

Automatic Modularization of Artificial Neural Networks

Eva Volna

University of Ostrava, 30th dubna st. 22, 701 03 Ostrava, Czech Republic

Abstract. The majority of this paper relies on some forms of automatic decomposition tasks into modules. Both described methods execute automatic neural network modularization. Modules in neural networks emerge; we do not build them straightforward by penalizing interference between modules. The concept of emergence takes an important role in the study of the design of neural networks. In the paper, we study an emergence of modular connectionist architecture of neural networks, in which networks composing the architecture compete to learn the training patterns directly from the interaction of reproduction with the task environment. Network architectures emerge from an initial set of randomly connected networks. In this way can be eliminated connections so as to dedicate different portions of the system to learn different tasks. Mentioned methods were demonstrated for experimental task solving.

1 Reasons for a Modular Approach

The primary reason for adopting an ensemble approach to combining nets into a modular architecture is that of improving performance. There are a number of possible justifications for taking a modular approach to combining artificial neural nets. First, a modular approach might be used to solve a problem which could not have been solved through the use of a unitary net. A modular system of nets can exploit the specialist capabilities of the modules, and consequently achieve results, which would not be possible in a single net. Another reason for adopting a modular approach is that of reducing model complexity, and making the overall system easier to understand. This justification is often common to engineering design in general. Other possible reasons include the incorporation of prior knowledge, which usually takes the form of suggesting an appropriate decomposition of the global task. A modular approach can also reduce training times and make subsequent modification and extension easier. Finally, a modular approach is likely to be adopted when there is concern to achieve some degree of neurobiological or psychological plausibility, since it is reasonable to suppose that most aspects of information processing involve modularity.

A modular neural network can be characterized by a series of independent neural networks moderated by some intermediary. Each independent neural network serves as a module and operates on separate inputs to accomplish some subtask of the task the network hopes to perform [1]. The intermediary takes the outputs of each module and processes them to produce the output of the network as a whole. The interme-

each other.

When a modular approach is adopted, for what ever reason, there are different ways in which a problem might be decomposed. In particular, task decomposition can be either explicit or automatic. Explicit decomposition is likely to depend on an understanding of the task and the capabilities of the modular components. It provides a way of incorporating prior knowledge and understanding of the task in question. For instance, a particular decomposition might be implied by the structure of the task, if for example, the data came from different sources or took different forms [3]. Similarly, modular decomposition might be guided by theories or evidence about the likely modular structures in the human brain, or the human information processing system. By contrast, automatic decomposition, where decomposition is accomplished through the blind application of a data partitioning algorithm, is particularly useful when expert knowledge of the task is not available.

There has been a considerable amount of research on automatic decomposition methods, for example, the mixture-of-experts [4] and hierarchical mixtures-of-experts approaches [6]. Under such methods, the input data is partitioned into several subspaces, and simple systems are trained to fit the local data. Such data partitioning is often more effective than training on the whole input data space. In general, the concern in this work is to improve performance, and as such it is closely related to the ensemble approach. Thus performance on a task could be improved by either taking a modular decompositional approach, or by creating an ensemble of parallel solutions to the problem, and combining them in some way. As yet, it is not clear where one approach is likely to be better than the other [7]. It is increasingly recognized that the effectiveness of ensemble approaches depends on the extent to which their failures are correlated and a decompositional approach promotes the reduction of such correlation. However, there are few direct comparisons of the relative effectiveness of a modular approach relying on automatic decomposition, and an ensemble-based approach. Neither are the two alternatives necessarily mutually exclusive, since it is possible to envisage an ensemble system, where each member was composed of a set of modules created through automatic decomposition. The majority of this paper relies on some forms of automatic decomposition tasks into modules. In this way can be eliminated connections so as to dedicate different portions of the system to learn different tasks.

