiments. Furthermore, the ability of the networks to gradually grow without any limits was tested in this phase, as a separate potential module–oriented mechanism.

In all the experiments, the task of the networks was to solve a modular problem consisting of two binary classification sub–problems, one simple and one harder. In both cases, the networks had to split XY space of size 100x100, $x, y \in < -50, 50 >$, into two classes. In the simpler problem, one class was the area between circles of radius 30 and 60, both located in the center of the space, whereas the second class was the rest of the space. In the harder problem, both classes formed intertwined spirals which cannot be linearly separated. A perfect division of the space performed by an example network is depicted in Figure 7.

(a) (b)

Figure 7. Correct division of the input space into two classes: (a) – simpler sub–problem, (b) – harder sub–problem

In order to make neural networks capable of solving the above–mentioned modular problem they contained two input, four output and maximally ninety four hidden neurons. The inputs were fed with (x,y) coordinates of K=196 learning data points, whereas outputs were responsible for identification, one output for one class. All neurons in the networks used sigmoid activation function.

In the tests with MN, the neural networks had a different architecture from the one described above. Instead of two inputs and four outputs they had three inputs and two outputs. The extra input was used to indicate the problem being currently solved: 0 meant the harder problem whereas 1 meant the simpler one. In order to solve both sub–problems, the network had to be run twice. As before, the outputs corresponded to two classes.

A different network architecture was also applied in the tests with GAP and GAPI. In this case, the networks also had three inputs, as the networks applied during the tests with MN, and four outputs. According to GAP/GAPI each sub–problem corresponded to other activity pattern of four output neurons.

In the first and third phase of the experiments, the effectiveness of each evolved neural network was measured as follows:

$$E(ANN) = \frac{1}{2}\sum_{j}^{2} E_j = \frac{1}{2}\sum_{j}^{2}\left(S_j + \frac{1}{1+\sum_{i}^{W_j} E_j^i}\right) \tag{9}$$

$$E_j^i = \begin{cases} 100 \text{ if } o_{2j-2} = o_{2j-1} \\ 1 - o_{2j-1} + o_{2j-2} \text{ if } c(i) = 1 \text{ and } o_{2j-2} \neq o_{2j-1} \\ 1 - o_{2j-2} + o_{2j-1} \text{ otherwise} \end{cases} \tag{10}$$

where

S_j – is the number of correct classifications in j–th sub–problem,
W_j – is the number of wrong classifications in j–th sub–problem,
E_j^i – is an error in i–th learning point, in j–th sub–problem,
$c(i)$ – is a class of i–th data point, $c(i) = 0$ for class no. 0 and $c(i) = 1$ for class no. 1,
o_k – is k–th output neuron, the outputs o_0, o_1 are assigned to sub–problem no. 1 whereas the outputs no. o_2, o_3 to sub–problem no. 2.

In turn, in the second phase, the effectiveness depended on the environmental change scenario and algorithm iteration. In the course of the evolutionary process, the effectiveness was measured according to equations (5)–(8), however, in order to calculate the final effectiveness of the network, (9) was applied.

Since in the tests with GAP and GAPI each task corresponded to an activity pattern of four output neurons, the effectiveness of each network in a single data point E_j^i was measured in this case as follows:

$$E_1^i = \begin{cases} 100 \text{ if } o_0 = o_1 \\ 1 - o_1 + o_0 + o_2 + o_3 \text{ if } c(i) = 1 \text{ and } o_0 \neq o_1 \\ 1 - o_0 + o_1 + o_2 + o_3 \text{ otherwise} \end{cases} \tag{11}$$

$$E_2^i = \begin{cases} 100 \text{ if } o_0 = o_1 \text{ or } o_2 = o_3 \\ 1 - o_1 + o_0 + 1 - o_3 + o_2 \text{ if } c(i) = 1 \text{ and } o_0 \neq o_1, o_2 \neq o_3 \\ 1 - o_0 + o_1 + 1 - o_2 + o_3 \text{ otherwise} \end{cases} \tag{12}$$

The outputs no. 0 and 1 corresponded to a common part of activity patters for both sub–problems, whereas, the outputs no. 2 and 3 were specific for each sub–problem.

