Optimal Genetic Representation of Complete Strictly-Layered Feedforward Neural Networks

Spyros Raptis, Spyros Tzafestas, and Hermione Karagianni

Intelligent Robotics and Automation Laboratory
Division of Signals, Control and Robotics
Department of Electrical and Electronic Engineering
National Technical University of Athens
Zographou Campus, 15773, Athens, GREECE
tzafesta@softlab.ece.ntua.gr

Abstract. The automatic evolution of neural networks is both an attractive and a rewarding task. The connectivity matrix is the most common way of directly encoding a neural network for the purpose of genetic optimization. However, this representation presents several disadvantages mostly stemming from its inherent redundancy and its lack of robustness. We propose a novel representation scheme for encoding complete strictly-layered feedforward neural networks and prove that it is optimal in the sense that it utilizes the minimum possible number of bits. We argue that this scheme has a number of important advantages over the direct encoding of the connectivity matrix. It does not suffer from the curse of dimensionality, it allows only legal networks to be represented which relieves the genetic algorithm from a number of checking and rejections, and the mapping from the genotypes to phenotypes is one-to-one. Additionally, the resulting networks have a simpler structure assuring an easier learning phase.

1 Introduction

As stochastic search processes, genetic algorithms (GA's) provide no estimation on the time required to locate an adequate solution to a given problem. They use very limited (if any) a priori information on the specific problem they are addressing. In this sense, they are inadequate for on-line execution in time-sensitive problems.

On the contrary, GA's have very often been used as a high-level tool to evolve other systems that are more suited for on-line performance. Indicative examples include the definition of the input partitions or other parameters of a fuzzy system, the determination of appropriate values for the parameters of another GA, the design of an appropriate set of detectors for pattern recognition, etc. The evolutionary design of neural networks by genetic means may very well be placed in this context.

In spite of the intense research in the area of neural networks, globally applicable rules for their design are still missing and their development is still based on heuristics and rules of thumb. Design parameters include the network topology, the determination of its connectivity pattern, the selection of the neuron activation functions, the training algorithm used to calculate the weights of the connections, etc.

The design of a neural network by genetic means restates the problem in the context of an optimization process. Automatically evolving a network that is "optimal" (in the sense of a certain criterion) is particularly attractive since it offers a general methodology for the design of a "well-behaving" neural system.

The work in the field of evolving neural networks is quite extensive and certain surveys are also available (e.g. [1]). The various approaches may be roughly divided into categories based on different characteristics of the process.

Based on the genotypic representation used we can identify (a) *direct encoding schemes* (e.g. [2], [3], [4]) where the GA uses a simple and easily decodable representation of the neural network such as its connectivity matrix and (b) *indirect encoding schemes* (e.g. [5], [6], [7]) such as production rules, grammars, etc. where decoding is not so trivial. Based on the type of network being evolved, we may identify approaches for the design of (a) *feedforward networks* or (b) *recurrent networks*. The selection of the required class of neural networks is, clearly, problem-dependent. A last classification can be identified that is based on the type of network parameters being evolved. We can identify methods that (a) optimize only the *network topology*, (b) optimize the values of the *synaptic weights* and other parameters of a fixed network, or (c) optimize *both* the topology and the parameter values of the network.

In the remainder of the paper we will only consider the case of *complete strictly-layered feedforward neural networks*. A neural network is said to be *layered* if its hidden neurons are organized in layers. No connection can appear among neurons of the same layer. A network is *strictly-layered* if it is layered and the neurons of a layer can only accept inputs from neurons of the immediately preceding layer. A network is *complete* if every neuron of a layer is connected to all the neuron in the previous layer.

2 Connectivity Matrices

The connectivity matrix is the most commonly used representation of a neural network. It is a square binary matrix, C, of dimension equal to the total number of neurons in the network (including input, output and hidden neurons). Each element of the matrix, say $C(i,j)$, indicates the presence or absence of a connection from neuron i to neuron j. For a feedforward network the connectivity matrix has an upper-triangular form.