2 Automatic Task Decomposition

An artificial neural network may show slow learning because it is being trained to simultaneously perform two or more tasks. For example, suppose that the mapping from the input units to each output unit constitute separate tasks and that the network is trained via backpropagation algorithm. During training, each output unit provides error information to the hidden units from which it receives a projection. It is possible that the error information from one output unit may indicate that a hidden unit's activation should be larger and, at the same time, the error information from another output unit may indicate that the same unit's activation should be smaller. This conflict in the error information is called spatial crosstalk. Although spatial crosstalk is clearly

seen in terms of the backpropagation algorithm, it is limited to networks trained using this algorithm. Therefore, spatial crosstalk may be considered as resulting from the connectivity of the network and not from the learning algorithm used to training the network. By maintaining short connections and eliminating long connections, spatial crosstalk can be reduced and tasks can be decomposed into subtasks. Although the three systems show in Fig. 1 [5] can be trained to perform the same mapping. System in Panel A has its hidden units fully interconnected with its output units and is most susceptible to spatial crosstalk. System in the Panel B has its hidden units on the top fully interconnected with its top output units and its hidden units on the bottom fully interconnected with its bottom output units. Thus, it consists of two separate networks (two 4-4-2 networks). If the mapping that this system is trained to perform can be decomposed so that the mapping from the input units to the top set of output units may be thought of as one task and the mapping from the input units to the bottom set of output units may be thought of as a second task, then this system has dedicated different networks to learn the different tasks. Because there is no spatial crosstalk between the two tasks, such a system may show rapid learning. The Panel C has hidden unit project to only a single output unit. It therefore consists of a separate network for each output unit (four 4-2-1 networks) and is immune to spatial crosstalk.

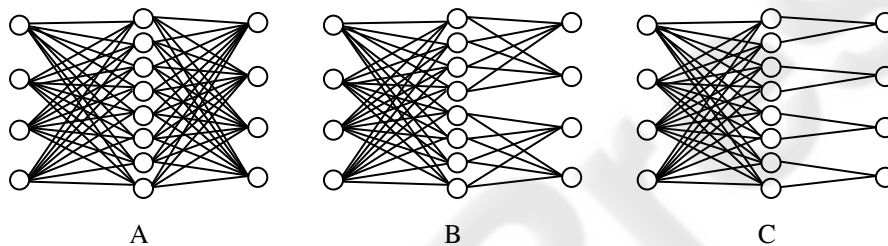


Fig. 1. A: One 4-8-4 network. B: Two 4-4-2 networks. C: Four 4-2-1 networks [5].

Artificial neural network with many adjustable weights may learn to training data quickly and accurately, but generalize poorly to novel data. One method of improving the generalization abilities of network with too many “degrees of freedom” is to decay or eliminate weights during training. A second method is to match the structure of the network with the structure task. For example, networks, whose units have local receptive fields, can learn to reliably, detect the local structure that is often present in pattern recognition tasks. A system that maintains short connections and eliminates long connections should generalize well because its degrees of freedom are reduced and because its units develop local receptive fields.

Artificial neural network often develop relatively non interpretable representations for at least two reasons. Networks whose units are densely connected tend to develop representations that are distributed over many units and, thus, are difficult to interpret. In addition, non interpretable representations often develop in networks that are trained to simultaneously perform multiple tasks. In contrast, networks, whose units tend to have local receptive fields, towards short connections may develop relatively local representations. Furthermore, such a system may be capable of eliminating connections so that different networks learn different tasks.

3 Evolutionary Module Acquisition

There is a simple model of evolutionary emergence of modular neural network topology introduced in the chapter [10]. We describe a method of optimization of the modular neural network architecture via evolutionary algorithms that uses a fix part of network architecture in the genome. Every individual is a multilayer neural network with one hidden layer of units. We have to fix its maximal architecture (e.g. number of input, hidden and output units) before the main calculation. Population P consists of $P = \{\alpha_1, \alpha_2, \dots, \alpha_p\}$, where p is equal to a number of chromosomes in P . Every chromosome consists of binary digits that are generated randomly with a probability 0.5. Chromosome, with m hidden units a n output units is shown in Fig. 2, where $e_{ij} = 0$, if the connection between i -th hidden unit and j -th output unit of the individual doesn't exist, and $e_{ij} = 1$, if the connection exists ($i = 1, \dots, m; j = 1, \dots, n$). Connections between input and hidden units are not included in chromosomes, because they are not necessary for modular network architecture creation. Each individual (e.g. the network architecture) is partially adapted by backpropagation, its fitness function is then calculated as follows (1):

$$Fitness_k = \frac{1}{E_k} \quad (1)$$

where $k = 1, \dots, p$ (p = number of individuals in the population);
 E_k is the error after backpropagation adaptation of the k -individual.