During the evolutionary process, the networks received fitness F for their effectiveness E, and in some cases, also for following the restrictions imposed on their architecture:

$$F(ANN) = E(ANN) + R(ANN) \tag{13}$$

where $R = RSNC, RSNOC, RSC, RDCNN$ for the module–oriented mechanisms no. 1,2,3,5 and $R = 0$ in the remaining cases.

After the learning process, each evolutionary run was represented by the highest–fitness neural

network that was then evaluated in terms of both the effectiveness E and the modularity $M \in < 0,1 >$. The modularity was measured according to the Louvain [5] method for community detection. Afterwards, in order to characterize each network by a single cumulative measure of both the effectiveness and modularity, the following formula was applied:

$$EM(ANN) = \frac{E_{<0,1>}(ANN) + M(ANN)}{2} \quad (14)$$

$$E_{<0,1>}(ANN) = \begin{cases} 0 \text{ if } E(ANN) < 160 \\ 0.25 \text{ if } E(ANN) < 170 \\ 0.5 \text{ if } E(ANN) < 180 \\ 0.75 \text{ if } E(ANN) < 190 \\ 1 \text{ otherwise} \end{cases} \quad (15)$$

Note that $E = 190$ means 2x190 data points (96%) correctly classified in both sub–problems.

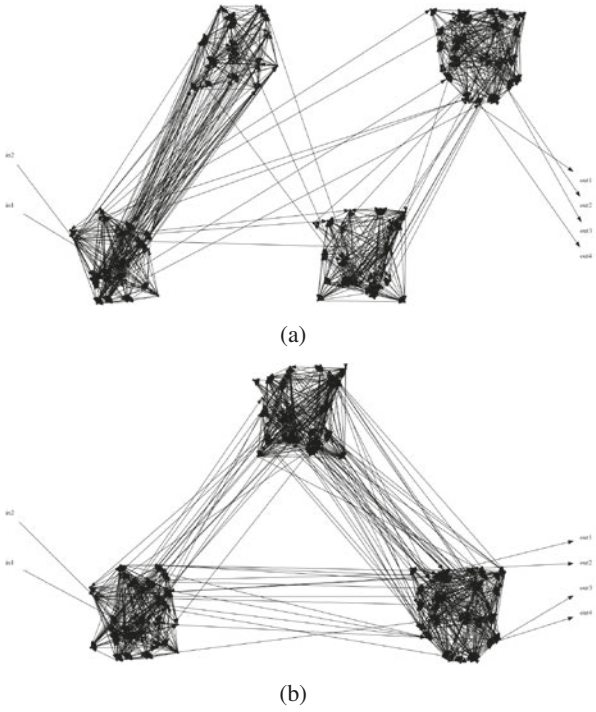
In all the examined cases, the first step was to find parameters which maximized fitness F. For each parameter setting the evolutionary process was run fifty times and it always ended after 4 million iterations. The results presented in the following sections correspond to the most fit settings.

5.1 Results in first phase of experiments

The results of the first phase of the experiments are summarized in Tab. 1 and Figure 8. They include results averaged over fifty runs of each module-oriented mechanism. As already mentioned each run was represented by the highest–fitness neural network that was then evaluated in terms of both the effectiveness E and modularity $M \in < 0,1 >$. In addition to EM, $E_{<0,1>}$, and M the table also includes coefficient C which indicates sparsity of the evolved networks and is measured as follows: $C(ANN) = \frac{N_c}{N_{All}} 100\%$, where N_{All} is the number of all possible connections in the network.