In the connectivity matrix of a feedforward neural network, the number of elements that can be non-zero is given by:

$$n_{gene} = (n+h)(m+h) - \tfrac{1}{2}h(h+1) \qquad (1)$$

where n is the number of network inputs, m is the number of network outputs, and h is the number of hidden neurons.

The representation of neural connectivity patterns through their respective connectivity matrix contains redundant information. Thus, while the two networks displayed in Fig. 1 are functionally equivalent, their respective connectivity matrices differ. Of course, by appropriate renumbering (i.e. exchanging columns and rows) the

two matrices can be made identical. The implementation of an algorithm that can transform the different versions of a matrix to a single is a relatively straightforward task.

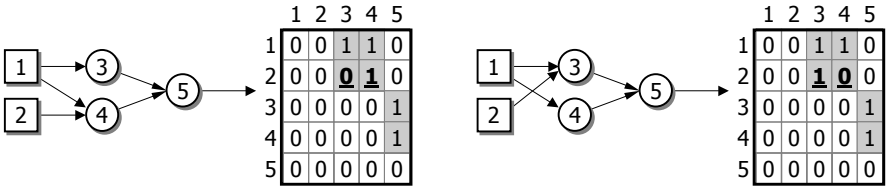

Fig. 1. Two equivalent neural networks

In the context of genetic search, a representation that relies on a direct coding of the significant bits of the connectivity matrix is bound to such a "symmetry" and the formulated mapping from the genotype domain (coded individuals) to the phenotype domain (neural networks) is, inherently, many-to-one. This is true not only for strictly layered neural networks but for any type of feedforward network as well.

For a feedforward network, the required number of bits for coding only the necessary information of the connectivity matrix is given by Eq. (1). Thus, a typical genetic representation would need to utilize n_{gene} bits.

It is important to note that random combinations of these bits do not always produce legal networks. To quantify this argument, it is interesting to get an estimate of the percentage of legal networks to the illegal ones. To this end, let us consider the simple problem of designing a feedforward neural network with two inputs (n=2), one output (m=1), and (up to) four hidden neurons (h=4). This gives a total of seven neurons. The dimension of the connectivity matrix of such a network will be 7×7 containing 20 significant bits (i.e. bits that can be non-zero) thus leading to 2^{20} different possible combinations. A set of 100000 such bit series were randomly produced and the respective networks were constructed and simplified. The statistics for the resulting networks are shown in Table 1.

Table 1. Statistics for 100000 neural networks of dimension 2×4×1 randomly produced using direct coding of the significant bits of the connectivity matrix

Hidden Neurons	%
0 (illegal)	11%
1	17%
2	25%
3	28%
4	19%
	100%

Num of Groups	%
0 (illegal)	11%
1	24%
2	37%
3	23%
4	5%
	100%

Num of Layers[1]	%
1	89.80%
2	9.30%
3	0.87%
4	0.03%
	100%

[1] Only 7% of generated networks presented layering. The percentages in this table are with respect to those layered networks.

The statistics become even more problematic for the case of more complex networks. Therefore, for realistic problems it is almost impossible to control the quality of the networks that are randomly created based on a direct coding of the significant bits of the connectivity matrix and evaluated by a genetic algorithm. Additionally, it is quite rare for a layered network to randomly come by. Thus, for networks of this type a different encoding scheme is obviously required.

From the discussion above, it is quite obvious that a different network representation scheme is necessary especially for the case of genetically evolving strictly layered neural networks.

3 The Proposed Representation Scheme

Let us consider again the case of a strictly-layered feedforward neural network of n inputs, m outputs and h hidden neurons. Then the total number of neurons on the network will be $N=n+m+h$.

Consider a string of h binary digits receiving the values "0" or "1". If we assign to the symbol "1" the meaning "*new layer*" and to the symbol "0" the meaning "*same layer*", then we may directly express through a string of h such symbols any *complete strictly-layered feedforward neural network*.