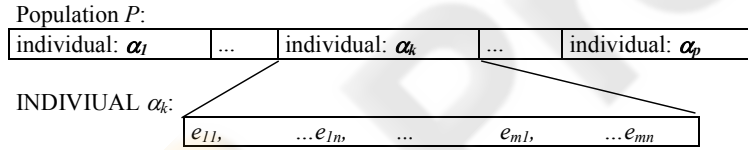


Fig. 2. A population of individuals.

Only two mutation operators are used, no crossover operators. The *first* mutation operator is defined in following way. In the every generation, one individual is randomly chosen and each bit is changed with probability 0.01 (e.g. if the connection exists – after mutation it does not exist and vice versa) in its chromosome. The *second* mutation operator is defined in following way, see Fig. 3. First, we define a pattern of t -consecutive zeroes that will be fixed during whole calculation. The pattern is determined by number of neurons in the output layer, which represent individual modules. Output neurons are organized into d modules, $t = \min(t_i, i = 1, \dots, d)$, where t is number of neurons in the pattern, and t_i is number of neurons in the i -th module. Defined pattern is represented as a continuous chain of t -zeros, which is not changed during applications of the second mutation operator. Fixation of t -zeros chain can be defended by biological motivation, where the protection against mutation is usually related to continuous section. Defined pattern in the chromosome allows temporary fixing the existing module against the application of the second mutation operator. Then we find the define pattern in each chromosome. If we find only

one continuous pattern, we fix it. If we find more than n -consecutive zeroes, we randomly choose n -consecutive zeroes from them and fix them. The fixed pattern represents a single atomic unit and the second mutation operator is not applied to it. Only to the rest of bits from chromosomes are changed with probability 0.01. Thus, each individual has a unique collection of fixed patterns. The second mutation operator is applied to every individual r -times, where r is a parameter and its value is define before calculation. Only the best individual or its best mutation is included into the next generation. Next, all individuals in the new generation release a portion of the pattern that was fixed that way they can once again be manipulated by reproduction operators. The process of evolutionary algorithms is ended when the population achieves the maximal generation or if there is no improvement in the objective function for a define sequence of consecutive generations.

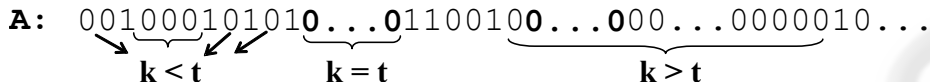
A: 001000101010...01100100...000...0000010...

B: 001000101010...01100100...000...0000010...
 001000101010...01100100...000...0000010...

Fig. 3. The second mutation operator. The fixed pattern is t -consecutive zeroes, k is number of consecutive zeroes in the chromosome. A: An individual before mutation. B: Possible chromosomal representation of the individual after mutation.

4 Modularization Via Evolutionary Hill – climbing Algorithm

The second presented method is based on hill-climbing algorithm with learning [8]. Evolution of the probability vector is modeled by a genetic algorithm on the basis of the best evaluated individuals in this algorithm, which are selected on the basis of the speed and quality of learning of the given tasks [11]. Population P is presented in Fig. 2 and is defined in the same way as in the previous chapter. Individuals in the next generation are generated from the updating probability vector. Every individual (e.g. its neural network architecture) is partially adapted by backpropagation [2] and evaluated by the quality of its adaptation. The number of epochs is a very important criterion in the described method, because modular architectures start to learn faster than fully connected multilayer connectionist networks [9]. Our goal is to produce such a neural network architecture that is able to learn a given problem with the smallest error. A backpropagation error is a fitness function parameter. A fitness function value F_i of the i -th individual is calculated as follows (2):

$$F_i = \frac{\sum_{k=1}^{con} f_{ik}}{con} \quad (2)$$

where $i = 1, \dots, p$ (p is number of individuals in a population);

$f_{ik} = \frac{1}{E_{ik}}$ is a fitness function value of the i -th individual in the k -th adaptation;
 $k = 1, \dots, con$ (con is a define constant, $con > 1$);
 E_{ik} is the backpropagation error of the i -th individual in the k -th adaptation.