The table shows that RM which forms neural networks through putting together small modules consisting of $N_m = 20$ densely connected neurons and one output neuron, is the only mechanism in which the increase in modularity entails an increase in efficiency. EM, in this case, clearly shows that networks evolved by HCAE–RM are not only effective but also modular. From the point of view of HCAE itself, it is also important that regardless of modularity, RM leads to an increase in algorithm efficiency. What is more, even though the networks are composed of overlapping densely connected modules, as a whole they are characterized by a high sparsity compared to the networks produced by the original HCAE.

The decomposition into modules made by the Louvain algorithm also reveals that HCAE–RM evolves architectures in which the modules constitute sequentially connected layers, each with a dense inner connectivity. The assignment of neurons to modules depends on the distance of the neuron to the input. Neurons closest to the input form the first layer, whereas the neurons situated further constitute subsequent layers. In majority of cases, the layers are connected only with the neighbors – the connections between distant layers are either very rare or none at all. Example networks generated by HCAE–RM are presented in Figure 9.



(a)

(b)

Figure 9. Example networks evolved by HCAE–RM, network (a) – four modules, network (b) – three modules determined by Louvain algorithm

RSNOC and RSC are the mechanisms which also contributed to the increase in modularity, however, without any positive effect on the effectiveness which either decreased or remained, more or less, at the same level. As in the case of RM, the networks are sparser in terms of connectivity than networks

Table 1. Results of the first phase of experiments

	HCAE	RSNC	RSNOC	RDCNN	RSC	LNC	RM	CCS	MN
Avg. EM	0.5	0.5	0.53	0.5	0.52	0.28	**0.7**	0.45	0.34
Std. EM	0.03	0.048	0.042	0.038	0.04	0.059	**0.04**	0.047	0.043
Avg. $E_{<0,1>}$	0.83	0.8	0.8	0.82	0.79	0.15	**0.87**	0.72	0.43
Std. $E_{<0,1>}$	0.03	0.048	0.042	0.038	0.04	0.059	**0.04**	0.047	0.043
Avg. M	0.18	0.21	0.26	0.18	0.27	0.41	**0.53**	0.2	0.25
Std. M	0.011	0.009	0.013	0.011	0.009	0.16	**0.012**	0.012	0.013
Avg. C[%]	42	30	28	42	20	8	**25**	39	28
Std. C[%]	1.7	1.3	1.5	1.5	1	0.8	**0.5**	1.8	1.6

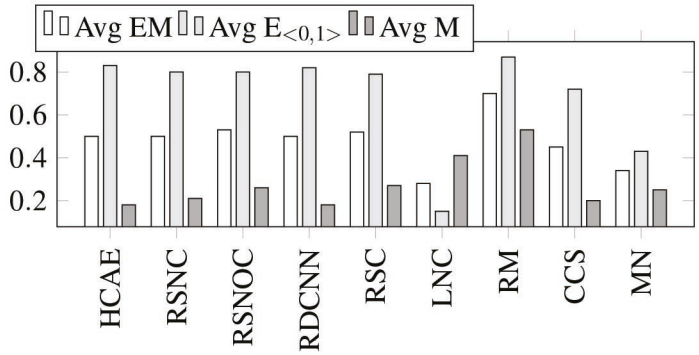


Figure 8. Average results in first phase of experiments

evolved by the original HCAE. In both cases, the highest fitness was achieved for the slightest pressure on reducing the number of connections. Any increase in the pressure resulted in an increase in modularity and a decrease in efficiency. There was not even a single case which would combine a high modularity and a high efficiency.

As regards the network architectures, the Louvain algorithm often splits them into a greater number of modules than in the case of RM. What is more, it is difficult to determine any rule of assigning neurons to modules. The assignments are not so chaotic as in the original HCAE, however, they often combine together distant neurons which rather did not take place in RM.

RSNC which is a counterpart of the connection cost mechanism, in spite of lighter architectures which were generated in this case, practically did not have effect on network modularity and effectiveness. As above, the increase in the network sparsity goes hand in hand with a greater modularity and a lower efficiency. The network architectures were, in this case, similar to those generated by RSNOC and RSC.