This way, the complete neural network of Fig. 2 having n=3, m=2 and h=7 (N=12), can be represented by the simple string "1000100". In its general form, the proposed representation can be depicted as shown in the Fig. 3.

Since, the number of 1's in the string explicitly controls the number of layers in the resulting network we may directly favor "shallower" network architectures by introducing an appropriate bias during the random instantiation of such strings which assigns higher probability to 0's than to 1's. Additionally, we may impose the demand for at least one hidden layer by fixing the leftmost bit of the string to "1" and allowing only the other h-1 bits to vary. In this case, however, the number of hidden neurons in all resulting networks will be *exactly h* as discussed below.

By relaxing the requirement for at least one hidden layer (thus allowing the leftmost digit of the string to receive the value "0") and omitting all the "0" that appear in the left side of the first "1" in the string, we may directly use the same scheme to represent networks of fewer hidden nodes or even without any hidden node (string consisting entirely of 0's). Such an approach offers additional properties to the representation scheme, such as its independence from the problem dimensions, since:

$$\underbrace{0\overbrace{0\ldots0}^{t}1\overbrace{**\ldots*}^{k}}_{t+k+1} \equiv \underbrace{1\overbrace{**\ldots*}^{k}}_{k+1} \qquad (2)$$

4 Properties and Advantages

Compared to the direct encoding of the significant digits of the connectivity matrix, the proposed representation scheme presents a set of significant advantages.

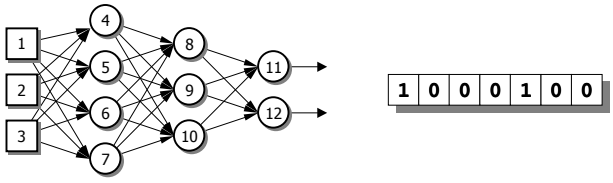

Fig. 2. The representation of the complete strictly-layered feedforward neural network of size 3×4×3×2 (n=3, m=2 and h=7)

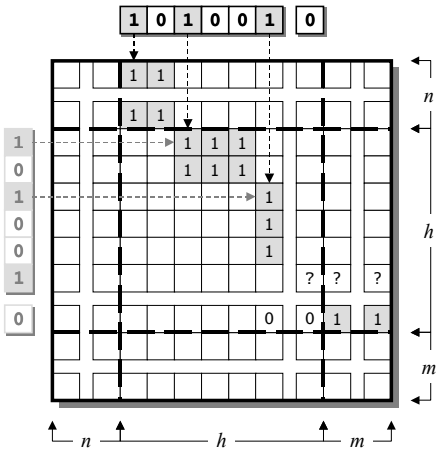

Fig. 3. The representation of a complete strictly-layered feedforward neural network

Adequacy: The direct encoding scheme of the connectivity matrix allows for the representation of legal neural networks that the proposed model cannot. Examples of such networks are networks that are not complete (i.e. networks whose neurons are not fully interconnected among successive layers) or not strictly layered (i.e. networks whose neurons may be arbitrarily interconnected, e.g. input neurons directly connected to output neurons). However, such an economy on the number of connections (as in the case of non-complete networks) or the generalized "sprawl" connectivity patterns (as in the case of networks that are not strictly layered), do not necessarily lead to more powerful or adequate networks. Not only is there no evidence that strictly-layered networks are by any means inferior to more generalized networks but, on the contrary, such networks tend to present benefits during the training phase due to their simplified structure.

Complexity: An important advantage of the proposed representation scheme is its computational simplicity: it requires $\mathbf{O}(h)$ digits as opposed to the $\mathbf{O}(h^2)$ required by the direct encoding scheme, thus avoiding the well known "curse of dimensionality" problem. So, the search space is effectively reduced and the problem of determining adequate network structures is kept tractable even for relatively large networks. For example, for the case of the network of Fig. 2, the proposed scheme requires just 7 bits (introducing a search space of 2^7 configurations) as opposed to the 62 bits

required by the direct encoding scheme (leading to a search space of 2^{62} configurations)!