Crossover and mutation operators are not used in the described method. This algorithm is based on the probability vector emergency. The probability vector is updated on the basis of well - evaluated individuals in the population. Entries $0 \leq w_{ij} \leq 1$ of the probability vector $\mathbf{w} = (w_{11}, \dots, w_{1n}, \dots, w_{m1}, \dots, w_{mn}) \in [0,1]^{mn}$, (m is number of hidden units; n is number of output units) determine probabilities of appearance of '1' entries in given positions.

Entries of the Probability Vector are Calculated in the Next Generation as follows:

- We calculate F_{avg} , e.g the average fitness value of the population in the given generation (3):

$$F_{avg} = \frac{\sum_{i=1}^p F_i}{p}, \quad (3)$$

where p is a number of individual in the population;
 F_i is a fitness function value of the i -th individual, see a formula (2).

- We choose a set of q individuals with $F_i \geq F_{avg}$, e.g. $\alpha_1, \alpha_2, \dots, \alpha_q$ ($1 \leq q \leq p$, where p is a number of individuals in the population).
- Entries of the probability vector of the population $w'_k \in [0,1]$ are calculated as follows (4):

$$w'_k = (1 - \lambda)w_k + \lambda w''_k \quad (4)$$

where $k = 1, \dots, mn$ (mn is a number of the probability vector \mathbf{w} entries);
 w_k is a value of the k -th entry of the probability vector in the last generation;
 λ is a constant ($0 < \lambda < 1$);
 w''_k is a value of the k -th bit of the probability vector \mathbf{w} that is calculated as follows (5):

$$w''_k = \frac{\sum_{i=1}^q (e_k)_i}{q} \quad (5)$$

where $(e_k)_i$ is a value of the k -th bit of the chromosome of the individual α_i ($i = 1, \dots, q$) and it is true $F_i \geq F_{avg}$ for these individuals.

The best individual in the population is included to the next population automatically. Values of the chromosomes of the rest of individuals α ($i = 2, \dots, p$) are calculated for the next generation as follows: if $w_k = 0(1)$, then $(e_k)_i = 0(1)$; if $0 < w_k < 1$ the corresponding $(e_k)_i$ is determined randomly by (6):

$$(e_k)_i = \begin{cases} 1 & \text{if } \text{random} < w_k \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where $k = 1, \dots, p$ ($p =$ number of individuals in the population).

The process of the evolutionary algorithm is ended if the saturation parameter $\tau(\mathbf{w})^*$ is greater than a predefined value.

5 Experiments

In the experimental task, a system (neural network) recognizes a binary pattern and its rotation. Neural network with one hidden layer of units with topology 9-13-8 adapted by backpropagation represents a system here. The creation of such modular system that would solve partial tasks correctly was our target. Basic set of training patterns are organized into a matrix (grid) 3x3, which is represented by binary vector. The direction of rotation is defined towards the basic pattern by four possibilities: (a) 0°-state without rotation, (b) turn 90°, (c) turn 180°, and (d) turn 270°. The training set includes four patterns that are defined in four different states, see Fig 4. Thus, we get 16 different combinations of shapes and their rotations. Eight output units are divided into two subsets of four units. Units in the “shape” subset are responsible for indicating the identity of the input. Each input is associated with one of the four “shape” units, and one of the four rotations. The system is considered to correctly recognize and locate an input.

Parameters of the Experimental part.

- Population (both methods):
Number of individuals: 100.
Neural network architecture: 9 – 13 – 8.
Training algorithm: Backpropagation
(learning rate: 1; momentum: 0; training times: 150 epochs in the partial training).
- Parameters of method from chapter 3:
Probability of mutations: 0.01.
Fix pattern in the second mutation: “0000”.
 r : 5.
Ending conditions: Maximal number of generations: 500.