RDCNN which is focused on clustering the neighboring neurons without any bias towards sparsity achieved, in principle, identical results as the original HCAE. The only thing that differs both approaches is the architecture of the networks. In the case of RDCNN, they are more like RSNOC and RSC architectures than those evolved by HCAE.

The remaining mechanisms examined in this phase significantly decrease effectiveness of the original algorithm. LNC generates very small networks which turned out to be insufficient to solve the problem. CCS doubles actions of HCAE which also appeared to be disadvantageous solution, at least, in the case of the considered modular problem. The same applies to MN which was unable to divide the networks into effectively cooperating modules.

5.2 Results in second phase of experiments

In order to test the influence of different forms of environmental fluctuations on network modularity/efficiency, RSNOC was used as a point of reference for all results obtained in this phase.

The results of this phase in the form of averages calculated for fifty runs of each HCAE variant with different scheme of environmental changes are given in Tab. 2 and Figure 10. They generally do not confirm earlier observations of other researchers that environmental changes positively affect modularity. The only case of the modularity increase, compared with RSNOC, is noticed for STS. However, a higher modularity, in this case, is rather due to a high sparsity of the networks, what is more, it goes hand in hand with very poor effectiveness.

Table 2. Results of the second phase of experiments

	GA	QC	STS	TS	GAP	GAPI
Avg. EM	0.27	0.52	0.21	0.48	0.35	0.45
Std. EM	0.029	0.017	0.007	0.021	0.033	0.023
Avg. $E_{<0,1>}$	0.29	0.75	0	0.68	0.42	0.65
Std. $E_{<0,1>}$	0.062	0.035	0	0.048	0.077	0.052
Avg. M	0.25	0.26	0.43	0.29	0.29	0.25
Std. M	0.011	0.01	0.014	0.015	0.016	0.01
Avg. C[%]	29	27	9	24	23	28
Std. C[%]	1.1	1	0.7	1.4	2.2	1.5

As it appeared, such result of STS is an effect of forgetting previously learned skills while learning new skills. The interneuron connections useful for one task were destroyed while learning the new task. The longer the break in learning a single task was, the worst the cumulative effectiveness also was. The best effectiveness of STS was achieved for very short intervals between the tasks.

The result of STS was improved in TS, which according to the assumptions should build skills in one task and gradually add skills in another task without forgetting previously acquired skills. Unfortunately, again, it was not possible to improve the modularity and efficiency compared to RSNOC.

One–time change of the environmental conditions applied in the remaining scenarios either deteriorates results or remains them at a more or less the same level compared to RSNOC. In this case, a better solution was a drastic immediate change of the conditions than a gradual change. As it turned out the gradual change applied in GA extended the process of adjusting to new conditions with the consequence that this process often did not completed up to the last iteration of the algorithm which disadvantageously affected the final effectiveness.

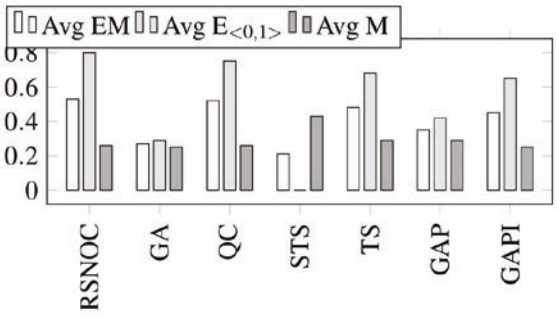


Figure 10. Average results in second phase of experiments

It was also impossible to repeat the results presented in [16]. Both GAP and GAPI which are an adaptation of the solution proposed in [16] produce networks which are less effective and similar in terms of the modularity to the networks evolved according to RSNOC.