Correctness. The proposed scheme does not allow the representation of illegal networks configurations. So, any arbitrary string of digits corresponds to a legal network configuration. This property relieves the genetic search process from the need to perform any checking either to the randomly generated individuals or to the individuals that result from the application of genetic operators during recombination. On the contrary, the direct encoding scheme cannot guarantee the validity of the network configurations involved in the search process, rendering the extensive use of checking absolutely necessary during the initialization and recombination phases. Such a checking will result in certain individuals being rejected or "punished" by assigning particularly low fitness values during their evaluation. However, such checking and rejection can introduce severe obstacles to the search process since the genotypic resemblance (which is a main driving force of a genetic algorithm) will no longer imply phenotypic resemblance of even similarity in the assigned fitness values.

Properties of the Mapping. It can be shown that the genotypic-to-phenotypic correspondence obtained by the proposed scheme is "1-1" and "over" mapping from the domain of binary strings of appropriate length to the domain of neural network configurations. This suggests that (a) two different strings can never correspond to the same network, and (b) for *any* complete strictly-layered feedforward neural network there is a corresponding string to describe it.

5 Mathematical Proof of the Optimality of the Mapping

The main target of the present paragraph is to determine the cardinality of the set of complete strictly-layered feedforward neural networks.

In Theorem 3, we prove that the cardinality of the set of complete strictly-layered feedforward neural networks that contain *at most h* hidden neurons equals 2^h. Similarly, the cardinality of the set of strings the are composed of h binary digits is also 2^h. Thus, since the cardinality of the set of such neural networks and of the set of binary strings is the same, and given that the performed mapping is 1-1, we can prove that the proposed representation scheme is *optimal* in the sense that it is *minimal to the size of the required bits*. I.e. there is no scheme that can represent the set of complete strictly-layered feedforward neural networks using a smaller number of bits.

Let us consider a neural network containing *at most h* hidden neurons. We wish to determine the number of all its possible configurations, i.e. the number of ways h or less neurons can be partitioned to any number of hidden layers. Obviously, each layer can contain any number of neurons between 1 and h and the sum of all the neurons must not be higher that h.

Fig. 4 displays all the possible configurations of a complete strictly-layered feedforward neural network having h=4 or less hidden neurons.

We will first calculate the number of possible configurations of a complete strictly-layered feedforward neural network of *exactly h* hidden neurons divided in *exactly l* hidden layers where, obviously, $l \leq h$.

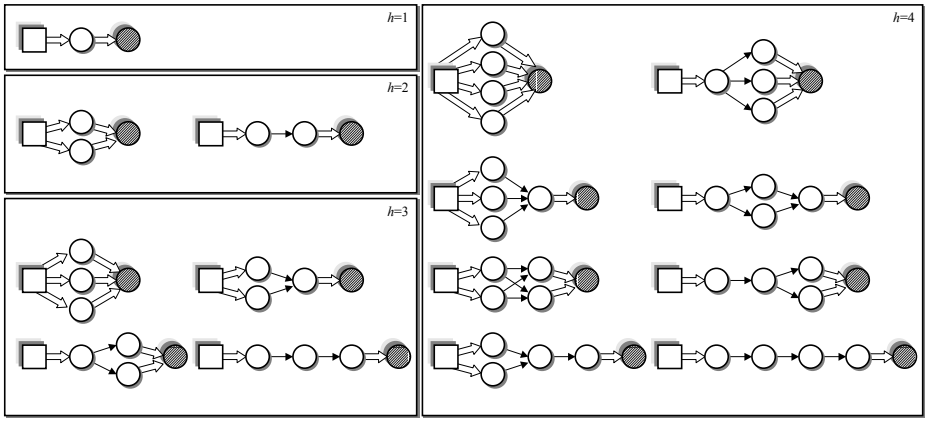

Fig. 4. Graphical representation of all the possible configurations of a complete strictly-layered feedforward neural network having *at most* h=4 hidden neurons. The subplots display all the configurations having *exactly* (a) h=1, (b) h=2, (c) h=3, and (d) h=4 neurons. The number of inputs and outputs is arbitrary. The thick arrows represent many-to-one and one-to-many connections.