* $\tau(\mathbf{w}) =$ a number of entries (w_i) of the probability vector \mathbf{w} that are less then w_{eff} or $(1 - w_{eff})$, where w_{eff} is a small positive number.

- Parameters of method from chapter 4:

con : 100; see formula (2).

λ : 0.2; see formula (4).

Ending conditions: The saturation parameter, $\tau(\mathbf{w})$: $0.99 * m * n$

($m=13$, number of hidden units; $n=8$, number of output units); $w_{eff} = 0.01$.

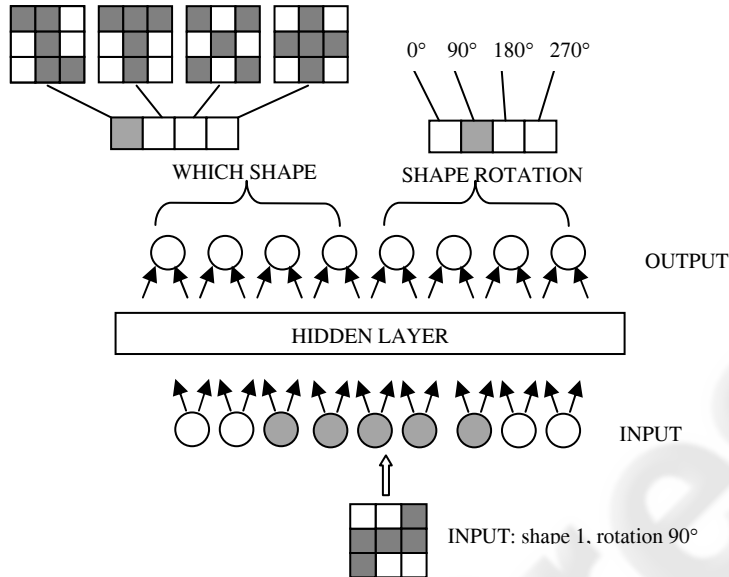


Fig. 4. A defined pattern in a training set.

Table 1 shows a table of results. The table shows an evolution of the best individual in the population. It is evidently seen, the connections among modules are eliminated faster than connection inside modules. These results support also the fact that systems were created dynamically during a learning process. Method from chapter 3 gives the following results: six hidden units of the best individual realise the “shape” task and its four units realise the “rotation” task in the last generation. Method from chapter 4 gives the following results: seven hidden units of the best individual realise the “shape” task and its four units realise the “rotation” task in the last generation. Calculation was terminated, when ending conditions were fulfilled, e.g. for method from chapter 3 was calculation terminated in the 498-th generation and for method from chapter 4 was calculation terminated in the 353-rd generation. Other numerical simulations give very similar results.

Table 1. Table of results.

generation	Method from chapter 3:			Method from chapter 4:		
	number of hidden units: „shape“ task:	number of hidden units: „rotate“ task:	number of interferences:	number of hidden units: „shape“ task:	number of hidden units: „rotate“ task:	number of interferences:
1	1	1	11	0	0	13
100	2	3	8	1	3	9
200	3	3	7	4	3	6
300	4	3	6	6	3	4
400	5	4	4	<i>GENERATION: 353</i>		
	<i>GENERATION: 498</i>			7	4	2
	6	4	3			

We made the following experiment. Neural network with modular architecture (the best individual) and network with the same arrangement of neurons, but by all connections between layers have been adapted via backpropagation to solve the above defined task. For each model was done 10 adaptations, the weight vector was at the beginning of each simulation generated randomly. In Fig. 5 the average error function values is shown: (a) modular neural network and (b) fully connected neural network during the whole calculation. Adaptation of each neural network was terminated after 1500 iterations. The figure shows that the network with a modular architecture, which includes only a limited number of connections, allows to learn the considered problem as efficiently as a monolithic networks designed within an appropriate architecture.