Table 3. Results of the third phase of experiments

	Gradual	300
Avg. EM	0.55	0.6
Std. EM	0.018	0.017
Avg. $E_{<0,1>}$	0.8	0.84
Std. $E_{<0,1>}$	0.037	0.035
Avg. M	0.3	0.38
Std. M	0.01	0.01
Avg. C[%]	13	5
Std. C[%]	0.8	0.7

5.3 Results in third phase of experiments

In this phase, again, the networks evolved according to RSNOC. However, in contrast to all the previous tests, the networks were allowed to contain more neurons. In the first and second phase, the networks could have maximum 100 neurons. In this phase, they could either grow infinitely or have maximum 300 neurons. In HCAE, limitations on the number of neurons result from the size of the matrix NDM which represents the networks. The final number of neurons is always a decision of HCAE, however, it cannot be greater than the number of rows in NDM.

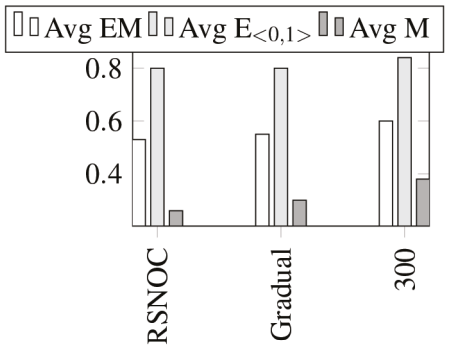
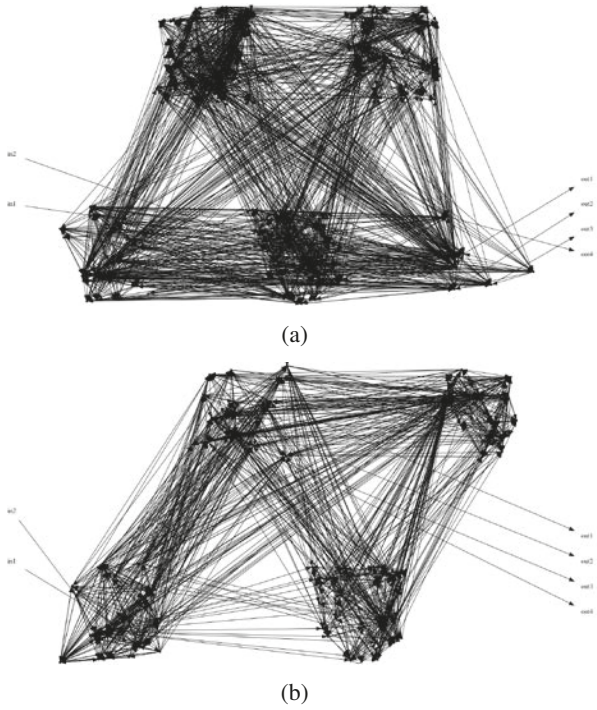


Figure 11. Average results in third phase of experiments



(a)

(b)

Figure 12. Example networks evolved by HCAE–300, network (a) – five modules, network (b) – four modules determined by Louvain algorithm

In the case of the growth of the networks, NDMs enlarged their size during the evolution which corresponded to the increase of the maximum number of neurons which was allowed at a given stage of the evolutionary process. The change of the size was gradual and it took place after a definite number of HCAE iterations without progress. The initial maximum number of neurons was equal to 50. The results of this phase are given in Tab. 3 and Figure 11.

They, generally, show a slight increase in all criteria compared to RSNOC. The networks encoded in larger NDMs were more modular, more effective and very sparse. Such result is in line with results of other researchers and it confirms the key role of the scale in the process of evolving modular neural networks. The enlarged NDMs provide much more space for HCAE programs which have more options for forming modules than the programs operating on smaller matrices.

Despite enlarged NDMs, even three times, the number of connections did not increase proportionally to the new size of the matrices. The average number of connections for RSNOC, Gradual, and 300 was as follows: 1372, 1574, 2023. Example networks evolved in this phase are depicted in Figure 12.

A slight number of connections in relation the maximum number is due to RSNOC mechanism applied during the evolutionary process. Combining this mechanism with extended space of possible neural networks seems to be the main driver of improved modularity and efficiency of the networks in this phase of the experiments.