Theorem 1. The number of possible configurations of a complete strictly-layered feedforward neural network that contains *exactly* h hidden neurons divided in *exactly* l hidden layers is given by:

$$N_l^h = \frac{(h-1)!}{(h-l)!(l-1)!} \tag{3}$$

Proof. The problem of determining the number of possible configurations for exactly h neurons divided in exactly l layers is equivalent to the following problem:

"Consider a binary string of h 0's. We want to determine the number of ways we can insert l-1 1's between the 0's. The insertions take place at the h-1 spaces between the zeros and each such space can only accept up to one 1."

In the above transformed version of the problem, the 0's play the role of neurons and the 1's the role of barriers separating consecutive layers.

The number possible combinations for placing r identical objects to n numbered boxes with no more than one object per box is given by:

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!} \tag{4}$$

So, for r=l-1 and n=h-1, Eq. (4) takes the form:

$$N_l^h = C(h-1,l-1) = \frac{(h-1)!}{(h-l)!(l-1)!}$$

Q.E.D.■

We will proceed with the calculation of the number of possible configurations of a complete strictly-layered feedforward neural network with exactly h hidden neurons distributed in any number of hidden layers.

Theorem 2[2]. The number of possible configurations of a complete strictly-layered feedforward neural network with exactly h hidden neurons distributed in any number of hidden layers is given by:

$$N^h = 2^{h-1} \tag{5}$$

Proof. Obviously, h hidden neurons can form any number from 1 (vertical form) to h (horizontal form) hidden layers as shown in Fig. 5. So, the total number of different configurations can be calculated as the sum of the quantities given by Theorem 1 Eq. (3) for l=1...h, i.e.

$$N^h = \sum_{l=1}^{h} N_l^h = \sum_{l=1}^{h} \frac{(h-1)!}{(h-l)!(l-1)!} = \sum_{l=1}^{h} C(h-1,h-l) = \sum_{l=1}^{h} \binom{h-1}{h-l} = \sum_{l=1}^{h} \binom{h-1}{l-1} \tag{6}$$

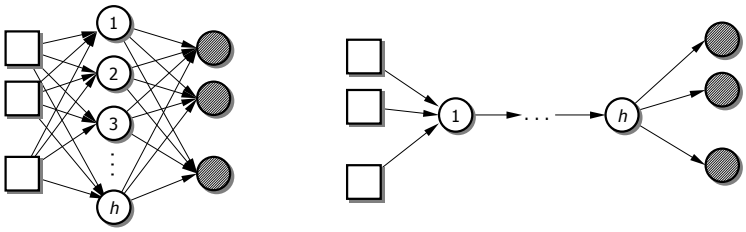

Fig. 5. Two extreme configurations of a network with h hidden neurons

It is known that:

$$\binom{h-1}{h-l} = \binom{h-1}{l-1} \Rightarrow \binom{h}{l} = \binom{h-1}{l} + \binom{h-1}{l-1} \tag{7}$$

Thus, summing from l=1 to h-1, we get:

$$\sum_{l=1}^{h-1} \binom{h-1}{l-1} = \sum_{l=1}^{h-1} \binom{h}{l} - \sum_{l=1}^{h-1} \binom{h-1}{l} \tag{8}$$

Employing the binomial expansion of $(x+y)^2$ for x=y=1 and n=h we can arrive at:

$$\sum_{l=1}^{h} \binom{h}{l} = 2^h - 1 \text{ and } \sum_{l=1}^{h-1} \binom{h-1}{l} = 2^{h-1} - 1 \tag{9}, \tag{10}$$