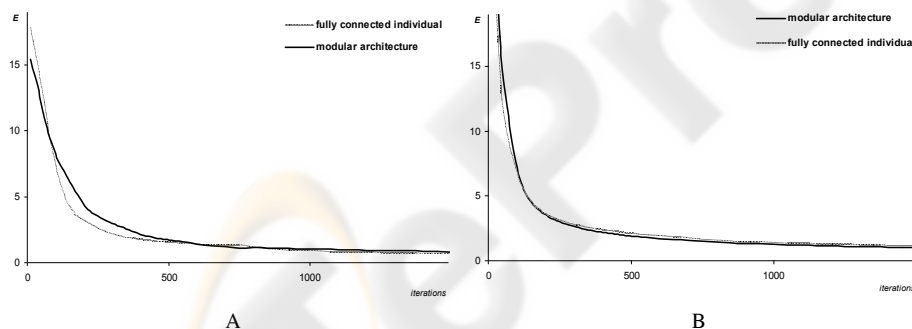


Fig. 5. The history of average error function value during whole calculation A: method from chapter 3; B: method from chapter 4.

6 Conclusions

Both described method are methods of automatic neural net modularization. The problem specific modularisations of the representation emerge through the iteration of the evolutionary algorithm directly with the problem.

When interpreting solutions, we have to be careful, because algorithms' parameters are not the object of the optimization process, but we obtain solutions just in dependence on these parameters. Both numerical simulations reflect the modular

structure significance as a tool of a negative influence interference rejection on neural network adaptation. As the hidden units in the not split network are perceived as some input information processing for output units, where a multiple pattern classification is realized on the basis of diametrically distinct criteria (e.g. neural network has to classify patterns according to their form, location, colors, ...), so in the beginning of an adaptation process the interference can be the reason that output units also get further information about general object classifications than the one which is desired from them. This negative interference influence on running the adaptive process is removed just at the modular neural network architecture, which is proved also by results of the performed experiment. The winning modular network architecture was the product of emergence using evolutionary algorithms. The neural network serves here as a special way of solving the evolutionary algorithm, because of its structure and properties it can be slightly transformed into an individual in evolutionary algorithm.

References

1. Di Fernando, A., Calebretta, R., and Parisi, D. (2001) Evolving modular architectures for neural networks. In French R., and Sougne, J. (eds.) Proceedings of the Sixth Neural Computation and Psychology Workshop: Evolution, Learning and Development. Springer Verlag, London.
2. Fausett, L. V. (1994) Fundamentals of neural networks. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Hampshire, J. and Waibel, A. The Meta-Pi network: Building distributed knowledge representation for robust pattern recognition. Technical Report CMU-CS-89-166. Pittsburgh, PA: Carnegie Mellon University.
4. Jacobs, R. A., Jordan, M. I., Nowlan, S.J., and Hinton, G. E. (1991) Adaptive mixtures of local experts. *Neural Computation*, 3, pp.79-97.
5. Jacobs, R. A., Jordan, M. I. (1992). Computational consequences of a bias toward short connections. *Journal of Cognitive Neuroscience*, 4, 323–336.
6. Jacobs, R. A. (1994) Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6, 181-214.
7. Jordan, M. I. and Jacobs, R. A. (1995) Modular and Hierarchical Learning Systems. In M. A. Arbib (Ed) *The Handbook of Brain Theory and Neural Networks*. pp 579-581.
8. Kvasnička, V; Pelikán, M.; Pospíchal, J. (1996) Hill climbing with learning (an abstraction of genetic algorithm). *Neural network world* 5, 773-796.
9. Rueckl, J. G. (1989) Why are “What” and “Where” processed by separate cortical visual systems? A computational investigation. *Journal of Cognitive Neuroscience* 2, 171-186.
10. Volna, E. (2002) Neural structure as a modular developmental system. In P. Sinčák, J. Vašček, V. Kvasnička, J. Pospíchal (eds.): *Intelligent technologies – theory and applications*. IOS Press, Amsterdam, pp.55-60.
11. Volna, E. (2007) Designing Modular Artificial Neural Network through Evolution. In J. Marques de Sá, L. A. Alexandre, W. Duch, and D. P. Mandic (eds.) *Artificial Neural Networks – ICANN’07, Lecture Notes in Computer Science*, vol. 4668, Springer-Verlag series, pp 299-308.