6 Summary

The paper deals with the issue of modularity of neural networks designed in the evolutionary way by means of a generative algorithm called Hill Climb Assembler Encoding. However, modularity is not an end in itself. The goal of the paper is to find mechanisms which lead to both modularity and efficiency.

In the paper, a number of module–oriented mechanisms are proposed. In addition to mechanisms suggested earlier by other researchers, new mechanisms are also proposed and examined like outgoing connection cost, connection length cost, inter–neuron distance cost, the ability to growth of the networks, assembling the networks from small evolutionary formed modules, grouping neurons in densely packed clusters, replicating neuron concentrations in different network locations, and a new type of neuron whose task is to split the networks into separate clusters.

Apart from the above–mentioned mechanisms, the influence of the network size and fluctuations

in the environment on network modularity and efficiency was also examined.

The experiments reported in the paper revealed positive impact of the connection cost mechanism on the combined modularity/efficiency. For small networks, the impact is hardly noticeable, however, it increases for larger networks. The outgoing connection cost (RSNOC) and connection length cost (RSC) turned out to be slightly more effective than the original solution (RSNC) that is oriented towards reducing the overall network connectivity.

The size of the space of possible neural solutions is a next key factor that decides about modularity/efficiency. A smaller space represented by smaller matrices NDM results in a greater density of connections and neurons and it is not conducive to arising separate loosely connected modules. There is simply no room for spreading neurons over a larger space. In turn, larger matrices make it possible to locate clusters of neurons far away from each other which seems to be very helpful in generating cooperating modules.

However, extending the matrices cannot be used in isolation. In order for modular/efficient neural networks to evolve, the connection cost must be applied as well. The lack of this mechanism results in heavier architectures in which it is difficult to form the modules.

The experiments also revealed that larger modular/effective networks can be successfully made up of small densely connected modules with only one output neuron (RM mechanism). As it turned out, the application of this mechanism produced the best results out of all examined solutions, including the original HCAE. Majority of networks with small modules incorporated were very effective as well as modular. What is more, most of them had a very interesting architecture with sequentially connected modules and with very rare connections between distant modules.

The remaining mechanisms along with the environmental changes either resulted in noticeable decrease in efficiency or had no serious influence on the achieved results. This is particularly surprising concerning the environmental changes which are commonly regarded as one of key factors leading to modularity.

Acknowledgements

The paper is supported by Polish Ministry of Defense within the framework of the program entitled ”Research Grant”.

References

[1] S. Ahmadian, S. Jalali, S. Islam, A. Khosravi, E. Fazli, and S. Nahavandi. A novel deep neuroevolution-based image classification method to diagnose coronavirus disease (covid-19). Comput Biol Med., (139:104994), 2021.

[2] A. Baldominos, Y. Saez, and P. Isasi. Evolutionary convolutional neural networks: an application to handwriting recognition. Neurocomputing, 283:38–52, 2018.

[3] C.Y. Baldwin and K.B. Clark. Design Rules: The power of modularity. Chapter 3: What Is Modularity? MIT Press, 2018.

[4] A. Billard and M. J. Mataric. Learning human movements by imitation: evaluation of a biologically inspired connectionist architecture. Robotics and Autonomous Systems, 941:1–16, 2001.

[5] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics Theory and Experiment, 10:10008, 2008.

[6] R. Calabretta and J. Neirotti. Adaptive agents in changing environments, the role of modularity. Neural Process Lett, 42:257–274, 2015.

[7] M. Carcenac. A modular neural network applied to image transformation and mental images. Neural Computing and Applications, 17:549–568, 2008.

[8] J. Clune, B.E. Beckmann, P.K. McKinley, and C. Ofria. Investigating whether hyperneat produces modular neural networks. In Proceedings of the 12th annual conference on Genetic and evolutionary computation, pages 635–642, 2010.

[9] J. Clune, J-B. Mouret, and H. Lipson. The evolutionary origins of modularity. In Proceedings of the Royal Society B, 2013.