We have:

$$\sum_{l=1}^{h} \binom{h}{l} = \sum_{l=1}^{h-1} \binom{h}{l} + \binom{h}{l} \overset{Eq.(9)}{\Rightarrow} 2^h - 1 = \sum_{l=1}^{h-1} \binom{h}{l} + 1 \Rightarrow \sum_{l=1}^{h-1} \binom{h}{l} = 2^h - 2 \tag{11}$$

---

[2] It is easy to see that the problem of determining N^h is equivalent to the problem of counting the additive partitions of integer h, which is a typical problem in the field of discrete mathematics.

So:

$$Eq.(8) \underset{Eq.(11)}{\overset{Eq.(10)}{\Rightarrow}} \sum_{l=1}^{h-1}\binom{h-1}{l-1} = \left(2^h - 2\right) - \left(2^{h-1} - 1\right) = 2^{h-1} - 1 \qquad (\mathbf{12})$$

Finally:

$$\sum_{l=1}^{h}\binom{h-1}{l-1} = \sum_{l=1}^{h-1}\binom{h-1}{l-1} + \binom{h-1}{h-1} \overset{Eq.(12)}{\Rightarrow} \sum_{l=1}^{h}\binom{h-1}{l-1} = \left(2^{h-1} - 2\right) + 1 = 2^{h-1}$$

Q.E.D.■

The proposed scheme uses h bits for the representation networks having *up to h* hidden neurons (and so does an $h \times h$ connectivity matrix). Such networks of lower dimension can be represented by padding an appropriate number of 0's to the left of the string (or by inserting rows and columns of all zeros in a connectivity matrix). The following theorem takes these lower dimension networks into account.

Theorem 3. The number of all possible configurations of a complete strictly-layered feedforward neural network having up to h hidden nodes distributed in any number of hidden layers is given by:

$$N^{[1,h]} = 2^h \qquad (\mathbf{13})$$

Proof. The number of configurations of *up to h* neurons is, clearly, the sum of the number of configurations of exactly i neurons, for i=1...h.
Obviously, this sum is:

$$N^{[1,h]} = \sum_{i=1}^{h} N^i = \sum_{i=1}^{h} 2^{i-1} = \tfrac{1}{2}\sum_{i=1}^{h} 2^i = 2^h$$

Q.E.D.■

References

1. X. Yao, "Evolving Artificial Neural Networks," *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1423-1447, September (1999)
2. T. R. J. Bossomaier and N. Snoad, "Evolution and Modularity in Neural Networks," in *Proc. IEEE Workshop on Non-Linear Signal and Image Processing*, I. Pitas, Ed., IEEE, Thessaloniki, Greece, pp. 282-292 (1992)
3. D. Dasgupta and D. R. McGregor, "Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm," in *Proc. COGANN-92: Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks*, L. D. Whitley and J. D. Schaffer (Eds.), IEEE Computer Society Press, June 5 (1992)
4. D. Whitley and C. Bogart, "The evolution of connectivity: pruning neural networks using genetic algorithms," in *Proc. Int'l Joint Conf. on Neural Networks*, Vol. I, pp. 134-137, Washington, DC, Lawrence Erlbaum Associates, Hillsdale, NJ (1990)
5. N. J. Radcliffe, "Genetic Set Recombination and its Application to Neural Network Topology Optimization," *Neural Computing and Applications*, Vol. 1, No. 1, pp. 67-90 (1993)
6. V. Maniezzo, "Genetic Evolution of the Network Topology and Weight Distribution of Neural Networks," *IEEE Trans. Neural Networks*, Vol. 5, No. 1, pp. 39-53, January (1994)
7. M. Mandischer, "Representation and evolution of neural networks," in R. F. Albrecht, C. R. Reeves and U. C. Steele (Eds.), *Artificial Neural Nets and Genetic Algorithms*, pp. 643-649 (1993)