[10] [10] A.S. Cofino, J.M. Gutierrez, and M.L. Ivanissevich. Evolving modular networks with genetic algorithms: application to nonlinear time series. Expert Systems, 21(4):208–216, 2004.

[11] Y. J. Cruz, M. Rivas, R. Quiza, A. Villalonga, R. E. Haber, and G. Beruvides. Ensemble of convolutional neural networks based on an evolutionary

algorithm applied to an industrial welding process. Computers in Industry, 133:103530, 2021.

[12] K. Deb, A. Pratap, S. Agarwal, , and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.

[13] S. Doncieux and J. Meyer. Evolving modular neural networks to solve challenging control problems. In Proceedings of the Fourth International ICSC Symposium on Engineering of Intelligent Systems, 2004.

[14] K. O. Ellefsen and J. Torresen. Evolving neural networks with multiple internal models. In Proceedings of the 14th European Conference on Artificial Life ECAL 2017, volume 14, pages 138–145, 2017.

[15] K.O. Ellefsen, J-B. Mouret, and J. Clune. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. PLoS Computational Biology, 11(4):e1004128, 2015.

[16] C. Espinosa-Soto and A. Wagner. Specialization can drive the evolution of modularity. PLoS Computational Biology, 6(3):e1000719, 2010.

[17] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D.Wierstra. Convolution by evolution: differentiable pattern producing networks. In Proceedings of the 2016 Genetic and Evolutionary Computation Conference, pages 109–116, 2016.

[18] D. Filan, S. Hod, C. Wild, A. Critch, and S. Russell. Pruned neural networks are surprisingly modular. Technical Report arXiv:2003.04881 [cs.NE], ArXiV, 2020.

[19] J. Gauci and K. Stanley. Generating large–scale neural networks through discovering geometric regularities. In Proceedings of the Genetic and Evolutionary Computation Conference, pages 997–1004, 2007.

[20] S. Han and S. Oh. An optimized modular neural network controller based on environment classification and selective sensor usage for mobile robot reactive navigation. Neural Computation and Application, 17:161–173, 2008.

[21] J. Huizinga, J.B. Mouret, and J. Clune. Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pages 697–704, 2014.

[22] M. Hulse, S. Wischmann, and F. Pasemann. Structure and function of evolved neuro– controllers for autonomous robots. Connection Science, 16(4):249–266, 2004.

[23] L. Kirsch, J. Kunze, and David Barber. Modular networks: Learning to decompose neural computation. Technical Report arXiv:1811.05249 [cs.LG], ArXiV, 2018.

[24] J. Koutnik, J. Schmidhuber, and F. Gomez. Evolving deep unsupervised convolutional networks for vision–based reinforcement learning. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pages 541–548, 2014.

[25] V. Landassuri-Moreno and J. A. Bullinaria. Biasing the evolution of modular neural networks. In 2011 IEEE Congress of Evolutionary Computation, 2011.

[26] J. Liang, E. Meyerson, and R. Miikkulainen. Evolutionary architecture search for deep multitask networks. In GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference, pages 466–473, 2018.

[27] I. Loshchilov and F. Hutter. CMA–ES for hyperparameter optimization of deep neural networks. Technical Report arXiv: abs/1604.07269 [cs.NE], ArXiV, 2016.

[28] P. Melin, D. Bravo, and O. Castillo. Fingerprint recognition using the fuzzy sugeno integral for response integration in modular neural networks. International Journal of General Systems, 37(4):499–515, 2008.

[29] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. Technical Report arXiv abs/1703.00548 [cs.NE], ArXiV, 2017.

[30] J-B. Mouret and S. Doncieux. Evolving modular neural networks through exaptation. In 2009 IEEE Congress on Evolutionary Computation, pages 1570–1577, 2009.

[31] H. Munn and M. Gallagher. Modularity in NEAT reinforcement learning networks, 2022.

[32] N. NourAshrafoddin, A. R. Vahdat, and M. M. Ebadzadeh. Automatic design of modular neural networks using genetic programming. In Proceedings of the 17th International Conference on Artificial Neural Networks ICANN 2007 Part I, pages 788–798, 2007.

[33] M. Potter. The Design and Analysis of a Computational Model of Cooperative Coevolution. PhD thesis, George Mason University, 1997.

[34] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation, 8(1):1–29, 2000.

[35] T. Praczyk. Probabilistic neural network application to warship radio stations identification. Computational Methods in Science and Technology, 13(1):53–58, 2007.

[36] T. Praczyk. Using augmenting modular neural networks to evolve neuro–controllers for a team of underwater vehicles. Soft Computing, 18(12):2445–2460, 2014.

[37] T. Praczyk. Cooperative co–evolutionary neural networks. Journal of Intelligent & Fuzzy Systems, 30(5):2843–2858, 2016.

[38] T. Praczyk. Hill climb modular assembler encoding: Evolving modular neural networks of fixed modular architecture. Knowledge-Based Systems, 232:107493, nov 2021.

[39] K. Soltanian, A. Ebnenasir, and M. Afsharchi. Modular grammatical evolution for the generation of artificial neural networks. Evolutionary Computation, 30(2):291–327, 06 2022.

[40] S. Sotirov, E. Sotirova, V. Atanassova, K. Atanassov, O. Castillo, P. Melin, T. Petkov, and S. Surchev. A hybrid approach for modular neural network design using intercriteria analysis and intuitionistic fuzzy logic. Complexity, 1:1–11, 2018.

[41] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10:99–127, 2002.

[42] Y. Sun, B. Xue, M. Zhang, and G. G. Yen. Automatically designing CNN architectures using genetic algorithm for image classification. Technical Report arXiv:1808.03818 [cs.NE], ArXiV, 2018.

[43] C. R. Tosh. Can computational efficiency alone drive the evolution of modularity in neural networks? Scientific Reports, 6:31982, 2016.

[44] C. R. Tosh and L. McNally. The relative efficiency of modular and non–modular networks of different size. In Proceedings of the Royal Society B: Biological Sciences, volume 282:20142568, 2015.

[45] A. Turan, S. D. Hinchberger, and M. H. El Naggar. Predicting the dynamic properties of glyben using a modular neural network (MNN). Canadian Geotechnical Journal, 45:1629–1638, 2008.

[46] V. K. Valsalam and R. Miikkulainen. Evolving symmetric and modular neural networks for distributed control. In Proceedings of the Genetic and Evolutionary Computation Conference, 2009.

[47] L. Xie and A. Yuille. Genetic CNN. Technical Report arXiv abs/1703.01513 [cs.NE], ArXiV, 2017.

[48] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks, 8(3):694– 713, 1997.

[49] S.R. Young, D.C. Rose, T.P. Karnowsky, S.H. Lim, and R.M. Patton. Optimizing deep learning hyper–parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High–Performance Computing Environments, number 4, pages 1–5, 2015.

[50] Z. Zhu, S. Guo, and M. Liao. Deep neuroevolution: Evolving neural network for character locomotion controller. In 2021 2nd International Conference on Artificial Intelligence and Information Systems, ICAIIS 2021, New York, NY, USA, 2021. Association for Computing Machinery.

Tomasz Praczyk was born in Grudziadz, Poland in 1971. He received the M.S. degree in computer science from the Military Technical Academy, Warszawa, in 1996, the Ph.D. degree in maritime navigation from the Maritime Academy, Szczecin, in 2001 and the habilitation in computer science from the Military Technical Academy, Warszawa, in 2012. From 1996 to 2002, he was a Programmer/Designer with the Computer Center of Polish Navy, from 2002 to 2013 he was a Senior Lecturer with the Polish Naval Academy, and since 2013, he has been an Assistant Professor with the same Academy. His research interests include neuro-evolution, swarm-intelligence, deep learning, underwater robotics.
https://orcid.org/0000-0003-0547-7935